SoftPython

Guida introduttiva alla pulizia e analisi dati con Python 3

David Leoni, Alessio Zamboni, Marco Caresia

Mar 08, 2021

Copyright © 2021 by David Leoni, Alessio Zamboni, Marco Caresia.

SoftPython is available under the Creative Commons Attribution 4.0 International License, granting you the right to copy, redistribute, modify, and sell it, so long as you attribute the original to David Leoni, Alessio Zamboni, Marco Caresia and identify any changes that you have made. Full terms of the license are available at:

http://creativecommons.org/licenses/by/4.0/

The complete book can be found online for free at:

https://it.softpython.org

CONTENTS

Prefazione . . . 1
News . . . 1

1 Panoramica **3**
1.1 A chi è rivolto . . . 3
1.2 Requisiti . . . 3
1.3 Contenuti . . . 3
1.4 Autori . . . 6
1.5 Licenza . . . 6
1.6 Ringraziamenti . . . 7

2 Come orientarsi **9**
2.1 Capitoli . . . 9
2.2 Perchè Python? . . . 10
2.3 Approccio e obiettivi . . . 10
2.4 Non funziona, che faccio ? . . . 11
2.5 Installazione e strumenti . . . 12
2.6 Cominciamo ! . . . 12

3 Installazione **13**
3.1 Installazione Python . . . 14
3.2 Installare pacchetti . . . 17
3.3 Notebook Jupyter . . . 18
3.4 Fare progetti Python con gli ambienti virtuali . . . 22
3.5 Approfondimenti . . . 24

4 A - Fondamenti **25**
4.1 Introduzione rapida a Python . . . 25
4.2 Strumenti e script . . . 64
4.3 Basi Python - variabili e interi . . . 84
4.4 Basi 2 - booleani . . . 101
4.5 Basi 3 - numeri reali . . . 115
4.6 Basi 4 - Challenges . . . 128
4.7 Stringhe 1 - introduzione . . . 133
4.8 Stringhe 2 - operatori . . . 151
4.9 Stringhe 3 - metodi . . . 177
4.10 Stringhe 4 - iterazione e funzioni . . . 206
4.11 Stringhe 5 - Challenges . . . 219
4.12 Liste 1 - Introduzione . . . 220
4.13 Liste 2 - operatori . . . 232
4.14 Liste 3 - Metodi . . . 264

4.15 Liste 4 - iterazione e funzioni . . . 298
4.16 Tuple . . . 317
4.17 Insiemi . . . 340
4.18 Dizionari 1 - Introduzione . . . 373
4.19 Dizionari 2 - operatori . . . 389
4.20 Dizionari 3 - metodi e classi . . . 415
4.21 Dizionari - iterazione e funzioni . . . 437
4.22 Dizionari - strutture composte . . . 455
4.23 Condizionali - if else . . . 465
4.24 Controllo di flusso - Cicli for . . . 491
4.25 Controllo di flusso - cicli while . . . 566
4.26 Sequenze e comprehensions . . . 602
4.27 Funzioni - soluzioni . . . 627
4.28 Gestione degli errori e testing . . . 639
4.29 Matrici - Liste di liste . . . 653
4.30 Matrici - Numpy . . . 704
4.31 Comandamenti . . . 735

5 B - Analisi dati 743
5.1 Formati dati . . . 743
5.2 Visualizzazione dati . . . 784
5.3 Ricerca - espressioni regolari . . . 817
5.4 Integrazione dati . . . 854
5.5 Estrazione dati . . . 880
5.6 Pandas . . . 904
5.7 Information retrieval . . . 938
5.8 Computer vision . . . 954

6 C - Applicazioni 959
6.1 Interfacce grafiche . . . 959
6.2 Mappe interattive . . . 984
6.3 Esempio webapp . . . 987
6.4 Database . . . 989
6.5 Web development . . . 1017

7 D - Progetti 1021
7.1 Come fare un progetto . . . 1021
7.2 Progetto SoftPython - Template . . . 1024
7.3 Markdown . . . 1029
7.4 Idee per progetti . . . 1034
7.5 Challenges . . . 1054

8 Riferimenti 1057
8.1 Riferimenti per argomento . . . 1057
8.2 Libro Passo dopo passo impariamo a programmare con Python . . . 1060
8.3 Lezioni di Nicola Cassetta . . . 1060
8.4 Pensare in Python seconda edizione . . . 1060
8.5 W3Resources website . . . 1061
8.6 Corso Python 3 di Nicola Zoppetti . . . 1061
8.7 Guida Introduttiva a Python 3 guida ufficiale . . . 1061
8.8 Immersione in Python 3 . . . 1062
8.9 Corso Scientific Programming Master Data Science - Trento . . . 1062
8.10 Geeks for Geeks . . . 1063
8.11 Introduction to Scientific Programming with Python . . . 1063

Prefazione

Guida introduttiva e gratuita alla pulizia e analisi dati con Python 3.

Ricca di esercizi svolti e adatta sia a principianti che studenti con conoscenze tecnico/scientifiche.

Al giorno d'oggi sempre più decisioni vengono prese in base a dati fattuali e oggettivi. Tutte le discipline, dall'ingegneria alle scienze sociali, necessitano oramai capacità di elaborare dati ed estrarre informazioni utili dall'analisi di fonti eterogenee. Questo sito di esercizi pratici fornisce quindi un'introduzione al processamento dati usando Python[1], un linguaggio di programmazione popolare sia nell'industria che nell'ambito della ricerca.

News

8 marzo 2021:

- aggiornato *stringhe*, aggiunto esercizi e challenge

28 febbraio 2021:

- Diviso basics in *1 - Variabili e interi* , *2- Booleani* , *3 - Numeri reali* e *4- Challenges*
- raccolto tutti i riferimenti sparsi nella pagina *Riferimenti*

18 ottobre 2020: le soluzioni sul sito adesso si vedono solo cliccando su bottoni 'Mostra soluzione'

3 ottobre 2020: Aggiunto pagina *Riferimenti* (spostata da Come orientarsi)

6 agosto 2020

- Creato pagina *Strumenti e script* **estraendola da basics**
- aggiunto paragrafo sui NaN in foglio *numpy*

27 Luglio 2020 - Restyling!

13 Luglio 2020 - Spostato sito su: it.softpython.org[2]

Vecchie news: link

[1] http://www.python.it/
[2] https://it.softpython.org

CHAPTER

ONE

PANORAMICA

1.1 A chi è rivolto

Il materiale in questo sito è rivolto sia a principianti assoluti che a studenti con background più tecnico che desiderino acquisire conoscenze pratiche riguardo l'estrazione, la pulizia, l'analisi e visualizzazione di dati (tra i framework Python usati vi saranno Pandas, Numpy e l'editor Jupyter). Per superare eventuali difficoltà e garantire risultati didattici concreti, le guide presentate sono passo-passo.

1.2 Requisiti

Qualche conoscenza base di programmazione, in un qualsiasi linguaggio è utile ma non strettamente necessaria. Tutti i materiali proposti sono in lingua italiana.

1.3 Contenuti

- *Come orientarsi*: Approccio e obbiettivi
- *Riferimenti*

1.3.1 A - Fondamenti

1. *Installazione*
2. *Introduzione veloce a Python* (per chi ha già conoscenze di programmazione)
3. *Strumenti e script*
4. Basi (per principianti assoluti)
    1. *variabili e interi*
    2. *booleani*
    3. *numeri reali*
    4. *challenges*
5. Stringhe
    1. *introduzione*
    2. *operatori*

    3. *metodi*
    4. *altri esercizi*
    5. *challenges*
6. Liste
    1. *introduzione*
    2. *operatori*
    3. *metodi*
    4. *iterazione e funzioni*
7. *Tuple*
8. *Insiemi*
9. Dizionari
    1. *introduzione*
    2. *operatori*
    3. *metodi*
    4. *iterazione e funzioni*
    5. *strutture composte*
10. Controllo di flusso
    1. *Condizionali if*
    2. *Cicli for*
    3. *Cicli while*
11. *Sequenze e comprehensions*
12. *Funzioni*
13. *Gestione errori e testing*
14. *Matrici - liste di liste*
15. *Matrici - Numpy*

1.3.2 B - Analisi dati

1. *Formati dei dati*
    - Presentazione di formati dati comuni (CSV, JSON, XML, file binari)
    - Convertire dati, trattare errori e dati mal-formattati
    - es. impianti funiviari Trentino
2. *Visualizzazione dati*
    - Plotting di grafici in Matplotlib.
    - Esporre dati online con DataWrapper, RawGraphs, UMap
3. *Ricerca*
    - Ricerca in dati testuali usando espressioni regolari

    - es. fermate autobus trentino
4. *Integrazione dati*
    - Esempio di integrazione dati: scaricamento di un dataset opendata (agritur del Trentino)
    - pulizia e posizionamento dei punti di interesse su OpenStreetMap usando un servizio di georeferenziazione
5. *Estrazione Dati*
    - Estrazione di testo rilevante da una pagina HTML usando BeautifulSoup
    - es. eventi del Trentino
6. *Pandas*
    - Analisi di un dataset contenente dati numerici con la libreria Pandas (es. statistiche sensori stazione spaziale)
    - Calcolo di correlazioni tra fattori (es. meteo trentino)
7. *Information retrieval*
    - Ricerca per rilevanza
    - Ricerca per similarità

1.3.3 C - Applicazioni

1. *Interfacce utente*
    - Creazione di interfacce per analisi interattiva di dati con Jupyter Widgets
    - Discussione di alternative (QT, wxWidget)
2. *Integrazione con database*
    - Estrazione di dati da un database SQLite con Pandas
    - Esecuzione di semplici ricerche SQL
3. *Web development*
    - Cenni sviluppo di un semplice server web

1.3.4 D - Progetti

1. *Realizzare progetti*
    - ambienti virtuali
    - installazione librerie
2. *Scrittura testo in Markdown / Jupyter*
3. *Idee per progetti*
4. *Challenges*

1.4 Autori

David Leoni (autore principale): Software engineer specializzato in data integration e web semantico, ha realizzato applicazioni in ambito open data e medico in Italia e all'estero. Dal 2019 è presidente dell'associazione CoderDolomiti, con cui insieme a Marco Caresia gestisce il movimento di volontariato CoderDojo Trento dove da anni insegnano programmazione creativa ai ragazzi. Email: david.leoni@unitn.it Sito: davidleoni.it[3]

Marco Caresia (assistente Edizione Autunno 2017 @DISI Università di Trento): E' stato docente di informatica presso la Scuola Professionale Einaudi di Bolzano. E' presidente della delegazione Trentino Alto Adige Südtirol dell'Associazione Italiana Formatori e vicepresidente dell'associazione CoderDolomiti.

Alessio Zamboni (assistente Edizione Marzo 2018 @Sociologia Università di Trento): Data scientist e software engineer con esperienza in NLP, GIS e gestione del knowledge. Ha collaborato in numerosi progetti di ricerca, collezionando esperienze in Europa e in Asia. Sostiene con convinzione che *"la programmazione è una forma d'arte"*

Luca Bosotti (assistente edizione Summerschool Data Science 2020 e seminari 2021 @Sociologia, Università di Trento): Developer, scientist e professore. Crede che il mondo stia diventando sempre più complicato e interessante. Perciò quello che dobbiamo fare è studiarlo e per farlo ci tocca sfruttare tutte le potenzialità ed il ragionamento possibile. Ha insegnato a giovani di tutte le età, dalle scuole elementari al livello universitario ed è sempre rimasto colpito da quanto siano differenti le persone che si approcciano alla programmazione.

Massimiliano Luca (docente Edizione Estate 2019 @Sociologia Università di Trento): Adora imparare nuove tecnologie ogni giorno. Particolarmente interessato in knowledge representation, data integration, data modeling e computational social science. Crede fermamente che sia vitale introdurre i giovani alla computer science, a tal fine è mentor al Coder Dojo DISI Master.

1.5 Licenza

Questo sito è stato realizzato con stanziamenti del Dipartimento di Ingegneria e Scienze dell'Informazione (DISI)[4] dell'Università di Trento

UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Tutto il materiale in questo sito è distribuito con licenza CC-BY 4.0 Internazionale https://creativecommons.org/licenses/by/4.0/deed.it

Si può quindi liberamente ridistribuire e modificare il contenuto, basta citare l'Università di Trento / DISI e *gli autori*

Note tecniche: tutte le pagine del sito sono fogli Jupyter facilmente modificabili - sono stati convertiti in pagine web tramite NBSphinx[5] usando il template Jupman[6]. I sorgenti del testo si trovano su Github all'indirizzo https://github.com/

[3] http://davidleoni.it
[4] https://www.disi.unitn.it
[5] https://nbsphinx.readthedocs.io
[6] https://github.com/DavidLeoni/jupman

DavidLeoni/softpython-it

1.6 Ringraziamenti

Ringraziamo in particolare il professor Alberto Montresor del Dipartimento di Informatica dell'Università di Trento per aver consentito la realizzazione dei corsi dai quali nasce questo materiale, e il progetto Trentino Open Data (dati.trentino.it[7]) per i numerosi dataset forniti.

Gli altri numerosi enti ed aziende che nel tempo hanno contribuito materiale ed idee sono citati in questa pagina

[7] https://dati.trentino.it

CHAPTER

TWO

COME ORIENTARSI

Per cominciare spenderemo due parole sull'approccio e gli obiettivi del libro, e poi ci tufferemo nel codice:

- Presentazione
- Installazione
- Guardiamo Python

2.1 Capitoli

Le guide trattano fondamenti di Python 3, analisi dati (intesa più come processamento dati grezzi che statistiche) e qualche applicazione (dashboard, database, ..)

Cosa *non* trattano:

- teoria programmazione a oggetti
- algoritmi, complessità computazionale
- performance (niente terabyte di dati …)

- debugging avanzato (pdb)
- il testing è solo accennato
- machine learning
- web development è solo accennato

2.2 Perchè Python?

Semplice a sufficienza per iniziare

Versatile, molto usato per

- calcolo scientifico
- applicazioni web
- scripting

diffuso sia nell'industria che nella ricerca - Indice Tiobe[8] - popolarità su Github[9]

Licenza open source & business friendly[10]

- tradotto: potete vendere prodotti commerciali basati su Python senza pagare royalties

2.3 Approccio e obiettivi

Se hai difficoltà con le basi di programmazione:

- **Difficoltà esercizi**: ⊛ , ⊛⊛
- Leggi SoftPython - Parte A - Fondamenti[11]

Se già sai programmare bene:

- **Difficoltà esercizi**: ⊛⊛⊛, ⊛⊛⊛⊛

[8] https://www.tiobe.com/tiobe-index/
[9] https://madnight.github.io/githut/#/pull_requests/2019/1
[10] https://www.python.it/doc/faq/#ci-sono-restrizioni-di-copyright-nell-uso-di-python
[11] https://it.softpython.org/index.html#A---Fondamenti

- Leggi Introduzione veloce a Python[12] e poi vai direttamente alla Parte B - Analisi dati

Ulteriori guide: Alla pagina *Riferimenti* trovi altro materiale sia introduttivo che più avanzato.

2.4 Non funziona, che faccio ?

Sicuramente mentre programmi incontrerai dei problemi, e ti capiterà di fissare misteriosi messaggi di errore sullo schermo . Lo scopo di questo corso non è dare una serie di ricette da imparare a memoria e che funzionano sempre, quanto piuttosto di mettere in grado di muovere i primi passi nel mondo Python con un minimo di disinvoltura. Quindi, quando qualcosa va storto, non perderti d'animo e prova a seguire la seguente lista di passi che potrebbero aiutarti. Cerca di seguire la lista nell'ordine proposto:

1. se in classe, chiedi al prof (se non in classe, vedi ultimi due punti)
2. se in classe, chiedi al vicino che ne sa di più
3. cerca messaggio di errore su Google
   - rimuovi nomi o parti troppo specifici al tuo programma, come numeri di linea, nomi di file, nomi di variabili
   - se sai l'inglese, Stack overflow[13] è il tuo migliore amico
   - se non sai l'inglese: cerca nel Forum italiano di python-it.org[14]
4. Guarda Appendice A - Debug del libro Pensare in Python[15]
   - Syntax error[16]
     - Continuo a fare modifiche ma non cambia nulla.[17]
   - Errori di runtime[18]
     - Il programma non fa assolutamente nulla[19]
     - Il programma si blocca[20]
     - Ciclo infinito[21]
     - Ricorsione infinita[22]
     - Flusso di esecuzione[23]
     - Quando eseguo il programma è sollevata un'eccezione.[24]
     - Ho aggiunto talmente tante istruzioni di stampa che sono sommerso di output[25]
   - Errori di semantica[26]
     - Il mio programma non funziona.[27]

[12] https://it.softpython.org/quick-intro/quick-intro-sol.html
[13] https://stackoverflow.com
[14] http://www.python-it.org/forum
[15] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html
[16] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec238
[17] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec239
[18] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec240
[19] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec241
[20] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec242
[21] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec243
[22] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec244
[23] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec245
[24] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec246
[25] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec247
[26] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec248
[27] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec249

  - Ho una grande e complicata espressione che non fa quello che voglio.[28]
  - Ho una funzione che non restituisce quello che voglio.[29]
  - Sono proprio bloccato e mi serve aiuto.[30]
  - No, ho davvero bisogno di aiuto.[31]

5. Guarda la pagina dei *Riferimenti*, per spunti che possono coprire l'argomento del problema
6. fatti coraggio e fai una domanda su un forum pubblico, come Stack overflow o python-it.org - vedi *come porre domande*.

2.4.1 Come porre domande

IMPORTANTE

Se vuoi fare domande per iscritto su chat/forum pubblici (per esempio quello di python-it[32]), PRIMA LEGGI ASSOLUTAMENTE sia il regolamento del forum / chat (vedi per es. sia Regolamento pyhon-it[33] che questo esempio di post[34])

In sostanza, ti si chiede sempre di esprimere chiaramente le circostanze del problema, mettendo un titolo esplicativo al post / mail e dimostrando di avere speso un po' di tempo (almeno 10 min) cercando una soluzione per conto tuo. Se hai seguito le regole di cui sopra, e per sfortuna trovi programmatori scorbutici che ti rispondono male, ignorali e lascia che se la prendano con la tastiera. Con i prof del corso SoftPython potete stare tranquilli, pur essendo programmatori sono molto pazienti :-)

2.5 Installazione e strumenti

- Se non hai già provveduto ad installare Python 3 e Jupyter, guarda *Installazione*

2.6 Cominciamo !

- **Se hai già conoscenze di programmazione**: puoi guardare *l'introduzione rapida a Python*
- **Se non hai conoscenze di programmazione**: vai a Strumenti e script[35]

[28] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec250
[29] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec251
[30] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec252
[31] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2021.html#sec253
[32] http://www.python-it.org/forum
[33] http://www.python-it.org/forum/index.php?topic=2592.0
[34] http://www.python-it.org/forum/index.php?topic=2580
[35] https://it.softpython.org/tools/tools-sol.html

CHAPTER

THREE

INSTALLAZIONE

In questa guida vedremo se e come installare Python, le librerie addizionali, il notebook Jupyter e infine la gestione degli ambienti virtuali.

A volte non serve nemmeno installare!

Volendo puoi programmare direttamente online con i seguenti servizi

NOTA 1: se vuoi provarne uno, ricordati sempre di controllare che sia Python **3** !

NOTA 2: Come per ogni servizio online, quando è offerto gratuitamente non abusarne. Se cerchi di processare un terabyte di dati al giorno senza pagare una sottoscrizione, rischi che ti neghino il servizio.

Python 3 su repl.it[36]: permette di editare codice Python anche in modo collaborativo con altri utenti, e supporta anche librerie come Matplotlib.

Python Tutor[37]: permette di eseguire una istruzione alla volta offrendo un'utile visualizzazione di cosa accade 'sotto il cofano'

Google Colab[38]: permette di editare in modo collaborativo fogli Jupyter e di salvarli sul Google Drive.

- NOTA 1: potrebbe darsi che non riusciate ad accedere coi vostri account universitari (es. `@studenti.unitn.it`). In tal caso, usate account personali `@gmail.com`
- NOTA 2: l'aspetto 'collaborativo' del Colab è cambiato nel corso del tempo, **fate bene attenzione a cosa succede quando operate in due sullo stesso documento**. Una volta (2017) le modifiche fatte da uno erano immediatamente viste dagli altri, ultimamente (2019) paiono visibili solo quando si salva - peggio, vanno a sovrascrivere eventuali modifiche fatti da altri.

Demo online di Jupyter[39]: a volte funziona non sempre è disponibile. Se riesci ad accedere, ricordati di selezionare nel menù *Kernel->Change kernel->Python 3*

[36] https://repl.it/languages/python3
[37] http://pythontutor.com/visualize.html#py=3
[38] https://colab.research.google.com
[39] http://try.jupyter.org

3.1 Installazione Python

Ci sono vari modi per installare Python 3 e le sue librerie: c'è la distribuzione ufficiale di Python 'liscia' come anche gestori di pacchetti (es. Anaconda) o ambienti preconfezionati (es. Python(x,y)) che forniscono sia Python più varie librerie extra. Inoltre, una volta completata l'installazione, Python 3 contiene un comando `pip` (a volte per python 3 si chiama `pip3`) che permette di installare in seguito da linea di comando eventuali librerie mancanti.

Il modo migliore per scegliere cosa installare dipende da quale sistema operativo si ha e cosa ci si intende fare. In questo libro useremo Python 3 e librerie scientifiche, quindi cercheremo di creare un ambiente adatto a supportare questo scenario.

Attenzione: Prima di installare roba a caso scaricata da internet, leggi bene questa guida!

Abbiamo cercato di renderla abbastanza generica, ma non essendoci stato possibile testare tutti i vari casi potrebbero insorgere dei problemi, se avete qualche dubbio chiedete ai mentor.

Attenzione: non sovrapporre distribuzioni di Python alla stessa versione !

Vista la varietà di metodi di installazione e il fatto che Python si trova già in molti programmi, può darsi che abbiate già Python installato senza saperlo, magari in versione 2, ma a noi serve la 3! Sovrapporre diversi ambienti Python con la stessa versione potrebbe causare problemi, di nuovo se avete qualche dubbio chiedete ai mentor.

3.1.1 Installazione Windows

Per Windows, ti consigliamo di installare la distribuzione Anaconda per Python 3.6[40] o maggiore, che oltre all'gestore di pacchetti nativo di Python `pip`, fornisce anche il più generico gestore di pacchetti da linea di comando `conda`.

Una volta installato, verifica che funzioni così:

1. clicca sull'icona di Windows in basso a sinistra e cerca 'Anaconda Prompt'. Dovrebbe apparire una console dove inserire comandi, con scritto qualcosa come `C:\Users\David>`. NOTA: per lanciare i comandi di Anaconda, usa solo questa console speciale. Se usie la console di default di Windows (`cmd`), Windows non sarà in grado di trovare Python.
2. Nella console di Anaconda, scrivi

```
conda list
```

Dovrebbe apparire una lista di pacchetti installati, tipo

```
# packages in environment at C:\Users\Jane\AppData\Local\Continuum\Anaconda3:
#
alabaster                 0.7.7                    py35_0
anaconda                  4.0.0                np110py35_0
anaconda-client           1.4.0                    py35_0
...
numexpr                   2.5                  np110py35_0
numpy                     1.10.4                   py35_0
odo                       0.4.2                    py35_0
...
yaml                      0.1.6                         0
```

(continues on next page)

[40] https://www.anaconda.com/download/#windows

(continued from previous page)

```
zeromq                    4.1.3                       0
zlib                      1.2.8                       0
```

3. Prova Python3, scrivendo sempre nella console di Anaconda:

```
C:>  python
```

Dovrebbe apparire qualcosa tipo

```
Python 3.6.3 (default, Sep 14 2017, 22:51:06)
MSC v.1900 64 bit (Intel)[GCC 5.4.0 20160609] on win64
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Attenzione: Con Anaconda, devi scrivere `python` invece che `python3`!

Sei hai installato Anaconda per Python3, userà automaticamente la versione di Python corretta semplicemente scrivendo `python`. Se scrivi `python3` riceverai un'errore di file non trovato !

Attenzione: Se hai Anaconda, usa sempre `conda` per installare i moduli Python ! Quindi se nelle istruzioni nei prossimi tutorial vedi scritto `pip3 install qualcosa`, dovrai usare invece `conda install qualcosa`

3.1.2 Installazione Mac

Per gestire al meglio le app installate sul Mac indipendentemente da Python, in genere conviene installare un cosiddetto *gestore di pacchetti*. Ce ne sono vari, e uno tra i più popolari su Mac è Homebrew[41]. Quindi suggeriamo di installare prima Homebrew e poi tramite esso installare Python 3, più eventualmente in seguito altri componenti che ci interessano. In linea di massima, per l'installazione abbiamo tradotto qua in italiano e semplificato questa guida[42].

Attenzione: controlla se hai già un gestore di pacchetti !

Se hai già installato un gestore di pacchetti come per esempio *Conda* (nella distribuzione *Anaconda*), *Rudix*, *Nix*, *Pkgsrc*, *Fink* o *MacPorts*, forse Homebrew non ti serve e ti conviene usare quello che già hai. In questi casi, chiedi ai mentor. Se hai già *Conda/Anaconda*, può andar bene a patto di avere la versione per Python 3.

— 1 Apri il Terminale

Il terminale di MacOS è un'applicazione che puoi usare per accedere alla linea di comando. Come qualunque altra applicazione, la puoi trovare andando in *Finder*, navigando nella cartella *Applicazioni*, e quindi nella cartella *Accessori*. Da lì, fai doppio click sul *Terminale* per aprirlo come ogni altra applicazione. In alternativa, puoi usare *Spotlight* tenendo premuto i tasti *Command* e *Spazio* per trovare il Terminale scrivendone il nome nella barra che appare.

— 2 Installa Homebrew eseguendo nel terminale questo comando:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

[41] https://brew.sh/
[42] https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-local-programming-environment-on-macos

— 3 Aggiungi `/usr/local/bin` al `PATH`

In questo passaggio dal nome inquietante, una volta che l'installazione di Homebrew è completata, ti assicurerai che vengano sempre sempre usate le applicazioni installate con Homebrew, invece che quelle che Mac OS X potrebbe selezionare automaticamente:

— 3.1 Apri un nuovo Terminale.

— 3.2 Da dentro il terminale, digita il comando

```
ls -a
```

Verrà visualizzata la lista di tutti i file presenti nella cartella della home. In questi file, verifica se esiste un file con il seguente nome: `.profile` (nota il punto all'inizio):

- Se esiste: salta al punto seguente
- Se non esiste, per crearlo digita il comando seguente:

  ```
  touch  $HOME/.profile
  ```

— 3.3 Apri con text edit il file `.profile` appena creato dando il comando:

```
open -e  $HOME/.profile
```

— 3.4 In text edit, aggiungi alla fine del file questa riga:

```
export PATH=/usr/local/bin:$PATH
```

— 3.5 Salva e chiudi sia Text Edit che il Terminale

— 4 Verifica che Homebrew sia installato correttamente, digitando in un nuovo Terminale:

```
brew doctor
```

Se non ci sono aggiornamenti da fare, il Terminale dovrebbe mostrare:

```
Your system is ready to brew.
```

Altrimenti, potresti vedere un warning che consiglia di eseguire un altro comando come `brew update` per assicurarti che l'installazione di Homebrew sia aggiornata.

— 5. Installa python3 (Ricordati il '3' !):

```
brew install python3
```

Assieme a Python 3, Homebrew installerà anche il gestore di pacchetti interno di Python `pip3` che useremo in seguito.

— 6 Verifica che Python3 sia installato correttamente. Eseguendo questo comando dovrebbe apparire la scritta "/usr/local/bin/python3":

```
which python3
```

Dopodichè, prova a lanciare

```
python3
```

Dovresti vedere qualcosa di simile:

```
Python 3.6.3 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on mac
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Per uscire da Python, scrivi `exit()` e invio.

3.1.3 Installazione Linux

Fortunatamente, tutte le distribuzioni Linux sono già fornite con gestori di pacchetti per installare facilmente le applicazioni.

- Se hai Ubuntu:
  1. segui la guida di Immersione in Python 3, capitolo 0 - Installare Python[43] in particolare andando alla sotto sezione installazione in Ubuntu[44]
  2. dopo aver completato la guida, installa anche `python3-venv`:

     ```
     sudo apt-get install python3-venv
     ```

- Se *non* hai Ubuntu, guarda questa nota[45] e/o chiedi ai mentor.

Per verificare l'installazione, prova a lanciare da terminale

```
python3
```

Dovresti vedere qualcosa di simile:

```
Python 3.6.3 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

3.2 Installare pacchetti

Si può estendere Python installando diversi pacchetti gratuiti. Il modo migliore per farlo, varia a seconda del sistema operativo e del gestore di pacchetti installato

> **ATTENZIONE**: Andremo ad usare comandi *di sistema*. Se vedi >>> sulla riga dei comandi, vuol dire che sei dentro l'interprete Python e devi prima uscirne: per farlo, scrivi `exit()` e premi Invio.

In quello che segue, per provare vedere se il tutto funziona, al posto di `NOMEPACCHETTO` puoi scrivere `requests` che è una libreria per il web.

Se hai Anaconda:

- clicca sull'icona di Windows in basso a sinistra e cerca 'Anaconda Prompt'. Dovrebbe apparire una console dove inserire comandi, con scritto qualcosa come `C:\Users\David>`. (NOTA: per lanciare i comandi di Anaconda,

[43] http://gpiancastelli.altervista.org/dip3-it/installare-python.html
[44] http://gpiancastelli.altervista.org/dip3-it/installare-python.html#ubuntu
[45] http://gpiancastelli.altervista.org/dip3-it/installare-python.html#other

usa solo questa console speciale. Se usi la console di default di Windows (cmd), Windows non sarà in grado di trovare Python)

- Nella console scrivi `conda install NOMEPACCHETTO`

Se hai Linux/Mac (`--user` installa nella propria home) apri il Terminal e dai questo comando:

- `python3 -m pip install --user NOMEPACCHETTO`
- **NOTA:** Se ricevi errori che dicono che il comando `python3` non viene trovato, togli il `3` dopo `python`

INFO: volendo esiste anche un comando di sistema `pip` (o `pip3` a seconda del sistema). Lo puoi chiamare direttamente con `pip install --user NOMEPACCHETTO`

Noi installiamo invece con comandi del tipo `python3 -m pip install --user NOMEPACCHETTO` per uniformità e per essere sicuri di installare pacchetti per la versione 3 di Python

---

3.3 Notebook Jupyter

3.3.1 Eseguire il notebook Jupyter

Un editor comodo che si può usare per Python è Jupyter[46]:

- Se hai installato Anaconda, dovresti trovarlo già nel menu di sistema e anche nell'Anaconda Navigator.
- Anche se non hai installato Anaconda, comunque cercalo nel menu di sistema chissamai che per qualche caso fortuito sia già installato
- Se non lo trovi nel menu di sistema, potresti comunque averlo dalla linea di comando,

Prova se funzionano:

```
jupyter notebook
```

Oppure, in alternativa

```
python3 -m notebook
```

ATTENZIONE: `jupyter` NON è un comando Python, bensì un comando *di sistema*.

Se vedi `>>>` sulla riga dei comandi vuol dire che devi prima uscire dall'interprete Python scrivendo `exit()` e premendo Invio!

ATTENZIONE: Se Jupyter non è installato appariranno dei messaggi di errore, in questo caso non ti spaventare e *vai all'installazione*.

Un browser dovrebbe aprirsi automaticamente con Jupyter, e nella console dovresti vedere dei messaggi come qua sotto. Nel browser dovresti vedere i file nella cartella da cui hai lanciato Jupyter. Se non parte nessun browser ma vedi un messaggio come qua sotto, allora copia l'indirizzo che vedi in un browser internet, preferibilmente Chrome, Safari o Firefox

[46] http://jupyter.org/

```
$ jupyter notebook
[I 18:18:14.669 NotebookApp] Serving notebooks from local directory: /home/da/Da/prj/
↪softpython/prj
[I 18:18:14.669 NotebookApp] 0 active kernels
[I 18:18:14.669 NotebookApp] The Jupyter Notebook is running at: http://localhost:
↪8888/?token=49d4394bac446e291c6ddaf349c9dbffcd2cdc8c848eb888
[I 18:18:14.669 NotebookApp] Use Control-C to stop this server and shut down all␣
↪kernels (twice to skip confirmation).
[C 18:18:14.670 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=49d4394bac446e291c6ddaf349c9dbffcd2cdc8c848eb888
```

ATTENZIONE 1: in questo caso l'indirizzo è `http://localhost:8888/?token=49d4394bac446e291c6ddaf349c9dbffcd2cdc8c848eb888`, ma il tuo sarà sicuramente diverso!

ATTENZIONE 2: Mentre il server Jupyter è attivo, non puoi mettere comandi nel terminale !

Nella console vedi l'output del server di Jupyter, che è attivo e in un certo senso 'ha preso il controllo' del terminale. Questo significa che se anche scrivi dei comandi dentro il terminale, questi **non** verranno eseguiti!

3.3.2 Salvare i fogli Jupyter

Puoi salvare il foglio corrente in Jupyter premendo `Control-S` mentre sei nel browser.

ATTENZIONE: NON APRIRE LO STESSO DOCUMENTO IN PIU' TAB !!

Fai attenzione a non aprire lo stesso notebook in più tab, modifiche su tab differenti possono sovrascriversi a caso ! Per evitare queste spiacevolissime situazioni, assicurati di avere una sola tab per docmento. Se apri accidentalmente lo stesso notebook in tab differenti, semplicemente chiudi la tab di troppo.

Salvataggi automatici

Le modifiche ai notebook sono automaticamente salvate dopo qualche minuto.

3.3.3 Spegnere il server di Jupyter

Prima di chiudere il server di Jupyter, ricordati di salvare nel browser i fogli modificati sinora.

Per chiudere correttamente Jupyter, *non* chiudere brutalmente il terminale. Piuttosto, nel terminale da dove lo hai lanciato premi `Control-c`, dovrebbe apparirti questa domanda, a cui devi rispondere `y` (se non rispondi entro 5 secondi, poi dovrai ripremere `Control-c` perchè ti venga rifatta la domanda):

```
Shutdown this notebook server (y/[n])? y
[C 11:05:03.062 NotebookApp] Shutdown confirmed
[I 11:05:03.064 NotebookApp] Shutting down kernels
```

3.3.4 Navigare i notebook

(Opzionale) Per migliorare l'esperienza di navigazione dei fogli Jupyter, potresti voler installare qualche estensione Jupyter, come `toc2` che mostra le intestazioni dei paragrafi nella barra a sinistra. Per installarlo:

Installa le Jupyter contrib extensions[47]:

1a. Se hai Anaconda: Apri Anaconda Prompt (o il Terminale se hai Mac / Linux), e scrivi:

```
conda install -c conda-forge jupyter_contrib_nbextensions
```

1b. Se non hai Anaconda: Apri il terminale e scrivi:

```
python3 -m pip install --user jupyter_contrib_nbextensions
```

2. Installalo in Jupyter:

```
jupyter contrib nbextension install --user
```

3. Abilita le estensioni:

```
jupyter nbextension enable toc2/main
```

Una volta installato: Per vedere le tabelle dei contenuti in un documento dovrai premere un bottone lista all'estremità destra della barra degli strumenti

Se per caso non lo vedi il bottone:

1. vai nell'interfaccia principale di Jupyter
2. controlla la tab `Nbextensions`
3. Assicurati che `Table of Contents(2)` sia abilitato
4. Poi chiudi Jupyter, riaprilo, e vai su un foglio, dovresti finalmente vedere il bottone

[47] https://github.com/ipython-contrib/jupyter_contrib_nbextensions

3.3.5 Installare il notebook Jupyter - tutti i sistemi operativi

Se non sei riuscito a *trovare e/o far partire Jupyter*, probabilmente vuol dire che bisogna installarlo !

Puoi provare ad installare jupyter con `pip` (il gestore di moduli nativo di Python).

Per installare, esegui questo comando:

```
python3 -m pip install --user jupyter -U
```

Una volta installato, segui la sezione *Eseguire il notebook Jupyter*

> **ATTENZIONE**: NON serve installare Jupyter dentro *ambienti virtuali*.
>
> Jupyter è da considerarsi un'applicazione a livello di sistema, dovrebbe essere indipendente dagli ambienti virtuali. Se sei all'interno di un ambiente virtuale (per es. la linea di comando inizia con una scritta tra parentesi tipo `(myprj)`), esci dall'ambiente scrivendo `deactivate`

HELP: Se hai problemi ad installare Jupyter, intanto che aspetti aiuto puoi provare la versione dimostrativa online[48] (nota che non è sempre disponibile) o Google Colab[49]

[48] https://try.jupyter.org/
[49] http://colab.research.google.com/

3.4 Fare progetti Python con gli ambienti virtuali

ATTENZIONE: Se sei alle prime armi con Python, puoi saltare questa sezione.

La dovresti leggere se hai già fatto progetti personali in Python che vuoi evitare di compromettere, oppure quando vuoi fare un progetto da consegnare a qualcuno.

Quando cominciamo un nuovo progetto in Python, di solito ci accorgiamo in fretta che bisogna estendere Python con delle librerie particolari, per esempio per disegnare grafici. Non solo, potremmo anche voler installare dei programmi Python non scritti da noi che a loro volta avranno bisogno delle loro librerie per funzionare.

Ora, potremmo installare tutte queste librerie extra in un unico calderone per tutto il computer, ma ogni progetto può richiedere le sue specifiche versioni di ogni libreria, e a volte potrebbe non gradire le versioni installate da altri progetti. Peggio ancora, potrebbe aggiornare automaticamente librerie usate da vecchi progetti, rendendo il codice dei vecchi progetti non più eseguibile. Quindi è PRATICAMENTE NECESSARIO separare per bene ogni progetto e le sue librerie da quelle degli altri: a tal fine si può creare un cosiddetto *virtual environment*.

3.4.1 Creare ambienti virtuali

- **Se hai installato Anaconda**, per creare ambienti virtuali ti conviene usare il suo gestore di pacchetti `conda`. Supponendo che vogliamo chiamare il nostro progetto `myprj` (ma potrebbe essere un nome qualunque), da mettere in una cartella dallo stesso nome `myprj`, per creare un ambiente virtuale possiamo dare questo comando:

  ```
  conda create -n myprj
  ```

  Il comando potrebbe chiederti di scaricare dei pacchetti, dai pure conferma.

- **Se *non* hai installato Anaconda**, per creare ambienti virtuali ti conviene usare il modulo nativo di Python `venv`:

  ```
  python3 -m venv myprj
  ```

Entrambi i metodi creano la cartella `myprj` e la riempiono con tutti i file Python necessari per avere un progetto completamente isolato dal resto del computer. Ma adesso, come possiamo dire a Python che vogliamo lavorare proprio con quel progetto lì? Dobbiamo *attivare* l'ambiente come segue.

3.4.2 Attivare un ambiente virtuale

Per attivare l'ambiente virtuale, dobbiamo usare comandi diversi a seconda del sistema operativo (ma sempre dal terminale)

Su Linux & Mac (senza Anaconda):

```
source myprj/bin/activate
```

Attivare environment su Windows con Anaconda:

```
activate myprj
```

Una volta che l'ambiente è attivato, nel prompt dei comandi dovremmo vedere il nome dell'ambiente (in questo caso `myprj`) in parentesi tonde all'inizio della riga:

```
(myprj) directory/corrente >
```

Il prefisso ci fa sapere che l'ambiene `myprj` è correntemente attivo, perciò tutti i comandi Python che daremo useranno le impostazioni e le librerie di quell'ambiente.

Nota: dentro l'ambiente virtuale, si può usare il comando `python` invece di `python3`, e `pip` invece di `pip3`, ma noi consigliamo comunque di essere espliciti e aggiungere sempre 3 così da evitare possibili confusioni.

Disattivare un ambiente:

Scrivere nella console il comando `deactivate`. Una volta disattivato l'ambiente, il nome dell'ambiente `(myprj)` all'inizio del prompt dovrebbe scomparire.

3.4.3 Eseguire gli ambienti da dentro Jupyter

Come abbiamo detto in precedenza, Jupyter è un'applicazione a livello di sistema, quindi dovrebbe esserci uno e un solo Jupyter. Ciononostante, durante l'esecuzione di Jupyter, potremmo voler eseguire i nostri comandi Python in un particolare ambiente Python. Per fare ciò, dobbiamo configurare Jupyter perchè usi l'ambiente desiderato. Nella terminologia di Jupyter, le configurazioni si chiamano *kernel*: esse definiscono i programmi lanciati da Jupyter (siano essi versioni di Python o anche altri linguaggi, tipo R). Il kernel corrente per un foglio è visibile in alto a destra. Per avere il kernel che vogliamo, ci sono diversi modi:

Se hai Anaconda

Jupyter dovrebbe essere disponibile nel Navigator. Se nel Navigator abiliti un environment (per es per Python 3), quando poi lanci Jupyter e crei un foglio dovresti avere l'ambiente desiderato attivo, o quantomeno poter scegliere un kernel con quell'ambiente.

Se non hai Anaconda

In questo caso, la procedura è un po' più complicata:

— 1 Dal Terminale, *attiva il tuo ambiente*

— 2 Crea un kernel Jupyter:

```
python3 -m ipykernel install --user --name myprj
```

NOTA: qua `myprj` è il nome *del kernel Jupyter*. Usiamo lo stesso nome dell'ambiente solo per questioni di praticità.

— 3 Disattiva il tuo ambiente, lanciando

```
deactivate
```

A questo punto, ogni volta che lanci Jupyter, se tutto è andato bene nella voce 'Kernel' dei notebook dovresti poter selezionare il kernel che hai appena creato (in questo esempio, avrebbe il nome `myprj`)

NOTA: il passaggio di creazione del kernel va fatto una sola volta per progetto

---

NOTA: Non serve attivare l'ambiente prima di lanciare Jupyter!

Durante l'esecuzione di Jupyter semplicemente seleziona il kernel desiderato. Ciononostante, conviene eseguire Jupyter dalla cartella del nostro ambiente virtuale, così vedremo tutti i file del progetto nella home di Jupyter.

3.5 Approfondimenti

Per approfondimenti su come usare editor diversi da Jupyter e l'architettura di Python, puoi leggere il foglio Strumenti e script[50].

[50] https://it.softpython.org/tools/tools-sol.html

CHAPTER

FOUR

A - FONDAMENTI

4.1 Introduzione rapida a Python

4.1.1 Scarica zip esercizi

Naviga file online[51]

REQUISITI:

- **QUESTO FOGLIO E' PER CHI HA GIA' COMPETENZE DI PROGRAMMAZIONE**, e in 3-4h vuole farsi rapidamente un'idea di Python
- **Aver installati Python 3 e Jupyter:** se non hai già provveduto, guarda Installazione[52]

SE SEI UN PRINCIPIANTE: Salta questo foglio e fai invece i tutorial che trovi nella sezione Fondamenti[53], a partire da Strumenti e script[54]

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
quick-intro
    quick-intro.ipynb
    quick-intro-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

[51] https://github.com/DavidLeoni/softpython-it/tree/master/quick-intro
[52] https://it.softpython.org/installation.html
[53] https://it.softpython.org/index.html#A---Fondamenti
[54] https://it.softpython.org/tools/tools-sol.html

ATTENZIONE: In questo libro usiamo **SOLO PYTHON 3**

Se per caso ottieni comportamenti inattesi, controlla di usare Python 3 e non il 2. Se per caso il tuo sistema operativo scrivendo `python` fa partire il 2, prova ad eseguire il tre scrivendo il comando: `python3`

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `quick-intro.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

ATTENZIONE: Ricordati di eseguire sempre la prima cella dentro il notebook. Contiene delle istruzioni come `import jupman` che dicono a Python quali moduli servono e dove trovarli. Per eseguirla, vedi le seguenti scorciatoie

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.1.2 Proviamo Jupyter

Vediamo brevemente come funzionano i fogli Jupyter.

ESERCIZIO: Proviamo a inserire un comando Python: scrivi nella cella qua sotto `3 + 5`, e poi mentre sei in quella cella premi i tasti speciali `Control+Invio`. Come risultato, dovresti vedere apparire il numero 8

[ ]:
```
```

ESERCIZIO: in Python possiamo scrivere commenti iniziando una riga con un cancelletto `#`. Come prima, scrivi nella cella sotto `3 + 5` ma questa volta scrivilo nella riga sotto la scritta `# scrivi qui`:

[2]:
```
# scrivi qui
```

ESERCIZIO: Jupyter per ogni cella mostra il risultato solo dell'ultima riga eseguita in quella cella. Prova a inserire questo codice nella cella sotto ed esegui premendo `Control+Invio`. Che risultato appare?

```
3 + 5
1 + 1
```

[3]:
```
# scrivi qui
```

ESERCIZIO: Proviamo adesso a creare noi una nuova cella.

- Mentre sei con il cursore in questa cella, premi `Alt+Invio`. Si dovrebbe creare una nuova cella dopo la presente.

- Nella cella appena creata, inserisci `2 + 3` e poi premi `Shift+Invio`. Cosa succede al cursore? prova la differenze con `Control+Invio`. Se non capisci la differenza, prova a premere ripetutamente `Shif+Invio` e vedi che succede.

4.1.3 Principali tipi di dati Python

Dato che il tema del corso è il trattamento dei dati, per cominciare ci focalizzeremo sui tipi di dati in Python.

Riferimenti:

- Pensare in Python, Capitolo 1, Lo scopo del programma[55]
- Pensare in Python, Capitolo 2, Variabili, istruzioni ed espressioni[56]

Quando leggiamo delle informazioni da una fonte esterna come un file, poi dovremo inevitabilmente incastonare i dati letti in qualche combinazione di questi tipi:

Tipo	Esempio 1	Esempio 2	Esempio 3
int	`0`	`3`	`-5`
numero in virgola mobile float	`0.0`	`3.7`	`-2.3`
bool	False	True	
string	`""`	`"Buon giorno"`	`'come stai?'`
list	`[]`	`[5, 8, 10]`	[``"qualcosa", 5, "altro ancora"]``
dict	`{}`	`{'chiave 1':'valore 1', 'chiave 2':'valore 2'}`	`{5:'un valore stringa', 'chiave stringa':7}`

A volte, useremo tipi più complessi, per esempio i valori temporali si potrebbero mettere in oggetti di tipo `datetime` che oltre alla data vera e propria possono contenere anche il fuso orario.

In quel che segue, forniremo alcuni esempi rapidi su quello che si può fare sui vari tipi di dato, mettendo i riferimenti alle spiegazioni più dettagliate dal libro Pensare in Python.

4.1.4 Numeri interi e con virgola

Nel libro 'Pensare in Python' non c'è un vero e proprio capitolo dedicato ai calcoli numerici, che sono sparse nei primi capitoli segnalati prima. Metto qua due note velocissime:

In Python abbiamo numeri interi:

```
[4]: 3 + 5
```

```
[4]: 8
```

La somma tra interi ovviamente ci da un intero:

```
[5]: type(8)
```

```
[5]: int
```

E se dividiamo interi? Ci troveremo con il tipo in virgola mobile *float*:

[55] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2002.html
[56] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2003.html

[6]:
```
3 / 4
```
[6]:
```
0.75
```

[7]:
```
type(0.75)
```
[7]:
```
float
```

> **ATTENZIONE al punto !**
>
> In Python e in molti formati dati, al posto della nostra virgola si usa il formato inglese con il punto '.'

⊛ **ESERCIZIO**: Prova a scrivere qua sotto `3.14` con il punto, e poi `3,14` con la virgola ed esegui con Ctrl+Invio. Cosa appare nei due casi?

[8]:
```
# scrivi qui con il punto
```

[9]:
```
# scrivi qui con la virgola
```

⊛ **ESERCIZIO**: Prova a scrivere qua sotto `3 + 1.0` ed esegui con Ctrl+Invio. Di quale tipo sarà il risultato? Controlla anche usando il comando `type`.

[10]:
```
# scrivi qui i comandi
```

⊛ **ESERCIZIO**: Qualche professore di matematica ti avrà sicuramente intimato di non dividere mai per zero. Neanche a Python piace molto, prova a scrivere nella cella qua sotto `1 / 0` e poi premi Ctrl+Invio per eseguire la cella. Nota come Python riporterà la riga dove è accaduto l'errore:

[11]:
```
# scrivi qui sotto il codice
```

4.1.5 Booleani - bool

I booleani rappresentano valori veri e falsi, e si possono usare per verificare quando certe condizioni avvengono.

Riferimenti

- Softpython: Basi - booleani[57]
- Softpython: controllo di flusso - if[58]

Per indicare i booleani Python ci fornisce due costanti `True` e `False`. Cosa ci possiamo fare?

[57] https://it.softpython.org/basics/basics-sol.html#Booleani
[58] https://it.softpython.org/control-flow/flow1-if-sol.html

and

Potremmo usarle per segnare in variabili se un certo fatto è avvenuto, per esempio possiamo fare un programma che al mattino ci dice che possiamo uscire di casa solo dopo aver fatto entrambe (`and`) colazione e e lavato i denti:

[12]:
```
fatto_colazione = True
lavato_denti = True

if fatto_colazione and lavato_denti:
    print("fatto tutto !")
    print("posso uscire di casa")
else:
    print("NON posso uscire di casa")
```

```
fatto tutto !
posso uscire di casa
```

⊛ **ESERCIZIO**: prova a scrivere qui sotto a mano il programma riportato nella cella precedente, ed eseguilo con Control+Invio. Prova a cambiare i valori da `True` a `False` e guarda che succede.

Assicurati di provare tutti i casi:

- True True
- True False
- False True
- False False

ATTENZIONE: Ricordati i `:` alla fine della riga con `if` !!!!

[13]:
```
# scrivi qui
```

Si può anche mettere un `if` dentro l'altro (*nested if*). Per esempio, questo programma qui funziona esattamente come il precedente:

[14]:
```
fatto_colazione = True
lavato_denti = True

if fatto_colazione:
    if lavato_denti:                        # NOTA: Questo blocco if è indentato␣
↪rispetto
        print("fatto tutto !")              #       all' if fatto_colazione
        print("posso uscire di casa!")      #
    else:
        print("NON posso uscire di casa")
else:
    print("NON posso uscire di casa")
```

```
fatto tutto !
posso uscire di casa!
```

⊛ **ESERCIZIO**: Prova a modificare il programma riportato nella cella precedente per fargli riportare lo stato delle varie azioni compiute. Elenchiamo qui i possibili casi e i risultati attesi:

- True False

```
ho fatto colazione
non ho lavato i denti
NON posso uscire di casa
```

- False True
- False False

```
non ho fatto colazione
NON posso uscire di casa
```

- True True

```
ho fatto colazione
ho lavato i denti
fatto tutto !
posso uscire di casa!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[15]:
```
# scrivi qui

fatto_colazione = True
lavato_denti = True

if fatto_colazione:
    print("ho fatto colazione")

    if lavato_denti:                        # NOTA: Questo blocco if è indentato␣
↪rispetto
        print("ho lavato i denti")          #
        print("fatto tutto !")              #       all' if fatto_colazione
        print("posso uscire di casa!")      #
    else:
        print("non ho lavato i denti")
        print("NON posso uscire di casa")

else:
    print("non ho fatto colazione")
    print("NON posso uscire di casa")
```

```
ho fatto colazione
ho lavato i denti
fatto tutto !
posso uscire di casa!
```

[15]:
```
# scrivi qui
```

```
ho fatto colazione
ho lavato i denti
fatto tutto !
posso uscire di casa!
```

or

Per verificare se almeno di due condizioni si è verificata, si usa `or`. Per esempio, possiamo stabilire che per fare colazione ci serve avere del latte intero o scremato (nota: se li abbiamo tutte e due riusciamo a fare colazione lo stesso !)

[16]:
```
ho_latte_intero = True
ho_latte_scremato = False

if ho_latte_intero or ho_latte_scremato:
    print("posso fare colazione !")
else:
    print("NON posso fare colazione :-(")
```

```
posso fare colazione !
```

⊛ **ESERCIZIO**: prova a scriverecqui sotto a mano il programma riportato nella cella presedente, ed eseguilo con Control+Invio. Prova a cambiare i valori da `True` a `False` e guarda che succede:

Assicurati di provare tutti i casi:

- True True
- True False
- False True
- False False

[17]:
```
# scrivi qui
```

⊛⊛ **ESERCIZIO**: prova a fare un programma che ti dice se puoi uscire di casa solo se hai fatto colazione (per cui devi avere hai almeno un tipo di latte) e lavato i denti

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
ho_latte_intero = False
ho_latte_scremato = True
lavato_denti = False


# scrivi qui


if ho_latte_intero or ho_latte_scremato:
    print("posso fare colazione !")
    fatto_colazione = True
else:
    print("NON posso fare colazione :-(")
    fatto_colazione = False

if fatto_colazione and lavato_denti:
    print("posso uscire di casa")
else:
    print("NON posso uscire di casa")
```

```
posso fare colazione !
NON posso uscire di casa
```

[18]:
```
ho_latte_intero = False
ho_latte_scremato = True
lavato_denti = False


# scrivi qui
```

```
posso fare colazione !
NON posso uscire di casa
```

not

Per le negazioni, puoi usare il `not`:

[19]:
```
not True
```

[19]:
```
False
```

[20]:
```
not False
```

[20]:
```
True
```

[21]:
```
fatto_colazione = False

if not fatto_colazione:
    print("Ho fame !")
else:
    print("che buoni che erano i cereali")
```

```
Ho fame !
```

[22]:
```
fatto_colazione = True

if not fatto_colazione:
    print("Ho fame !")
else:
    print("che buoni che erano i cereali")
```

```
che buoni che erano i cereali
```

⊛⊛ ESERCIZIO: prova a fare un programma che ti dice se puoi nuotare se NON hai fatto colazione E hai il salvagente

Assicurati di provare tutti i casi:

- True True
- True False
- False True
- False False

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
hai_salvagente = True
fatto_colazione = True

# scrivi qui

if hai_salvagente and not fatto_colazione:
    print("puoi nuotare")
else:
    print("NON puoi nuotare")
```

```
NON puoi nuotare
```

[23]:
```
hai_salvagente = True
fatto_colazione = True

# scrivi qui
```

```
NON puoi nuotare
```

Non solo True e False

ATTENZIONE ai booleani diversi da True e False !

In Python, il numero `0` e altri oggetti 'nulli' (come l'oggetto `None`, la stringa vuota `""` e la lista vuota `[]`) sono considerati `False`, e tutto ciò che non è 'nullo' è considerato `True`!

Facciamo degli esempi per mostrare quanto sopra riportato:

[24]:
```
if True:
    print("questo")
    print("sarà")
    print("stampato")
else:
    print("quest'altro")
    print("non sarà stampato")
```

```
questo
sarà
stampato
```

Tutto ciò che non è 'nullo' è considerato `True`, verifichiamolo per esempio con la stringa `"ciao"`:

[25]:
```
if "ciao":
    print("anche questo")
    print("sarà stampato!!")
```

(continues on next page)

(continued from previous page)

```
else:
    print("e questo no")
```

```
anche questo
sarà stampato!!
```

[26]:
```
if False:
    print("io non sarò stampato")
else:
    print("io sì")
```

```
io sì
```

[27]:
```
if 0:
    print("anche questo non sarà stampato")
else:
    print("io sì")
```

```
io sì
```

[28]:
```
if None:
    print("neppure questo sarà stampato")
else:
    print("io sì")
```

```
io sì
```

[29]:
```
if "":  # stringa vuota
    print("Neanche questo sarà stampato !!!")
else:
    print("io sì")
```

```
io sì
```

⊛ **ESERCIZIO**: Copia qua sotto l'`if` una stringa con uno spazio dentro `" "`nella condizione dell'`if`. Cosa succederà?

- prova anche a mettere una lista vuota `[]`, che succede?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[30]:
```
# scrivi qui l'if


if " ": # spazio
    print("Niente stampa!")
else:
    print("io sì")


if []: # lista vuota
    print("Niente stampa!")
else:
    print("io sì")
```

```
Niente stampa!
io sì
```

[30]:
```
# scrivi qui l'if
```

```
Niente stampa!
io sì
```

4.1.6 Stringhe - string

Le stringhe sono sequenze *immutabili* di caratteri.

Riferimenti:

SoftPython:

- stringhe 1 - introduzione[59]
- stringhe 2 - operatori[60]
- stringhe 3 - metodi[61]
- stringhe 4 - altri esercizi[62]

Concatenare stringhe

Una delle cose che si fanno più frequentemente è concatenre delle stringhe:

[31]:
```
"ciao " + "mondo"
```

[31]:
```
'ciao mondo'
```

Ma nota che quando concateniamo una stringa e un numero, Python si arrabbia:

```
"ciao " + 5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-e219e8205f7d> in <module>()
----> 1 "ciao " + 5

TypeError: Can't convert 'int' object to str implicitly
```

Questo succede perchè Python vuole che convertiamo esplicitamente il numero '5' in una stringa. Porrà simili lamentela anche con altri tipi di oggetti. Quindi, quando concatenate oggetti che non sono stringhe, per evitare problemi racchiudete l'oggetto da convertire nella funzione `str` come qui:

[32]:
```
"ciao " + str(7)
```

[32]:
```
'ciao 7'
```

Un modo alternativo e più veloce è usare l'operatore di formattazione percentuale `%`, che sostituisce alle occorrenze di `%s` quello che mettete dopo un `%` dopo la stringa :

[59] https://it.softpython.org/strings/strings1-sol.html
[60] https://it.softpython.org/strings/strings2-sol.html
[61] https://it.softpython.org/strings/strings3-sol.html
[62] https://it.softpython.org/strings/strings4-sol.html

[33]:
```
"ciao %s" % 7
```

[33]:
```
'ciao 7'
```

Meglio ancora, il `%s` può stare all'interno della stringa e venire ripetuto. Per ogni occorrenza si può passare un sostituto diverso, come per esempio nella tupla `("bello", "Python")` (una tupla è semplicemente una sequenza immutabile di elementi racchiusi tra parentesi tonde e separati da virgole)

[34]:
```
"Che %s finalmente imparo %s" % ("bello", "Python")
```

[34]:
```
'Che bello finalmente imparo Python'
```

⊛ **ESERCIZIO**: il `%s` funziona con le stringhe ma anche con quasiasi altro tipo di dato, per esempio un intero. Scrivi qua sotto il comando sopra, aggiunendo un `%s` alla fine della stringa, e aggiungendo alla fine della tupla il numero `3` (separandolo dagli altri con una virgola).

Domanda: i `%s` possono stare uno dopo l'altro senza spazi tra di loro? Prova.

[35]:
```
# scrivi qui
```

Usare metodi degli oggetti

Quasi tutto in Python è un oggetto, faremo qui una velocissima introduzione tanto per dare l'idea.

Riferimenti

- Pensare in Python, Capitolo 15, Classi e oggetti[63]
- Pensare in Python, Capitolo 16, Classi e funzioni[64]
- Pensare in Python, Capitolo 17, Classi e metodi[65]

Quasi tutto in Python è un oggetto. Per esempio, le stringhe sono oggetti. Ogni tipo di oggetto ha delle azioni chiamati *metodi* che si possono eseguire su quello oggetto. Per esempio, per le stringhe che rappresentano nomi potremmo voler rendere in maiuscolo la prima lettera: a tal fine possiamo cercare se per le stringhe ci sono giò metodi esistenti che fanno questo. Proviamo il metodo esistente `capitalize()` sulla *stringa* `"trento"` (notare che la stringa è tutta in minuscolo e *capital* in inglese vuol anche dire 'maiuscolo'):

[36]:
```
"trento".capitalize()
```

[36]:
```
'Trento'
```

Python ci ha appena fatto la cortesia di rendere la prima lettera della parola in maiuscolo `'Trento'`

⊛ **ESERCIZIO**: Scrivi nella cella qua sotto `"trento".` e premi TAB: Jupyter dovrebbe suggerirti dei metodi disponibili per la stringa. Prova il metodo `upper()` e `count("t")`

[37]:
```
# scrivi qui
```

[63] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2016.html
[64] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2017.html
[65] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2017.html

4.1.7 Liste - list

Una lista in python è una sequenza di elementi eterogenei, in cui possiamo mettere gli oggetti che vogliamo.

Riferimenti - SoftPython:

- Liste 1 - introduzione[66]
- Liste 2 - operatori[67]
- Liste 3 - metodi[68]
- Liste 4 - iterazione e funzioni[69]

Creiamo una lista di stringhe:

[38]:
```
x = ["ciao", "soft", "python"]
```

[39]:
```
x
```

[39]:
```
['ciao', 'soft', 'python']
```

Le liste sono sequenze di oggetti possibilmente eterogenei, quindi dentro ci potete buttare di tutto, interi, stringhe, dizionari …:

[40]:
```
x = ["ciao", 123, {"a":"b"}]
```

[41]:
```
x
```

[41]:
```
['ciao', 123, {'a': 'b'}]
```

Per accedere ad un elemento in particolare dentro una lista, si può usare un indice tra parentesi quadre che indica un elemento:

[42]:
```
# primo elemento

x[0]
```

[42]:
```
'ciao'
```

[43]:
```
# secondo elemento

x[1]
```

[43]:
```
123
```

[44]:
```
# terzo elemento

x[2]
```

[44]:
```
{'a': 'b'}
```

In una lista possiamo cambiare gli elementi con l'assegnazione:

[45]:
```
# Cambiamo il _secondo_ elemento:

x[1] = "soft"
```

[66] https://it.softpython.org/lists/lists1-sol.html
[67] https://it.softpython.org/lists/lists2-sol.html
[68] https://it.softpython.org/lists/lists3-sol.html
[69] https://it.softpython.org/lists/lists4-sol.html

[46]:
```
x
```
[46]:
```
['ciao', 'soft', {'a': 'b'}]
```

[47]:
```
x[2] = "python"
```

[48]:
```
x
```
[48]:
```
['ciao', 'soft', 'python']
```

Per ottenere la lunghezza di una lista, possiamo usare `len`:

[49]:
```
x = ["ciao", "soft", "python"]

len(x)
```
[49]:
```
3
```

⊛ **ESERCIZIO**: prova ad accedere ad un elemento fuori dalla lista, e vedi che succede.

- `x[3]` è dentro o fuori dalla lista?
- C'è una qualche lista `x` per cui possiamo scrivere `x[len(x)]` senza problemi ?
- se usi indici negativi, che succede? Prova -1, -2, -3, -4 …

[50]:
```
# scrivi qui
```

Possiamo aggiungere elementi alla fine di una lista usando il comando `append`:

[51]:
```
x = []
```

[52]:
```
x
```
[52]:
```
[]
```

[53]:
```
x.append("ciao")
```

[54]:
```
x
```
[54]:
```
['ciao']
```

[55]:
```
x.append("soft")
```

[56]:
```
x
```
[56]:
```
['ciao', 'soft']
```

[57]:
```
x.append("python")
```

[58]:
```
x
```
[58]:
```
['ciao', 'soft', 'python']
```

Ordinamento liste

Le liste possono ordinare comodamente con il metodo `.sort`, che funziona su tutti gli oggetti ordinabili. Per esempio, possiamo ordinare i numeri:

IMPORTANTE: `.sort()` modifica la lista su cui è chiamato, *non* ne genera una nuova !

[59]:
```
x = [ 8, 2, 4]

x.sort()
```

[60]:
```
x
```

[60]:
```
[2, 4, 8]
```

Come altro esempio, possiamo ordinare le stringhe:

[61]:
```
x = [ 'mondo', 'python',  'ciao',]

x.sort()
x
```

[61]:
```
['ciao', 'mondo', 'python']
```

Se non volessimo modificare la lista originale e invece volessimo generarne una nuova, useremmo la funzione `sorted()`. **NOTA**: `sorted` è una *funzione*, non un *metodo*:

[62]:
```
x = [ 'mondo', 'python', 'ciao',]

sorted(x)
```

[62]:
```
['ciao', 'mondo', 'python']
```

[63]:
```
# l'x originale non è cambiato:
x
```

[63]:
```
['mondo', 'python', 'ciao']
```

⊛ **ESERCIZIO**: Che succede se ordini stringhe contenenti gli stessi caratteri ma maiscoli invece di minuscoli? Come vengono ordinati? Fai delle prove.

[64]:
```
# scrivi qui
```

⊛ **ESERCIZIO**: Che succede se nella stessa lista metti sia stringhe che numeri e provi ad ordinarla? Fai delle prove.

[65]:
```
# scrivi qui
```

Ordine rovesciato

Supponiamo di voler ordinare la lista alla rovescia usando `sorted`. Per fare cioò possiamo indicare a Python il parametro booleano `reverse` e il suo valore, che in questo caso sarà `True`. Questo ci permette di notare come Python consenta di usare parametri opzionali specificandoli *per nome* :

[66]:
```
sorted(['mondo', 'python', 'ciao'], reverse=True)
```

[66]:
```
['python', 'mondo', 'ciao']
```

⊛ **ESERCIZIO**: Per cercare informazioni su `sorted`, avremmo potuto chiedere a Python dell'aiuto. Per fare ciò Python mette a disposizione una comoda funzione chiamata `help`, che potresti usare così `help(sorted)`. Prova ad

eseguirla nella cella qua sotto. A volte l'help è piuttosto complesso, e sta a noi sforzarci un po' per individuare i parametri di interesse

[67]:
```
# scrivi qui
```

Rovesciare liste non ordinate

E se volessimo rovesciare una lista così com'è, senza ordinarla in senso decrescente, per esempio per passare da `[6,2,4]` a `[2,4,6]` Cercando un po' nella libreria di Python, vediamo che c'è una comoda funzione `reversed()` che prende come paramatro la lista che vogliamo rovesciare e ne genera una nuova rovesciata.

⊛ **ESERCIZIO**: Prova ad eseguire `reversed([6,2,4])` nella cella qua sotto, e guarda che output ottieni. E' quello che ti aspetti ? In genere, e specialmente in Python 3, quando ci aspettiamo una lista e per caso vediamo invece un oggetto col nome `iterator`, possiamo risolvere passando il risultato come parametro della funzione `list()`

[68]:
```
# scrivi qui il codice
```

4.1.8 Dizionari - dict

I dizionari sono dei contenitori che ci consentono di abbinare dei *valori* a delle voci dette *chiavi*. Qua faremo un esempio rapidissimo per dare un'idea.

Riferimenti:

- Pensare in Python, Capitolo 11, Dizionari[70]
- SoftPython - dizionari:
  1. introduzione[71]
  2. operatori[72]
  3. metodi[73]
  4. iterazione e funzioni[74]
  5. strutture composte[75]

Possiamo creare un dizionario con le graffe `{ }`, separando le chiavi dai valori con i due punti `:`, e separando le coppie chiave/valore con la virgola `,`:

[69]:
```
d = {  'chiave 1':'valore 1', 'chiave 2':'valore 2'}
```

Per accedere ai valori, possiamo usare le chiavi tra parentesi quadre:

[70]:
```
d['chiave 1']
```

[70]:
```
'valore 1'
```

[70] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2012.html
[71] https://it.softpython.org/dictionaries/dictionaries1-sol.html
[72] https://it.softpython.org/dictionaries/dictionaries2-sol.html
[73] https://it.softpython.org/dictionaries/dictionaries3-sol.html
[74] https://it.softpython.org/dictionaries/dictionaries4-sol.html
[75] https://it.softpython.org/dictionaries/dictionaries5-sol.html

[71]:
```
d['chiave 2']
```

[71]:
```
'valore 2'
```

Valori: come valori nei dizionari possiamo mettere quello che ci pare, numeri, stringhe, tuple, liste, altri dizionari ..

[72]:
```
d['chiave 3'] = 123
```

[73]:
```
d
```

[73]:
```
{'chiave 1': 'valore 1', 'chiave 2': 'valore 2', 'chiave 3': 123}
```

[74]:
```
d['chiave 4'] = ('io','sono', 'una', 'tupla')
```

[75]:
```
d
```

[75]:
```
{'chiave 1': 'valore 1',
 'chiave 2': 'valore 2',
 'chiave 3': 123,
 'chiave 4': ('io', 'sono', 'una', 'tupla')}
```

⊛ **ESERCIZIO**: prova ad inserire nel dizionario delle coppie chiave valore con chiavi stringa e come valori delle liste e altri dizionari,

[76]:
```
# scrivi qui:
```

Chiavi: Per le chiavi abbiamo qualche restrizione in più. Dentro i dizionari possiamo anche inserire dei numeri interi come chiavi:

[77]:
```
d[123] = 'valore 3'
```

[78]:
```
d
```

[78]:
```
{123: 'valore 3',
 'chiave 1': 'valore 1',
 'chiave 2': 'valore 2',
 'chiave 3': 123,
 'chiave 4': ('io', 'sono', 'una', 'tupla')}
```

o persino delle sequenze, purchè siano *immutabili* come le tuple:

[79]:
```
d[('io','sono','una','tupla')] = 'valore 4'
```

[80]:
```
d
```

[80]:
```
{('io', 'sono', 'una', 'tupla'): 'valore 4',
 123: 'valore 3',
 'chiave 1': 'valore 1',
 'chiave 2': 'valore 2',
 'chiave 3': 123,
 'chiave 4': ('io', 'sono', 'una', 'tupla')}
```

ATTENZIONE: Non tutti i tipi vanno bene a Python come chiavi. Senza andare nei dettagli, in genere non puoi inserire nei dizionari dei tipi che possono essere modificati dopo che sono stati creati.

⊛ **ESERCIZIO**:

- Prova ad inserire in un dizionario una lista tipo `['a','b']`come chiave, e come valore metti quello che vuoi. Python non dovrebbe permetterti di farlo, e ti dovrebbe mostrare la scritta `TypeError: unhashable type: 'list'`
- prova anche ad inserire un dizionario come chiave (per es. anche il dizionario vuoto `{}`). Che risultato ottieni?

4.1.9 Visualizzare l'esecuzione con Python Tutor

Abbiamo visto i principali tipi di dati. Prima di procedere oltre, è bene vedere gli strumenti giusti per comprendere al meglio cosa succede quando si esegue il codice. Python tutor[76] è un ottimo sito online per visualizzare online l'esecuzione di codice Python, permettendo di andare avanti e *indietro* nell'esecuzione del codice. Sfruttatelo più che potete, dovrebbe funzionare con parecchi degli esempi che tratteremo a lezione. Vediamo un esempio.

Python tutor 1/4

Vai sul sito pythontutor.com[77] e seleziona *Python 3*

Python tutor 2/4

[76] http://pythontutor.com/
[77] http://pythontutor.com/

Python tutor 3/4

Python tutor 4/4

Debuggare codice in Jupyter

Python tutor è fantastico, ma quando esegui del codice in Jupyter e non funziona, come si può fare? Per ispezionare l'esecuzione, gli editor di solito mettono a disposizione uno strumento chiamato *debugger*, che permette di eseguire le istruzioni una per una. Al momento (Agosto 2018), il debugger di Jupyter che si chiama pdb[78] è estramamente limitato . Per superarne le limitazioni, in questo corso ci siamo inventati una soluzione di ripiego, che sfrutta Python Tutor.

Se inserisci del codice Python in una cella, e poi **alla fine della cella** scrivete l'istruzione `jupman.pytut()`, come per magia il codice precedente verrà visualizzato all'interno del foglio Jupyter con il debugger di Python Tutor.

> **ATTENZIONE**: `jupman` è una collezione di funzioni di supporto che ci siamo inventati apposta per questo corso.
>
> Quando vedi comandi che iniziano con `jupman`, affinchè funzionino devi prima eseguire la cella in cima al documento. Riportiamo tale cella qua per comodità. Se non lo hai già fatto, eseguila adesso.

[81]:
```
# Ricordati di eseguire questa cella con Control+Invio
# Questi comandi dicono a Python dove trovare il file jupman.py
import jupman;
```

Adesso siamo pronti a provare Python tutor con la funzione magica `jupman.pytut()`:

> **ATTENZIONE**: Per usare Python tutor dentro Jupyter dovete essere online. Una volta eseguita la cella successivo, dopo qualche secondo dovrebbe apparire il debugger di Python tutor:

[82]:
```
x = 5
y = 7
z = x + y
```

(continues on next page)

[78] https://davidhamann.de/2017/04/22/debugging-jupyter-notebooks/

(continued from previous page)

```
jupman.pytut()
```

[82]:
```
<IPython.core.display.HTML object>
```

Python Tutor : Limitazione 1

Python tutor è comodo, ma ci sono importanti limitazioni:

> **ATTENZIONE**: Python Tutor guarda dentro una cella sola!
>
> Quando usi Python tutor dentro Jupyter, l'unico codice che viene considerato da Python tutor è quello dentro la cella dove sta il comando jupman.pytut().

Quindi per esempio in queste due celle che seguono, solo `print(w)` apparirà dentro Python tutor senza il `w = 3`. Se proverai a cliccare *Forward* in Python tutor, ti verrà segnalato che non è stata definita `w`

[83]:
```
w = 3
```

[84]:
```
print(w)

jupman.pytut()
```

```
3
```

```
Traceback (most recent call last):
  File "../jupman.py", line 2305, in _runscript
    self.run(script_str, user_globals, user_globals)
  File "/usr/lib/python3.5/bdb.py", line 431, in run
    exec(cmd, globals, locals)
  File "<string>", line 2, in <module>
NameError: name 'w' is not defined
```

[84]:
```
<IPython.core.display.HTML object>
```

Per avere tutto in Python tutor devi mettere tutto il codice nella stessa cella:

[85]:
```
w = 3
print(w)

jupman.pytut()
```

```
3
```

[85]:
```
<IPython.core.display.HTML object>
```

Python Tutor : Limitazione 2

Un'altra limitazione è la seguente:

> **ATTENZIONE**: Python tutor usa solo funzioni dalla distribuzione standard di Python**
>
> Python Tutor va bene per ispezionare semplici algoritmi che usano funzioni di base di Python.

Se usate qualche libreria tipo `numpy`, potete provare **solo online** a selezionare `Python 3.6 with anaconda`

4.1.10 Iterazione

Spesso è utile compiere azioni su ogni elemento di una sequenza.

Riferimenti

- SoftPython:
  - controllo di flusso - cicli for[79]
  - controllo di flusso - cicli while[80]
- Pensare in Python, Capitolo 7, Iterazione[81]
- Nicola Cassetta, Lezione 8, L'istruzione while[82]

Cicli for

Tra i vari modi per farlo, uno è usare i cicli `for`

[79] https://it.softpython.org/control-flow/flow2-for-sol.html
[80] https://it.softpython.org/control-flow/flow3-for-sol.html
[81] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2008.html
[82] http://ncassetta.altervista.org/Tutorial_Python/Lezione_08.html

[86]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

for animale in animali:
    print("Nella lista ci sono:")
    print(animale)
```

```
Nella lista ci sono:
cani
Nella lista ci sono:
gatti
Nella lista ci sono:
scoiattoli
Nella lista ci sono:
alci
```

Qua abbiamo definito la variabile `animale` (avremmo potuto chiamarla con qualunque nome, anche `pippo`). Per ogni elemento nella lista `animali`, vengono eseguite le istruzioni dentro il blocco. Ogni volta che le istruzioni vengono eseguite, la variabile `animale` assume uno dei valori della lista `animali`

ATTENZIONE: RICORDATI I DUE PUNTI `:` ALLA FINE DELLA LINEA DEL FOR !!!

ATTENZIONE: Per indentare il codice, usa SEMPRE sequenze di 4 spazi bianchi. Sequenze di 2 soli spazi per quanto consentite non sono raccomandate.

ATTENZIONE: A seconda dell'editor che usi, premendo TAB potresti ottenere una sequenza di spazi bianchi come accade in Jupyter (4 spazi che sono raccomandati), oppure un carattere speciale di tabulazione (da evitare)! Per quanto noiosa questa distinzione ti possa apparire, ricordatela perchè potrebbe generare errori molto difficili da scoprire.

[87]:
```
# Guardiamo cosa succede con Python tutor:

animali = ['cani', 'gatti', 'scoiattoli', 'alci']

for animale in animali:
    print("Nella lista ci sono:")
    print(animale)

jupman.pytut()
```

```
Nella lista ci sono:
cani
Nella lista ci sono:
gatti
Nella lista ci sono:
scoiattoli
Nella lista ci sono:
alci
```

[87]:
```
<IPython.core.display.HTML object>
```

⊛ **ESERCIZIO**: Proviamo a capire meglio tutti gli *Attenzione* qua sopra. Scrivi qui sotto il `for` con gli animali di prima (niente copia e incolla!), vedi se funziona. Ricordati di usare 4 spazi per le indentazioni.

- Poi prova a togliere i due punti alla fine e vedi che errore ti da Python

- Riaggiungi i due punti, e adesso prova a variare l'indentazione. Prova a mettere due spazi all'inizio di entrambi i print, vedi se esegue
- Adesso prova a mettere due spazi prima del primo print e 4 spazi prima del secondo, e vedi se esegue

[88]:
```
# scrivi qui - copia il for di sopra
```

for in range

Un'altra iterazione molto comune è incrementare un contatore ad ogni ciclo. Python rispetto ad altri linguaggi offre un sistema piuttosto particolare che usa la funzione `range(n)`, che ritorna una sequenza con i primi numeri da 0 incluso a `n` *escluso*. La possiamo usare così:

[89]:
```
for indice in range(3):
    print(indice)
```

```
0
1
2
```

[90]:
```
for indice in range(6):
    print(indice)
```

```
0
1
2
3
4
5
```

Guardiamo meglio con Python tutor:

[91]:
```
for indice in range(6):
    print(indice)

jupman.pytut()
```

```
0
1
2
3
4
5
```

[91]:
```
<IPython.core.display.HTML object>
```

Quindi possiamo usare questo stile come un'alternativa per listare i nostri animali:

[92]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

for indice in range(3):
    print("Nella lista ci sono:")
    print(animali[indice])
```

```
Nella lista ci sono:
cani
Nella lista ci sono:
```

(continues on next page)

(continued from previous page)

```
gatti
Nella lista ci sono:
scoiattoli
```

Guardiamo meglio con Python tutor:

[93]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

for indice in range(3):
    print("Nella lista ci sono:")
    print(animali[indice])

jupman.pytut()
```

```
Nella lista ci sono:
cani
Nella lista ci sono:
gatti
Nella lista ci sono:
scoiattoli
```

[93]:
```
<IPython.core.display.HTML object>
```

4.1.11 Funzioni

Una funzione prende dei parametri e li usa per produrre o riportare qualche risultato.

Riferimenti

- Pensare in Python, Capitolo 3, Funzioni[83]
- Pensare in Python, Capitolo 6, Funzioni produttive[84] puoi fare tutto saltando la parte 6.5 sulla ricorsione. **NOTA**: nel libro viene usato il termine strano 'funzioni produttive' per quelle funzioni che ritornano un valore, ed il termine ancora più strano 'funzioni vuote' per funzioni che non ritornano nulla ma fanno qualche effetto tipo stampa a video: ignora e dimentica questi termini !
- Nicola Cassetta, Lezione 4, Funzioni[85]
- SoftPython, Esercizi sulle funzioni[86]

Per definire una funzione, possiamo usare la parola chiave `def`:

[94]:
```
def mia_stampa(x,y):   # RICORDATI I DUE PUNTI ':' ALLA FINE DELLA RIGA !!!!!
    print('Ora stamperemo la somma di due numeri')
    print('La somma è %s' % (x + y))
```

Possiamo chiamare la funzione così:

[95]:
```
mia_stampa(3,5)
```

```
Ora stamperemo la somma di due numeri
La somma è 8
```

[83] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2004.html
[84] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2007.html
[85] http://ncassetta.altervista.org/Tutorial_Python/Lezione_04.html
[86] https://it.softpython.org/functions/functions-sol.html

Vediamo meglio che succede con Python Tutor:

[96]:
```
def mia_stampa(x,y):     # RICORDATI I DUE PUNTI ':' ALLA FINE DELLA RIGA !!!!!
    print('Ora stamperemo la somma di due numeri')
    print('La somma è %s' % (x + y))

mia_stampa(3,5)

jupman.pytut()
```

```
Ora stamperemo la somma di due numeri
La somma è 8
```

[96]:
```
<IPython.core.display.HTML object>
```

La funzione appena dichiarata stampa dei valori, ma non ritorna nulla. Per fare una funzione che ritorni un valore, dobbiamo usare la parola chiava `return`

Trovi questo tipo di funzioni anche sul libro Pensare in Python, Capitolo 6, Funzioni produttive[87], di cui puoi fare tutto saltando la parte 6.5 sulla ricorsione. NOTA: nel libro viene usato il termine strano "funzioni produttive" per quelle funzioni che ritornano un valore, ed il termine ancora più strano "funzioni vuote" per funzioni che non ritornano nulla ma fanno qualche effetto tipo stampa a video: ignora questi strani termini !

[97]:
```
def mia_somma(x,y):
    s = x + y
    return s
```

[98]:
```
mia_somma(3,5)
```

[98]:
```
8
```

[99]:
```
# Vediamo meglio  che succede con Python Tutor:

def mia_somma(x,y):
    s = x + y
    return s

print(mia_somma(3,5))

jupman.pytut()
```

```
8
```

[99]:
```
<IPython.core.display.HTML object>
```

⊛ **ESERCIZIO**: Se proviamo ad assegnare ad una variabile `x` il valore di ritorno delle funzione `mia_stampa` che apparantemente non ritorna nulla, che valore ci sarà in `x`? Prova a capirlo qua sotto:

[100]:
```
# scrivi qui
```

⊛ **ESERCIZIO**: Scrivi qua sotto una funzione `media` che calcola e ritorna la media tra due numeri x e y in input

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[87] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2007.html

[101]:
```
# scrivi qui



def media(x, y):
    return (x + y) / 2
```

[101]:
```
# scrivi qui
```

⊛⊛ **ESERCIZIO**: Scrivi qua sotto una funzione che chiameremo `iniziab` che prende una stringa `x` in ingresso. Se la stringa inizia con la lettera 'b', per esempio `'bianco'` la funzione stampa la scritta `bianco inizia con b`, altrimenti stampa `bianco non inizia con la b`.

- Per controllare se il primo carattere è uguale alla `'b'`, usa l'operatore == (ATTENZIONE: è un DOPPIO uguale !)
- Se la stringa è vuota la tua funzione può avere problemi? Come potresti risolverli? (Per separare più condizioni nell'`if`, usa l'operatore `and` oppure `or` a seconda di come hai costruito l'if).

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[102]:
```
# scrivi qui

def iniziab(x):
    if len(x) != 0 and x[0] == 'b':
        print(x + ' inizia con b')
    else:
        print(x + ' non inizia con b')

iniziab('bianco')
iniziab('verde')
iniziab('')
```

```
bianco inizia con b
verde non inizia con b
 non inizia con b
```

[102]:
```
# scrivi qui
```

```
bianco inizia con b
verde non inizia con b
 non inizia con b
```

Funzioni lambda

In Python una variabile può contenere una funzione. Per esempio, sappiamo che `len("ciao")` ci dà la lunghezza della stringa `"ciao"`

[103]:
```
len("ciao")
```
[103]:
```
4
```

Proviamo a creare una variabile `mia_variabile` che punta alla funzione `len`.

NOTA: *non* abbiamo aggiunto parametri a `len`!

[104]:
```
mia_variabile = len
```

Adesso possiamo usare `mia_variabile` esattamente come usiamo la funzione `len`, che da la lunghezza di sequenze come le stringhe

[105]:
```
mia_variabile("ciao")
```
[105]:
```
4
```

Possiamo anche riassegnare `mia_variabile` ad altre funzioni, per esempio `sorted`. Vediamo che succede:

[106]:
```
mia_variabile = sorted
```

chiamando `mia_variabile`, ci aspettiamo di vedere i caratteri di `"ciao"` in ordine alfabetico:

[107]:
```
mia_variabile("ciao")
```
[107]:
```
['a', 'c', 'i', 'o']
```

In Python possiamo definire funzioni in una sola riga, con le cosiddette *funzioni lambda*:

[108]:
```
mia_f = lambda x: x + 1
```

Cosa fa `mia_f` ? Prende un parametro `x` e ritorna il risultato di calcolare l'espressione `x + 1`:

[109]:
```
mia_f(5)
```
[109]:
```
6
```

Possiamo anche passare due parametri:

[110]:
```
mia_somma = lambda x,y: x + y
```

[111]:
```
mia_somma(3,5)
```
[111]:
```
8
```

⊛ **ESERCIZIO**: Prova a definire qua sotto una funzione lambda per calcolare la media tra due numeri `x` e `y`, e assegnala alla variabile `media`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[112]:
```
# scrivi qui

media = lambda x,y: (x + y)/2

media(2,7)
```

[112]:
```
4.5
```

[112]:
```
# scrivi qui
```

[112]:
```
4.5
```

4.1.12 Trasformazioni sulle liste

Supponiamo di voler prendere la lista di animali e generarne una nuova in cui tutti i nomi iniziano con la lettera maiuscola. Questa che vogliamo fare di fatto è la creazione di una nuova lista operando una trasformazione sulla precedente. Per fare ciò esistono diversi modi, quello più semplice è usare un ciclo `for` così

Trasfomazioni con il for

[113]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']


nuova_lista = []
for animale in animali:  # ad ogni ciclo la variabile 'animale' contiene un nome␣
↪preso dalla lista 'animali'
    nuova_lista.append(animale.capitalize())   # aggiungiamo alla nuova lista il nome␣
↪dell'animale corrente, con la prima lettera maiuscola
nuova_lista

#vediamo che succede con Python tutor
jupman.pytut()
```

[113]:
```
<IPython.core.display.HTML object>
```

Nota importante: i metodi sulle stringhe non modificano mai la stringa originale, ma ne generano sempre una nuova. Quindi la lista originale `animali` conterrà ancora le stringhe originale senza modifiche:

[114]:
```
animali
```

[114]:
```
['cani', 'gatti', 'scoiattoli', 'alci']
```

⊛ **ESERCIZIO**: Prova a scrivere qua sotto un ciclo `for` (senza usare il copia e incolla!) che scorre la lista dei nomi degli animali e crea un'altra lista che chiameremo `m` in cui tutti i caratteri dei nomi degli animali sono in maiuscolo (usa il metodo `.upper()`)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[115]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

# scrivi qui

animali = ['cani', 'gatti', 'scoiattoli', 'alci']
# Scrivi qua
m = []
for animale in animali:  # ad ogni ciclo la variabile 'animale' contiene un nome␣
↪preso dalla lista 'animali'
    m.append(animale.upper())   # aggiungiamo alla nuova lista il nome dell'animale␣
↪corrente, con la prima lettera maiuscola
m
```

[115]:
```
['CANI', 'GATTI', 'SCOIATTOLI', 'ALCI']
```

[115]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

# scrivi qui
```

[115]:
```
['CANI', 'GATTI', 'SCOIATTOLI', 'ALCI']
```

Trasformazioni con le *list comprehension*

Riferimenti: SoftPython - Sequenze[88]

La stessa identica trasformazione di sopra si potrebbe attuare con una cosiddetta *list comprehension*, che servono per generare nuove liste eseguendo la stessa operazione su tutti gli elementi di una lista esistente di partenza. Come sintassi imitano le liste, infatti iniziano e finiscono con le parentesi quadre, ma dentro contengono un for speciale per ciclare dentro un sequenza :

[116]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

nuova_lista = [animale.capitalize() for animale in animali]
```

[117]:
```
nuova_lista
```

[117]:
```
['Cani', 'Gatti', 'Scoiattoli', 'Alci']
```

Vediamo che succede con Python tutor:

[118]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

nuova_lista = [animale.capitalize() for animale in animali]

jupman.pytut()
```

[118]:
```
<IPython.core.display.HTML object>
```

⊛ **ESERCIZIO**: Prova qua sotto ad usare una list comprehension per mettere tutti i caratteri in maiuscolo

[88] https://it.softpython.org/sequences/sequences-sol.html#List-comprehensions

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[119]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']
# scrivi qui


animali = ['cani', 'gatti', 'scoiattoli', 'alci']

nuova_lista = [animale.upper() for animale in animali]
```

[119]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']
# scrivi qui
```

Filtrare con le comprehension: Volendo, possiamo anche filtrare i dati usando un `if` speciale da mettere alla fine della comprehension. Per esempio potremmo selezionare solo gli animali la cui lunghezza del nome è di 4 caratteri:

[120]:
```
[animale.upper() for animale in animali if len(animale) == 4]
```

[120]:
```
['CANI', 'ALCI']
```

Trasformazioni con le map

Un'altro modo ancora per trasformare una lista in una nuova è usare l'operazione `map`, che a partire da una lista, ne genera un'altra applicando ad ogni elemento della lista di partenza una funzione `f` che passiamo come parametro. Per risolvere lo stesso esercizio precedente, si potrebbe per esempio creare al volo con `lambda` una funzione `f` che mette la prima lettera di una stringa in maiuscolo, e poi si potrebbe chiamare la `map` passando la `f` appena creata:

[121]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']


f = lambda animale: animale.capitalize()

map(f, animali)
```

[121]:
```
<map at 0x7fac244d7358>
```

Purtroppo il risultato non è esattamente la lista che volevamo. Il problema è che Python 3 aspetta a ritornarci una lista vera e propria, e invece ci ritorna un *iteratore*. Come mai? Python 3 per ragioni di efficienza spera che non non andremo mai ad usare nessun elemento della nuova lista, evitandogli di fatto la fatica di applicare la funzione a tutti gli elementi della lista originale. Ma noi possiamo forzarlo a darci la lista che vogliamo usando la funzione `list`:

[122]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

f = lambda animale: animale.capitalize()

list(map(f, animali))
```

[122]:
```
['Cani', 'Gatti', 'Scoiattoli', 'Alci']
```

Per avere un esempio totalmente equivalente ai precedenti, possimao assegnare il risultato a `nuova_lista`:

[123]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

f = lambda animale: animale.capitalize()

nuova_lista = list(map(f, animali))
```

[124]:
```
nuova_lista
```

[124]:
```
['Cani', 'Gatti', 'Scoiattoli', 'Alci']
```

Un vero hacker Python probabilmente preferirà scrivere tutto in una sola linea, così:

[125]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

nuova_lista = list(map(lambda animale: animale.capitalize(), animali))
```

[126]:
```
nuova_lista
```

[126]:
```
['Cani', 'Gatti', 'Scoiattoli', 'Alci']
```

⊛ **ESERCIZIO**: la lista originale `animali` è cambiata? Controlla.

⊛ **ESERCIZIO**: Data una lista di numeri `numeri = [3, 5, 2, 7]` prova a scrivere una `map` che genera una nuova lista con i numeri raddoppiati, come `[6, 10, 4, 14]`:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[127]:
```
numeri = [3, 5, 2, 7]

# scrivi qui

numeri = [3,5,2,7]

list(map(lambda x: x * 2, numeri))
```

[127]:
```
[6, 10, 4, 14]
```

[127]:
```
numeri = [3, 5, 2, 7]

# scrivi qui
```

[127]:
```
[6, 10, 4, 14]
```

4.1.13 Matrici

Finita la presentazione, è ora di sforzarsi un po' di più. Vediamo brevemente matrici come liste di liste. Per approfondire, guardare i riferimenti.

Riferimenti:

- SoftPython - matrici come liste di liste[89]
- SoftPython - matrici numpy[90]

⊛⊛ **ESERCIZIO**: Date le due liste con nomi di animali e corrispondente aspettativa di vita in anni:

```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]
```

Scrivere nella cella sotto del codice che generi una lista di liste da due elementi, così:

`[ ['cane', 12], ['gatto', 14], ['pellicano', 30], ['scoiattolo', 6], ['aquila', 25] ]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[128]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui


coppie = []
for i in range(len(animali)):
    coppie.append([animali[i], anni[i]])
coppie
```

[128]:
```
[['cane', 12],
 ['gatto', 14],
 ['pellicano', 30],
 ['scoiattolo', 6],
 ['aquila', 25]]
```

[128]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui
```

[128]:
```
[['cane', 12],
 ['gatto', 14],
 ['pellicano', 30],
 ['scoiattolo', 6],
 ['aquila', 25]]
```

[89] https://it.softpython.org/matrices-lists/matrices-lists-sol.html
[90] https://it.softpython.org/matrices-numpy/matrices-numpy-sol.html

⊛⊛ **ESERCIZIO** modificare scrivendolo qua sotto il codice dell'esercizio precedente nella versione con il normale ciclo `for` per filtrare solo le specie con aspettativa di vita superiore ai 13 anni, così da ottenere questo risultato:

```
[['gatto', 14], ['pellicano', 30], ['aquila', 25]]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[129]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui

animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

coppie = []
for i in range(len(animali)):
    if anni[i] > 13:
        coppie.append([animali[i], anni[i]])
coppie
```

[129]:
```
[['gatto', 14], ['pellicano', 30], ['aquila', 25]]
```

[129]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui
```

[129]:
```
[['gatto', 14], ['pellicano', 30], ['aquila', 25]]
```

ESERCIZIO Scrivi qua sotto del codice con un normale ciclo `for` filtri solo le specie con aspettativa di vita superiore ai 10 anni e inferiore ai 27, così da ottenere questo risultato:

```
[['cane', 12], ['gatto', 14], ['aquila', 25]]
```

SUGGERIMENTO: in Pyhton per imporre due condizioni in un if si usa la parola chiave **and**

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[130]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui

animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]


coppie = []
for i in range(len(animali)):
```

(continues on next page)

(continued from previous page)

```
    if anni[i] > 10 and anni[i] < 27 :
        coppie.append([animali[i], anni[i]])
coppie
```

[130]:
```
[['cane', 12], ['gatto', 14], ['aquila', 25]]
```

[130]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui
```

[130]:
```
[['cane', 12], ['gatto', 14], ['aquila', 25]]
```

Funzione zip

La funzione `zip` prende due liste e ne restituisce una terza nuova, in cui mette coppie di elementi come tuple (che ricordiamo sono come le liste ma immutabilii), abbinando il primo elemento della prima lista a primo elemento della seconda lista, il secondo elemento della prima lista al secondo elemento della secondo e così via :

[131]:
```
list(zip(['a','b','c'], [5,2,7]))
```

[131]:
```
[('a', 5), ('b', 2), ('c', 7)]
```

Perchè abbiamo messo anche `list` nell'esempio? Perchè `zip` ha lo stesso problema della map, cioè non ritorna subito una lista come vorremmo noi:

[132]:
```
zip(['a','b','c'], [5,2,7])
```

[132]:
```
<zip at 0x7fac244d5548>
```

⊛⊛⊛ **ESERCIZIO**: Come vedi con la `zip` abbiamo ottenuto un risultato simile a quello del precedente esercizio, ma abbiamo tuple con parentesi tonde invece di liste con parentesi quadre. Riusciresti aggiungendo una `list comprehension` o una `map` ad ottenere lo stesso identico risultato?

- per convertire una tupla in una lista, usa la funzione `list`:

[133]:
```
list(  ('ciao', 'soft', 'python')  )  # all'interno abbiamo messo una tupla,␣
↪delimitata da parentesi tonde
```

[133]:
```
['ciao', 'soft', 'python']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[134]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui - soluzione con list comprehension

[ list(c) for c in zip(animali, anni) ]
```

[134]:
```
[['cane', 12],
 ['gatto', 14],
 ['pellicano', 30],
 ['scoiattolo', 6],
 ['aquila', 25]]
```

[134]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui - soluzione con list comprehension
```

[134]:
```
[['cane', 12],
 ['gatto', 14],
 ['pellicano', 30],
 ['scoiattolo', 6],
 ['aquila', 25]]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[135]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui - soluzione con map

animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]
list(map(list, zip(animali, anni)))
```

[135]:
```
[['cane', 12],
 ['gatto', 14],
 ['pellicano', 30],
 ['scoiattolo', 6],
 ['aquila', 25]]
```

[135]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui - soluzione con map
```

[135]:
```
[['cane', 12],
 ['gatto', 14],
 ['pellicano', 30],
 ['scoiattolo', 6],
 ['aquila', 25]]
```

⊛⊛⊛ **ESERCIZIO**: svolgi l'esercizio precedente filtrando gli animali con aspettativa di vita superiore ai 13 anni, usando la `zip` e una list comprehension

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra

soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[136]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]


# scrivi qui

animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

[ list(c) for c in zip(animali, anni) if c[1] > 13 ]
```

[136]:
```
[['gatto', 14], ['pellicano', 30], ['aquila', 25]]
```

[136]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]


# scrivi qui
```

[136]:
```
[['gatto', 14], ['pellicano', 30], ['aquila', 25]]
```

⊛⊛ **ESERCIZIO**: Date le due liste con nomi di animali e corrispondente aspettativa di vita in anni come sopra, scrivere nella cella sotto del codice che con un normale ciclo `for` generi un dizionario che associa alla specie l'aspettativa di vita, così:

```
{
 'aquila': 25,
 'cane': 12,
 'gatto': 14,
 'pellicano': 30,
 'scoiattolo': 6
}
```

ATTENZIONE: A seconda della versione esatta di Python che avete e di come il dizionario viene creato, l'ordine dei campi quando viene stampato potrebbe differire dall'esempio. Questo è perfettamente normale, perchè le chiavi di un dizionario sono da intendersi come un insieme senza ordine particolare. Se volete essere sicuri di trovare le chiavi stampate nell'ordine in cui sono state inserite, dovete usare un OrderedDict

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[137]:
```

animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui
```

(continues on next page)

(continued from previous page)

```
d = {}
for i in range(len(animali)):
    d[animali[i]] = anni[i]
d
```

[137]:
```
{'aquila': 25, 'cane': 12, 'gatto': 14, 'pellicano': 30, 'scoiattolo': 6}
```

[137]:
```
animali = ['cane', 'gatto', 'pellicano', 'scoiattolo', 'aquila']
anni = [12,14,30,6,25]

# scrivi qui
```

[137]:
```
{'aquila': 25, 'cane': 12, 'gatto': 14, 'pellicano': 30, 'scoiattolo': 6}
```

Per ottenere lo stesso risultato in una linea, è possibile usare la funzione `zip` come fatto negli esercizi precedenti, e poi la funzione `dict` che crea un dizionario a partire dalla lista di coppie di elementi generata dallo `zip`:

[138]:
```
dict(zip(animali, anni))
```

[138]:
```
{'aquila': 25, 'cane': 12, 'gatto': 14, 'pellicano': 30, 'scoiattolo': 6}
```

⊛⊛ **ESERCIZIO**: Data una lista di prodotti contenente a sua volta liste ciascuna con tipologia, marca e quantità di confezioni vendute:

```
vendite = [
    ['pomodori', 'Santini', 5],
    ['pomodori', 'Cirio', 1],
    ['pomodori', 'Mutti', 2],
    ['cereali', 'Kelloggs', 3],
    ['cereali', 'Choco Pops', 8],
    ['cioccolata','Novi', 9],
    ['cioccolata','Milka', 4],
]
```

Usando un normale ciclo for, scrivere del codice Python nella cella sotto per creare un dizionario in cui le chiavi sono le tipologie e i valori sono la somma delle confezioni vendute per quella categoria:

```
{
 'cereali': 11,
 'cioccolata': 13,
 'pomodori': 8
}
```

SUGGERIMENTO: fare attenzione ai due casi, quando il dizionario da ritornare ancora non contiene la tipologia estratta dalla lista correntemente in esame e quando invece già la contiene:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[1]:
```
vendite = [
    ['pomodori', 'Santini', 5],
    ['pomodori', 'Cirio', 1],
    ['pomodori', 'Mutti', 2],
    ['cereali', 'Kelloggs', 3],
    ['cereali', 'Choco Pops', 8],
    ['cioccolata','Novi', 9],
    ['cioccolata','Milka', 4],
]

# scrivi qui

vendite = [
    ['pomodori', 'Santini', 5],
    ['pomodori', 'Cirio', 1],
    ['pomodori', 'Mutti', 2],
    ['cereali', 'Kelloggs', 3],
    ['cereali', 'Choco Pops', 8],
    ['cioccolata','Novi', 9],
    ['cioccolata','Milka', 4],
]


d = {}
for vendita in vendite:
    if vendita[0] in d:
        d[vendita[0]] += vendita[2]
    else:
        d[vendita[0]] = vendita[2]
d
```

[1]:
```
{'cereali': 11, 'cioccolata': 13, 'pomodori': 8}
```

[1]:
```
vendite = [
    ['pomodori', 'Santini', 5],
    ['pomodori', 'Cirio', 1],
    ['pomodori', 'Mutti', 2],
    ['cereali', 'Kelloggs', 3],
    ['cereali', 'Choco Pops', 8],
    ['cioccolata','Novi', 9],
    ['cioccolata','Milka', 4],
]

# scrivi qui
```

[1]:
```
{'cereali': 11, 'cioccolata': 13, 'pomodori': 8}
```

4.1.14 Approfondimenti

Strumenti e script: Se vuoi approfondire come eseguire il codice in editor diversi da Jupyter e avere un'idea più precisa dell'architettura di Python, ti invitiamo a leggere Strumenti e script[91]

Gestione errori e testing: Per capire come si gestiscono le situazioni di errore in genere guarda il foglio separato Gestione errori e testing[92], serve anche per capire come eseguire alcuni esercizi della Parte A - Fondamenti.

4.2 Strumenti e script

4.2.1 Scarica zip esercizi

Naviga file online[93]

REQUISITI:

- **Aver installati Python 3 e Jupyter:** se non hai già provveduto, guarda Installazione[94]

4.2.2 L'interprete Python

In queste guide usiamo estensivamente l'editor di notebook Jupyter, perchè ci permette di eseguire comodamente codice Python, mostrare grafici e prendere note. Ma se vogliamo solo far calcolare il computer non è affatto obbligatorio!

Il modo più immediato (per quanto non molto pratico) per eseguire codice Python è usando l'interprete *da linea di comando* nella cosiddetta *modalità interattiva* , cioè facendo in modo che attenda i comandi che verranno inseriti manualmente uno per uno. Questo uso *non* richiede Jupyter, basta avere installato Python. Nota che in Mac OS X e parecchi sistemi linux come Ubuntu, Python è installato di *default* , ma a volte può capitare che non sia la versione 3. Cerchiamo di capire che versione abbiamo installata noi.

Apriamo la console di sistema

Apri una console (in Windows: menu di sistema -> Anaconda Prompt,in Mac OS X: lancia il Terminale)

Nella console trovi il cosidetto *prompt* dei comandi. In questo *prompt* puoi dare comandi direttamente al sistema operativo.

ATTENZIONE: I comandi che dai nel prompt, sono comandi nel linguaggio del sistema operativo che stai usando, **NON** nel linguaggio Python !!!!!

In Windows dovresti vedere qualcosa del genere:

```
C:\Users\David>
```

In Mac / Linux potrebbe essere una cosa simile a questa:

```
david@mio-computer:~$
```

[91] https://it.softpython.org/tools/tools-sol.html
[92] https://it.softpython.org/exercises/errors-and-testing/errors-and-testing-sol.html
[93] https://github.com/DavidLeoni/softpython-it/tree/master/tools
[94] https://it.softpython.org/installation.html

Listare i file e cartelle

Nella console di sistema, prova per esempio a

su Windows: scrivere il comando `dir` e digitare il tasto Invio.

su Mac o Linux: scrivere il comando `ls` e digitare il tasto Invio.

Dovrebbe apparire un elenco di tutti i file nella cartella corrente. Nel mio caso appare una lista del genere:

> **RIPETO**: in questo contesto `dir` ed `ls` sono comandi *del sistema operativo*, **NON** di Python !!

Windows:

```
C:\Users\David> dir
Arduino                     gotysc                          program.wav
a.txt                       index.html                      Public
CART                        java0.log                       RegDocente.pdf
backupsys                   java1.log
BaseXData                   java_error_in_IDEA_14362.log
```

Mac / Linux:

```
david@david-computer:~$ ls
Arduino                           gotysc                          program.wav
a.txt                             index.html                      Public
CART                              java0.log                      ␣
↪RegistroDocenteStandard(1).pdf
backupsys                         java1.log                      ␣
↪RegistroDocenteStandard.pdf
BaseXData                         java_error_in_IDEA_14362.log
```

Lanciamo l'interprete Python

Sempre nella console di sistema aperta, scrivi semplicemente il comando `python`:

> **ATTENZIONE**: Se Python non parte, prova a scrivere `python3` con il `3` alla fine di `python`

```
C:\Users\David> python
```

Dovresti veder apparire qualcosa di simile (ma molto probabilmente NON uguale). Nota che nella prima riga è contenuta la versione di Python. Se inizia con `2.`, allora non stai usando quella giusta per questo corso - in quel caso prova a uscire dall'interprete (*vedi come uscire*) e poi scrivi `python3`

```
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on windows
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

> **ATTENZIONE** al triplo maggiore >>> all'inizio!

Il triplo maggiore >>> all'inizio indica che a differenza di prima adesso la console si sta aspettando comandi *in linguaggio Python*. Quindi, i comandi di sistema che abbiamo usato prima (`cd`, `dir`, ...) NON funzioneranno più, o daranno risultati diversi !!!!

Adesso la console si aspetta comandi Python, prova ad inserire `3 + 5` e poi premi `Invio`:

ATTENZIONE tu NON scrivere >>>, scrivi solo il comando che appare dopo !

```
>>> 3 + 5
```

Dovrebbe apparire la scritta `8`

```
8
```

Oltre ai calcoli, potremmo dire a Python di stampare qualcosa con la funzione `print("ciao")`

```
>>> print("ciao")
ciao
```

Uscire dall'interprete

Per uscire dall'interprete Python e tornare al prompt di sistema (per intenderci, quello che accetta i comandi `cd` e `dir`), scrivi il comando Python `exit()`

Dopo che sei effettivamente uscito dall'interprete Python, il triplo >>> deve essere sparito (cioè non deve stare all'inizio della linea dove darai nuovi comandi)

In Windows, dovresti vedere un risultato simile:

```
>>> exit()
C:\Users\David>
```

in Mac / Linux potrebbe essere così:

```
>>> exit()
david@mio-computer:~$
```

Adesso potresti tornare ad eseguire comandi per il sistema operativo come `dir` e `cd`

Windows:

```
C:\Users\David> dir
Arduino                    gotysc                          program.wav
a.txt                      index.html                      Public
CART                       java0.log                       RegDocente.pdf
backupsys                  java1.log
BaseXData                  java_error_in_IDEA_14362.log
```

Mac:

```
david@david-computer:~$ ls
Arduino                            gotysc                          program.wav
a.txt                              index.html                      Public
```

(continues on next page)

(continued from previous page)

```
CART                                java0.log                    ␣
↪RegistroDocenteStandard(1).pdf
backupsys                           java1.log                    ␣
↪RegistroDocenteStandard.pdf
BaseXData                           java_error_in_IDEA_14362.log
```

4.2.3 Moduli

I moduli Python sono semplicemente dei file di testo che hanno l'estensione **.py** (per es. `prova.py`). Quando stai scrivendo codice in un editor, di fatto stai implementando il modulo corrispondente.

In Jupyter usiamo file notebook con estensione `.ipynb`, ma per editarli serve necessariamente Jupyter.

Con i file `.py` (detti anche *script*) possiamo invece usare un qualsiasi editor di testo, e possiamo poi dire all'interprete di eseguire il file. Vediamo come si fa.

Editor di testo semplice

1. Con un editor di testo (*Blocco note* in Windows, o *TextEdit* in Mac Os X) crea un file di testo, e metti all'interno questo codice

```
x = 3
y = 5
print(x + y)
```

2. Proviamo a salvarlo - sembra semplice, ma non lo è sempre, leggi bene!

> **ATTENZIONE**: Al momento di salvare il file, **assicurati che il file abbia estensione `.py` !!**

Supponiamo di creare il file `prova.py` all'interno di una cartella chiamata `CART`:

- **WINDOWS**: Se usi *Blocco Note*, nella finestra di salvataggio devi impostare *Salva come* a *Tutti i file* (altrimenti il file verrà salvato erroneamente come `prova.py.txt` !)
- **MAC**: se usi *TextEdit*, prima di salvare clicca *Formato* e poi *Converti in formato Solo testo*: **se dimentichi questo passaggio, TextEdit nella finestra di salvataggio non ti darà modo di salvarlo nel formato giusto e probabilmente finirai con un file .rtf che non ci interessa**

3. Apri una console (in Windows: menu di sistema -> Anaconda Prompt,in Mac OS X: lancia il Terminale)

La console apre il cosidetto *prompt* dei comandi. In questo *prompt* puoi dare comandi direttamente al sistema operativo (vedi *paragrafo precedente*)

> **ATTENZIONE**: I comandi che dai nel prompt, sono comandi nel linguaggio del sistema operativo che stai usando, **NON** nel linguaggio Python !!!!!

In Windows dovresti vedere qualcosa del genere:

```
C:\Users\David>
```

In Mac / Linux :

```
david@david-mio-computer:~$
```

Prova per esempio a scrivere il comando `dir` (o `ls` per Mac / Linux) che mostra tutte i file nella cartella corrente. Nel mio caso appare una lista del genere:

RIPETO: in questo contesto `dir` / `ls` sono comando *del sistema operativo*, **NON** di Python.

```
C:\Users\David> dir
Arduino                    gotysc                          program.wav
a.txt                      index.html                      Public
CART                       java0.log                       RegDocente.pdf
backupsys                  java1.log
BaseXData                  java_error_in_IDEA_14362.log
```

Se noti, nella lista appare `CART`, dove ho messo `prova.py`. Per *entrare* nella cartella dal *prompt*, devi usare il comando del sistema operativo `cd` come segue

4. Per entrare in una cartella che si chiama CART, scrivere `cd CART`:

```
C:\Users\David> cd CART
C:\Users\David\CART>
```

E se sbaglio cartella?

Se per caso entri nella cartella sbagliata, tipo `CAVOLATE`, per andare indietro di una cartella, scrivere `cd ..` (NOTA: `cd` è seguito da spazio e DUE punti `..` *uno dopo l'altro*)

```
C:\Users\David\CAVOLATE> cd ..
C:\Users\David\>
```

5. Assicurati di essere nella cartella che contiene `prova.py`. Se non lo sei, usa i comandi `cd` e `cd ..` come sopra per navigare le cartelle.

Vediamo cosa c'è in CART con il comando di sistema `dir` (o `ls` se usi Mac/Linux):

RIPETO: in questo contesto `dir` (o `ls`) è un comando *del sistema operativo*, **NON** di Python.

```
C:\Users\David\CART> dir

prova.py
```

`dir` ci sta dicendo che in `CART` c'è il nostro file `prova.py`.

6. Da dentro la cartella `CART`, scrivi `python prova.py`

```
C:\Users\David\CART>python prova.py
```

ATTENZIONE: Se Python non parte, prova a scrivere `python3 prova.py` con il `3` alla fine di `python`

Se tutto è andato bene, dovresti veder apparire

```
8
C:\Users\David\CART>
```

ATTENZIONE: Dopo l'esecuzione di uno script in questo modo, la console si aspetta di nuovo comandi *di sistema*, **NON** comandi Python (quindi non dovrebbe esserci il triplo maggiore >>>)

IDE

In queste guide lavoriamo su notebook Jupyter con estensione `.ipynb`, ma per editare file sorgenti `.py` lunghi conviene usare editor più tradizionali, detti anche IDE (*Integrated Development Environment*). Per Python possiamo usare Spyder[95], Visual Studio Code[96] o PyCharme Community Edition[97].

Rispetto a Jupyter, questi editor permettono più agevolmente di *debuggare* e *testare* il codice.

Facciamo una prova con Spyder, che è il più semplice - se hai Anaconda, lo trovi disponibile dentro l'Anaconda Navigator.

INFO: Quando lanci Spyder, potrebbe richiederti di effettuare un aggiornamento (*upgrade*), ma puoi tranquillamente cliccare No.

Nella parte sinistra dell'editor si vede il codice del file `.py` che stai editando. Tali file sono detti anche *script* . Nella parte destra in basso si vede la console con l'interprete IPython (che è lo stesso alla base di Jupyter, qui in versione testuale). Quando esegui lo script, è come immettere i comandi in quell'interprete.

- Per eseguire tutto lo script: premi `F5`
- Per eseguire solo la linea corrente o la selezione: premi `F9`
- Per ripulire la memoria: dopo molte esecuzioni le variabili nella memoria dell'interprete potrebbero assumere valori che non ti aspetti. Per ripulire la memoria, clicca sull'ingranaggio a destra del riquadro della console, e seleziona *Restart kernel*

ESERCIZIO: fai delle prove, riprendendo il file `prova.py` di prima:

```
x = 3
y = 5
print(x + y)
```

- una volta che il codice è nello script, premi `F5`
- seleziona solo `print(x+y)` e premi F9
- seleziona solo `x=3`e premi F9
- clicca sull'ingranaggio a destra del riquadro della console, e seleziona *Restart kernel*, poi seleziona solo `print(x+y)` e premi `F9`. Che succede?

Ricordati che se la memoria dell'interprete è stata ripulita con *Restart kernel* e poi provi ad eseguire una riga di codice con variabili definite in linee che non sono stati eseguite prima, Python non saprà a che variabili ti stai riferendo e mostrerà un `NameError`

[95] https://www.spyder-ide.org/
[96] https://code.visualstudio.com/Download
[97] https://www.jetbrains.com/pycharm/download/

4.2.4 Jupyter

Jupyter è un editor che ti permette di lavorare sui cosiddetti *notebook* , che sono file che finiscono con l'estensione `.ipynb`. Sono documenti divisi in celle dove per ogni cella puoi immettere dei comandi e vederne subito il rispettivo output. Proviamo ad aprire questo.

1. scompatta lo zip degli esercizi in una cartella, dovresti ottenere qualcosa del genere:

```
tools
    tools-sol.ipynb
    tools.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

2. apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `tools.ipynb`

ATTENZIONE: apri quello SENZA il `-sol` alla fine!

Vedere subito le soluzioni è troppo facile ;-)

ATTENZIONE: Se non trovi Jupyter / qualcosa non funziona, guarda la guida per l'installazione[98]

[98] https://it.softpython.org/installation.html#Notebook-Jupyter

3. Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

> **ATTENZIONE**: In questo libro usiamo **SOLO PYTHON 3**
>
> Se per caso ottieni comportamenti inattesi, controlla di usare Python 3 e non il 2. Se per caso il tuo sistema operativo scrivendo `python` fa partire il 2, prova ad eseguire il tre scrivendo il comando: `python3`

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

ESERCIZIO: Proviamo a inserire un comando Python: scrivi nella cella qua sotto `3 + 5`, e poi mentre sei in quella cella premi i tasti speciali `Control+Invio`. Come risultato, dovresti vedere apparire il numero `8`

[ ]:
```
```

ESERCIZIO: in Python possiamo scrivere commenti iniziando una riga con un cancelletto `#`. Come prima, scrivi nella cella sotto `3 + 5` ma questa volta scrivilo nella riga sotto la scritta `# scrivi qui`:

[2]:
```
# scrivi qui
```

ESERCIZIO: Jupyter per ogni cella mostra il risultato solo dell'ultima riga eseguita in quella cella. Prova a inserire questo codice nella cella sotto ed esegui premendo `Control+Invio`. Che risultato appare?

```
3 + 5
1 + 1
```

[3]:
```
# scrivi qui
```

ESERCIZIO: Proviamo adesso a creare noi una nuova cella.

- Mentre sei con il cursore in questa cella, premi `Alt+Invio`. Si dovrebbe creare una nuova cella dopo la presente.
- Nella cella appena creata, inserisci `2 + 3` e poi premi `Shift+Invio`. Cosa succede al cursore? prova la differenze con `Control+Invio`. Se non capisci la differenza, prova a premere ripetutamente `Shif+Invio` e vedi che succede.

Stampare una espressione

Proviamo ad assegnare una espressione ad una variabile:

[4]:
```
monete = 3 + 2
```

Nota che l'assegnazione di per sè non produce nessun output nella cella di Jupyter. Possiamo chiedere a Jupyter quanto vale la variabile semplicemente riscrivendo il nome in una cella:

[5]:
```
monete
```

[5]:
```
5
```

L'effetto è (quasi sempre) lo stesso che otterremmo chiamando esplicitamente la funzione `print`:

[6]:
```
print(monete)
```

```
5
```

Qual'è la differenza? Jupyter per nostra comodità ci mostra direttamente il risultato dell'ultima espressione eseguita in una cella, ma solo dell'ultima:

[7]:
```
monete = 4
2 + 5
monete
```

[7]:
```
4
```

Se vogliamo essere sicuri di stampare entrambe, dobbiamo usare la `print`:

[8]:
```
monete = 4
print(2 + 5)
print(monete)
```

```
7
4
```

Inoltre, il risultato dell'ultima espressione è mostrato solo nei fogli Jupyter, se stai scrivendo un normale script `.py` e vuoi vedere risultati devi in ogni caso usare la `print`.

Volendo stampare più espressioni su una stessa riga, possiamo passarle come parametri differenti a `print` separandole da una virgola:

[9]:
```
monete = 4
print(2+5,monete)
```

```
7 4
```

A `print` possiamo passare quante espressioni vogliamo:

[10]:
```
monete = 4
print(2 + 5, monete, monete*3)
```

```
7 4 12
```

Se vogliamo mostrare anche del testo, possiamo scriverlo creando delle cosiddette *stringhe* tra doppie virgolette (in seguito vedremo le stringhe molto più nel dettaglio):

[11]:
```
monete = 4
print("Abbiamo", monete, "monete d'oro, ma ci piacerebbe averne il doppio:", monete *␣
↪2)
```

```
Abbiamo 4 monete d'oro, ma ci piacerebbe averne il doppio: 8
```

DOMANDA: Guarda le espressioni seguenti, e per ciascuna cerca di indovinare quale risultato produce. Prova a verificare le tue supposizioni sia in Jupyter che un'altro editor di file `.py` come Spyder:

1.
```
x = 1
x
x
```

2.
```
x = 1
x = 2
print(x)
```

3.
```
x = 1
x = 2
x
```

4.
```
x = 1
print(x)
x = 2
print(x)
```

5.
```
print(zam)
print(zam)
zam = 1
zam = 2
```

6.
```
x = 5
print(x,x)
```

7.
```
x = 5
print(x)
print(x)
```

8.
```
tappeti = 8
lunghezza = 5
print("Se ho", tappeti, "tappeti in fila cammino per", tappeti * lunghezza,
↪"metri.")
```

9.
```
tappeti = 8
lunghezza = 5
print("Se", "ho", tappeti, "tappeti", "in","fila","cammino", "per", tappeti *␣
↪lunghezza, "metri.")
```

Esercizio - castelli per aria

Date due variabili

```
castelli = 7
dirigibili = 4
```

scrivi del codice che stampa:

```
Ho costruito 7 castelli per aria
Ho 4 dirigibili a vapore
Ne voglio parcheggiare uno per castello
perciò ne acquisterò altri 3 al Mercato del Vapore
```

- **NON** mettere costanti numeriche nel tuo codice come `7`, `4` o `3`! Scrivi del codice generico che usa solo le variabili fornite.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
castelli = 7
dirigibili = 4
# scrivi qui
print("Ho costruito",castelli, "castelli per aria")
print("Ho", dirigibili, "dirigibili a vapore")
print("Ne voglio parcheggiare uno per castello")
print("perciò ne acquisterò altri", castelli - dirigibili, "al Mercato del Vapore.")
```

```
Ho costruito 7 castelli per aria
Ho 4 dirigibili a vapore
Ne voglio parcheggiare uno per castello
perciò ne acquisterò altri 3 al Mercato del Vapore.
```

[12]:
```
castelli = 7
dirigibili = 4
# scrivi qui



```

```
Ho costruito 7 castelli per aria
Ho 4 dirigibili a vapore
Ne voglio parcheggiare uno per castello
perciò ne acquisterò altri 3 al Mercato del Vapore.
```

4.2.5 Visualizzare l'esecuzione con Python Tutor

Abbiamo visto i principali tipi di dati. Prima di procedere oltre, è bene vedere gli strumenti giusti per comprendere al meglio cosa succede quando si esegue il codice. Python tutor[99] è un ottimo sito online per visualizzare online l'esecuzione di codice Python, permettendo di andare avanti e *indietro* nell'esecuzione del codice. Sfruttatelo più che potete, dovrebbe funzionare con parecchi degli esempi che tratteremo a lezione. Vediamo un esempio.

Python tutor 1/4

[99] http://pythontutor.com/

Vai sul sito pythontutor.com[100] e seleziona *Python 3*

Python tutor 2/4

Assicurati che sia selezionato almeno Python 3.6:

[100] http://pythontutor.com/

Python tutor 3/4

Inserisci questo codice:

```
x = 5
y = 7
z = x + y
```

e clicca Visualize Execution:

Python tutor 4/4

Cliccando su Next visualizzerai i cambiamenti nella memoria di Python:

Debuggare codice in Jupyter

Python tutor è fantastico, ma quando esegui del codice in Jupyter e non funziona, come si può fare? Per ispezionare l'esecuzione, gli editor di solito mettono a disposizione uno strumento chiamato *debugger*, che permette di eseguire le istruzioni una per una. Al momento (Agosto 2018), il debugger di Jupyter che si chiama pdb[101] è estramamente limitato . Per superarne le limitazioni, in questo corso ci siamo inventati una soluzione di ripiego, che sfrutta Python Tutor.

Se inserisci del codice Python in una cella, e poi **alla fine della cella** scrivi l'istruzione `jupman.pytut()`, come per magia il codice precedente verrà visualizzato all'interno del foglio Jupyter con il debugger di Python Tutor.

> **ATTENZIONE**: `jupman` è una collezione di funzioni di supporto che ci siamo inventati apposta per questo corso.
>
> Quando vedi comandi che iniziano con `jupman`, affinchè funzionino devi prima eseguire la cella in cima al documento. Riportiamo tale cella qua per comodità. Se non lo hai già fatto, eseguila adesso.

[13]:
```
# Ricordati di eseguire questa cella con Control+Invio
# Questi comandi dicono a Python dove trovare il file jupman.py
import jupman;
```

Adesso siamo pronti a provare Python tutor con la funzione magica `jupman.pytut()`:

[14]:
```
x = 5
y = 7
z = x + y

jupman.pytut()
```

[14]:
```
<IPython.core.display.HTML object>
```

Python Tutor : Limitazione 1

Python tutor è comodo, ma ci sono importanti limitazioni:

> **ATTENZIONE**: Python Tutor guarda dentro una cella sola!
>
> Quando usi Python tutor dentro Jupyter, l'unico codice che viene considerato da Python tutor è quello dentro la cella dove sta il comando `jupman.pytut()`.

Quindi per esempio in queste due celle che seguono, solo `print(w)` apparirà dentro Python tutor senza il `w = 3`. Se proverai a cliccare *Forward* in Python tutor, ti verrà segnalato che non è stata definita `w`

[15]:
```
w = 3
```

[16]:
```
print(w)

jupman.pytut()
```

```
3
```

```
Traceback (most recent call last):
  File "../jupman.py", line 2305, in _runscript
```

(continues on next page)

[101] https://davidhamann.de/2017/04/22/debugging-jupyter-notebooks/

(continued from previous page)

```
    self.run(script_str, user_globals, user_globals)
  File "/usr/lib/python3.5/bdb.py", line 431, in run
    exec(cmd, globals, locals)
  File "<string>", line 2, in <module>
NameError: name 'w' is not defined
```

[16]:
```
<IPython.core.display.HTML object>
```

Per funzionare in Python Tutor devi mettere TUTTO il codice nella STESSA cella:

[17]:
```
w = 3
print(w)

jupman.pytut()
```

```
3
```

[17]:
```
<IPython.core.display.HTML object>
```

Python Tutor : Limitazione 2

ATTENZIONE: Python Tutor usa solo funzioni dalla distribuzione standard di Python

Python Tutor va bene per ispezionare semplici algoritmi con funzioni di base di Python, se usi librerie fatte da terze parti non funzionerà.

Se usi qualche libreria tipo `numpy`, puoi provare **solo online** a selezionare `Python 3.6 with Anaconda`:

Write code in Python 3.6 with Anaconda (experimental)

```
1
2 import numpy as np
3
4 a = np.arange(15).reshape(3,5)
5 print(a)
```

Visualize Execution

Esercizio - taverna

Date le variabili

```
pirati = 10
ognuno_vuole = 5      # boccali di grog
barili = 4
capienza_barile = 20  # boccali di grog
```

Prova a scrivere qua sotto del codice che stampi questo:

```
Nella taverna ci sono 10 pirati, ognuno vuole 5 boccali di grog
Abbiamo 4 barili di grog pieni
Da ogni barile possiamo prendere 20 boccali
Stasera i pirati berranno 50 boccali, ne avanzano 30 per domani
```

- **NON** usare costanti numeriche nel tuo codice, usa invece le variabili proposte
- Per tener traccia dei boccali avanzati, crea una variabile apposita `boccali_avanzati`
- Se stai usando Jupyter, prova ad usare `jupman.pytut()` alla fine della cella per visualizzare l'esecuzione

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
pirati = 10
ognuno_vuole = 5      # boccali di grog
barili = 4
capienza_barile = 20  # boccali di grog

# scrivi qui
print("Nella taverna ci sono", pirati, "pirati, ognuno vuole", ognuno_vuole, "boccali␣
↪di grog")
print("Abbiamo", barili, "barili di grog pieni")
print("Da ogni barile possiamo prendere", capienza_barile,"boccali")
boccali_avanzati = barili*capienza_barile - pirati*ognuno_vuole
print("Stasera i pirati berranno", pirati * ognuno_vuole, "boccali, ne avanzano",␣
↪boccali_avanzati, "per domani")

#jupman.pytut()
```

```
Nella taverna ci sono 10 pirati, ognuno vuole 5 boccali di grog
Abbiamo 4 barili di grog pieni
Da ogni barile possiamo prendere 20 boccali
Stasera i pirati berranno 50 boccali, ne avanzano 30 per domani
```

[18]:
```
pirati = 10
ognuno_vuole = 5      # boccali di grog
barili = 4
capienza_barile = 20  # boccali di grog

# scrivi qui
```

```
Nella taverna ci sono 10 pirati, ognuno vuole 5 boccali di grog
Abbiamo 4 barili di grog pieni
Da ogni barile possiamo prendere 20 boccali
Stasera i pirati berranno 50 boccali, ne avanzano 30 per domani
```

4.2.6 Architettura di Python

La parte che segue non è strettamente fondamentale ai fini del libro, ma serve per capire cosa succede sotto il cofano quando esegui dei comandi.

Torniamo a Jupyter: l'editor di notebook Jupyter è uno strumento molto potente e flessibile, permette di eseguire codice Python, ma non solo, anche codice scritto in altri linguaggi di programmazione (R, Bash, etc) e linguaggi di formattazione (Markdown, Latex, etc).

Quindi dobbiamo tenere presente che il codice Python che scrivi nelle celle di questi notebook Jupyter (che sono file con estensione `.ipynb`) non viene certo magicamente compreso dal tuo computer. Sotto il cofano, vengono effettuate diverse trasformazioni che permettono al processore del tuo computer di capire che istruzioni vuoi che esegua. Riportiamo qua le principali trasformazioni che avvengono, a partire da Jupyter fino al processore:

Python è un linguaggio d'alto livello

Cerchiamo di capire bene che succede quando esegui una cella:

1. **codice sorgente**: Prima il notebook Jupyter guarda se hai scritto del *codice sorgente* Python nella cella (invece che altri linguaggi come di programmazione come R, Bash o formattazione come Markdown …). Di default Jupyter assume che il codice che stai scrivendo è Python. Supponiamo che ci sia il seguente codice:

```
x = 3
y = 5
print(x + y)
```

ESERCIZIO: Senza andare nel dettaglio del codice, prova a copia/incollarlo nella cella qua sotto. Assicurandoti di avere il cursore nella cella, eseguila digitando `Control + Invio`. Quando la esegui dovrebbe apparire un `8` come risultato del calcolo. Il `# scrivi qui sotto` come tutte le righe che iniziano con un cancelletto `#` è solo un commento che verrà ignorato da Python:

```
[19]: # scrivi qui sotto
```

Se sei riuscito ad eseguire il codice, puoi fare i complimenti a Python! Ti ha permesso di eseguire un programma scritto in un linguaggio abbastanza comprensibile *indipendentemente* dal tuo sistema operativo (Windows, Mac Os X, Linux …) e dal processore del tuo computer (x86, ARM, …)! Non solo, l'editor di notebook Jupyter ti ha anche fatto la cortesia di mostrare il risultato nel tuo browser.

Più in dettaglio, cos'è successo ? Vediamo meglio:

2. **bytecode**: Al momento della richiesta d'esecuzione, Jupyter ha preso il testo scritto dalla cella, e lo ha mandato al cosidetto *compilatore Python* che lo ha trasformato in *bytecode*. Il *bytecode* è una sequenza di istruzioni più lunga e meno comprensibile per noi umani (**è solo un esempio, non serve che lo capisci !!**):

```
2           0 LOAD_CONST               1 (3)
            3 STORE_FAST               0 (x)

3           6 LOAD_CONST               2 (5)
            9 STORE_FAST               1 (y)

4          12 LOAD_GLOBAL              0 (print)
           15 LOAD_FAST                0 (x)
           18 LOAD_FAST                1 (y)
           21 BINARY_ADD
           22 CALL_FUNCTION            1 (1 positional, 0 keyword pair)
           25 POP_TOP
           26 LOAD_CONST               0 (None)
           29 RETURN_VALUE
```

3. **codice macchina**: L'*interprete Python* ha preso il *bytecode* qua sopra una istruzione alla volta, e lo ha convertito in *codice macchina* che può essere effettivamente compreso dal processore (CPU) del computer. A noi il *codice macchina* può sembrare ancora più lungo e più brutto del *bytecode* ma il processore è contento e leggendolo produce i risultati del programma. Esempio di *codice macchina* (**è solo un esempio, non serve che lo capisci !!**):

```
mult:
    push rbp
    mov rbp, rsp
    mov eax, 0
mult_loop:
    cmp edi, 0
```

(continues on next page)

(continued from previous page)

```
    je mult_end
    add eax, esi
    sub edi, 1
    jmp mult_loop
mult_end:
    pop rbp
    ret
```

Riportiamo quanto detto qua sopra in questa tabella. Nella tabella esplicitiamo anche l'estensione dei file in cui possiamo scrivere i vari formati del codice.

- Quelle che interessano a noi sono i notebook Jupyter `.ipynb` ed i file di codice sorgente Python `.py`
- I `.pyc` possono venire generati dal compilatore quando legge i file `.py`, ma a noi non interessano - non avremo mai la necessità di editarli
- Il codice macchina `asm` pure non ci interessa

Strumento	Linguaggio	File	esempio
Notebook Jupyter	Python	.ipynb	
Compilatore Python	codice sorgente Python	.py	`x = 3 y = 5print(x + y )`
Interprete Python	bytecode Python	.pyc	`0 LOAD_CONST 1 (3)3 STORE_FAST 0 (x)`
Processore (CPU)	Codice macchina	.asm	`cmp edi, 0je mult _end`

Adesso che abbiamo un idea di quello che succede, possiamo forse capire meglio l'affermazione che *Python è un linguaggio d'alto livello*, cioè sta alto nella tabella qua sopra: quando scriviamo codice Python, non ci interessa per nulla del *bytecode* o del *codice macchina* che viene generato, ma **possiamo concentrarci sulla logica del programma**. Inoltre, il codice Python che scriviamo è **indipendente dall'architettura dei pc**: se abbiamo un interprete Python installato su un computer, si prenderà cura lui di convertire il codice d'alto livello nel codice macchina particolare di quella architettura, che include sistema operativo (Windows / Mac Os X / Linux) e processore (x86, ARM, PowerPC, etc).

Performance

Tutto ha un prezzo. Se vogliamo poter scrivere programmi concentrandoci sulla *logica ad alto livello* senza entrare nei dettagli di come viene interpretato dal processore, dobbiamo tipicamente rinunciare alle *performance*. Python, essendo un linguaggio *interpretato* ha il difetto di essere lento. Ma se abbiamo bisogno di efficienza, che facciamo ? Per fortuna Python può essere esteso con codice scritto in *linguaggio C* che tipicamente è molto più performante. Infatti, anche se non te ne accorgerai, tante delle funzionalità di Python sotto sotto sono state scritte direttamente nel veloce linguaggio C. Se proprio abbiamo bisogno di performance (ma non in questo corso!) può valer la pena prima scrivere un prototipo in Python e poi, una volta stabilito che funziona, compilarlo in *linguaggio C* con il compilatore Cython[102] e ottimizzare a mano il codice generato.

```
[ ]:
```

[102] http://cython.org/

4.3 Basi Python - variabili e interi

4.3.1 Scarica zip esercizi

Naviga file online[103]

REQUISITI:

- **Installato Python 3 e Jupyter:** se non hai già provveduto, guarda Installazione[104]
- **Letto** Strumenti e script[105]

4.3.2 Jupyter

Jupyter è un editor che ti permette di lavorare sui cosiddetti *notebook*, che sono file che finiscono con l'estensione `.ipynb`. Sono documenti divisi in celle dove per ogni cella puoi immettere dei comandi e vederne subito il rispettivo output. Proviamo ad aprire questo.

1. scompatta lo zip degli esercizi in una cartella, dovresti ottenere qualcosa del genere:

```
basics
    basics1-ints.ipynb
    basics1-ints-sol.ipynb
    basics2-bools.ipynb
    basics2-bools-sol.ipynb
    basics3-floats.ipynb
    basics3-floats-sol.ipynb
    basics4-chal.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

2. apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `basics1-ints.ipynb`

ATTENZIONE: apri quello SENZA il `-sol` alla fine!

Vedere subito le soluzioni è troppo facile ;-)

ATTENZIONE: Se non trovi Jupyter / qualcosa non funziona, guarda la guida per l'installazione[106]

3. Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

[103] https://github.com/DavidLeoni/softpython-it/tree/master/basics
[104] https://it.softpython.org/installation.html
[105] https://it.softpython.org/tools/tools-sol.html
[106] https://it.softpython.org/installation.html#Notebook-Jupyter

ATTENZIONE: In questo libro usiamo **SOLO PYTHON 3**

Se per caso ottieni comportamenti inattesi, controlla di usare Python 3 e non il 2. Se per caso il tuo sistema operativo scrivendo `python` fa partire il 2, prova ad eseguire il tre scrivendo il comando: `python3`

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.3.3 Oggetti

Per Python qualunque cosa è un oggetto. Gli oggetti hanno **proprietà** (campi dove salvare valori) e **metodi** (cose che possono fare). Per esempio, un oggetto **auto** ha le *proprietà* modello, marca, colore, numero di porte etc …, e i *metodi* sterza a destra, sterza a sinistra, accelera, frena, cambia marcia …

Secondo la documentazione ufficiale di Python:

```
Gli oggetti sono l'astrazione Python per i dati. Tutti i dati in un programma Python␣
↪sono rappresentati da oggetti o da relazioni tra gli oggetti.
```

Tutto quello che devi sapere per adesso è che gli oggetti Python hanno un **identificatore** (per es, il loro nome), un **tipo** (numeri, testo, collezioni, …) e un **valore** (il dato attuale rappresentato dagli oggetti). Una volta che un'oggetto è stato creato l' *identificatore* e il *tipo* non cambiano mai, mentre il *valore* può cambiare (**oggetti mutabili**) o rimanere costante (**oggetti immutabili**).

Python fornisce questi tipi predefiniti (*built-in*):

Tipo	Significato	Dominio	Mutabile?
`bool`	Condizione	`True`, `False`	no
`int`	Intero	$\mathbb{Z}$	no
`long`	Intero	$\mathbb{Z}$	no
`float`	Razionale	$\mathbb{Q}$ (più o meno)	no
`str`	Testo	Testo	no
`list`	Sequenza	Collezione di oggetti	sì
`tuple`	Sequenza	Collezione di oggetti	no
`set`	Insieme	Collezione di oggetti	sì
`dict`	Mappatura	Mappatura tra oggetti	sì

Per il momento guarderemo i più semplici, più avanti li approfondiremo tutti.

4.3.4 Variabili

Le variabili sono associazioni tra nomi e oggetti (possiamo anche chiamarli valori)

Le variabili possono essere associate o in termine più tecnico *assegnate* agli oggetti usando l'operatore di assegnazione =.

L'istruzione

```
[2]: diamanti = 4
```

può rappresentare quante pietre preziose teniamo in cassaforte. Cosa succede quando l'eseguiamo in Python?

- un oggetto è creato
- il suo tipo è impostato a `int` (un numero intero)
- il suo valore è impostato a 4
- un nome `diamanti` è creato nell'ambiente e assegnato a quell'oggetto

Rilevare il tipo di una variabile

Quando vedi una variabile o una costante e hai dubbi sul suo tipo, puoi usare la funzione predefinita `type`:

```
[3]: type(diamanti)
[3]: int
```

```
[4]: type(4)
[4]: int
```

```
[5]: type(4.0)
[5]: float
```

```
[6]: type("Ciao")
[6]: str
```

Riassegnare una variabile

Considera adesso il codice seguente:

```
[7]: diamanti = 4

     print(diamanti)
     4
```

```
[8]: diamanti = 5

     print(diamanti)
     5
```

Il valore della variabile `diamanti` è stato cambiato da `4` a `5`, ma come riportato nella tabella sopra, il tipo `int` è **immutabile**. Fortunatamente, questo non ci ha ostacolato nel cambiare il valore di `diamanti` da `4` a `5`. Cosa è successo dietro le scene? Quando abbiamo eseguito le istruzioni `diamanti = 5`, un nuovo oggetto di tipo `int` è stato creato (`5` è un intero) e quindi reso disponibile con lo stesso nome `diamanti`, ma dato che è un oggetto differente (l'intero `5`).

Riusare una variabile

Quando riassegni una variabile ad un'altro valore, per calcolare il nuovo valore puoi tranquillamente riusare il valore vecchio della variabile che vuoi cambiare. Per esempio, supponi di avere la variabile

[9]:
```
fiori = 4
```

e vuoi aumentare il numero di `fiori` corrente di uno. Puoi scrivere così:

[10]:
```
fiori = fiori + 1
```

Cos'è successo? Quando Python incontra un comando con l'=, PRIMA calcola il valore dell'espressione che trova a destra dell'=, e POI assegna quel valore alla variabile che trova a sinistra dell'=.

Dato quest'ordine, nell'espressione a destra viene usato il valore vecchio della variabile (in questo caso `4`) a cui viene sommato `1` per ottenere `5` che viene quindi assegnato a `fiori`:

[11]:
```
fiori
```

[11]:
```
5
```

In modo del tutto equivalente, potremmo riscrivere il codice così, usando una variabile d'appoggio `x`. Vediamolo in Python Tutor:

[12]:
```
# ATTENZIONE: per usare la funzione jupman.pytut() di seguito,
# è necessario eseguire prima questa cella con Shift+Invio

# basta eseguirla una volta sola, la trovi presente anche in tutti i fogli nella␣
↪prima cella

import jupman
```

[13]:
```

fiori = 4

x = fiori + 1

fiori = x

jupman.pytut()
```

[13]:
```
<IPython.core.display.HTML object>
```

Puoi eseguire una somma e fare un assegnamento contemporaneamente con la notazione +=

[14]:
```
fiori = 4
fiori += 1
print(fiori)
```

```
5
```

Questa notazione vale anche per altri operatori aritmetici:

[15]:
```
fiori = 5
fiori -= 1      # sottrazione
print(fiori)
```

```
4
```

[16]:
```
fiori *= 3      # moltiplicazione
print(fiori)
```

```
12
```

[17]:
```
fiori /= 2      # divisione
print(fiori)
```

```
6.0
```

Assegnazioni - domande

DOMANDA: Guarda le espressioni seguenti, e per ciascuna cerca di indovinare quale risultato produce (o se da errore). Prova a verificare le tue supposizioni sia in Jupyter che un'altro editor di file `.py` come Spyder:

1.
```
x = 1
x
x
```

2.
```
x = 1
x = 2
print(x)
```

3.
```
x = 1
x = 2
x
```

4.
```
x = 1
print(x)
x = 2
print(x)
```

5.
```
print(zam)
print(zam)
zam = 1
zam = 2
```

6.
```
x = 5
print(x,x)
```

7.
```
x = 5
print(x)
print(x)
```

8.
```
x = 3
print(x,x*x,x**x)
```

9.
```
3 + 5 = x
print(x)
```

10.
```
3 + x = 1
print(x)
```

11.
```
x + 3 = 2
print(x)
```

12.
```
x = 2
x =+ 1
print(x)
```

13.
```
x = 2
x = +1
print(x)
```

14.
```
x = 2
x += 1
print(x)
```

15.
```
x = 3
x *= 2
print(x)
```

Esercizio - scambia

⊛ Date due variabili `a` e `b`:

```
a = 5
b = 3
```

scrivi del codice che scambia i loro valori, per cui dopo il tuo codice deve risultare

```
>>> print(a)
3
>>> print(b)
5
```

- Bastano due variabili? Se non sono sufficienti, prova ad introdurne una terza.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
a = 5
b = 3

# scrivi qui
temp = a    # associamo 5 alla variabile temp, facendo di fatto una copia
a = b       # riassegniamo a al valore di b, cioè 3
b = temp    # riassegniamo b al valore di temp, cioè 5
#print(a)
#print(b)
```

[18]:
```
a = 5
b = 3

# scrivi qui
```

Esercizio - ciclare

⊛ Scrivi un programma che date tre variabili con i numeri `a`, `b`, `c`, cicla i valori, cioè, mette il valore di `a` in `b`, il valore di `b` in `c`, e alla fine il valore di `c` in `a`.

Perciò se hai iniziato così:

```
a = 4
b = 7
c = 9
```

Dopo il codice che scriverai tu, eseguendo questo:

```
print(a)
print(b)
print(c)
```

Dovresti vedere:

```
9
4
7
```

Ci sono vari modi di farlo, prova ad usare **solo una** variabile temporanea e fai attenzione a non perdere i valori !

SUGGERIMENTO: per aiutarti, scrivi i commenti sullo stato della memoria, e pensa a quale comando usare.

```
# a b c t    di quale comando ho bisogno?
# 4 7 9
# 4 7 9 7    t = b
#
#
#
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
a = 4
b = 7
c = 9

# scrivi qui

print(a)
print(b)
print(c)
```

```
4
7
9
```

[19]:
```
a = 4
b = 7
c = 9

# scrivi qui
```

```
4
7
9
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[20]:
```
# SOLUZIONE

a = 4
b = 7
c = 9


# a b c t  di quale comando ho bisogno?
# 4 7 9
# 4 7 9 7  t = b
# 4 4 9 7  b = a
# 9 4 9 7  a = c
# 9 4 7 7  c = t


t = b
b = a
a = c
c = t

print(a)
print(b)
print(c)
```

```
9
4
7
```

[20]:

```
9
4
7
```

Cambiare il tipo durante l'esecuzione

Puoi anche cambiare il tipo di una variabile durante l'esecuzione ma quello normalmente è una **cattiva idea** perchè rende il codice più difficile da comprendere, e aumenta la probabilità di commettere errori. Facciamo un esempio:

[21]:
```
diamanti = 4          # intero
```

[22]:
```
diamanti + 2
```

[22]:
```
6
```

[23]:
```
diamanti = "quattro"  # testo
```

Adesso che `diamanti` è diventato testo, se per sbaglio proviamo a trattarlo come se fosse un numero avremo un errore !!

```
diamanti + 2

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-6124a47997d7> in <module>
----> 1 diamanti + 2

TypeError: can only concatenate str (not "int") to str
```

Comandi multipli su una stessa linea

E' possibile mettere più comandi su una stessa linea (non solo assegnazioni) separandoli da un punto e virgola `;`

[24]:
```
a = 10; print('Tanti comandi!'); b = a + 1;
```

```
Tanti comandi!
```

[25]:
```
print(a,b)
```

```
10 11
```

NOTA: I comandi multipli su stessa linea sono 'poco pythonici'

Per quanto volte utili e meno verbose che quelle con definizione esplicita, sono uno stile sconsigliato.

Inizializzazioni multiple

Cosa diversa sono le inizializzazioni multiple, separate da virgola `,` come:

[26]:
```
x,y = 5,7
```

[27]:
```
print(x)
```

```
5
```

[28]:
```
print(y)
```

```
7
```

A differenza dei comandi multipli, le assegnazioni multiple sono uno stile più accettabile.

Esercizio - scambiare come un ninja

Prova a scambiare il valore di due variabili `a` e `b` in una riga usando una riassegnazione multipla.

```
a,b = 5,3
```

Dopo il tuo codice, deve risultare

```
>>> print(a)
3
>>> print(b)
5
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

```
[29]: a,b = 5,3

      # scrivi qui
      a,b = b,a
      #print(a)
      #print(b)
```

```
[29]: a,b = 5,3

      # scrivi qui
```

Nomi di variabile

NOTA IMPORTANTE:

Puoi scegliere il nome che preferisci per le tue variabili (consiglio di scegliere qualcosa che ci ricorda il loro significato), ma devi aderire a semplici regole:

1. I nome possono solo contenere caratteri in maiuscolo/minuscolo (`A-Z`, `a-z`), numeri (`0-9`) o *underscore* _
2. I nomi non possono iniziare con un numero
3. i nomi di variabile dovrebbero iniziare con lettera minuscola
4. I nomi non possono essere uguali a parole riservate

Parole riservate:

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

Funzioni di sistema: Oltre alle parole riservate (impossibili da ridefinire), Python ha anche diverse funzioni di sistema predefinite:

- `bool`, `int`,`float`,`tuple`,`str`,`list`,`set`,`dict`
- `max`, `min`, `sum`
- `next`, `iter`
- `id`, `dir`, `vars`,`help`

Purtroppo, Python consente agli incauti di ridefinirle, ma **noi no**:

V COMANDAMENTO[107] **: Non ridifinerai mai funzioni di sistema**

Mai dichiarare variabili con questi nomi!

Nomi di variabile - domande

Per ciascuno dei nomi seguenti, prova a immaginare se è un valido *nome di variabile* oppure no, e poi prova ad assegnarlo nella cella seguente

1. `my-variable`
2. `my_variable`
3. `theCount`
4. `the count`
5. `some@var`
6. `MacDonald`
7. `7channel`
8. `channel7`
9. `stand.by`
10. `channel45`
11. `maybe3maybe`
12. `"ciao"`
13. `'hello'`
14. `as` CERCA DI CAPIRE LA DIFFERENZA *MOLTO IMPORTANTE* TRA QUESTO E I SEGUENTI DUE !!!
15. `asino`
16. `As`

[107] https://it.softpython.org/commandments.html#V-COMANDAMENTO

17. `lista` CERCA DI COMPRENDERE LA DIFFERENZA *MOLTO IMPORTANTE* TRA QUESTO E I SEGUENTI DUE !!!
18. `list` NON PROVARE NEMMENO AD ASSEGNARE QUESTO NELL'INTERPRETE (SCRIVENDO PER ES `list = 5`), SE LO FAI RENDERAI L'INTERPRETE PRATICAMENTE INUTILIZZABILE !
19. `List`
20. `black&decker`
21. `black & decker`
22. `glab()`
23. `caffè` (NOTA LA è ACCENTATA `è` !)
24. `):-]`
25. `€zone` (NOTA IL SEGNO DELL'EURO)
26. `some:pasta`
27. `aren'tyouboredyet`
28. `<angular>`

[30]:
```
# scrivi qui
```

4.3.5 Tipi numerici

Abbiamo già menzionato che i numeri sono **oggetti immutabili**. Python fornisce diversi tipi numerici: interi (`int`), reali (`float`), booleani, frazioni e numeri complessi.

E' possibile compiere operazioni aritmetiche con i seguenti operatori, in ordine di precedenza:

Operatore	Descrizione
`**`	Potenza
`+ -`	Più e meno unari
`* / // %`	Moltiplicazione, divisione, divisione intera, modulo
`+ -`	Addizione e sottrazione

Vi sono inoltre diverse funzioni predefinite:

Funzione	Descrizione
`min(x,y, ...)`	il minimo tra i numeri passati
`max(x,y, ...)`	il massimo tra i numeri passati
`abs(x)`	il valore assoluto

e altre sono disponibili nel modulo `math` (ricordati che per usarle devi prima importare il modulo `math` scrivendo `import math`:

Funzione	Descrizione
`math.floor(x)`	arrotonda x all'intero inferiore
`math.ceil(x)`	arrotonda x all'intero superiore
`math.sqrt(x)`	la radice quadrata
`math.log(x)`	il logaritmo naturale di n
`math.log(x,b)`	il logaritmo di n in base b

… più molte altre che qui non riportiamo.

4.3.6 Numeri interi

La gamma di valori che gli interi possono avere è limitata solo dalla memoria disponibile. Per lavorare con i numeri, Python fornisce degli operatori:

[31]:
```
7 + 4
```
[31]:
```
11
```

[32]:
```
7 - 4
```
[32]:
```
3
```

[33]:
```
7 // 4
```
[33]:
```
1
```

NOTA: la seguente divisione tra interi produce un risultato **float**, che come separatore per i decimali usa **il punto** (vedremo meglio in seguito):

[34]:
```
7 / 4
```
[34]:
```
1.75
```

[35]:
```
type(7 / 4)
```
[35]:
```
float
```

[36]:
```
7 * 4
```
[36]:
```
28
```

NOTA: per quanto in molti linguaggi la potenza si indica con il cappuccio `^`, invece in Python si indica con il doppio asterisco `**`:

[37]:
```
7 ** 4   # potenza
```
[37]:
```
2401
```

Esercizio - scadenza 1

⊛ Ci viene data una importante scadenza tra:

[38]:
```
giorni = 4
ore = 13
minuti = 52
```

Scrivi del codice che stampa i minuti in totale. Eseguendolo, deve risultare:

```
In totale mancano 6592 minuti
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[39]:
```
giorni = 4
ore = 13
minuti = 52

# scrivi qui
print("In totale mancano", giorni*24*60 + ore*60 + minuti, "minuti")
```

```
In totale mancano 6592 minuti
```

[39]:
```
giorni = 4
ore = 13
minuti = 52

# scrivi qui
```

```
In totale mancano 6592 minuti
```

Operatore modulo

Per trovare il resto della divisione tra due interi, possiamo usare l'operatore modulo che indichiamo con `%`:

[40]:
```
5 % 3  # 5 diviso 3 da resto 2
```

[40]:
```
2
```

[41]:
```
5 % 4
```

[41]:
```
1
```

[42]:
```
5 % 5
```

[42]:
```
0
```

[43]:
```
5 % 6
```

[43]:
```
5
```

[44]:
```
5 % 7
```

[44]:
```
5
```

Esercizio - scadenza 2

⊛ Ad un'altra importantissima scadenza importante mancano:

```
tot_minuti = 5000
```

Scrivi del codice che stampa:

```
Mancano:
   3 giorni
   11 ore
   20 minuti
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[45]:
```
tot_minuti = 5000

# scrivi qui
print('Mancano:')
print('  ', tot_minuti // (60*24), 'giorni')
print('  ', (tot_minuti % (60*24)) // 60, 'ore')
print('  ', (tot_minuti % (60*24)) % 60, 'minuti')
```

```
Mancano:
   3 giorni
   11 ore
   20 minuti
```

[45]:
```
tot_minuti = 5000

# scrivi qui
```

```
Mancano:
   3 giorni
   11 ore
   20 minuti
```

min e max

Il minimo tra due numeri può essere calcolato con la funzione`min`

[46]:
```
min(7,3)
```

[46]:
```
3
```

e il massimo con la funzione `max`:

[47]:
```
max(2,6)
```

[47]:
```
6
```

A `min` e `max` possiamo passare un numero arbitrario di parametri, anche negativi:

[48]:
```
min(2,9,-3,5)
```

[48]:
```
-3
```

[49]:
```
max(2,9,-3,5)
```

[49]:
```
9
```

V COMANDAMENTO[108] **Non ridifinerai mai funzioni di sistema come** `min` **e** `max`

[108] https://it.softpython.org/commandments.html#V-COMANDAMENTO

Se usi `min` e `max` come variabili, le funzioni corrispondenti smetteranno letteralmente di funzionare!

```
min = 4    # NOOOO !
max = 7    # NON FARLO !
```

DOMANDA: dati due qualsiasi interi `a` e `b`, quali delle seguenti espressioni risultano equivalenti?

```
1. max(a,b)
2. max(min(a,b),b)
3. -min(-a,-b)
4. -max(-a,-b)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 1. e la 3. sono equivalenti

Esercizio - trasporti

⊛ Una ditta dispone di un camion che usa abitualmente per effettuare consegne presso il suo miglior cliente. Il camion può trasportare 10 tonnellate di materiale, ma sfortunatamente le strade che può percorrere hanno dei tratti su ponti che limitano il tonnellaggio massimo. Questi limiti sono indicati in 5 variabili:

```
p1,p2,p3,p4,p5 = 7,2,4,3,6
```

Il camion deve sempre percorrere il ponte `p1`, poi per il tragitto vi sono a disposizione tre itinerari possibili:

- Nel primo itinerario, il camion percorre anche il ponte `p2`
- Nel secondo itinerario, il camion percorre anche i ponti `p3` e `p4`
- Nel terzo itinerario, il camion percorre anche il ponte `p5`

La ditta vuole sapere qual'è il tonnellaggio massimo che può far arrivare a destinazione con un viaggio. Scrivi del codice che stampa questo numero.

NOTA: non vogliamo sapere qual'è l'itinerario migliore, ci basta trovare il tonnellaggio massimo che possiamo far giungere a destinazione con un viaggio.

Esempio - dati:

```
p1,p2,p3,p4,p5 = 7,2,4,6,3
```

deve stampare

```
In un viaggio possiamo trasportare al massimo 4 tonnellate.
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[50]:
```
p1,p2,p3,p4,p5 = 7,2,4,6,3    # 4
#p1,p2,p3,p4,p5 = 2,6,2,4,5   # 2
#p1,p2,p3,p4,p5 = 8,6,2,9,5   # 6
#p1,p2,p3,p4,p5 = 8,9,9,4,7   # 8
```

(continues on next page)

(continued from previous page)

```
# scrivi qui

print('In un viaggio possiamo trasportare',
       max(min(p1,p2), min(p1,p3,p4),min(p1,p5)),
      'tonnellate')
```

```
In un viaggio possiamo trasportare 4 tonnellate
```

[50]:
```
p1,p2,p3,p4,p5 = 7,2,4,6,3   # 4
#p1,p2,p3,p4,p5 = 2,6,2,4,5  # 2
#p1,p2,p3,p4,p5 = 8,6,2,9,5  # 6
#p1,p2,p3,p4,p5 = 8,9,9,4,7  # 8


# scrivi qui
```

```
In un viaggio possiamo trasportare 4 tonnellate
```

Esercizio - divani

⊛ Il magnate De Industrionis possiede due fabbriche di divani, una a Belluno e una a Rovigo. Per realizzare un divano servono tre componenti principali: un materasso, uno schienale e una copertura di stoffa. Ogni fabbrica produce tutti i componenti necessari, impiegando un certo tempo per costruire ciascun componente:

[51]:
```
b_mat, b_sch, b_sto, r_mat, r_sch, r_sto = 23,54,12,13,37,24
```

Belluno impiega 23h a produrre un materasso, 54h lo schienale e 12h la stoffa. Rovigo, rispettivamente, 13, 37 e 24 ore. Quando i 3 componenti sono pronti, assemblarli nel divano finito richiede un'ora.

Ogni tanto arrivano richieste particolari da parte di nobili troppi facoltosi, che pretendono di vedersi consegnati nel giro di poche ore divani con modifiche stravaganti come schienali in platino massiccio e altre fesserie.

Se le due fabbriche iniziano la produzione dei componenti allo stesso tempo, De Industrionis vuole sapere in quanto tempo si produrrà il primo divano. Scrivi del codice che calcola tale numero.

- **NOTA 1**: non ci interessa sapere quale fabbrica produrrà il divano, vogliamo solo sapere il tempo più breve nel quale si otterrà un divano
- **NOTA 2**: supponi che entrambi le fabbriche **non** abbiano componenti in magazzino
- **NOTA 3**: le due fabbriche **non** si scambiano componenti

Esempio 1 - dati:

```
b_mat, b_sch, b_sto, r_mat, r_sch, r_sto = 23,54,12,13,37,24
```

il tuo codice deve stampare:

```
il primo divano verrà prodotto in 38 ore.
```

Esempio 2 - dati:

```
b_mat, b_sch, b_sto, r_mat, r_sch, r_sto = 81,37,32,54,36,91
```

il tuo codice deve stampare:

```
il primo divano verrà prodotto in 82 ore.
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[52]:
```
b_mat, b_sch, b_sto, r_mat, r_sch, r_sto = 23,54,12,13,37,24   # 38
#b_mat, b_sch, b_sto, r_mat, r_sch, r_sto = 81,37,32,54,36,91  # 82
#b_mat, b_sch, b_sto, r_mat, r_sch, r_sto = 21,39,47,54,36,91  # 48

# scrivi qui

t = min(max(b_mat, b_sch, b_sto) + 1, max(r_mat, r_sch, r_sto) + 1)

print('il primo divano verrà prodotto in', t,'ore.')
```

```
il primo divano verrà prodotto in 38 ore.
```

[52]:
```
b_mat, b_sch, b_sto, r_mat, r_sch, r_sto = 23,54,12,13,37,24   # 38
#b_mat, b_sch, b_sto, r_mat, r_sch, r_sto = 81,37,32,54,36,91  # 82
#b_mat, b_sch, b_sto, r_mat, r_sch, r_sto = 21,39,47,54,36,91  # 48

# scrivi qui
```

```
il primo divano verrà prodotto in 38 ore.
```

4.3.7 Prosegui

Continua con Basi 2 - booleani[109]

[ ]:
```

```

4.4 Basi 2 - booleani

4.4.1 Scarica zip esercizi

Naviga file online[110]

REQUISITI:

[109] https://it.softpython.org/basics/basics2-bools-sol.html
[110] https://github.com/DavidLeoni/softpython-it/tree/master/basics

- **Letto** Basi 1 - variabili e interi[111]

I booleani sono usati nell'algebra booleana e hanno il tipo `bool`.

I valori di verità sono rappresentati in Python con le parole chiave `True` e `False`: un oggetto booleano può solo avere i valori `True` o `False`.

[2]:
```
x = True
```

[3]:
```
x
```

[3]:
```
True
```

[4]:
```
type(x)
```

[4]:
```
bool
```

[5]:
```
y = False
```

[6]:
```
type(y)
```

[6]:
```
bool
```

4.4.2 Operatori tra booleani

Possiamo operare sui valori booleani con gli operatori booleani `not`, `and`, `or`:

and

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

or

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

[111] https://it.softpython.org/basics/basics1-ints-sol.html

not

a	not a
False	True
True	False

4.4.3 Booleani - Domande con costanti

DOMANDA: Per ciascuna delle seguenti espressioni booleane, prova a indovinare il risultato (*prima* indovina, e *poi* provale !):

1. `not (True and False)`
2. `(not True) or (not (True or False))`
3. `not (not True)`
4. `not (True and (False or True))`
5. `not (not (not False))`
6. `True and (not (not((not False) and True)))`
7. `False or (False or ((True and True) and (True and False)))`

4.4.4 Domande con variabili

DOMANDA: Per ciascuna di queste espressioni, per quali valori di `x` e `y` danno `True`? Prova a pensare una risposta prima di provare !!!

NOTA: ci possono essere più combinazioni che producono `True`, trovale tutte.

1. `x or (not x)`
2. `(not x) and (not y)`
3. `x and (y or y)`
4. `x and (not y)`
5. `(not x) or  y`
6. `y or not (y and x)`
7. `x and ((not x) or not(y))`
8. `(not (not x)) and not (x and y)`

9. `x and (x or (not(x) or not(not(x or not (x)))))`

DOMANDA: Per ciascuna di queste espressioni, per quali valori di `x` e `y` danno `False` ?

NOTA: ci possono essere più combiinazioni che producono `False`, trovale tutte

1. `x or ((not y) or z)`
2. `x or (not y) or (not z)`
3. `not (x and y and (not z))`
4. `not (x and (not y) and (x or z))`
5. `y or ((x or y) and (not z))`

4.4.5 De Morgan

Ci sono un paio di leggi che a volte tornano utili:

Formula	Equivalente a
`x or y`	`not(not x and not y)`
`x and y`	`not(not x or not y)`

DOMANDA: Guarda la seguenti espressioni, e prova a riscriverla in una equivalente usando le leggi di De Morgan, eventualmente semplificando il risultato ove possibile. Verifica poi se la traduzione produce un risultato uguale all'originale per tutti i possibili valori di `x` e `y`

1. `(not x) or y`
2. `(not x) and (not y)`
3. `(not x) and (not (x or y))`

Esempio:

```
x,y = False, False
#x,y = False, True
#x,y = True, False
#x,y = True, True

orig = x or y
trasf = not((not x) and (not y))
print('orig=',orig)
print('trasf=',trasf)
```

[7]:
```
# scrivi qui
```

4.4.6 Conversione

Possiamo convertire booleani in interi con la funzione predefinita `int`. Ogni intero può essere convertito in un booleano (e vice versa) con `bool`:

```
[8]: bool(1)
[8]: True
```

```
[9]: bool(0)
[9]: False
```

```
[10]: bool(72)
[10]: True
```

```
[11]: bool(-5)
[11]: True
```

```
[12]: int(True)
[12]: 1
```

```
[13]: int(False)
[13]: 0
```

Ogni intero è valutato a `True`, eccetto `0`. Nota che, i valori di verità `True` e `False` si comportano rispettivamente come gli interi `1` e `0`

4.4.7 Domande - cos'è un booleano?

DOMANDA: Per ciascuna di queste espressioni, che risultato produce?

1. `bool(True)`
2. `bool(False)`
3. `bool(2 + 4)`
4. `bool(4-3-1)`
5. `int(4-3-1)`
6. `True + True`
7. `True + False`
8. `True - True`
9. `True * True`

4.4.8 Ordine di valutazione

Per questioni di efficienza, se durante la valutazione di un'espressione booleana Python scopre che il risultato possibile può essere solo uno, allora evita di calcolare le espressioni seguenti. Per esempio, in questa espressione:

```
False and x
```

leggendo da sinistra a destra, nel momento in cui incontriamo `False` sappiamo già che il risultato della `and` sarà sempre `False` indipendentemente dal valore della `x` (convinciti!).

Se invece leggendo da sinistra a destra Python trova prima `True`, continua la valutazione delle espressioni seguenti *e come risultato dell'intera* `and` *restituisce la valutazione dell'* ***ultima*** *espressione*. Se usiamo sempre booleani, non ci accorgeremo delle differenze, ma cambiando tipi potremmo ottenere delle sorprese:

```
[14]: True and 5
[14]: 5
```

```
[15]: 5 and True
[15]: True
```

```
[16]: False and 5
[16]: False
```

```
[17]: 5 and False
[17]: False
```

Pensiamo a quale può essere l'ordine di valutazione nella `or`. Guarda l'espressione:

```
True or x
```

leggendo da sinistra a destra, appena troviamo il `True` possiamo stabilire che il risultato di tutta la `or` dovrà essere `True` indipendentemente dal valore di `x` (convinciti!).

Se invece il primo valore è `False`, Python continuerà la valutazione finchè trova un valore logico `True`, quando questo accade quel valore sarà il risultato dell'intera espressione. Ce ne possiamo accorgere se usiamo costanti diverse da `True` e da `False`:

```
[18]: False or 5
[18]: 5
```

```
[19]: 7 or False
[19]: 7
```

```
[20]: 3 or True
[20]: 3
```

I numeri che vedi hanno sempre un risultato logico coerente con le operazioni fatte, cioè se vedi `0` si intende che il risultato dell'espressione abbia valore logico `False` e se vedi un numero diverso da `0` il risultato si intende `True` (convinciti)

DOMANDA: Guarda le espressioni seguenti, e per ciascuna cerca di indovinare quale risultato produce (o se da errore):

1. ```
   0 and True
   ```
2. ```
   1 and 0
   ```
3. ```
   True and -1
   ```
4. ```
   0 and False
   ```
5. ```
   0 or False
   ```
6. ```
   0 or 1
   ```
7. ```
   False or -6
   ```
8. ```
   0 or True
   ```

4.4.9 Errori nella valutazione

Cosa succede se un'espressione booleana contiene del codice che genererebbe un errore? Intuitivamente, il programma dovrebbe terminare, ma non è sempre così.

Proviamo a generare di proposito un errore. Come ti avranno sicuramente detto più e più volte alle lezioni di matematica, provare a dividere un numero per zero è un errore perchè il risultato è indeterminato. Quindi se proviamo a chiedere a Python il risultato di `1/0` otterremo (prevedibilmente) lamentele:

```
print(1/0)
print('dopo')

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-51-9e1622b385b6> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

Nota che `'dopo'` *non* viene stampato perchè il programma si interrompe prima.

E se proviamo a scrivere così, cosa otteniamo?

```
[21]: False and 1/0
[21]: False
```

Python produce un risultato senza lamentarsi ! Perchè? Valutando da sinistra a destra ha incontrato un `False` e ha quindi concluso in anticipo che il risultato dell'espressione debba essere `False`. Molte volte non ti accorgerai di questi potenziali problemi ma è bene conoscerli perchè vi sono situazioni in cui puoi persino sfruttare l'ordine di esecuzione per prevenire errori (per esempio negli `if` e `while` che vedremo in seguito).

DOMANDA: Guarda le espressioni seguenti, e per ciascuna cerca di indovinare quale risultato produce (o se da errore):

1. ```
   True and 1/0
   ```
2. ```
   1/0 and 1/0
   ```

3. `False or 1/0`
4. `True or 1/0`
5. `1/0 or True`
6. `1/0 or 1/0`
7. `True or (1/0 and True)`
8. `(not False) or not 1/0`
9. `True and 1/0 and True`
10. `(not True) or 1/0 or True`
11. `True and (not True) and 1/0`

4.4.10 Operatori di comparazione

Gli operatori di comparazione permettono di costruire *espressioni* che ritornano un valore booleano:

Comparatore	Descrizione
`a == b`	`True` se e solo se $a = b$
`a != b`	`True` se e solo se $a \neq b$
`a < b`	`True` se e solo se $a < b$
`a > b`	`True` se e solo se $a > b$
`a <= b`	`True` se e solo se $a \leq b$
`a >= b`	`True` se e solo se $a \geq b$

```
[22]: 3 == 3
[22]: True
```

```
[23]: 3 == 5
[23]: False
```

```
[24]: a,b = 3,5
```

```
[25]: a == a
[25]: True
```

```
[26]: a == b
[26]: False
```

```
[27]: a == b - 2
```

[27]:
```
True
```

[28]:
```
3 != 5  # 3 è diverso da 5 ?
```

[28]:
```
True
```

[29]:
```
3 != 3  # 3 è diverso da 3 ?
```

[29]:
```
False
```

[30]:
```
3 < 5
```

[30]:
```
True
```

[31]:
```
5 < 5
```

[31]:
```
False
```

[32]:
```
5 <= 5
```

[32]:
```
True
```

[33]:
```
8 > 5
```

[33]:
```
True
```

[34]:
```
8 > 8
```

[34]:
```
False
```

[35]:
```
8 >= 8
```

[35]:
```
True
```

Dato che le comparazioni sono espressioni che producono booleani, possiamo anche assegnare il risultato ad una variabile:

[36]:
```
x = 5 > 3
```

[37]:
```
print(x)
```

```
True
```

DOMANDA: Guarda le espressioni seguenti, e per ciascuna cerca di indovinare quale risultato produce (o se produce un errore).

1. ```
   x = 3 == 4
   print(x)
   ```

2. ```
   x = False or True
   print(x)
   ```

3. ```
   True or False = x or False
   print(x)
   ```

4. ```
   x,y = 9,10
   z = x < y and x == 3**2
   print(z)
   ```

5.
```
a,b = 7,6
a = b
x = a >= b + 1
print(x)
```

6.
```
x = 3^2
y = 9
print(x == y)
```

4.4.11 Esercizio - Il Tempio di Lava

Durante i tuoi studi scopri una mappa di un antico tempio, ove sono custoditi mirabolanti tesori.

Il tempio misura `d=80` metri di lato ed è un labirinto di corridoi. Sai per certo che alcune aree mostrate in rosso contengono pavimento fragile sotto il quale scorrono fiumi di lava incandescente: per avvertirti del pericolo mentre cammini, ti costruisci un detector che emetterà un suono quando sarai in zone rosse.

Scrivi un'espressione booleana che restituisce `True` se sei in una zona di pericolo, `False` altrimenti.

- **NON usare istruzioni** `if`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[38]:
```
d = 80

x,y = 0, 0      # False
#x,y = 20, 20   # False
#x,y = 60, 10   # True
#x,y = 10, 60   # True
#x,y = 20, 70   # False
#x,y = 70, 20   # False
```

(continues on next page)

(continued from previous page)

```
#x,y = 70, 70  # False
#x,y = 0,60    # True
#x,y = 60,0    # True

# scrivi qui

((x > d//2) or (y > d//2)) and  (x < -y + d)
```

[38]:
```
False
```

[38]:
```

d = 80

x,y = 0, 0      # False
#x,y = 20, 20  # False
#x,y = 60, 10  # True
#x,y = 10, 60  # True
#x,y = 20, 70  # False
#x,y = 70, 20  # False
#x,y = 70, 70  # False
#x,y = 0,60    # True
#x,y = 60,0    # True

# scrivi qui
```

[38]:
```
False
```

4.4.12 Esercizio - La Torre di Gradius I

La lancetta dell'orologio sulla prima Torre di Gradius ha compiuto finora una rotazione di `n` gradi. Scrivi del codice che mostra `True` se la lancetta si trova nelle zone evidenziate, `False` altrimenti.

- **NON usare istruzioni** `if`
- `n` può essere maggiore di `360`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[39]:
```
n = 20    # False
#n = 405  # False
#n = 70   # False
#n = 100  # True
#n = 460  # True
#n = 225  # True
#n = 182  # False
#n = 253  # False
#n = 275  # False
#n = 205  # False
#n = 350  # True
#n = 925  # False
#n = 815  # True
#n = 92   # True

# scrivi qui
n % 120 > 90
```

[39]:
```
False
```

[39]:
```
n = 20    # False
#n = 405  # False
#n = 70   # False
#n = 100  # True
#n = 460  # True
#n = 225  # True
#n = 182  # False
#n = 253  # False
#n = 275  # False
#n = 205  # False
#n = 350  # True
#n = 925  # False
#n = 815  # True
#n = 92   # True

# scrivi qui
```

[39]:
```
False
```

4.4.13 Esercizio - Il Salto del Tubo

Un idraulico italiano vede 3 tubi di altezza rispettivamente `t1`, `t2` e `t3`, sopra cui vi sono rispettivamente sono `10`, `20` e `30` monete. Esaltato, spicca un balzo di altezza `h`. Scrivi del codice che stampa quante monete raccoglie (`10`,`20` o `30`).

- **NON usare istruzioni** `if`
- **SUGGERIMENTO**: Se non sai come fare, riguardati il paragrafo *Ordine di valutazione* e prova a pensare come produrre dei numeri solo quando una certa condizione è vera …

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[40]:
```
t1,t2,t3 = 200,500,600

h=450    # 10
#h=570   # 20
#h=610   # 30
#h=50    # 0

# scrivi qui
(h > t1 and ((h <= t2 and 10) or (h <= t3 and 20) or 30)) or 0
```

[40]:
```
10
```

[40]:
```
t1,t2,t3 = 200,500,600

h=450    # 10
#h=570   # 20
#h=610   # 30
#h=50    # 0

# scrivi qui
```

[40]:
```
10
```

4.4.14 Esercizio - La Torre di Gradius II

Le lancette dell'orologio sulla seconda Torre di Gradius hanno compiuto finora una rotazione rispettivamente di `n` e `m` gradi. Scrivi del codice che mostra `True` se entrambe le lancette si trovano nella **stessa** zona tra quelle evidenziate, `False` altrimenti.

- **NON usare istruzioni** `if`
- `n` e `m` possono essere maggiori di `360`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[41]:
```
n,m = 160,170   # True
#n,m = 135, 140 # False
#n,m = 160,190  # False
#n,m = 70,170   # False
#n,m = 350,260  # False
#n,m = 350,340  # True
#n,m = 350,340  # True
#n,m = 430,530  # False
#n,m = 520,510  # True
#n,m = 730,740  # False

# scrivi qui
(n % 90 > 60) and ((n // 90) % 4 == (m // 90) % 4)
```

[41]:
```
True
```

[41]:
```
n,m = 160,170   # True
#n,m = 135, 140 # False
#n,m = 160,190  # False
#n,m = 70,170   # False
```

(continues on next page)

(continued from previous page)

```
#n,m = 350,260  # False
#n,m = 350,340  # True
#n,m = 350,340  # True
#n,m = 430,530  # False
#n,m = 520,510  # True
#n,m = 730,740  # False

# scrivi qui
```

[41]: True

4.4.15 Prosegui

Continua con Basi 3 - numeri reali[112]

4.5 Basi 3 - numeri reali

4.5.1 Scarica zip esercizi

Naviga file online[113]

REQUISITI:

- **Letto** Basi 1 - variabili e interi[114]
- **Letto** Basi 2 - booleani[115]

4.5.2 Numeri reali

Python salva i numeri reali (numeri in virgola mobile) in 64 bit di informazione divisi in segno, esponente e mantissa. Vediamo un esempio:

[2]:
```
3.14
```

[2]: 3.14

[3]:
```
type(3.14)
```

[3]: float

[112] https://it.softpython.org/basics/basics3-floats-sol.html
[113] https://github.com/DavidLeoni/softpython-it/tree/master/basics
[114] https://it.softpython.org/basics/basics1-ints-sol.html
[115] https://it.softpython.org/basics/basics2-bools-sol.html

ATTENZIONE: bisogna usare il punto invece della virgola!

Quindi scriverai `3.14` invece di `3,14`

Fai molta attenzione quando copi numeri da documenti in italiano, potrebbero contenere insidiose virgole !

Notazione scientifica

Quando i numeri sono molto grandi o molto piccoli, per evitare di scrivere troppi zero conviene usare la notazione scientitifica con la *e* come xen, che moltiplica il numero *x* per 10^n

Con questa notazione, in memoria vengono messe solo le cifre più significative (*la mantissa*) e l'esponente, evitando quindi di sprecare spazio.

```
[4]: 75e1
[4]: 750.0
```

```
[5]: 75e2
[5]: 7500.0
```

```
[6]: 75e3
[6]: 75000.0
```

```
[7]: 75e123
[7]: 7.5e+124
```

```
[8]: 75e0
[8]: 75.0
```

```
[9]: 75e-1
[9]: 7.5
```

```
[10]: 75e-2
[10]: 0.75
```

```
[11]: 75e-123
[11]: 7.5e-122
```

DOMANDA: Guarda le seguenti espressioni, e cerca di indovinare qule risultato producono (o se danno errore):

1. `print(1.000.000)`
2. `print(3,000,000.000)`
3. `print(2000000.000)`
4. `print(2000000.0)`

5. `print(0.000.123)`
6. `print(0.123)`
7. `print(0.-123)`
8. `print(3e0)`
9. `print(3.0e0)`
10. `print(7e-1)`
11. `print(3.0e2)`
12. `print(3.0e-2)`
13. `print(3.0-e2)`
14. `print(4e2-4e1)`

Numeri troppo grandi o troppo piccoli

A volte i calcoli su numeri estremamente piccoli o enormi possono produrre come risultato `math.nan` (Not a Number) o `math.inf`. Per il momento li menzioniamo e basta, trovi una discussione dettagliata nel foglio su Numpy[116]

Esercizio - cerchio

Calcola l'area del cerchio al centro di un pallone da calcio (raggio = 9.15m), ricordandoti che $area = pi * r^2$ (come operatore per la potenza, usa `**`):

Il tuo codice dovrebbe stampare come risultato `263.02199094102605`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
# SOLUZIONE

r = 9.15
pi = 3.1415926536
area = pi*(r**2)
print(area)
```

```
263.02199094102605
```

[12]:

```
263.02199094102605
```

[116] https://it.softpython.org/matrices-numpy/matrices-numpy-sol.html#NaN-e-infinit%C3%A0

Nota che le parentesi intorno al `r**2` non sono necessarie perchè l'operatore `**` ha la precedenza, ma possono aiutare la leggibilità del codice.

Ricordiamo qua la precedenza degli operatori:

Operatore	Descrizione
`**`	Potenza (massima precedenza)
`+ -`	Più e meno unari
`* / // %`	Moltiplicazione, divisione, divisione intera, modulo
`+ -`	Addizione e sottrazione
`<= < > >=`	Operatori di comparazione
`== !=`	Operatori di uguaglianza
`not or and`	Operatori logici (minima precedenza)

Esercizio - frazionamento

Scrivi del codice per calcolare il valore della seguente formula per `x = 0.000003`, dovresti ottenere `2.753278226511882`

$$-\frac{\sqrt{x+3}}{\frac{(x+2)^3}{\log x}}$$

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[13]:
```
x = 0.000003

# scrivi qui
import math
- math.sqrt(x+3) / (((x+2)**3)/math.log(x))
```

[13]:
```
2.753278226511882
```

[13]:
```
x = 0.000003

# scrivi qui
```

[13]:
```
2.753278226511882
```

Esercizio - sommatoria

Scrivi del codice per calcolare il valore della seguente espressione (non usare cicli, scrivi per esteso tutto i calcoli), dovresti ottenere `20.53333333333333`

$$\sum_{j=1}^{3} \frac{j^4}{j+2}$$

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[14]:
```
# scrivi qui
((1**4) / (1+2)) + ((2**4) / (2+2)) + ((3**4) / (3+2))
```

[14]:
```
20.53333333333333
```

[14]:
```
# scrivi qui
```

[14]:
```
20.53333333333333
```

Reali - conversione

Se vogliamo convertire un reale ad un intero, abbiamo a disposizione diversi modi:

Funzione	Descrizione	Simbolo matematico	Risultato
`math.floor(x)`	arrotonda x all'intero inferiore	$\lfloor 8.7 \rfloor$	8
`int(x)`	arrotonda x all'intero inferiore	$\lfloor 8.7 \rfloor$	8
`math.ceil(x)`	arrotonda x all'intero superiore	$\lceil 5.3 \rceil$	6
`round(x)`	arrotonda x all'intero più vicino	$\lfloor 2.49 \rceil$	2
		$\lfloor 2.51 \rceil$	3

DOMANDA: Guarda le espressioni seguenti, e per ciascuna cerca di indovinare quale risultato produce (o se da errore).

1. `math.floor(2.3)`
2. `math.floor(-2.3)`
3. `round(3.49)`

4. `round(3.51)`
5. `round(-3.49)`
6. `round(-3.51)`
7. `math.ceil(8.1)`
8. `math.ceil(-8.1)`

DOMANDA: Dato un qualsiasi float `x`, la seguente formula è :

```
math.floor(math.ceil(x)) == math.ceil(math.floor(x))
```

1. sempre `True`
2. sempre `False`
3. a volte `True` e a volte `False` (fornire esempi)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 3: per interi come `x=2.0` è `True`, negli altri casi come `x=2.3` è `False`

DOMANDA: Dato un qualsiasi float `x`, la seguente formula è :

```
math.floor(x) == -math.ceil(-x)
```

1. sempre `True`
2. sempre `False`
3. a volte `True` e a volte `False` (fornire esempi)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 1.

Esercizio - tonificante

Gli studi eccessivi ti hanno portato a cercare su internet ricette di bevande energetiche. Fortunatamente, una guru della nutrizione ha appena postato sul suo canale Instagram @BeviCheTiFaBene la ricetta di un frappè portentoso:

> Versare in un contenitore 2 decilitri di spremuta di kiwi, 4 decilitri di salsa di soia, e 3 decilitri di shampoo al karitè bio. Miscelare vigorosamente e poi versare metà del composto in un bicchiere. Rabboccare il bicchiere fino al decilitro intero superiore. Tracannare avidamente.

Corri a comprare gli ingredienti, e ti appresti a miscelarli. Hai a disposizione un misurino in cui travasi i preziosi fluidi, ad uno ad uno. Nell'effettuare il travaso, versi sempre un po' più del necessario (ma mai più di 1 decilitro), e ad ogni ingrediente elimini poi l'eccesso.

- **NON** usare sottrazioni, prova ad usare solo gli operatori per gli arrotondamenti

Esempio:

Dati

```
kiwi = 2.4
soia = 4.8
shampoo = 3.1
misurino = 0.0
mixer = 0
bicchiere = 0.0
```

Il tuo codice deve stampare:

```
Verso nel misurino 2.4 dl di kiwi, tolgo eccesso fino a tenere 2 dl
Travaso nel mixer, adesso contiene 2 dl
Verso nel misurino 4.8 dl di soia, tolgo eccesso fino a tenere 4 dl
Travaso nel mixer, adesso contiene 6 dl
Verso nel misurino 3.1 dl di shampoo, tolgo eccesso fino a tenere 3 dl
Travaso nel mixer, adesso contiene 9 dl
Verso metà della miscela ( 4.5 dl ) nel bicchiere
Rabbocco il bicchiere fino al decilitro intero superiore, adesso contiene 5 dl
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[15]:
```
kiwi = 2.4
soia = 4.8
shampoo = 3.1
misurino = 0.0
mixer = 0.0
bicchiere = 0.0


# scrivi qui
print('Verso nel misurino', kiwi, 'dl di kiwi, tolgo eccesso fino a tenere',␣
↪int(kiwi), 'dl')
mixer += int(kiwi)
print('Travaso nel mixer, adesso contiene', mixer, 'dl')
print('Verso nel misurino', soia, 'dl di soia, tolgo eccesso fino a tenere',␣
↪int(soia), 'dl')
mixer += int(soia)
print('Travaso nel mixer, adesso contiene', mixer, 'dl')
print('Verso nel misurino', shampoo, 'dl di shampoo, tolgo eccesso fino a tenere',␣
↪int(shampoo), 'dl')
mixer += int(shampoo)
print('Travaso nel mixer, adesso contiene', mixer, 'dl')
bicchiere = mixer/2
print('Verso metà della miscela (', bicchiere, 'dl ) nel bicchiere')
print('Rabbocco il bicchiere fino al decilitro intero superiore, adesso contiene:',␣
↪math.ceil(bicchiere), 'dl')
```

```
Verso nel misurino 2.4 dl di kiwi, tolgo eccesso fino a tenere 2 dl
Travaso nel mixer, adesso contiene 2.0 dl
Verso nel misurino 4.8 dl di soia, tolgo eccesso fino a tenere 4 dl
Travaso nel mixer, adesso contiene 6.0 dl
```

(continues on next page)

(continued from previous page)

```
Verso nel misurino 3.1 dl di shampoo, tolgo eccesso fino a tenere 3 dl
Travaso nel mixer, adesso contiene 9.0 dl
Verso metà della miscela ( 4.5 dl ) nel bicchiere
Rabbocco il bicchiere fino al decilitro intero superiore, adesso contiene: 5 dl
```

[15]:
```
kiwi = 2.4
soia = 4.8
shampoo = 3.1
misurino = 0.0
mixer = 0.0
bicchiere = 0.0


# scrivi qui
```

```
Verso nel misurino 2.4 dl di kiwi, tolgo eccesso fino a tenere 2 dl
Travaso nel mixer, adesso contiene 2.0 dl
Verso nel misurino 4.8 dl di soia, tolgo eccesso fino a tenere 4 dl
Travaso nel mixer, adesso contiene 6.0 dl
Verso nel misurino 3.1 dl di shampoo, tolgo eccesso fino a tenere 3 dl
Travaso nel mixer, adesso contiene 9.0 dl
Verso metà della miscela ( 4.5 dl ) nel bicchiere
Rabbocco il bicchiere fino al decilitro intero superiore, adesso contiene: 5 dl
```

Esercizio - arrotondamente

Scrivi del codice per calcolare il valore della seguente formula per `x = -5.51`, dovresti ottenere `41`

$$|\lceil x \rceil| + \lfloor x \rfloor^2$$

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[16]:
```
import math

x = -5.51   # 41
#x = -5.49  # 30

# scrivi qui
abs(math.ceil(x)) + round(x)**2
```

[16]:
```
41
```

[16]:
```
import math

x = -5.51   # 41
#x = -5.49  # 30
```

(continues on next page)

(continued from previous page)

```
# scrivi qui
```

[16]:
```
41
```

Reali - uguaglianza

> **ATTENZIONE: Ciò che segue vale per *tutti* i linguaggi di programmazione!**
>
> Alcuni risultati ti parranno curiosi ma è il modo in cui la maggior parte dei processori (CPU) opera, indipendentemente da Python.

Quando si effettuano calcoli con i numeri in virgola mobile, il processore può commettere degli errori di arrotondamento dovuto ai limiti della rappresentazione interna. Sotto il cofano i numeri float vengono memorizzati in una sequenza in codice binario di 64 bit, secondo lo standard *IEEE-754 floating point arithmetic* : questo impone un limite fisico alla precisione dei numeri, e a volte possono anche capitare sorpese dovute alla conversione da decimale a binario. Per esempio, proviamo a stampare `4.1`:

[17]:
```
print(4.1)
```

```
4.1
```

Per nostra convenienza Python ci mostra `4.1`, ma in realtà nella memoria del processore è finito un numero diverso ! Quale? Per scoprire cosa nasconde, con `format` possiamo esplicitamente formattare il numero con lo specificatore di formato `f` usando per esempio `55` cifre di precisione:

[18]:
```
format(4.1, '.55f')
```

[18]:
```
'4.0999999999999996447286321199499070644378662109375000000'
```

Possiamo quindi chiederci quale potrà essere il risultato di un calcolo:

[19]:
```
print(7.9 - 3.8)
```

```
4.1000000000000005
```

Notiamo che il risultato è ancora diverso da quello atteso ! Indagando meglio, notiamo che Python non ci sta nemmeno mostrando tutte le cifre:

[20]:
```
format(7.9 - 3.8, '.55f')
```

[20]:
```
'4.1000000000000005329070518200751394033432006835937500000'
```

E se volessimo sapere se due calcoli coi float producono lo 'stesso' risultato?

> **ATTENZIONE: EVITA IL == CON I FLOAT!**

Per capire se il risultato tra due calcoli con i float è uguale, **NON** puoi usare l'operatore ==:

[21]:
```
7.9 - 3.8 == 4.1     # GUAI IN VISTA !
```

[21]:
```
False
```

Invece, devi preferire alternative che valutano se un numero float è *vicino* ad un altro, come per esempio la comoda funzione math.isclose[117]:

[22]:
```
import math

math.isclose(7.9 - 3.8, 4.1)    # MOLTO MEGLIO
```

[22]:
```
True
```

Di default `math.isclose` usa una precisione di `1e-09` ma volendo, a `math.isclose` puoi anche passare un limite di tolleranza entro la quale deve essere la differenza tra i due numeri affinchè siano considerati uguali:

[23]:
```
math.isclose(7.9 - 3.8, 4.1, abs_tol=0.000001)
```

[23]:
```
True
```

DOMANDA: Possiamo rappresentare perfettamente il numero $\sqrt{2}$ in un `float`?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: $\sqrt{2}$ è irrazionale e pertanto non possiamo sperare di rappresentarlo perfettamente, qualunque calcolo useremo per ottenerlo avrà sempre un certo grado di imprecisione.

DOMANDA: Quali di queste tre espressioni di codice danno lo stesso risultato?

```
import math
print('a)', math.sqrt(3)**2 == 3.0)
print('b)', abs(math.sqrt(3)**2 - 3.0) < 0.0000001)
print('c)', math.isclose(math.sqrt(3)**2, 3.0, abs_tol=0.0000001))
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la b) e la c) danno `True`. La a) da `False`, perchè durante i calcoli in virgola mobile vengono compiuti errori di arrotondamento.

Esercizio - quadratica

Scrivi del codice che calcola gli zero dell'equazione $ax^2 - b = 0$

- Mostra i numeri con **20 cifre** di precisione
- Alla fine controlla che sostituendo il valore ottenuto di `x` nell'equazione si ottenga effettivamente zero.

Esempio - dati:

```
a = 11.0
b = 3.3
```

dopo il tuo codice deve stampare:

[117] https://docs.python.org/3/library/math.html#math.isclose

```
11.0 * x**2 - 3.3 = 0  per x1 = 0.54772255750516607442
11.0 * x**2 - 3.3 = 0  per x2 = -0.54772255750516607442
0.54772255750516607442 è una soluzione? True
-0.54772255750516607442 è una soluzione? True
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[24]:
```
a = 11.0
b = 3.3

# scrivi qui

import math

x1 = math.sqrt(b/a)
x2 = -math.sqrt(b/a)

print(a, "* x**2 -",b,"= 0  per x1 =", format(x1, '.20f'))
print(a, "* x**2 -",b,"= 0  per x2 =", format(x2, '.20f'))

# abbiamo bisogno di cambiare il valore di default di tolleranza
print(format(x1, '.20f'), "è una soluzione?", math.isclose(a*(x1**2) - b,0, abs_tol=0.
↪00001))
print(format(x2, '.20f'), "è una soluzione?", math.isclose(a*((x2)**2) - b,0, abs_
↪tol=0.00001))
```

```
11.0 * x**2 - 3.3 = 0  per x1 = 0.54772255750516607442
11.0 * x**2 - 3.3 = 0  per x2 = -0.54772255750516607442
0.54772255750516607442 è una soluzione? True
-0.54772255750516607442 è una soluzione? True
```

[24]:
```
a = 11.0
b = 3.3

# scrivi qui
```

```
11.0 * x**2 - 3.3 = 0  per x1 = 0.54772255750516607442
11.0 * x**2 - 3.3 = 0  per x2 = -0.54772255750516607442
0.54772255750516607442 è una soluzione? True
-0.54772255750516607442 è una soluzione? True
```

Esercizio - alla moda

Stai già pensando alle prossime vacanze, ma c'è un problema: dove vai, se un *selfie-stick* non ce l'hai? Non puoi partire con questo grave turbamento: per uniformarti al fenomeno di massa devi comprare lo stick più simile agli altri. Conduci quindi prima un'indagine statistica tra turisti malati di selfie-stick con l'obiettivo di scoprire i brand più frequenti, in altre parole, la *moda* delle frequenze. Ottieni questi risultati:

```
[25]: b1,b2,b3,b4,b5 = 'TroppiLike', 'ErMejo United', 'Perditempo SpA', 'Vanità 3.0',
      ↪'TronistiPerCaso' # brand
      f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.3   # frequenze (in percentuale)
```

Desumiamo quindi che le masse prediligono i selfie-stick dei brand `'ErMejo United'` e `'TronistiPerCaso'`, entrambi a parimerito con un 30% di turisti ciascuno. Scrivi quindi del codice che stampa questo risultato:

```
TroppiLike è il più frequente? False ( 25.0 % )
ErMejo United è il più frequente? True ( 30.0 % )
Perditempo Inc è il più frequente? False ( 10.0 % )
Vanità 3.0 è il più frequente? False ( 5.0 % )
TronistiPerCaso è il più frequente? True ( 30.0 % )
```

- **ATTENZIONE**: il tuo codice deve funzionare con **QUALUNQUE** serie di variabili !!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

```
[26]: b1,b2,b3,b4,b5 = 'TroppiLike', 'ErMejo United', 'Perditempo Inc', 'Vanità 3.0',
      ↪'TronistiPerCaso'    # brand
      f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.3         # frequenze (in percentuale) ␣
      ↪False True False False True
      # OCCHIO, sembrano uguali ma deve funzionare anche con queste ;-)
      #f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.1 + 0.2  #  False True False False True

      # scrivi qui
      mx = max(f1,f2,f3,f4,f5)
      print(b1, 'è il più frequente?', math.isclose(f1,mx), '(', f1*100.0, '% )')
      print(b2, 'è il più frequente?', math.isclose(f2,mx), '(', f2*100.0, '% )')
      print(b3, 'è il più frequente?', math.isclose(f3,mx), '(', f3*100.0, '% )')
      print(b4, 'è il più frequente?', math.isclose(f4,mx), '(', f4*100.0, '% )')
      print(b5, 'è il più frequente?', math.isclose(f5,mx), '(', f5*100.0, '% )')
```

```
TroppiLike è il più frequente? False ( 25.0 % )
ErMejo United è il più frequente? True ( 30.0 % )
Perditempo Inc è il più frequente? False ( 10.0 % )
Vanità 3.0 è il più frequente? False ( 5.0 % )
TronistiPerCaso è il più frequente? True ( 30.0 % )
```

```
[26]: b1,b2,b3,b4,b5 = 'TroppiLike', 'ErMejo United', 'Perditempo Inc', 'Vanità 3.0',
      ↪'TronistiPerCaso'    # brand
      f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.3         # frequenze (in percentuale) ␣
      ↪False True False False True
      # OCCHIO, sembrano uguali ma deve funzionare anche con queste ;-)
      #f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.1 + 0.2  #  False True False False True

      # scrivi qui
```

(continues on next page)

(continued from previous page)

```
TroppiLike è il più frequente? False ( 25.0 % )
ErMejo United è il più frequente? True ( 30.0 % )
Perditempo Inc è il più frequente? False ( 10.0 % )
Vanità 3.0 è il più frequente? False ( 5.0 % )
TronistiPerCaso è il più frequente? True ( 30.0 % )
```

4.5.3 Numeri decimali

Per moltissime applicazioni i `float` sono sufficienti, avendo coscienza dei loro limiti di rappresentazione e uguaglianza. Se proprio hai bisogno di maggiore precisione e/o prevedibilità, Python offre un tipo numerico apposito chiamato `Decimal`, che permette una precisione arbitraria. Per usarlo, devi prima importarlo dalla libreria `decimal`:

```
[27]: from decimal import Decimal
```

Possiamo creare un `Decimal` a partire da una stringa:

```
[28]: Decimal('4.1')
[28]: Decimal('4.1')
```

ATTENZIONE: se crei un Decimal a partire da una costante, indicala come stringa!

Se gli passi un `float` rischi di perdere l'utilità dei Decimal:

```
[29]: Decimal(4.1)  # così mi ritrovo coi problemi dei float ...
[29]: Decimal('4.0999999999999996447286321199499070644378662109375')
```

Operazioni tra `Decimal` producono altri `Decimal`:

```
[30]: Decimal('7.9') - Decimal('3.8')
[30]: Decimal('4.1')
```

Questa volta, possiamo tranquillamente usare l'operatore di uguaglianza e ottenere il risultato atteso:

```
[31]: Decimal('4.1') == Decimal('7.9') - Decimal('3.8')
[31]: True
```

Sono anche supportate alcune funzioni matematiche, e spesso si comportano in modo più predicibile (nota che **non** stiamo usando `math.sqrt`):

```
[32]: Decimal('2').sqrt()
[32]: Decimal('1.4142135623730950488016887242')
```

Ricordati comunque che la memoria del computer è finita!

I Decimal non possono essere la panacea per tutti i mali: per es, $\sqrt{2}$ non ci starà mai completamente nella memoria di nessun computer ! Possiamo verificare le limitazioni elevando al quadrato il risultato:

[33]:
```
Decimal('2').sqrt()**Decimal('2')
```

[33]:
```
Decimal('1.999999999999999999999999999')
```

L'unica cosa che possiamo avere in più coi Decimal è un maggiore numero di cifre per rappresentarli, che volendo possiamo aumentare a piacere[118] finchè non esauriamo la memoria del nostro pc. In questo libro non parleremo oltre dei Decimal perchè tipicamente servono solo per applicazioni specifiche, per esempio, se devi fare calcoli finanziari vorrai probabilmente usare cifre esatte !

[ ]:
```

```

4.6 Basi 4 - Challenges

4.6.1 Scarica zip esercizi

Naviga file online[119]

Proponiamo ora degli esercizi senza soluzione, accetti la sfida? Prova ad eseguirli sia in Jupyter che in un editor di testo come Spyder per familiarizzarti con entrambi gli ambienti.

4.6.2 Sfida - quali booleani 1?

⊛ Trova la riga che assegnando valori a `x` e `y` faccia in modo che la stampa stampi `True`. Esiste una sola combinazione o più di una?

[1]:
```
x = False; y = False
#x = False; y = True
#x = True; y = False
#x = True; y = True

print(x and y)
```

```
False
```

4.6.3 Sfida - quali booleani 2?

⊛ Trova la riga che assegnando valori a `x` e `y` faccia in modo che la stampa stampi `True`. Esiste una sola combinazione o più di una?

[2]:
```
x = False; y = False; z = False
#x = False; y = True; z = False
#x = True; y = False; z = False
#x = True; y = True; z = False
#x = False; y = False; z = True
#x = False; y = True; z = True
#x = True; y = False; z = True
#x = True; y =True; z =True

print((x or y) and (not x and z))
```

[118] https://docs.python.org/3/library/decimal.html
[119] https://github.com/DavidLeoni/softpython-it/tree/master/basics

```
False
```

4.6.4 Sfida - Aereoporto

⊛⊛ Finalmente decidi di prenderti una vacanza e vai all'aereoporto, ma già sai che dovrai fare varie code. Fortunatamente hai solo il bagaglio a mano, quindi ti rechi subito ai controlli di sicurezza, dove puoi scegliere tra tre file di `sic1`, `sic2` e `sic3` persone. Ogni persona in media ci mette 4 minuti ad essere esaminata, te incluso, e ovviamente scegli la fila più corta. Dopodichè vai al gate, dove trovi due file di `ga1` e `ga2` persone, e sai che ogni persona te incluso in media ci mette 20 secondi a passare: di nuovo scegli la fila più corta. Fortunatamente l'aereo è prossimo al gate quindi puoi subito scegliere se salire dalla fila in testa all'aereo con `bo1` persone o dalla fila in coda all'aereo con `bo2` persone. Per salire ogni passeggero te incluso in media ci mette 30 secondi, e scegli la coda più corta.

Scrivi del codice per calcolare quanto tempo ci impieghi in totale ad entrare nell'aereo, mostrandolo in minuti e secondi

Esempio - dati:

```
sic1,sic2,sic3,ga1,ga2,bo1,bo2 = 4,5,8,5,2,7,6
```

il tuo codice deve stampare:

```
24 minuti e 30 secondi
```

[3]:
```
sic1,sic2,sic3,ga1,ga2,bo1,bo2 = 4,5,8,5,2,7,6   # 24 minuti e 30 secondi
#sic1,sic2,sic3,ga1,ga2,bo1,bo2 = 9,7,1,3,5,2,9  # 10 minuti e 50 secondi

# scrivi qui
```

```
24 minuti e 30 secondi
```

4.6.5 Sfida - Space mining

⊛⊛ Un robot è atterrato sul pianeta Zorbon 5, con il compito di trivellare il terreno per estrarre il prezioso metallo Tiberium. La zona di ricerca ha lato `d` ed è divisa in settori, con l'origine (0,0) al centro. I colori nella mappa rappresentano la densità di Tiberium:

- rosso: 3
- arancione: 2
- giallo: 1

Il robot conosce le proprie coordinate `x` e `y`: scrivi del codice che stampa la densità di Tiberium corrispondente alla posizione del robot.

- **NON usare l'istruzione** `if`
- **SUGGERIMENTO**: usa la funzione `abs(x)` per calcolare il valore assoluto

[4]:
```
d = 30

x,y = 0, 0        # 3
#x,y = 2, 2       # 3
#x,y = -2, -2     # 3
#x,y = 7, 0       # 2
#x,y = -7, 0      # 2
#x,y = 0, 8       # 2
#x,y = -12, 0     # 2
#x,y = 2, 13      # 2
#x,y = 2, 13      # 2
#x,y = 13, 13     # 1
#x,y = 14, -12    # 1
#x,y = -11, -13   # 1
#x,y = -10, 11    # 1

# scrivi qui
```

[4]: 3

4.6.6 Sfida - La fabbrica infinita

⊛⊛ Nel regno di Steamland esiste una fabbrica che si dice si estenda all'infinito.

La fabbrica è popolata da robot che in prossimità degli incroci dei corridoi grigio chiari e gialli hanno il compito di azionare i complessi meccanismi a vapore che governano la fabbrica. Ogni robot conosce la sua posizione `x`, `y`. L'origine è in basso a sinistra e ogni corridoio è largo `m`.

Scrivi del codice che stampa `True` quando i robot sono negli incroci, e `False` altrimenti.

- **NON usare l'istruzione** `if`
- Il tuo codice deve funzionare anche per valori di `x` e `y` enormi.

[5]:
```
m = 10

x,y = 2.5 * m, 0.5 * m        # True
#x,y = 2.5 * m, 1.5 * m       # False
#x,y = 0.5 * m, 0.5 * m       # False
#x,y = 5.5 * m, 0.5 * m       # True
#x,y = 5.5 * m, 2.5 * m       # True
#x,y =  7.0 * m, 4.5 * m      # False
#x,y = 5.5 * m, 2.5 * m       # True
#x,y = 299.5 * m, 200.5 * m   # True
#x,y = 300.5 * m, 200.5 * m   # False

# scrivi qui
```

[5]: True

4.6.7 Sfida - L'Anello di Luce

⊛⊛⊛ Il malvagio stregone Tenebron ha scagliato una maledizione contro il tuo regno, e Re Tribolonius corre trafelato nel tuo studio alchemico implorandoti di contrastare queste forze oscure. Per fortuna, leggendo i tuoi grimori scopri il rituale del *RendeRing* che è in grado di erigere un anello di forza intorno al regno. Il sortilegio prescrive di tracciare un anello con un'intensità che varia da `0.0` (buio) fino all'intensità massima `5.0` (bianco). L'intensità aumenta linearmente fino a che si raggiunge il raggio `r=5.0`, dopodichè descresce linearmente fino a raggiungere lo `0.0`

Data la posizione `x,y`, scrivi del codice che calcola l'intensità di luce.

- **NON usare l'istruzione** `if`
- **NOTA**: l'intensità non può mai essere negativa
- **SUGGERIMENTO**: usa `math.sqrt`

[6]:
```
import math

r = 5.0

x,y = 0,0     #  0.0
#x,y = 0,3     #  3.0
#x,y = -4,0    #  4.0
#x,y = 3,-3    #  4.242640687119285
#x,y = 3,4     #  5.0
#x,y = 4,-3    #  5.0
#x,y = -4,3    #  5.0
#x,y = 6,0     #  4.0
#x,y = 0,7     #  3.0
#x,y = 0,-8    #  2.0
#x,y = 10,0    #  0.0
#x,y = -6,7    #  0.7804555427071129
#x,y = -7,9    #  0.0  mai negativa ...

# scrivi qui
```

[6]:
```
0.0
```

[ ]:
```

```

4.7 Stringhe 1 - introduzione

4.7.1 Scarica zip esercizi

Naviga file online[120]

Le stringhe sono sequenze *immutabili* di caratteri, e costituiscono uno dei tipi di base di Python. In questo foglio vedremo come manipolarle.

4.7.2 Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
strings
    strings1.ipynb
    strings1-sol.ipynb
    strings2.ipynb
    strings2-sol.ipynb
    strings3.ipynb
    strings3-sol.ipynb
    strings4.ipynb
    strings4-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `strings1.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

[120] https://github.com/DavidLeoni/softpython-it/tree/master/strings

4.7.3 Creare stringhe

Ci sono diversi modi di definire una stringa.

Doppie virgolette, su una linea

[2]:
```
a = "la mia prima stringa, in doppie virgolette"
```

[3]:
```
print(a)
```

```
la mia prima stringa, in doppie virgolette
```

Virgolette singole, su una linea

Questo modo è equivalente al precedente.

[4]:
```
b = 'la mia seconda stringa, in virgolette singole'
```

[5]:
```
print(b)
```

```
la mia seconda stringa, in virgolette singole
```

Tre doppie virgolette, su più linee

[6]:
```
c = """la mia terza stringa
in triple doppie virgolette
quindi la posso mettere

su più righe"""
```

[7]:
```
print(c)
```

```
la mia terza stringa
in triple doppie virgolette
quindi la posso mettere

su più righe
```

Tre apici, su più linee

[8]:
```
d = '''la mia quarta stringa,
in tripli apici
può pure essere messa

su più linee
'''
```

[9]:
```
print(d)
```

```
la mia quarta stringa,
in tripli apici
può pure essere messa

su più linee
```

4.7.4 Stampare - le celle

Per stampare una stringa possiamo usare la funzione `print`:

[10]:
```
print('ciao')
```
```
ciao
```

Notare che nella stampa *non* sono riportati gli apici.

Se invece scriviamo direttamente la stringa senza la `print`, vedremo gli apici:

[11]:
```
'ciao'
```
[11]:
```
'ciao'
```

Cosa succede se scriviamo la stringa coi doppi apici?

[12]:
```
"ciao"
```
[12]:
```
'ciao'
```

Vediamo che di default Jupyter la mostra con gli apici singoli

Lo stesso discorso si applica se assegnamo una stringa a una variabile:

[13]:
```
x = 'ciao'
```

[14]:
```
print(x)
```
```
ciao
```

[15]:
```
x
```
[15]:
```
'ciao'
```

[16]:
```
y = "ciao"
```

[17]:
```
print(y)
```
```
ciao
```

[18]:
```
y
```
[18]:
```
'ciao'
```

4.7.5 La stringa vuota

La stringa di lunghezza zero si rappresenta con due doppie virgolette `""` o due apici singoli.

Nota che se anche scriviamo due doppie virgolette, Jupyter la mostra che inizia e finisce con un apice solo:

[19]:
```
""
```
[19]:
```
''
```

Lo stesso vale se associamo una stringa vuota ad una variabile:

[20]:
```
x = ""
```

[21]:
```
x
```

[21]:
```
''
```

Nota anche che se chiediamo a Jupyter di stampare con la `print`, non vedremo nulla:

[22]:
```
print("")
```

[23]:
```
print('')
```

4.7.6 Stampare più stringhe

Per stampare più stringhe su una linea ci sono diversi modi, cominciamo dal più semplice con la `print`

[24]:
```
x = "ciao"
y = "Python"

print(x,y)    # nota che nella stampa Python ha inserito uno spazio
```

```
ciao Python
```

Alla `print` possiamo aggiungere quanti parametri vogliamo, e possono anche essere mischiati con altri tipi come i numeri:

[25]:
```
x = "ciao"
y = "Python"
z = 3

print(x,y,z)
```

```
ciao Python 3
```

4.7.7 Lunghezza di una stringa

Per ottenere la lunghezza di una stringa (o in genere di qualunque sequenza), possiamo usare la funzione `len`:

[26]:
```
len("ciao")
```

[26]:
```
4
```

[27]:
```
len("")    # stringa vuota
```

[27]:
```
0
```

[28]:
```
len('')    # stringa vuota
```

[28]:
```
0
```

DOMANDA: Possiamo scrivere una cosa del genere?

```
"len"("ciao")
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: no, `"len"` tra virgolette verrà interpretato come una stringa, non come una funzione, e quindi Python si lamenterà dicendoci che non possiamo applicare una stringa ad un'altra stringa. Prova a vedere che errore viene fuori qua riscrivendo l'espressione qua sotto.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[29]:
```
# scrivi qui

#"len"("ciao")
```

[29]:
```
# scrivi qui
```

DOMANDA: possiamo scrivere una cosa del genere? Cosa produce? un errore? un numero ? Quale?

```
len("len('ciao')")
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: restituisce il numero `11`: mettendo il codice Python `len('ciao')` tra doppie virgolette, è diventato una stringa come tutte le altre. Quindi scrivendo `len("len('ciao')")` andiamo a contare quanto è lunga la stringa `"len('ciao')"`, cioè 11.

DOMANDA: Cosa otteniamo se scriviamo così:

```
len((((((("ciao"))))))
```

1. un errore
2. la lunghezza della stringa
3. qualcos'altro

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 2: `"ciao"` è una espressione, come tale possiamo racchiuderla in quante parentesi tonde vogliamo.

Contare sequenze di escape: Nota che alcune sequenze particolari dette di *escape* come per esempio `\t` occupano meno spazi di quello che sembra (con `len` contano come 1), ma stampate ne occupano addirittura di più di 2 !!

Vediamo un esempio, nel prossimo paragrafo scenderemo nei dettagli:

```
[30]: len('a\tb')
```

```
[30]: 3
```

```
[31]: print('a\tb')
```

```
a       b
```

4.7.8 Stampare - sequenze di escape

Alcune sequenze di caratteri dette *sequenze di escape* sono speciali perchè invece di mostrare caratteri, forzano la stampa a fare cose particolari come andare a capo o inserire spazi extra. Queste sequenze sono sempre precedute da il carattere di *backslash* `\`:

Descrizione	Sequenza di *escape*
Ritorno a capo (*linefeed*)	`\n`
Tabulazione (*ASCII tab*)	`\t`

Esempio - ritorno a capo

```
[32]: print("ciao\nmondo")
```

```
ciao
mondo
```

Notare che il ritorno a capo si verifica SOLO quando stampiamo con `print`, se invece immettiamo direttamente la stringa nella cella la vediamo tale e quale:

```
[33]: "ciao\nmondo"
```

```
[33]: 'ciao\nmondo'
```

In una stringa si possono mettere quante sequenze si vuole:

```
[34]: print("La nebbia a gl'irti colli\npiovigginando sale,\ne sotto il maestrale\nurla e␣
↪biancheggia il mar;")
```

```
La nebbia a gl'irti colli
piovigginando sale,
e sotto il maestrale
urla e biancheggia il mar;
```

Esempio - tabulazione

```
[35]: print("ciao\tmondo")
```

```
ciao    mondo
```

```
[36]: print("ciao\tmondo\tcon\ttante\ttab")
```

```
ciao    mondo   con     tante   tab
```

ESERCIZIO: Visto che le *sequenze di escape* sono speciali, ci si può chiedere quanto siano lunghe. Usa la funzione `len` per stampare la lunghezza delle stringhe. Noti qualcosa di strano?

- `'ab\ncd'`
- `'ab\tcd'`

[37]:
```
# scrivi qui
```

ESERCIZIO: Prova a selezionare la sequenza di caratteri stampata nella precedente cella. Cosa ottieni ? Una sequenza di spazi, oppure un carattere singolo di tabulazione? Nota che questo può variare a seconda del programma che ha effettivamente stampato la stringa.

ESERCIZIO: trova UNA SOLA stringa che stampata con `print` venga mostrata come la seguente.

- USA SOLO combinazioni di `\t` e `\n`
- NON usare spazi
- inizia e concludi la stringa con apice singolo

```
Questa  è
una

sfida   apparentemente      semplice
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[38]:
```
# scrivi qui

print('Questa\tè\nuna\n\nsfida\tapparentemente\t\tsemplice')
```

```
Questa  è
una

sfida   apparentemente          semplice
```

[38]:
```
# scrivi qui
```

```
Questa  è
una

sfida   apparentemente          semplice
```

ESERCIZIO: Prova a trovare una stringa che stampata con `print` venga mostrata così, SENZA usare nessuno spazio, SENZA triple apici e usando SOLO combinazioni di `\t` e `\n` !!

```
At  te
n   tis
simame

n
te
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[39]:
```
# scrivi qui
print("At\tte\nn\ttis\nsimame\n\nn\nte")
```

```
At      te
n       tis
simame

n
te
```

[39]:
```
# scrivi qui
```

```
At      te
n       tis
simame

n
te
```

Caratteri speciali: Per mettere caratteri speciali come il singolo apice `'` o le doppie virgolette `"` , dobbiamo creare una cosiddetta *sequenza di escape*, cioè prima scriviamo il carattere *backslash* \ e poi lo facciamo seguire dal carattere speciale che ci interessa:

Descrizione	Sequenza di *escape*	Risultato a stampa
Apice singolo	\'	'
Doppio apice	\"	"
Backslash	\\	\

Esempio:

Stampiamo una stringa contenente un apice singolo `'` e una doppia virgoletta `"`

[40]:
```
stringa = "Così metto \'apici\' e \"doppie virgolette\" nelle stringhe"
```

[41]:
```
print(stringa)
```

```
Così metto 'apici' e "doppie virgolette" nelle stringhe
```

Se una stringa inizia con doppi apici, all'interno possiamo usare liberamente singoli apici, anche senza *backslash* \:

[42]:
```
print("Non c'è problema")
```

```
Non c'è problema
```

Se inizia con singoli apici, possiamo liberamente usare doppie virgolette anche senza *backslash* \:

[43]:
```
print('Così è "se vi pare"')
```

```
Così è "se vi pare"
```

ESERCIZIO: Trova una stringa da stampare con la `print` che stampata mostri la sequenza seguente

- la stringa DEVE iniziare e finire con singoli apici `'`

```
Questo "genio" delle stringhe vuole  /\\/ fregarmi \//\ con esercizi atroci O_o'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[44]:
```
# scrivi qui

print('Questo "genio" delle stringhe vuole  /\\\\/ fregarmi \\//\\ con esercizi␣
↪atroci O_o\'')
```

```
Questo "genio" delle stringhe vuole  /\\/ fregarmi \//\ con esercizi atroci O_o'
```

[44]:
```
# scrivi qui
```

```
Questo "genio" delle stringhe vuole  /\\/ fregarmi \//\ con esercizi atroci O_o'
```

Caratteri Unicode: Quando abbiamo bisogno di caratteri particolari come ⊛ che non sono disponibili sulla tastiera, possiamo rivolgerci ai caratteri Unicode. Ce ne sono tantissimi[121], e spesso per metterli in Python 3 è sufficiente un copia e incolla. Per esempio, andando su questa pagina[122] possiamo copia incollare il carattere ⊛. In altri casi il carattere può essere così speciale da non essere nemmeno correttamente visualizzato, in tali situazioni si può usare una sequenza più complessa in formato `\uxxxx` come questa:

Descrizione	Sequenza di *escape*	Risultato a stampa
Esempio stella in cerchio in formato `\uxxxx`	`\u272A`	⊛

ESERCIZIO: Prova a cercare in google *Unicode heart* e prova a far stampare a Python un cuoricino , sia copia-incollando direttamente il carattere che usando la notazione `\uxxxx`

4.7.9 Encodings

Caratteri ASCII

Quando usi le stringhe nei tuoi programmi quotidiani, tipicamente non hai bisogno di pensare molto a come i caratteri sono fisicamente rappresentati come bit in memoria, ma a volte è importante. La rappresentazione è chiamata *encoding* e va tenuta in considerazione in particolare quando leggi dati da sorgenti esterne come file e siti web.

L'encoding più usato e famoso è lo ASCII[123] (American Standard Code for Information Interchange), che offre 127 caselle fatte da caratteri stampabili dall'alfabeto inglese (`a-z`, `A-Z`, punteggiatura come `.;,!` e caratteri come `(`, `@` ...) e sequenze di controllo (come `\t`, `\n`)

- Vedi caratteri stampabili[124] (Wikipedia)
- Codici di controllo ASCII[125] (Wikipedia)

[121] http://www.fileformat.info/info/unicode/char/a.htm
[122] https://www.fileformat.info/info/unicode/char/272a/index.htm
[123] https://en.wikipedia.org/wiki/ASCII
[124] https://it.wikipedia.org/wiki/ASCII#Stampabili
[125] https://en.wikipedia.org/wiki/C0_and_C1_control_codes#Basic_ASCII_control_codes

Visto che la tabella originale ASCII manca del supporto per linguaggi non-inglesi (per esempio, non ha le lettere accentate italiane come è,à, ...), parecchie estensioni sono state sviluppate per supportare altri linguaggi, per esempio vedere la pagina Extended ASCII[126] su Wikipedia.

Caratteri Unicode

Quando abbiamo bisogno di caratteri particolari come ⊛ che non sono disponibili sulla tastiera, possiamo guardare i caratteri Unicode. Ce ne sono parecchi[127], e possiamo spesso usarli in Python 3 semplicemente copia-incollandoli. Per esempio, se vai a questa pagina[128] puoi copia-incollare il carattere ⊛. In altri casi potrebbe essere così speciale che non può nemmeno essere correttamente visualizzato, perciò in questi casi puoi usare una sequenza più complessa nel formato `\uxxxx` come questo:

Descrizione	Sequenza di escape	Risultato stampato
Esempio stella in un cerchio nel formato `\uxxxx`	`\u272A`	⊛

ESERCIZIO: Cerca su Google *Unicode heart* e prova a stampare un cuore in Python, sia copia-incollando direttamente il carattere che usando la notazione `\uxxxx`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[45]:
```
# scrivi qui

print("I ♥ Python, con copia e incolla")
print("I \u2665 Python, anche in formato \\uxxxx")
```

```
I ♥ Python, con copia e incolla
I ♥ Python, anche in formato \uxxxx
```

[45]:
```
# scrivi qui
```

```
I ♥ Python, con copia e incolla
I ♥ Python, anche in formato \uxxxx
```

Riferimenti Unicode: abbiamo solo menzionato Unicode ma può essere un argomento complesso, se avrai mai a che fare con insiemi di caratteri giapponesi o *encodings* eterogenei trovi qua un paio di riferimenti che dovresti leggere:

- prima parte sull'encoding Unicode dal capitolo Stringhe nel libro Dive into Python 3[129]
- Documentazione unicode di Python 3[130]

[126] https://it.wikipedia.org/wiki/ASCII_esteso
[127] http://www.fileformat.info/info/unicode/char/a.htm
[128] https://www.fileformat.info/info/unicode/char/272a/index.htm
[129] http://gpiancastelli.altervista.org/dip3-it/stringhe.html
[130] https://docs.python.org/3/howto/unicode.html

4.7.10 Le stringhe sono immutabili

Le stringhe sono oggetti *immutabili*, quindi una volta create non si possono più cambiare. Questo può apparire restrittivo, ma non è poi così tragico, perchè abbiamo comunque a disposizione queste alternative:

- generare una nuova stringa a partire da stringhe precedenti
- se abbiamo una variabile a cui abbiamo assegnato una stringa, possiamo assegnare un'altra stringa a quella variabile

Proviamo a generare una nuova stringa a partire da precedenti, per esempio concatendone due con l'operatore +

[46]:
```
x = 'ciao'
```

[47]:
```
y = x + 'mondo'
```

[48]:
```
x
```

[48]:
```
'ciao'
```

[49]:
```
y
```

[49]:
```
'ciaomondo'
```

L'operazione + quando eseguita *tra stringhe* le attacca creando una NUOVA stringa. Questo significa che l'associazione per `x` non è affatto cambiata, l'unica vera modifica osservabile è che abbiamo associato la stringa `'ciaomondo` alla variabile `y`. Prova a sincerartene in Python Tutor cliccando ripetutamente sul bottone *Next*:

[50]:
```
# ATTENZIONE: per usare la funzione jupman.pytut() di seguito,
# è necessario eseguire prima questa cella con Shift+Invio

# basta eseguirla una volta sola, la trovi presente anche in tutti i fogli nella␣
↪prima cella

import jupman
```

[51]:
```
x = 'ciao'
y = x + 'mondo'

print(x)
print(y)

jupman.pytut()
```

```
ciao
ciaomondo
```

[51]:
```
<IPython.core.display.HTML object>
```

Riassegnare variabili

Altre variazioni allo stato della memoria possono essere ottenute riassegnando le variabili, per esempio:

[52]:
```
x = 'ciao'
```

[53]:
```
y = 'mondo'
```

[54]:
```
x = y         # assegnamo a x la stessa stringa contenuta in y
```

[55]:
```
x
```

[55]:
```
'mondo'
```

[56]:
```
y
```

[56]:
```
'mondo'
```

Se una stringa che è stata creata non risulta assegnata a nessuna variabile, Python si occuperà automaticamente di eliminarla dalla memoria. Nel caso qui sopra, la stringa `'ciao'` in sè non è mai cambiata, semplicemente a un certo punto si è ritrovata non più assegnata ad alcuna variabile, e perciò Python ha provveduto ad eliminarla dalla memoria. Guarda cosa succede in Python Tutor:

[57]:
```
x = 'ciao'
y = 'mondo'
x = y

jupman.pytut()
```

[57]:
```
<IPython.core.display.HTML object>
```

Riassegnare una variabile a sè stessa

Possiamo chiederci cosa succede quando scriviamo qualcosa del genere:

[58]:
```
x = 'ciao'

x = x
```

[59]:
```
print(x)
```

```
ciao
```

Non è successo molto, l'assegnazione di `x` è rimasta inalterata.

Ma cosa succede se a destra dell'= mettiamo una formula più complessa?

[60]:
```
x = 'ciao'

x = x + 'mondo'

print(x)
```

```
ciaomondo
```

Cerchiamo di capire bene che è successo.

Nella prima linea, Python ha generato la stringa `'ciao'` e l'ha assegnata alla variabile `x`. Ma fin qui, niente di straordinario.

Poi, nella seconda linea, Python ha fatto due cose:

1. ha calcolato il risultato dell'espressione `x + 'mondo'`, generando una NUOVA stringa `ciaomondo`
2. ha assegnato la stringa appena generata `ciaomondo` alla variabile `x`

E' fondamentale capire bene che in presenza di riassegnamenti avvengono i due passaggi appena citati, che quindi ribadiamo:

- PRIMA viene calcolato il risultato dell'espressione a destra dell'= (avendo quindi a disposizione il vecchio valore associato a `x`)
- POI il risultato viene associato alla variabile a sinistra dell'=.

Se proviamo a guardare in Python Tutor, questo doppio passaggio viene eseguito in un colpo solo:

[61]:
```
x = 'ciao'
x = x + 'mondo'

jupman.pytut()
```

[61]:
```
<IPython.core.display.HTML object>
```

ESERCIZIO: Scrivi del codice che cambia lo stato della memoria in modo che alla fine risulti la stampa seguente:

```
z =  Questo
w =  era
x =  un problema
y =  era
s =  Questo era un problema
```

- per scrivere il codice, USA SOLO i simboli `=`,`+`,`z`,`w`,`x`,`y`,`s` E NIENT'ALTRO
- puoi usare quante linee di codice ritieni opportune
- puoi usare ogni simbolo quante volte vuoi

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[62]:
```
# queste variabili ti sono date

z = "Questo"
w = 'è'
x = 'un problema'
y = 'era'
s = ' '

# scrivi qui il codice

w = y

s = z + s + y + s + x
```

[62]:
```
# queste variabili ti sono date

z = "Questo"
w = 'è'
x = 'un problema'
y = 'era'
s = ' '

# scrivi qui il codice
```

(continues on next page)

(continued from previous page)

```
[63]: print("z = ", z)
print("w = ", w)
print("x = ", x)
print("y = ", y)
print("s = ", s)
```

```
z =  Questo
w =  era
x =  un problema
y =  era
s =  Questo era un problema
```

4.7.11 Stringhe e numeri

Le stringhe in Python hanno il tipo `str`:

```
[64]: type("ciao mondo")
```

```
[64]: str
```

Nelle stringhe possiamo inserire dei caratteri che rappresentano cifre:

```
[65]: print("Il carattere 5 rappresenta la cifra cinque, il carattere 3 rappresenta la␣
↪cifra tre")
```

```
Il carattere 5 rappresenta la cifra cinque, il carattere 3 rappresenta la cifra tre
```

Ovviamente possiamo anche costruire una sequenza di cifre, per avere ciò che sembra un numero:

```
[66]: print("La sequenza di caratteri 7583 rappresenta il numero settemila cinquecento␣
↪ottanta tre")
```

```
La sequenza di caratteri 7583 rappresenta il numero settemila cinquecento ottanta tre
```

Detto ciò, possiamo domandarci come si comporta Python quando abbiamo una *stringa* che contiene *solo* una sequenza di caratteri che rappresenta un numero, come per esempio `'254'`

Possiamo usare `'254'` (che abbiamo scritto come se fosse una stringa) anche come se fosse un numero? Per esempio, possiamo sommarci 3 ?

```
'254' + 3

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-29-d39aa62a7e3d> in <module>
----> 1 "254" + 3

TypeError: can only concatenate str (not "int") to str
```

Come vedi, Python is offende immediatamente, perchè stiamo cercando di mischiare tipi differenti

QUINDI:

- Scrivendo `'254'` tra doppi apici creiamo una *stringa* di tipo `str`

- scrivendo `254` creiamo un *numero* di tipo `int`

```
[67]: type('254')
[67]: str
```

```
[68]: type(254)
[68]: int
```

ATTENZIONE ALLA `print` !!

Se provi a stampare una stringa che contiene solo cifre, Python la mostrerà senza apici, e questo potrebbe ingannarti riguardo la sua vera natura !!

```
[69]: print('254')
      254
```

```
[70]: print(254)
      254
```

Solo in Jupyter, per mostrare costanti, variabili o risultati di calcoli, come alternativa alla `print` puoi immettere direttamente una formula nella cella. In questo caso siamo semplicmente mostrando una costante, e quando è una stringa vedrai degli apici:

```
[71]: '254'
[71]: '254'
```

```
[72]: 254
[72]: 254
```

Lo stesso discorso vale anche per le variabili:

```
[73]: x = '254'
```

```
[74]: x
[74]: '254'
```

```
[75]: y = 254
```

```
[76]: y
[76]: 254
```

Quindi, *solo in Jupyter*, quando devi mostrare una costante, una variabile o un calcolo spesso conviene scriverlo direttamente nella cella senza usare la `print`.

4.7.12 Conversioni - da stringa a numero

Torniamo al problema di sommare `'254' + 3`. La prima è una stringa, il secondo un numero. Se fossero entrambi numeri la somma sicuramente funzionerebbe:

```
[77]: 254 + 3
[77]: 257
```

Quindi possiamo provare a convertire la stringa `'254'` in un vero intero. Per farlo, possiamo usare `int` come se fosse una funzione, e passarci come argomento la stringa da convertire:

```
[78]: int('254') + 3
[78]: 257
```

ATTENZIONE: le stringhe e i numeri sono immutabili !!

Questo significa che scrivendo `int('254')` viene generato un *nuovo* numero senza intaccare minimamente la stringa `'254'` da cui si è partiti. Vediamo meglio con un esempio:

```
[79]: x = '254'      # assegnamo alla variabile x la stringa '254'
```

```
[80]: y = int(x)     # assegnamo alla variabile y il numero ottenuto convertendo '254' in int
```

```
[81]: x              # la variabile x risulta sempre assegnata alla stringa '254'
[81]: '254'
```

```
[82]: y              # in y invece adesso c'è un numero (notare che qua non abbiamo apici)
[82]: 254
```

Può essere utile rivedere l'esempio in Python tutor:

```
[83]: x = "254"

      y = int(x)

      print(y + 3)


      jupman.pytut()
      257
[83]: <IPython.core.display.HTML object>
```

ESERCIZIO: Prova a convertire una stringa che rappresenta un numero malformato (per es un numero con dentro un carattere: `'43K12'`) in un `int`. Che succede?

```
[84]: # scrivi qui
```

4.7.13 Conversioni - da numero a stringa

Un qualsiasi oggetto può essere convertito a stringa usando `str` come se fosse una funzione e passandogli l'oggetto da convertire. Proviamo quindi a convertire un numero a stringa:

[85]:
```
str(5)
```

[85]:
```
'5'
```

notare gli apici nel risultato, che testimoniano che abbiamo effettivamente ottenuto una stringa.

Se per caso vogliamo ottenere una stringa che è la concatenzione di oggetti di tipo diverso dobbiamo stare attenti:

```
x = 5
s = 'I giorni di lavoro settimanali sono ' + x
print(s)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-154-5951bd3aa528> in <module>
      1 x = 5
----> 2 s = 'I giorni di lavoro settimanali sono ' + x
      3 print(s)

TypeError: can only concatenate str (not "int") to str
```

Un modo di ovviare al problema (anche se non il più comodo) è convertire a stringa ciascuno degli oggetti che usiamo nella concatenazione:

[86]:
```
x = 3
y = 1.6
s = 'questa settimana ho fatto jogging ' + str(x) + ' volte correndo a una media di '␣
↪+ str(y) + 'km/h'
print(s)
```

```
questa settimana ho fatto jogging 3 volte correndo a una media di 1.6km/h
```

DOMANDA: Visto quanto detto in precedenza, dopo l'esecuzione del codice nella cella precedente, alla variabile `x` risulterà associato un *numero* oppure una *stringa* ?

Se hai dubbi, usa Python Tutor.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: i numeri, come le stringhe, sono immutabili. Quindi chiamando la funzione `str(x)` è impossibile che il numero `5` associato a `x` venga in alcun modo modificato. `str(x)` semplicemente *genererà* una NUOVA stringa `'5'` che verrà usata poi nella concatenazione.

4.7.14 Formattare stringhe

Concatenare stringhe con un segno + come sopra è macchinoso e si presta ad errori. Ci sono diverse soluzioni migliori, per una recensione completa ci riferiamo al sito Real Python[131]. In particolare, guardare le più utili che sono le f-strings[132] che sono disponibili per Python $\geq$ 3.6

Formattare con %

Vediamo come formattare stringhe con l'operatore %. Questa soluzione non è la migliore, ma è molto usata e supportata in tutte le versioni di Python, perciò l'abbiamo adottata nel libro:

[87]:
```
x = 3
"Ho saltato %s volte" % x
```

[87]:
```
'Ho saltato 3 volte'
```

Nota che abbiamo messo un cosìdetto *place-holder* (segnaposto) %s dentro la stringa, per dire a Python di sostituirlo con una variabile. Per passare a Python la variabile, *dopo* la stringa abbiamo messo un simbolo % seguito da una variabile, in questo caso x.

Se vogliamo mettere più di una variabile, dobbiamo solo aggiungere più segnaposti %s e dopo il % esterno mettiamo le variabili richieste in parentesi tonde, separandole con virgole:

[88]:
```
x = 3
y = 5
"Ho saltato %s volte e fatto %s flessioni" % (x,y)
```

[88]:
```
'Ho saltato 3 volte e fatto 5 flessioni'
```

Possiamo mettere quante variabili vogliamo, anche non-numeriche:

[89]:
```
x = 3
y = 5
premio = 'Migliore Atleta della Valle'
"Ho saltato %s volte, fatto %s flessioni e vinto il premio '%s'" % (x,y,premio)
```

[89]:
```
"Ho saltato 3 volte, fatto 5 flessioni e vinto il premio 'Migliore Atleta della Valle'
↪"
```

Esercizio - supercar

Hai ricevuto dei soldi, perciò decidi di comprare due modelli di auto sportive. Visto lo stile di guida scellerato prevedi già incidenti, perciò per ciascun modello comprerai tante auto quanti caratteri ci sono nel nome di ciascun modello.

Scrivi del codice che salva nella stringa s il numero di auto che comprerai.

- **USA** il segnaposto %s

Esempio - dati:

```
auto1 = 'Jaguar'
auto2 = 'Ferrari'
```

Dopo il tuo codice, dovrebbe mostrare:

[131] https://realpython.com/python-formatted-output/
[132] https://realpython.com/python-formatted-output/#the-python-formatted-string-literal-f-string

```
>>> s
'Comprerò 6 Jaguar e 7 Ferrari perchè tengo i dindi'
```

```
[90]: auto1, auto2 = 'Jaguar','Ferrari'        # Comprerò 6 Jaguar e 7 Ferrari perchè tengo␣
↪i dindi
#auto1, auto2 = 'Porsche','Lamborghini'  # Comprerò 7 Porsche e 11 Lamborghini perchè␣
↪tengo i dindi

# write here

s = 'Comprerò %s %s e %s %s perchè tengo i dindi' % (len(auto1), auto1, len(auto2),␣
↪auto2)
print(s)
```

```
Comprerò 6 Jaguar e 7 Ferrari perchè tengo i dindi
```

4.7.15 Prosegui

Trovi ulteriori esercizi nel foglio Stringhe 2 - operatori[133]

4.8 Stringhe 2 - operatori

4.8.1 Scarica zip esercizi

Naviga file online[134]

Python offre diversi operatori per lavorare con le stringhe:

Operatore	Uso	Risultato	Significato
len	`len`(str)	int	Ritorna la lunghezza della stringa
concatenazione	str + str	str	Concatena due stringhe
inclusione	str `in` str	bool	Controlla se la stringa è presente in un'altra stringa
`indice <#Leggere-caratteri>`__	str [int]	str	Legge il carattere all'indice specificato
slice	str [int : int]	str	Estrae una sotto-stringa
uguaglianza	==, !=	`bool`	Controlla se due stringhe sono uguali o differenti
replicazione	str * int	str	Replica la stringa

[133] https://it.softpython.org/strings/strings2-sol.html
[134] https://github.com/DavidLeoni/softpython-it/tree/master/strings

4.8.2 Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
strings
    strings1.ipynb
    strings1-sol.ipynb
    strings2.ipynb
    strings2-sol.ipynb
    strings3.ipynb
    strings3-sol.ipynb
    strings4.ipynb
    strings4-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `strings1.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.8.3 Leggere caratteri

Una stringa è una sequenza di caratteri, e spesso potremmo voler accedere ad un singolo carattere specificando la posizione del carattere che interessa

E' importante ricordarsi che le posizioni dei caratteri nelle stringhe iniziano da 0. Per leggere un carattere ad una certa posizione, bisogna scrivere la stringa seguita da parentesi quadre con all'interno la posizione. Esempi:

[2]:
```
'ciao'[0]
```

[2]: 'c'

[3]:
```
'ciao'[1]
```

[3]: 'i'

[4]:
```
#0123
'ciao'[2]
```

[4]: 'a'

[5]:
```
#0123
'ciao'[3]
```

[5]:
```
'o'
```

Se cerchiamo di andare oltre all'ultimo carattere, otterremo un errore:

```
#0123
'ciao'[4]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-106-b8f1f689f0c7> in <module>
      1 #0123
----> 2 'ciao'[4]

IndexError: string index out of range
```

Prima abbiamo usato una stringa specificandola letteralmente, ma possiamo anche usare variabili:

[6]:
```
    #01234
x = 'panca'
```

[7]:
```
x[0]
```

[7]:
```
'p'
```

[8]:
```
x[2]
```

[8]:
```
'n'
```

Come viene rappresentato il carattere letto? Se hai notato, è circondato da apici come se fosse una stringa. Controlliamo:

[9]:
```
type(x[0])
```

[9]:
```
str
```

Infatti è proprio una stringa. Questo potrebbe forse sorprendere alcuni, anche da un punto di vista filosico: le stringhe in Python sono a loro volta composte da… stringhe! In altri linguaggi di programmazione per i singoli caratteri si possono trovare tipi appositi, ma Python usa stringhe per gestire meglio anche alfabeti con simboli complicati come quello giapponese.

DOMANDA: Supponiamo che `x` sia una stringa *qualsiasi*. Se proviamo ad eseguire questo codice:

```
x[0]
```

otterremo:

1. sempre un carattere
2. sempre un errore
3. a volte un carattere, a volte un errore a seconda della stringa

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 3: potremmo ottenere un errore con la stringa vuota (prova)

DOMANDA: Supponiamo che `x` sia una stringa qualsiasi. Se proviamo ad eseguire questo codice:

```
x[len(x)]
```

otterremo:

1. sempre un carattere
2. sempre un errore
3. a volte un carattere, a volte un errore a seconda della stringa

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 2: poichè l'inidicizzazione inizia da 0, `len` ci da sempre un numero che è superiore di 1 al più grande indice usabile.

Esercizio - intercalare

Date due stringhe che hanno entrambe lunghezza `3`, stampa una stringa che inercala caratteri da entrambe le stringhe. Il tuo codice deve poter funzionare con qualsiasi stringa di questa lunghezza

Esempio:

Dati

```
x="say"
y="hi!"
```

dovrebbe stampare

```
shaiy!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[10]:
```
# scrivi qui

x="say"
y="hi!"
print(x[0] + y[0] + x[1] + y[1] + x[2] + y[2])
```

```
shaiy!
```

[10]:
```
# scrivi qui
```

```
shaiy!
```

Indici negativi

In Python possiamo anche usare indici negativi, che invece di partire *dall'inizio* partono *dalla fine*:

[11]:
```
#4321
"ciao"[-1]
```

[11]:
```
'o'
```

[12]:
```
#4321
"ciao"[-2]
```

[12]:
```
'a'
```

[13]:
```
#4321
"ciao"[-3]
```

[13]:
```
'i'
```

[14]:
```
#4321
"ciao"[-4]
```

[14]:
```
'c'
```

Se andiamo troppo in là, otteniamo un errore:

```
#4321
"ciao"[-5]

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-126-668d8a13a324> in <module>
----> 1 "ciao"[-5]

IndexError: string index out of range
```

DOMANDA: Supponiamo che `x` sia una stringa NON vuota qualsiasi, con la seguente espressione cosa otterremmo ?

```
x[-len(x)]
```

1. sempre un carattere
2. sempre un errore
3. a volte un carattere, a volte un errore a seconda della stringa

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 1. (abbiamo supposto che la stringa non sia mai vuota)

DOMANDA: Supponendo che `x` sia una stringa qualsiasi (anche vuota), le espressioni

```
x[len(x)-1]
```

e

```
x[-len(x)]
```

sono equivalenti ? Che cosa fanno ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: le espressioni sono equivalenti: se la stringa è vuota entrambe producono un errore, se è piena entrambe restituiscono l'ultimo carattere

DOMANDA: Se `x` è una stringa non vuota, l'espressione seguente cosa produce? La possiamo semplificare in una più breve?

```
(x + x)[len(x)]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: corrisponde a `x[0]`

DOMANDA: Se `x` è una stringa non vuota, cosa produce la seguente espressione? Un errore? Altro? La possiamo semplificare?

```
'ciao'[0][0]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Sappiamo che `'ciao'[0]` produce un carattere, ma sappiamo anche che in Python i caratteri estratti dalle stringhe sono a loro volta stringhe di lunghezza 1. Quindi se dopo l'espressione `"ciao"[0]` che produce la stringa `'c'` aggiungiamo un'altro `[0]` è come se stessimo scrivendo `'c'[0]`, che ritorna il carattere zeresimo trovato nella stringa `'c'`, cioè `c` stesso.

DOMANDA: Se `x` è una stringa non vuota, cosa produce la seguente espressione? Un errore? Altro? La possiamo semplificare?

```
(x[0])[0]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: `x[0]` è un espressione che produce il primo carattere della stringa `x`. In Python, possiamo mettere le espressioni tra parentesi che vogliamo. Quindi in questo caso la parentesi non produce nessun effetto, e l'espressione è equivalente a `x[0][0]` che come abbiamo visto prima è uguale a scrivere `x[0]`

4.8.4 Sostituire caratteri

Abbiamo detto che in Python le stringhe sono immutabili, quindi se abbiamo una stringa

[15]:
```
    #01234
x = 'ciao'
```

e vogliamo per esempio cambiare il carattere in posizione 2 (in questo caso, la `a`) perchè diventi una `b`, come facciamo ?

Potremmo essere tentati di scrivere così, ma Python ci punirebbe con un errore:

```
x[2] = 'b'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-113-e5847c6fa4bf> in <module>
----> 1 x[2] = 'b'

TypeError: 'str' object does not support item assignment
```

La soluzione corretta è assegnare a `x` una stringa completamente nuova, ottenuta prendendo pezzi della precedente:

[16]:
```
x = x[0] + x[1] + 'b' + x[3]
```

[17]:
```
x
```

[17]:
```
'cibo'
```

Se il fatto che la `x` compaia a destra dell'uguale ti sconcerta, possiamo scomporre il codice così e funzionerebbe ugualmente

[18]:
```
x = "ciao"
y = x
x = y[0] + y[1] + 'b' + y[3]
```

Provalo in Python Tutor:

[19]:
```
x = "ciao"
y = x
x = y[0] + y[1] + 'b' + y[3]

jupman.pytut()
```

[19]:
```
<IPython.core.display.HTML object>
```

4.8.5 Slice

Potremmo voler leggere solo una sottosequenza che parte da una posizione e finisce in un'altra. Per esempio, supponiamo di avere la scritta

[20]:
```
    #0123456789
x = 'mercantile'
```

e vogliamo estrarre la stringa `'canti'`, che parte dall'indice 3 **incluso** e arriva all'indice 8 **escluso**. Potremmo estrarre i singoli caratteri e concatenarli con il `+`, ma scriveremmo tantissimo codice. Un opzione migliore è usare le cosiddette slice[135]: basta scrivere la stringa seguita da parentesi quadre contenti i due indici di inzio (**incluso**) e fine (**escluso**) separati dai due punti `:`

[135] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2009.html#sec98

[21]:
```
    #0123456789
x = 'mercantile'

x[3:8]   # notare i : in mezzo ai due indici di inizio e fine
```

[21]:
```
'canti'
```

ATTENZIONE: Estrarre una sottostringa con le slice NON modifica la stringa originale !!

Vediamo un esempio:

[22]:
```
    #0123456789
x = 'mercantile'

print('              x è', x)
print('La slice x[3:8] è', x[3:8])
print('              x è', x)       # nota che x continua a puntare alla vecchia␣
↪stringa!
```

```
              x è mercantile
La slice x[3:8] è canti
              x è mercantile
```

DOMANDA: se `x` è una stringa qualsiasi di lunghezza almeno 5, cosa fa questo codice? Da errore? Funziona? Possiamo abbreviarlo?

```
x[3:4]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Se la stringa ha lunghezza almeno 5, potremmo avere una situazione del genere:

```
    #01234
x = 'abcde'
```

La slice `x[3:4]` estrarrà i caratteri da posizione 3 **incluso** fino alla posizione 4 **esclusa**, quindi di fatto estrarrà solo un carattere alla posizione 3. Il codice è quindi equivalente a `x[3]`

Esercizio - garalampog

Scrivi del codice per estrarre e stampare `alam` dalla stringa `"garalampog"`. Prova a indovinare correttamente gli indici.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
x = "garalampog"

# scrivi qui
```

(continues on next page)

(continued from previous page)

```
#      0123456789
print(x[3:7])
```

```
alam
```

[23]:
```
x = "garalampog"

# scrivi qui
```

```
alam
```

Esercizio - ifEweEfav lkSD lkWe

Scrivi del codice per estrarre e stampare `kS` dalla stringa `"ifE\te\nfav lkD lkWe"`. Fai attenzione agli spazi e caratteri speciali (prima potresti voler stampare `x`). Prova a indovinare gli indici corretti.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[24]:
```
x = "ifE\te\nfav  lkD lkWe"

# scrivi qui

#     0123 45 67890123456789
#x = "ifE\te\nfav  lkD lkWe"

print(x[12:14])
```

```
kD
```

[24]:
```
x = "ifE\te\nfav  lkD lkWe"

# scrivi qui
```

```
kD
```

Slice - limiti

Quando operiamo con le slice dobbiamo stare attenti ai limiti degli indici. Vediamo come si comportano:

[25]:
```
#012345
"sedia"[0:3]  # da indice 0 *incluso* a 3 *escluso*
```

[25]:
```
'sed'
```

[26]:
```
#012345
"sedia"[0:4]  # da indice 0 *incluso* a 4 *escluso*
```

[26]:
```
'sedi'
```

[27]:
```
#012345
"sedia"[0:5]  # da indice 0 *incluso* a 5 *escluso*
```

[27]:
```
'sedia'
```

[28]:
```
#012345
"sedia"[0:6]   # se andiamo oltre la lunghezza della stringa Python non si arrabbia
```

[28]:
```
'sedia'
```

DOMANDA: se `x` è una stringa qualsiasi (anche vuota), questa espressione cosa fa? Può dare errore? o ritorna qualcosa di utile?

```
x[0:len(x)]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Ritorna sempre una NUOVA copia dell'intera stringa, perchè parte da indice 0 *incluso* e finisce a indice `len(x)` *escluso*

Funziona anche con la stringa vuota, perchè `''[0:len('')]` equivale a `''[0:0]` cioè sottostringa da 0 *incluso* a 0 *escluso*, quindi non prendiamo nessun carattere e non sforiamo i limiti della stringa. In realtà, anche se sforassimo, Python non avrebbe problemi (prova a scrivere `''[0:100]`)

Slice - omissione limiti

Volendo, è possibile omettere l'indice di partenza, in tal caso Python supporrà sia 0:

[29]:
```
#0123456789
"traghetto"[:3]
```

[29]:
```
'tra'
```

E' anche possibile omettere l'indice di fine, in tal caso Python estrarrà fino alla fine della stringa:

[30]:
```
#0123456789
"traghetto"[3:]
```

[30]:
```
'ghetto'
```

Omettendo entrambi gli indici si ottiene l'intera stringa:

[31]:
```
"traghetto"[:]
```

[31]:
```
'traghetto'
```

Esercizio - isteroister

Scrivere del codice che data una stringa `x` stampa la stringa composta da tutte le lettere di `x` eccetto la prima seguita da tutte le lettere di `x` eccetto l'ultima

- il tuo codice deve funzionare con qualunque stringa

Esempio:

Dato

```
x = "mistero"
```

deve stampare

```
isteromister
```

Dato

```
x = "parlare"
```

deve stampare

```
arlareparlar
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[32]:
```
x = "mistero"

# scrivi qui

print(x[1:] + x[0:len(x)-1])
```

```
isteromister
```

[32]:
```
x = "mistero"

# scrivi qui
```

```
isteromister
```

Slice - limiti negativi

Volendo è anche possibile impostare limiti negativi, per quanto non sia sempre molto intuitivo.

[33]:
```
#0123456
"foresta"[3:0]    # da indice 3 a indici positivi <= 3 non produce nulla
```

[33]:
```
''
```

[34]:
```
#0123456
"foresta"[3:1]    # da indice 3 a indici positivi <= 3 non produce nulla
```

[34]:
```
''
```

[35]:
```
#0123456
"foresta"[3:2]  # da indice 3 a indici positivi <= 3 non produce nulla
```

[35]:
```
''
```

[36]:
```
#0123456
"foresta"[3:3]  # da indice 3 a indici positivi <= 3 non produce nulla
```

[36]:
```
''
```

Vediamo cosa succede con indici negativi:

[37]:
```
#0123456   indici positivi
#7654321   indici negativi
"foresta"[3:-1]
```

[37]:
```
'est'
```

[38]:
```
#0123456   indici positivi
#7654321   indici negativi
"foresta"[3:-2]
```

[38]:
```
'es'
```

[39]:
```
#0123456   indici positivi
#7654321   indici negativi
"foresta"[3:-3]
```

[39]:
```
'e'
```

[40]:
```
#0123456   indici positivi
#7654321   indici negativi
"foresta"[3:-4]
```

[40]:
```
''
```

[41]:
```
#0123456   indici positivi
#7654321   indici negativi
"foresta"[3:-5]
```

[41]:
```
''
```

Esercizio - javarnanda

Data una stringa `x`, scrivi del codice per estrarre e stampare i suoi ultimi 3 caratteri e unirli ai primi 3.

- Il tuo codice deve funzionare per qualsiasi stringa di lunghezza uguale o superiore a 3

Esempio 1 - dato:

```
x = "javarnanda"
```

dovrebbe stampare:

```
javnda
```

Esempio 2 - dato:

```
x = "abcd"
```

dovrebbe stampare:

```
abcbcd
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[42]:
```
x = "javarnanda"
#x = "abcd"

# scrivi qui

print(x[:3] + x[-3:])
```

```
javnda
```

[42]:
```
x = "javarnanda"
#x = "abcd"

# scrivi qui
```

```
javnda
```

Slice - modifica

Supponiamo di avere la stringa

[43]:
```
    #0123456789
s = "il tavolo bianco è al centro della stanza"
```

e di voler cambiare l'assegnazione della variabile `s` in modo che diventi associata alla stringa

```
#0123456789
"il divano bianco è al centro della stanza"
```

Visto che le due stringhe sono simili, potremmo essere tentati di ridefinire solo la sequenza di caratteri corrispondente alla parola `"tavolo"`, che va dall'indice `3` incluso all'indice `9` escluso:

```
s[3:9] = "divano"   # ATTENZIONE! SBAGLIATO!

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-57-0de7363c6882> in <module>
----> 1 s[3:9] = "divano"   # ATTENZIONE! SBAGLIATO!

TypeError: 'str' object does not support item assignment
```

Purtroppo, otterremo un errore, perchè come ripetuto più volte le stringhe sono IMMUTABILI, quindi non possiamo individuare una parte in una particolare stringa e provare a cambiare la stringa originale. Quello che invece possiamo fare è costruire una NUOVA stringa a partire da pezzi della stringa originale, concatenare i caratteri desiderati e associare il risultato alla variabile di cui vogliamo modificare l'assegnazione:

[44]:
```
    #0123456789
s = "il tavolo bianco è al centro della stanza"
s = s[0:3] + "divano" + s[9:]
print(s)
```

```
il divano bianco è al centro della stanza
```

Quando Python trova la linea

```
s = s[0:3] + "divano" + s[9:]
```

PRIMA calcola il risultato della parte a destra dell' =, e POI associa il risultato alla variabile che sta a sinistra. Nell'espressione a destra dell' = vengono solo solo generate e concatenate NUOVE stringhe, che quindi una volta costruite possono essere assegnate alla variabile `s`.

Esercizio - la corsa

Scrivere del codice per cui data una stringa `s`

```
s = "la corsa all'oro è iniziata."
```

e delle variabili

```
cosa = "atomo"
verbo = "finita"
```

sostituisce la sottostringa in corrispondenza di `"oro"` con la stringa presente nella variabile `cosa` e sostituisce la sottostringa in corrispondenza di `"iniziata"` con la stringa presente nella variabile `verbo`.

Dopo l'esecuzione del tuo codice, la stringa associata ad `s` dovra essere

```
>>> print(s)
"la corsa all'atomo è finita."
```

- **NON** usare caratteri costanti nel tuo codice, quindi niente punto `'.'` !

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[45]:
```
    #01234567890123456789012345678
s = "la corsa all'oro è iniziata."
cosa = "atomo"
verbo = "finita"

# scrivi qui

s = s[:13] + cosa + s[16:19] + verbo + s[27:]
print(s)
```

```
la corsa all'atomo è finita.
```

[45]:
```
    #01234567890123456789012345678
s = "la corsa all'oro è iniziata."
cosa = "atomo"
verbo = "finita"

# scrivi qui
```

```
la corsa all'atomo è finita.
```

4.8.6 Operatore `in`

Per verificare se una stringa è contenuta in un'altra, possiamo usare l'operatore `in`.

Nota che il risultato di questa espressione è un booleano:

[46]:
```
'tra' in 'Cantando per le strade'
```

[46]:
```
True
```

[47]:
```
'ca' in 'Cantando per le strade'  # l'operatore in   distingue tra maiuscole/minuscole
```

[47]:
```
False
```

[48]:
```
'Ca' in 'Cantando per le strade'
```

[48]:
```
True
```

Esercizio - contenute 1

Ti vengono date due stringhe `x` e `y`, e una terza `z`. Scrivi del codice che stampa `True` se `x` e `y` sono entrambe contenute in `z`.

Esempio 1 - date:

```
x = 'cad'
y = 'ra'
z = 'abracadabra'
```

dovrebbe stampare

```
True
```

Esempio 2 - date:

```
x = 'zam'
y = 'ra'
z = 'abracadabra'
```

dovrebbe stampare

```
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[49]:
```
x,y,z = 'cad','ra','abracadabra'   # True
#x,y,z = 'zam','ra','abracadabra'  # False

# scrivi qui

print((x in z) and (y in z))
```

```
True
```

[49]:
```
x,y,z = 'cad','ra','abracadabra'   # True
#x,y,z = 'zam','ra','abracadabra'  # False

# scrivi qui



```

```
True
```

Esercizio - contenute 2

Date tre stringhe `x`, `y`, `z`, scrivere del codice che stampa `True` se una stringa `x` è contenuta almeno in una delle stringhe `y` o `z`, altrimenti stampa `False`

- il tuo codice deve funzionare con qualunque insieme di stringhe

Esempio 1 - dati:

```
x = 'tti'
y = 'patti chiari amicizia lunga'
z = 'tutto chiaro?'
```

deve mostrare:

```
True
```

Esempio 2 - dati:

```
x = 'zio'
y = 'patti chiari amicizia lunga'
z = 'tutto chiaro?'
```

deve mostrare:

```
False
```

Esempio 3 - dati:

```
x = 'chiaro'
y = 'patti chiari amicizia lunga'
z = 'tutto chiaro?'
```

deve mostrare:

```
True
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[50]:
```
x,y,z = 'tti', 'patti chiari amicizia lunga', 'tutto chiaro?'   # True
#x,y,z = 'zio','patti chiari amicizia lunga','tutto chiaro?'    # False
#x,y,z = 'chiaro','patti chiari amicizia lunga','tutto chiaro?' # True

# scrivi qui

print((x in y) or (x  in z))
```

```
True
```

[50]:
```
x,y,z = 'tti', 'patti chiari amicizia lunga', 'tutto chiaro?'   # True
#x,y,z = 'zio','patti chiari amicizia lunga','tutto chiaro?'    # False
#x,y,z = 'chiaro','patti chiari amicizia lunga','tutto chiaro?' # True

# scrivi qui



```

```
True
```

4.8.7 Comparazioni

Python ci offre la possibilità di eseguire una *comparazione lessicografica* tra stringhe, come faremmo scrivendo nomi in una agenda di indirizzi. Sebbene ordinare nomi sia qualcosa di intuitivo che facciamo spesso, dobbiamo prestare attenzione ai casi speciali.

Per prima cosa, vediamo come determinare quando due stringhe sono uguali.

4.8.8 Operatori uguaglianza

Per verificare se due stringhe sono uguali, possiamo usare l'operatore == che come risultato produce un booleano `True` oppure `False`

ATTENZIONE: si scrive == con DUE uguali !!!

[51]:
```
"gatto" == "gatto"
```

[51]:
```
True
```

[52]:
```
"gatto" == "cane"
```

[52]:
```
False
```

L'operatore di uguaglianza distingue tra maiuscole e minuscole:

[53]:
```
"gatto" == "GATTO"
```

[53]:
```
False
```

Per vedere se due stringhe NON sono uguali, possiamo usare l'operatore `!=`, che come possiamo attenderci si comporta esattamente all'opposto di `==`:

[54]:
```
"gatto" != "gatto"
```
[54]:
```
False
```

[55]:
```
"gatto" != "cane"
```
[55]:
```
True
```

[56]:
```
"gatto" != "GATTO"
```
[56]:
```
True
```

In alternativa, potremmo anche usare l'operatore `not`:

[57]:
```
not "gatto" == "gatto"
```
[57]:
```
False
```

[58]:
```
not "gatto" == "cane"
```
[58]:
```
True
```

[59]:
```
not "gatto" == "GATTO"
```
[59]:
```
True
```

DOMANDA: Il codice seguente cosa stampa?

```
x = "fiume" == "fiume"
print(x)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Quando Python incontra `x = "fiume" == "fiume"` vede che è un'assegnazione, che dovrebbe associare alla variabile `x` il risultato dell'espressione `"fiume" == "fiume"`. Quindi PRIMA calcola l'espressione `"fiume" == "fiume"` che produce il booleano `True`, e POI associa il valore `True` alla variabile `x`. Quindi vedremo stampato `True`

DOMANDA: Per ciascuna delle espressioni seguenti, prova a indovinare se produce `True` o `False`

1. `'cappello' != 'Cappello'`
2. `'cappello' == 'CAPPELLO'`
3. `'pensieroso'[4:8] == 'forestiero'[6:]`
4. `'CoStOsO'[4:] == 'oSo'`
5. `'chiaro'[9:20] == 'scuro'[10:15]`

6. `'ramarro'[-1] == 'giglio'[-1]`

7. `('cappello' != 'giacca') == ('pantaloni' != 'cravatta')`

8. `('stra' in 'stradivarius') == ('div' in 'divano')`

9. `len('nota') in '5436'`

10. `str(len('nota') in '5436'`

11. `len('cartellone') in '5436'`

12. `str(len('cartellone')) in '5436'`

Esercizio - rockettaro

Scrivi del codice che stampa `True` se una `parola` inizia con le stesse due lettere con cui finisce

- Il tuo codice deve funzionare per qualsiasi `parola`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[60]:
```
parola = 'rockettaro'  # True
#parola = 'massima'    # True
#parola = 'stima'      # False
#parola = 'baobab'     # False

# scrivi qui

print(parola[:2] == parola[-2:len(parola)])
```

```
True
```

[60]:
```
parola = 'rockettaro'  # True
#parola = 'massima'    # True
#parola = 'stima'      # False
#parola = 'baobab'     # False

# scrivi qui
```

```
True
```

Comparare caratteri

I caratteri hanno un ordine che possiamo sfruttare. Vediamo un esempio:

```
[61]: 'a' < 'g'
[61]: True
```

un altro:

```
[62]: 'm' > 'c'
[62]: True
```

Sembrano comparazioni ragionevoli! Ma cosa dire di queste (nota la `Z` maiuscola)?

```
[63]: 'a' < 'Z'
[63]: False
```

Forse questa non sembra così ovvia. E se proviamo ad essere creativi e compariamo con simboli come parentesi quadre o cuori Unicode[136] ??

```
[64]: 'a' > '♥'
[64]: False
```

Per determinare come gestire questi casi speciali, dobbiamo ricordare che la codifica ASCII[137] assegna un numero di posizione ad ogni carattere, definendo di fatto *un ordine* tra tutti i suoi caratteri.

Se vogliamo sapere il numero corrispondente di un carattere, possiamo usare la funzione `ord`:

```
[65]: ord('a')
[65]: 97
```

```
[66]: ord('b')
[66]: 98
```

```
[67]: ord('z')
[67]: 122
```

Se vogliamo andare nell'altro senso, dato un numero di posizione possiamo ottenere il carattere corrispondente con la funzione `chr`:

```
[68]: chr(97)
[68]: 'a'
```

I caratteri maiuscoli hanno posizioni differenti:

```
[69]: ord('A')
[69]: 65
```

[136] https://it.softpython.org/strings/strings1-sol.html#Caratteri-Unicode
[137] https://it.softpython.org/strings/strings1-sol.html#Caratteri-ASCII

[70]:
```
ord('Z')
```

[70]:
```
90
```

ESERCIZIO: Usando le funzioni sopra, prova a trovare quali caratteri sono *tra* la `Z` maiuscola e la `a` minuscola.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[71]:
```
# scrivi qui
#print(chr(91),chr(92), chr(93),chr(94), chr(95),chr(96))
```

[71]:
```
# scrivi qui
```

L'ordine ci permette di eseguire *comparazioni lessicografiche* tra caratteri singoli:

[72]:
```
'a' < 'b'
```

[72]:
```
True
```

[73]:
```
'g' >= 'm'
```

[73]:
```
False
```

ESERCIZIO: Scrivi del codice che:

1. stampa i valori `ord` di `'A'`, `'Z'` e un dato carattere `car`
2. stampa `True` se `car` è maiuscolo, e `False` altrimenti

- il tuo codice funzionerebbe anche con caratteri accentati capitalizzati come `'Á'`?
- **NOTA**: gli insiemi di caratteri possibili sono parecchi, perciò la soluzione appropriata sarebbe usare il metodo isupper[138] che vedremo nel prossimo tutorial.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[74]:
```
car = 'G'    # True
#car = 'g'   # False

#car = 'Á'   # True ?? Nota l'accento!

# scrivi qui
print('A:', ord('A'), ' Z:', ord('Z'))
print(car+':',ord(car))
ord(car) >= ord('A') and ord(car) <= ord('Z')   #  only checks simple English alphabet␣
↪cases
```

```
A: 65  Z: 90
G: 71
```

[138] https://it.softpython.org/strings/strings3-sol.html#Metodi-isupper-e-islower

[74]:
```
True
```

[74]:
```
car = 'G'    # True
#car = 'g'   # False

#car = 'Á'   # True ?? Nota l'accento!

# scrivi qui
```

```
A: 65  Z: 90
G: 71
```

[74]:
```
True
```

Inoltre, dato che l'insieme dei caratteri Unicode *include* l'ASCII, l'ordine dei caratteri ASCII può essere usato per compararli in sicurezza con i caratteri unicode, perciò comparare i caratteri o i loro `ord` dovrebbe sempre essere equivalente:

[75]:
```
ord('a')    # ascii
```

[75]:
```
97
```

[76]:
```
ord('♥')    # unicode
```

[76]:
```
9829
```

[77]:
```
'a' > '♥'
```

[77]:
```
False
```

[78]:
```
ord('a') > ord('♥')
```

[78]:
```
False
```

Python offre anche comparazioni lessigrafiche su stringhe con più di un carattere. Per capire quale dovrebbe essere il risultato atteso, dobbiamo però distinguere diversi casi:

- stringhe di lunghezza uguale / differente
- stringhe con capitalizzazione uguale / diversa

Cominciamo con stringhe della stessa lunghezza:

[79]:
```
'mario' > 'luigi'
```

[79]:
```
True
```

[80]:
```
'mario' > 'wario'
```

[80]:
```
False
```

[81]:
```
'Mario' > 'Wario'
```

[81]:
```
False
```

[82]:
```
'Wario' < 'mario'    # il maiuscolo è *prima* del minuscolo in ASCII
```

[82]:
```
True
```

Comparare lunghezze diverse

Stringhe corte che sono incluse in altre più lunghe appaiono prima nell'ordinamento:

[83]:
```
'troll' < 'trolley'
```

[83]:
```
True
```

Se condividono solo un prefisso con una stringa più lunga, Python compara i caratteri dopo il prefisso comune, in questo caso rileva che `e` precede la `s` corrispondente:

[84]:
```
'trolley' < 'trolls'
```

[84]:
```
True
```

Esercizio - Intervalli di caratteri

Ti vengono fornite due coppie di stringhe `i1` e `i2` da due caratteri ciascuna.

Supponi che rappresentino intervalli di caratteri: il primo carattere di un intervallo ha sempre un numero d'ordine inferiore o uguale al secondo.

Ci sono cinque possibilità: il primo intervallo 'è contenuto in', oppure 'contiene', oppure 'si sovrappone parzialmente a', oppure 'è prima' o 'è dopo' il secondo intervallo. Scrivi del codice che dice quale relazione di contenimento abbiamo.

Esempio 1 - dati:

```
i1 = 'gm'
i2 = 'cp'
```

Il tuo programma dovrebbe stampare:

```
gm è contenuto in cp
```

Per vedere perchè, puoi guardare a questa piccola rappresentazione (**non** è necessario stamparla!):

```
  c   g     m  p
abcdefghijklmnopqrstuvwxyz
```

Esempio 2 - dati:

```
i1 = 'mr'
i2 = 'pt'
```

Il tuo programma dovrebbe stampare:

```
mr si sovrappone parzialmente a pt
```

perchè `mr` non è contenuto nè contiene ne completamente precede nè completamente segue `pt` (**non** è necessario stampare questo:)

```
            m  p r t
abcdefghijklmnopqrstuvwxyz
```

- se l'intervallo `i1` coincide con `i2`, è considerato come contenente `i2`
- **NON** usare cicli nè `if`
- **SUGGERIMENTO**: per soddisfare i vincoli qui sopra, pensa all'ordine di valutazione dei booleani[139]:

```
'g' >= 'c' and 'm' <= 'p' and 'è contenuto in'
```

produce come risultato la stringa `'è contenuto in'`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[85]:
```
i1,i2 = 'gm','cp'   # gm è contenuto in cp
#i1,i2 = 'dh','dh'  # gm è contenuto in cp  #(caso speciale)
#i1,i2 = 'bw','dq'  # bw contiene dq
#i1,i2 = 'ac','bd'  # ac si sovrappone parzialmente a bd
#i1,i2 = 'mr','pt'  # mr si sovrappone parzialmente  pt
#i1,i2 = 'fm','su'  # fm è prima di su
#i1,i2 = 'xz','pq'  # xz è dopo pq

# scrivi qui
res = (i1[0] >= i2[0] and i1[1] <= i2[1] and 'è contenuto in')  \
      or (i1[0] < i2[0] and i1[1] >i2[1] and 'contiene')         \
      or (i1[0] >= i2[0] and i1[0] <= i2[1] and 'si sovrappone parzialmente a') \
      or (i1[1] >= i2[0] and i1[1] <= i2[1] and 'si sovrappone parzialmente a')  \
      or (i1[1] < i2[0] and 'è prima di')   \
      or (i1[0] > i2[1] and 'è dopo')

print(i1, res, i2)
```

```
gm è contenuto in cp
```

[85]:
```
i1,i2 = 'gm','cp'   # gm è contenuto in cp
#i1,i2 = 'dh','dh'  # gm è contenuto in cp  #(caso speciale)
#i1,i2 = 'bw','dq'  # bw contiene dq
#i1,i2 = 'ac','bd'  # ac si sovrappone parzialmente a bd
#i1,i2 = 'mr','pt'  # mr si sovrappone parzialmente  pt
#i1,i2 = 'fm','su'  # fm è prima di su
#i1,i2 = 'xz','pq'  # xz è dopo pq

# scrivi qui
```

```
gm è contenuto in cp
```

[139] https://it.softpython.org/basics/basics2-bools-sol.html#Ordine-di-valutazione

Esercizio - La Biblioteca di Encodicus

Nello studio dell'algoritmista Encodicus è presente uno scaffale diviso in 26 sezioni ordinate alfabeticamente, ove custodisce gelosamente i suoi preziosi testi alchemici. Ciascuna sezione può contenere al massimo 9 libri. Un giorno, Encodicus decide di acquistare un nuovo tomo da aggiungere alla sua collezione: scrivi del codice che data una stringa rappresentante le `scaffale` con i conteggi dei libri e un nuovo `libro`, trovi la giusta posizione del libro e aggiorni `scaffale` di conseguenza

- assumi che nessuna sezione contenga `9` libri
- assumi che i nomi dei libri siano sempre in minuscolo
- **NON usare cicli, `if`, o metodi delle stringhe**
- **NON** scrivere manualmente stringhe con 26 caratteri, o peggio creare 26 variabili !!!
- **USA** `ord` per trovare la posizione della sezione

Esempio - dati:

```
scaffale = "|a 7|b 5|c 5|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r␣
↪2|s 4|t 6|u 1|v 3|w 3|x 5|y 7|z 6|"
libro = "ciclare all'aperto"
```

dopo il tuo codice `scaffale` deve risultare aggiornato con `|c 6|`:

```
>>> print(scaffale)
|a 7|b 5|c 6|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r 2|s 4|t 6|u␣
↪1|v 3|w 3|x 5|y 7|z 6|
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[86]:
```
libro = "ciclare all'aperto"
#libro = "algoritmi occulti"
#libro = "teoria dello zippo"
#libro = "zoologia dello sviluppatore software"

scaffale = "|a 7|b 5|c 5|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r␣
↪2|s 4|t 6|u 1|v 3|w 3|x 5|y 7|z 6|"

# scrivi qui

c = libro[0]

i = ord(c) - ord('a')
n = int(scaffale[(i*4)+3:((i)*4)+4])

scaffale = scaffale[:i*4] + '|' + c + ' ' + str(n+1)  + scaffale[(i+1)*4:]

print(scaffale)
```

```
|a 7|b 5|c 6|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r 2|s 4|t 6|u␣
↪1|v 3|w 3|x 5|y 7|z 6|
```

[86]:
```
libro = "ciclare all'aperto"
#libro = "algoritmi occulti"
```

(continues on next page)

(continued from previous page)

```
#libro = "teoria dello zippo"
#libro = "zoologia dello sviluppatore software"

scaffale = "|a 7|b 5|c 5|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r␣
↪2|s 4|t 6|u 1|v 3|w 3|x 5|y 7|z 6|"

# scrivi qui
```

```
|a 7|b 5|c 6|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r 2|s 4|t 6|u␣
↪1|v 3|w 3|x 5|y 7|z 6|
```

4.8.9 Operatore di replicazione

Con l'operatore `*` puoi replicare una stringa n volte, per esempio:

[87]:
```
'birra' * 4
```

[87]:
```
'birrabirrabirrabirra'
```

Nota come venga creata una NUOVA stringa, l'originale non viene intaccata:

[88]:
```
beviamo = "birra"
```

[89]:
```
print(beviamo * 4)
```

```
birrabirrabirrabirra
```

[90]:
```
beviamo
```

[90]:
```
'birra'
```

Esercizio - za za za

Data una `sillaba` e una `frase` che termina con una carattere `n` come cifra, scrivi del codice che stampa una stringa con la `sillaba` ripetuta `n` volte, separate da spazi.

- Il tuo codice deve funzionare con qualsiasi stringa assegnata a `sillaba` e `frase`

Esempio - date:

```
frase = 'Il numero 7'
sillaba = 'za'
```

dopo il tuo codice, deve stampare:

```
za za za za za za za
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[91]:
```
sillaba = 'za'
frase = 'Il numero 7'    # za za za za za za za
#frase = 'Dammi il 5'    # za za za za za


# scrivi qui

print((sillaba + ' ') * (int(frase[-1])))
```

```
za za za za za za za
```

[91]:
```
sillaba = 'za'
frase = 'Il numero 7'    # za za za za za za za
#frase = 'Dammi il 5'    # za za za za za


# scrivi qui
```

```
za za za za za za za
```

4.8.10 Prosegui

Trovi ulteriori esercizi nel foglio Stringhe 3 - metodi[140]

[ ]:
```
```

4.9 Stringhe 3 - metodi

4.9.1 Scarica zip esercizi

Naviga file online[141]

Ogni tipo di dati ha associati dei metodi particolari per quel tipo, vediamo quelli associati al tipo stringa (`str`)

ATTENZIONE: TUTTI i metodi sulle stringhe generano SEMPRE una NUOVA stringa

L'oggetto stringa originale non viene MAI modificato (perchè le stringhe sono immutabili).

[140] https://it.softpython.org/strings/strings3-sol.html
[141] https://github.com/DavidLeoni/softpython-it/tree/master/strings

Risultato	Metodo	Significato
str	*str.upper()*	Ritorna la stringa con tutti i caratteri maiuscoli
str	*str.lower()*	Ritorna la stringa con tutti i caratteri minuscoli
str	*str.capitalize()*	Ritorna la stringa con il primo carattere maiuscolo
bool	*str.startswith(str)*	Controlla se la stringa inizia con un'altra
bool	*str.endswith(str)*	Controlla se la stringa finisce con un'altra
bool	*str.isalpha(str)*	Controlla se tutti i caratteri sono alfabetici
bool	*str.isdigit(str)*	Controlla se tutti i caratteri sono cifre
bool	*str.isupper*	Controlla se tutti i caratteri sono minuscoli
bool	*str.islower*	Controlla se tutti i caratteri sono maiuscoli
str	*str.strip(str)*	Rimuove stringhe dai lati
str	*str.lstrip(str)*	Rimuove stringhe da sinistra
str	*str.rstrip(str)*	Rimuove stringhe da destra
int	*str.count(str)*	Conta il numero di occorrenze di una sottostringa
int	*str.find(str)*	Ritorna la prima posizione di una sottostringa a partire da sinistra
int	*str.rfind(str)*	Ritorna la prima posizione di una sottostringa a partire da destra
str	*str.replace(str, str)*	Sostituisce sottostringhe

4.9.2 Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
strings
    strings1.ipynb
    strings1-sol.ipynb
    strings2.ipynb
    strings2-sol.ipynb
    strings3.ipynb
    strings3-sol.ipynb
    strings4.ipynb
    strings4-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `strings1.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.9.3 Esempio - upper

Un metodo è una funzione di un oggetto che prende in ingresso l'oggetto a cui è applicata e vi fa dei calcoli.

Il tipo stringa (`str`) ha metodi predefiniti come `str.upper()` che possono essere applicati agli oggetti stringa (es: `'ciao'` è un oggetto stringa).

Il metodo `str.upper()` prende la stringa su cui è applicato e crea una NUOVA stringa in cui tutti i caratteri sono in maiuscolo. Per applicare un metodo come `str.upper()` al particolare oggetto stringa `'ciao'`, dobbiamo scrivere

```
'ciao'.upper()
```

cioè dobbiamo prima scrivere l'oggetto su cui applicare il metodo (`'ciao'`), poi un punto `.` , quindi il nome del metodo seguito da parentesi tonde. Le tonde possono contenere ulteriori parametri a seconda del metodo.

Esempi:

[2]:
```
'ciao'.upper()
```

[2]:
```
'CIAO'
```

[3]:
```
'prova'.upper()
```

[3]:
```
'PROVA'
```

ATTENZIONE: come TUTTI i metodi delle stringhe, l'oggetto stringa originale su cui è chiamato NON viene modificato.

Esempio:

[4]:
```
x = "ciao"
y = x.upper()     #  genera una NUOVA stringa e la associa alla variabile y
```

[5]:
```
x     # la variabile x è ancora associata alla vecchia stringa
```

[5]:
```
'ciao'
```

[6]:
```
y     # la variabile y è associata alla nuova stringa
```

[6]:
```
'CIAO'
```

Guarda lo stesso esempio in Python Tutor:

[7]:
```
x = "ciao"
y = x.upper()
print(x)
print(y)

jupman.pytut()
```

```
ciao
CIAO
```

[7]:
```
<IPython.core.display.HTML object>
```

4.9.4 Esercizio - cammina

Scrivere del codice che data una stringa `x` (per es: `x='cammina'`) stampa due volta la riga

```
cammina CAMMINA cammina CAMMINA
cammina CAMMINA cammina CAMMINA
```

- NON creare nuove variabili
- il tuo codice deve funzionare con qualsiasi stringa

[8]:
```
x = 'cammina'

print(x, x.upper(), x, x.upper())
print(x, x.upper(), x, x.upper())
```

```
cammina CAMMINA cammina CAMMINA
cammina CAMMINA cammina CAMMINA
```

Help: Se non sei sicuro riguardo a un metodo (per esempio `strip`), puoi chiedere aiuto a Python in questo modo:

ATTENZIONE: con help dopo il nome del metodo NON ci sono parentesi !!

[9]:
```
help("ciao".strip)
```

```
Help on built-in function strip:

strip(chars=None, /) method of builtins.str instance
    Return a copy of the string with leading and trailing whitespace removed.

    If chars is given and not None, remove characters in chars instead.
```

4.9.5 Metodo lower

Ritorna la stringa con tutti i caratteri minuscoli

[10]:
```
stringa = "ciAo MonDo"

altra_stringa = stringa.lower()

print(altra_stringa)
```

```
ciao mondo
```

[11]:
```
print(stringa)  # non è cambiata
```

```
ciAo MonDo
```

Esercizio - lowermezzo

Scrivi del codice che data una stringa qualsiasi `x` di lunghezza dispari, stampa una nuova stringa uguale a `x` eccetto al carattere in mezzo che deve essere in minuscolo.

- Il tuo codice deve funzionare con qualunque stringa !
- **SUGGERIMENTO**: per calcolare la posizione del carattere in mezzo, usa la divisione intera con l'operatore `//`

Esempi:

Dato

```
x = 'ASCENSORE'
```

deve stampare

`ASCEnSORE`

Dato

```
x = 'LAMPADA'
```

deve stampare

`LAMpADA`

[12]:
```
    #012345678
x = 'ASCENSORE'
k = len(x) // 2

print(x[:k] + x[k].lower() + x[k+1:])
```

```
ASCEnSORE
```

4.9.6 Metodo capitalize

Il metodo `capitalize()` crea una NUOVA stringa avente solo il PRIMO carattere in maiuscolo:

[13]:
```
"ciao".capitalize()
```

[13]:
```
'Ciao'
```

[14]:
```
"mondo".capitalize()
```

[14]:
```
'Mondo'
```

[15]:
```
x = 'capra'
y = 'capra'.capitalize()
```

[16]:
```
x      #  x rimane associata al vecchio valore
```

[16]:
```
'capra'
```

[17]:
```
y      #  y viene associata alla nuova stringa
```

[17]:
```
'Capra'
```

Esercizio - Vostra Eccellenza

Scrivere del codice che date due stringhe qualsiasi `x` e `y` restituisce le due stringhe concatenate, separandole con uno spazio ed entrambe tutte in minuscolo eccetto le rispettive prime lettere che devono essere maiuscole.

Esempio 1 - dati:

```
x = 'vosTRA'
y = 'ecCeLLeNza'
```

deve stampare:

```
Vostra Eccellenza
```

Esempio 2 - dati:

```
x = 'sUa'
y = 'maEStà'
```

deve stampare:

```
Sua Maestà
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
x,y = 'vosTRA', 'ecCeLLeNza'
#x,y = 'sUa', 'maEStà'

# scrivi qui

print(x.capitalize() + " " + y.capitalize())
```

```
Vostra Eccellenza
```

[18]:
```
x,y = 'vosTRA', 'ecCeLLeNza'
#x,y = 'sUa', 'maEStà'

# scrivi qui



```

```
Vostra Eccellenza
```

4.9.7 Metodo startswith

`str.startswith` prende come parametro una stringa e ritorna `True` se la stringa prima del punto inizia con la stringa passata come parametro. Esempio:

[19]:
```
"il cane abbaia nella strada".startswith('il cane')
```

[19]:
```
True
```

[20]:
```
"il cane abbaia nella strada".startswith('abbaia')
```

[20]:
```
False
```

[21]:
```
"il cane abbaia nella strada".startswith('IL CANE')  # le maiuscole sono diverse␣
↪dalle minuscole
```

[21]:
```
False
```

[22]:
```
"IL CANE ABBAIA SULLA STRADA".startswith('IL CANE')  # le maiuscole sono diverse␣
↪dalle minuscole
```

[22]:
```
True
```

Esercizio - per Giove

Scrivere del codice che date tre stringhe qualsiasi `x` e `y` e `z`, stampa `True` se entrambe `x` e `y` iniziano con la stringa `z`, altrimenti stampa `False`

Esempi 1 - Dati:

```
x = 'per Giove'
y = 'per Zeus'
z = 'per'
```

deve stampare

```
True
```

Esempio 2 - Dati:

```
x = 'per Giove'
y = 'per Zeus'
z = 'da'
```

deve stampare

```
False
```

Esempio 3 - Dati:

```
x = 'da Giove'
y = 'per Zeus'
z = 'per'
```

deve stampare

```
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
x = 'per Giove'
y = 'per Zeus'
z = 'per'
```

(continues on next page)

(continued from previous page)

```
# scrivi qui

print(x.startswith(z) and y.startswith(z))
```

```
True
```

[23]:
```
x = 'per Giove'
y = 'per Zeus'
z = 'per'

# scrivi qui
```

```
True
```

4.9.8 Metodo endswith

`str.endswith` prende come parametro una stringa e ritorna `True` se la stringa prima del punto finisce con la stringa passata come parametro. Esempio:

[24]:
```
"I miei più cari saluti".endswith('cari saluti')
```

[24]:
```
True
```

[25]:
```
"I miei più cari saluti".endswith('cari')
```

[25]:
```
False
```

[26]:
```
"I miei più cari saluti".endswith('SALUTI')  # le maiuscole sono diverse dalle␣
↪minuscole
```

[26]:
```
False
```

[27]:
```
"I MIEI PIU' CARI SALUTI".endswith('SALUTI')  # le maiuscole sono diverse dalle␣
↪minuscole
```

[27]:
```
True
```

Esercizio - Snobbonis

Dati nomi di coppie `marito` e `moglie`, scrivi del codice che stampa `True` se condividono il cognome, `False` altrimenti.

- assumi che il cognome sia sempre alla stessa posizione
- il tuo codice deve funzionare per qualsiasi coppia `marito moglie`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
                  #0123456789                #0123456789
marito, moglie = 'Antonio  Snobbonis',      'Carolina Snobbonis'    # True
#marito, moglie = 'Camillo  De Spaparanzi', 'Matilda  Degli Agi'   # False

# scrivi qui

print(moglie.endswith(marito[9:]))
```

```
True
```

[28]:
```
                  #0123456789                #0123456789
marito, moglie = 'Antonio  Snobbonis',      'Carolina Snobbonis'    # True
#marito, moglie = 'Camillo  De Spaparanzi', 'Matilda  Degli Agi'   # False

# scrivi qui
```

```
True
```

4.9.9 Metodo isalpha

Il metodo `isalpha` ritorna `True` se tutti i caratteri nella stringa sono alfabetici:

[29]:
```
'BarrieraCorallina'.isalpha()
```

[29]:
```
True
```

I numeri non sono considerati alfabetici:

[30]:
```
'Route 666'.isalpha()
```

[30]:
```
False
```

Nota che le spaziature (i cosiddetti *blank*) *non* sono alfabetiche:

[31]:
```
'Barriera Corallina'.isalpha()
```

[31]:
```
False
```

… nè la punteggiatura:

[32]:
```
'!'.isalpha()
```

[32]:
```
False
```

… e neanche strani caratteri Unicode:

[33]:
```
'♥'.isalpha()
```

[33]:
```
False
```

[34]:
```
''.isalpha()
```

[34]:
```
False
```

4.9.10 Metodo isdigit

Il metodo `isdigit` ritorna `True` se una stringa è solo composta di cifre:

```
[35]: '391'.isdigit()
[35]: True
```

```
[36]: '400m'.isdigit()
[36]: False
```

Le notazioni a virgola mobile e scientifica non sono riconosciute:

```
[37]: '3.14'.isdigit()
[37]: False
```

```
[38]: '4e29'.isdigit()
[38]: False
```

4.9.11 Metodi `isupper` e `islower`

Possiamo controllare se un carattere è maiuscolo o minuscolo con i metodi `isupper` e `islower`:

```
[39]: 'q'.isupper()
[39]: False
```

```
[40]: 'Q'.isupper()
[40]: True
```

```
[41]: 'b'.islower()
[41]: True
```

```
[42]: 'B'.islower()
[42]: False
```

Funzionano anche con stringhe lunghe, controllando se tutti i caratteri soddisfano il criterio:

```
[43]: 'GREAT'.isupper()
[43]: True
```

```
[44]: 'NotSoGREAT'.isupper()
[44]: False
```

Nota che spaziatura in genere (i *blank*) e punteggiatura non sono prese in considerazione:

```
[45]: 'TROPPO\nGIUSTO !'.isupper()
[45]: True
```

Potremmo controllare se un carattere è maiuscolo/minuscolo guardando il codice ASCII ma il miglior modo per coprire tutti gli alfabeti è usare i metodi `isupper` e `islower`, per esempio funzionano anche con lettere accentate:

```
[46]: 'à'.isupper()
[46]: False
```

```
[47]: 'Á'.isupper()
[47]: True
```

4.9.12 Metodi di ricerca e trasformazione

Le stringhe mettono a disposizione diversi metodi per effettuare ricerche e trasformazioni al loro interno, ma attenzione: il potere è nulla senza il controllo! Spesso ti troverai con l'esigenza di usarli, e potrebbero anche funzionare con qualche piccolo esempio, ma molto spesso nascondono tranelli che poi si rimpiangono amaramente. Quindi quando scrivi codice con uno di questi metodi, **poniti sempre le domande che evidenzieremo!**

4.9.13 Metodo strip

Elimina spazi bianchi, tab e ritorni a capo dalle *estremità* della stringa. In genere, questo insieme di caratteri viene definito *blank*

NOTA: NON rimuove i *blank* tra le parole della stringa! Solo quelli all'estremità destra e sinistra.

```
[48]: x = ' \t\n\n\t ciao mondo \t  '   # alle estremità abbiamo messo spazi bianchi, tab e␣
      ↪ritorni a capo
```

```
[49]: x
[49]: ' \t\n\n\t ciao mondo \t  '
```

```
[50]: print(x)

          ciao mondo
```

```
[51]: len(x)   # ricorda che i caratteri speciali come \t e \n occupano 1 carattere
[51]: 20
```

```
[52]: y = x.strip()
```

```
[53]: y
[53]: 'ciao mondo'
```

```
[54]: print(y)
      ciao mondo
```

```
[55]: len(y)
```

[55]:
```
10
```

[56]:
```
x      # IMPORTANTE: x è ancora associato alla vecchia stringa !
```

[56]:
```
' \t\n\n\t ciao mondo \t  '
```

Specificare i caratteri da rimuovere

Se vuoi che Python si limiti a rimuovere solo qualche carattere specifico, puoi specificarli in parentesi. Proviamo a specificarne solo uno:

[57]:
```
'salsa'.strip('s')     #  nota che la `s` interna non è rimossa
```

[57]:
```
'alsa'
```

Se ne specifichiamo due o più, Python rimuove tutti i caratteri che può trovare dai lati

Nota che l'ordine in cui specifichi i caratteri **non** conta:

[58]:
```
'caustic'.strip('aci')
```

[58]:
```
'ust'
```

ATTENZIONE: Se specifici i caratteri, Python non proverà più a rimuovere i *blank!*

[59]:
```
'bouquet  '.strip('b')     # non rimuoverà gli spazi a destra!
```

[59]:
```
'ouquet  '
```

[60]:
```
'\tbouquet  '.strip('b')     # ... nè rimuoverà i blank a sinistra come il tab
```

[60]:
```
'\tbouquet  '
```

Secondo lo stesso principio, se specifichi uno spazio `' '`, allora Python rimuoverà **solo** gli spazi e non cercherà gli altri *blank* !!

[61]:
```
'  attento! \t'.strip(' ')    # rimuove solo dalla sinistra!
```

[61]:
```
'attento! \t'
```

DOMANDA: per ciascuna delle espressioni seguenti, prova ad indovinare che risultato produce (o se da errore):

1. `'\ttumultuoso\n'.strip()`
2. `' a b c '.strip()`
3. `'\ta\tb\t'.strip()`
4. `'\\tMmm'.strip()`
5. `'sky diving'.strip('sky')`

6.
```
'anacondas'.strip('sad')
```

7.
```
'\nno way '.strip(' ')
```

8.
```
'\nno way '.strip('\\n')
```

9.
```
'\nno way '.strip('\n')
```

10.
```
'salsa'.strip('as')
```

11.
```
'\t ACE '.strip('\t')
```

12.
```
' e allora? '.strip("")
```

13.
```
str(-3+1).strip("+"+"-")
```

4.9.14 Metodo lstrip

Elimina spazi bianchi, tab e ritorni a capo dall' *estremità sinistra* della stringa.

NOTA: NON rimuove i *blank* tra le parole della stringa! Solo quelli all'estremità sinistra.

```
[62]: x = '\n \t la strada \t '
```

```
[63]: x
[63]: '\n \t la strada \t '
```

```
[64]: len(x)
[64]: 16
```

```
[65]: y = x.lstrip()
```

```
[66]: y
[66]: 'la strada \t '
```

```
[67]: len(y)
[67]: 12
```

```
[68]: x        # IMPORTANTE: x è ancora associato alla vecchia stringa !
[68]: '\n \t la strada \t '
```

4.9.15 Metodo rstrip

Elimina spazi bianchi, tab e ritorni a capo dall' *estremità destra* della stringa.

NOTA: NON rimuove i *blank* tra le parole della stringa! Solo quelli all'estremità destra.

```
[69]: x = '\n \t il faro \t '
```

```
[70]: x
[70]: '\n \t il faro \t '
```

```
[71]: len(x)
[71]: 14
```

```
[72]: y = x.rstrip()
```

```
[73]: y
[73]: '\n \t il faro'
```

```
[74]: len(y)
[74]: 11
```

```
[75]: x        # IMPORTANTE: x è ancora associato alla vecchia stringa !
[75]: '\n \t il faro \t '
```

Esercizio - hatespace

Data una stringa `x` che può contenere dei *blank* (spazi, caratteri di controllo come `\t` e `\n`, ...) all'inizio e alla fine, scrivi del codice che stampa la stringa senza i *blank* e le stringhe `INIZIO` e `FINE` alle estremità.

Esempio - dato:

```
x = ' \t  \n \n hatespace\n   \t \n'
```

stampa:

```
INIZIOhatespaceFINE
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

```
[76]: # scrivi qui

x = ' \t  \n \n hatespace\n   \t \n'

print('INIZIO' + x.strip() + 'FINE')
INIZIOhatespaceFINE
```

[76]:
```
# scrivi qui
```

```
INIZIOhatespaceFINE
```

Esercizio - Bad to the bone

Hai una stringa in maiuscolo `s` che contiene ai lati dei caratteri che vuoi rimuovere: `punteggiatura`, un `carattere` minuscolo e dei *blank*. Scrivi del codice per eseguire la rimozione

Esempio - dati:

```
carattere = 'b'
punteggiatura = '!?.;,'
s = ' \t\n...bbbbbBAD TO THE BONE\n!'
```

il tuo codice dovrebbe mostrare:

```
'BAD TO THE BONE'
```

- usa solo i metodi `strip` (o `lstrip` e `rstrip`) (se necessario, puoi fare chiamate ripetute)

[77]:
```
carattere = 'b'
punteggiatura = '!?.;,'
s = ' \t\n...bbbbbBAD TO THE BONE\n!'

# write here
s.strip().strip(carattere + punteggiatura).strip()
```

[77]:
```
'BAD TO THE BONE'
```

4.9.16 Metodo count

Il metodo `count` prende una sottostringa e conta quante occorrenze vi sono nella stringa prima del punto.

[78]:
```
"raggi astrali".count('a')
```

[78]:
```
3
```

[79]:
```
"raggi astrali".count('A')     # distingue maiuscole/minuscole
```

[79]:
```
0
```

[80]:
```
"raggi astrali".count('st')
```

[80]:
```
1
```

Opzionalmente, puoi passare due altri parametri per denotare un indice da quale partire a contare (incluso) e dove terminare (escluso):

[81]:
```
#012345678901
"raggi astrali".count('a',4)
```

[81]:
```
2
```

[82]:
```
#0123456789012
"raggi astrali".count('a',4,9)
```

[82]:
```
1
```

Non abusare count

ATTENZIONE: `count` **viene spesso usato in modo errato / inefficiente**

Chiediti sempre:

1. la stringa su cui operare potrebbe contenere duplicati? Ricordati che verranno tutti conteggiati!
2. `count` compie una ricerca in tutta la stringa, che può essere inefficiente: è proprio necessaria, o sappiamo già l'intervallo in cui cercare?

Esercizio - denaro astrale

Durante il lockdown 2020, mentre guardavi le stelle nella notte, hai cominciato a sentire… onde. Rimuginandoci sopra, hai deciso che LORO vogliono comunicare con te, perciò hai montato un antenna satellitare sul tuo tetto per ricevere messaggi dagli alieni. Dopo mesi di rumori apparentemente irrilevanti, un giorno hai finalmente ricevuto un radio-messaggio che pensi di poter tradurre. Gli alieni *ovviamente* stanno cercando di dirti i numeri vincenti della lotto!

Un messaggio è una sequenza di esattamente 3 *diverse* ripetizioni di caratteri: il numero di caratteri in ogni ripetizione è un numero che proverai alla lotteria. Inizi freneticamente a sviluppare il traduttore per mostrare questi numeri fortunati sul terminale.

Esempio - data:

```
s = '$$$$€€€€€!!'
```

dovrebbe stampare:

```
$ € !
4 5 2
```

- **IMPORTANTE: puoi assumere che tutte le sequenze abbiano caratteri *differenti***
- **NON** usare cicli nè comprehension
- per semplicità assumi che ogni sequenza di caratteri abbia al più 9 ripetizioni

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[83]:
```
    #01234567890       # $ € !
s = '$$$$€€€€€!!'      # 4 5 2

                       # a b c
#s = 'aaabbbbbbccc'    # 3 6 3

                       # H A L
#s = 'HAL'             # 1 1 1
```

(continues on next page)

(continued from previous page)

```
# scrivi qui
p1 = 0
d1 = s.count(s[p1])
p2 = p1 + d1
d2 = s.count(s[p2])
p3 = p2 + d2
d3 = s.count(s[p3])

print(s[p1],s[p2],s[p3])
print(d1,d2,d3)
```

```
$ € !
4 5 2
```

[83]:
```
    #01234567890       # $ € !
s = '$$$$€€€€€!!'      # 4 5 2

                       # a b c
#s = 'aaabbbbbbccc'    # 3 6 3

                       # H A L
#s = 'HAL'             # 1 1 1

# scrivi qui
```

```
$ € !
4 5 2
```

4.9.17 Metodo find

`find` ritorna l'indice della *prima* occorrenza di qualche sottostringa data.

[84]:
```
#0123456789012345
'bingo bongo bong'.find('ong')
```

[84]:
```
7
```

Se nessuna occorrenza è trovata, ritorna `-1`:

[85]:
```
#0123456789012345
'bingo bongo bong'.find('bang')
```

[85]:
```
-1
```

[86]:
```
#0123456789012345
'bingo bongo bong'.find('Bong')     #  differenzia tra maiuscole/minuscole
```

[86]:
```
-1
```

Opzionalmente, puoi specificare un indice da dove iniziare a cercare (incluso):

[87]:
```
#0123456789012345
'bingo bongo bong'.find('ong',10)
```

[87]:
```
13
```

E anche dove finire (escluso):

[88]:
```
#0123456789012345
'bingo bongo bong'.find('g',4, 9)
```

[88]:
```
-1
```

Metodo rfind

Come il *metodo find*, ma la ricerca inizia dalla destra.

Non abusare find

ATTENZIONE: `find` **viene spesso usato in modo errato / inefficiente**

Chiediti sempre:

1. la stringa su cui operare potrebbe contenere duplicati? Ricordati che solo il *primo* verrà trovato!
2. `find` compie una ricerca in tutta la stringa, che può essere inefficiente: è proprio necessaria, o sappiamo già l'intervallo in cui cercare?
3. Se sappiamo che c'è un solo elemento da cercare, e che se c'è, è in una specifica posizione nota, `find` non serve a nulla!

Esercizio - Il porto di Monkey Island

L'isola di Monkey Island ha un porto che dispone di 4 moli a cui attraccano le navi provenienti da tutto l'arcipelago. Il punto di attracco non è mai preciso, e possono esserci degli spazi arbitrari con i bordi dei moli. Vi possono anche essere navi dal nome duplicato.

1) Supponiamo che ad ogni molo può contenere una sola nave, e vogliamo scrivere del codice che mostri `True` se la `"The Jolly Rasta"` è attraccata al molo `2` oppure `False` altrimenti.

Guarda i porti seguenti, e per ciascun cerca di indovinare se le righe di codice seguenti produrranno risultati corretti oppure no. Prova poi a scrivere del codice da te che non abbia i problemi che riscontrerai.

- **NON** istruzioni `if`, cicli nè comprehension
- **NON** usare liste (quindi niente split)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[89]:
```
larghezza = 21  # larghezza di un molo,  INCLUSO il `|` di destra
molo = 2


# moli     :  1                    2                    3                    4
porto =      "The Mad Monkey      |  The Jolly Rasta   |  The Sea Cucumber  |LeChuck
↪'s Ghost Ship|"
```

(continues on next page)

(continued from previous page)

```
#porto =     "  The Mad Monkey    |                    | The Sea Cucumber   |LeChuck
↪'s Ghost Ship|"
#porto =     "    The Mad Monkey  |The Jolly Rasta     |   The Sea Cucumber |        ␣
↪            |"
#porto =     "The Jolly Rasta     |                    |    The Sea Cucumber|LeChuck
↪'s Ghost Ship|"
#porto =     "                    | The Mad Monkey     |   The Jolly Rasta  |LeChuck
↪'s Ghost Ship|"
#porto =     "    The Jolly Rasta |                    | The Jolly Rasta    |   The␣
↪Jolly Rasta  |"

print('La Jolly Rasta è attraccata al molo', molo, '?')
print()
print(porto)

print()
print('                     in:', 'The Jolly Rasta' in porto)

print()
print('          find su tutto:', porto.find('The Jolly Rasta') != -1)

print()
print('  find dal secondo molo:', porto.find('The Jolly Rasta', larghezza*(molo-1)) !
↪= -1)

# scrivi qui
print()
sub = porto[larghezza*(molo-1):larghezza*molo-1]
print('              Soluzione:', sub.find('The Jolly Rasta') != -1)
```

```
La Jolly Rasta è attraccata al molo 2 ?

The Mad Monkey      |  The Jolly Rasta   |  The Sea Cucumber  |LeChuck's Ghost Ship|

                     in: True

          find su tutto: True

  find dal secondo molo: True

              Soluzione: True
```

[89]:
```
larghezza = 21  # larghezza di un molo,  INCLUSO il `|` di destra
molo = 2


# moli     :  1                    2                    3                    4
porto =      "The Mad Monkey      |  The Jolly Rasta   |  The Sea Cucumber  |LeChuck
↪'s Ghost Ship|"
#porto =     "  The Mad Monkey    |                    | The Sea Cucumber   |LeChuck
↪'s Ghost Ship|"
#porto =     "    The Mad Monkey  |The Jolly Rasta     |   The Sea Cucumber |        ␣
↪            |"
#porto =     "The Jolly Rasta     |                    |    The Sea Cucumber|LeChuck
↪'s Ghost Ship|"
```

(continues on next page)

(continued from previous page)

```
#porto =     "                    | The Mad Monkey     |   The Jolly Rasta  |LeChuck
↪'s Ghost Ship|"
#porto =     "    The Jolly Rasta |                    | The Jolly Rasta    |   The␣
↪Jolly Rasta  |"

print('La Jolly Rasta è attraccata al molo', molo, '?')
print()
print(porto)

print()
print('                     in:', 'The Jolly Rasta' in porto)

print()
print('          find su tutto:', porto.find('The Jolly Rasta') != -1)

print()
print('  find dal secondo molo:', porto.find('The Jolly Rasta', larghezza*(molo-1)) !
↪= -1)

# scrivi qui
```

```
La Jolly Rasta è attraccata al molo 2 ?

The Mad Monkey      |  The Jolly Rasta   |  The Sea Cucumber  |LeChuck's Ghost Ship|

                     in: True

          find su tutto: True

  find dal secondo molo: True

              Soluzione: True
```

2) Supponiamo ora che ogni molo possa accogliere più di una nave, anche con lo stesso nome. Scrivi del codice che mostri `True` se **una sola** Grog Ship è attraccata al secondo molo, `False` altrimenti.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[90]:
```
# moli     :  1                    2                    3                    4
porto =      "The Mad Monkey      |The Jolly Rasta     |  The Sea Cucumber  |LeChuck
↪'s Ghost Ship|"
#porto =     "The Mad Monkey      | Grog Ship Grog Ship| The Jolly Rasta    |   The␣
↪Sea Cucumber "
#porto =     "   The Jolly Rasta  |   Grog Ship        | The Jolly Rasta    |   The␣
↪Jolly Rasta  "
#porto =     "   Grog Ship        |   Grog Ship        |LeChuck's Ghost Ship|    Grog␣
↪Ship        "
#porto =     "LeChuck's Ghost Ship|                    |   Grog Ship        |   The␣
↪Jolly Rasta  "
#porto =     "The Jolly Rasta     | Grog Ship Grog Ship|       Grog Ship    |   The␣
↪Jolly Rasta  "

print()
```

(continues on next page)

(continued from previous page)

```
print('Una sola Grog Ship è attraccata al molo', molo, '?')
print()

# scrivi qui

sub = porto[larghezza*(molo-1):larghezza*molo-1]
print('Soluzione Grog Ship:', sub.count('Grog Ship') == 1)
```

```
Una sola Grog Ship è attraccata al molo 2 ?

Soluzione Grog Ship: False
```

[90]:
```
# moli     :  1                    2                    3                    4
porto =      "The Mad Monkey      |The Jolly Rasta     |  The Sea Cucumber  |LeChuck
↪'s Ghost Ship|"
#porto =     "The Mad Monkey      | Grog Ship Grog Ship| The Jolly Rasta    |   The␣
↪Sea Cucumber "
#porto =     "   The Jolly Rasta  |   Grog Ship        | The Jolly Rasta    |   The␣
↪Jolly Rasta  "
#porto =     "   Grog Ship        |   Grog Ship        |LeChuck's Ghost Ship|    Grog␣
↪Ship        "
#porto =     "LeChuck's Ghost Ship|                    |   Grog Ship        |   The␣
↪Jolly Rasta  "
#porto =     "The Jolly Rasta     | Grog Ship Grog Ship|       Grog Ship    |   The␣
↪Jolly Rasta  "

print()
print('Una sola Grog Ship è attraccata al molo', molo, '?')
print()

# scrivi qui
```

```
Una sola Grog Ship è attraccata al molo 2 ?

Soluzione Grog Ship: False
```

Esercizio - banane

Durante l'esplorazione di una regione tropicale remota, un etologo scopre una popolazione di scimmie che sembra abbiano qualche concetto dei numeri. Collezionano centinaia di banane che sono poi scambiate con noci di cocco collezionate da un'altro gruppo. Per comunicare quantità fino a 999 banane, usano una serie di esattamente tre suoni gutturali. L'etologo scrive le sequenze e formula la teoria seguente: ogni suono è rappresentato da una sequenza dello stesso carattere, ripetuto un certo numero di volte. Il numero di caratteri nella prima sequenza è la prima cifra (le centinaia), il numero di caratteri nella seconda sequenza è la seconda cifra (le decine), mentre l'ultima sequenza rappresenta le unità.

Scrivi del codice che mette nella variabile `banane` **un intero** che rappresenti il numero.

Per esempio - dati:

```
s = 'bb bbbbb aaaa'
```

il tuo codice dovrebbe stampare:

```
>>> banane
254
>>> type(banane)
int
```

- **IMPORTANTE 1: sequenze differenti possono usare lo *stesso* carattere!**
- **IMPORTANTE 2: non puoi assumere quali caratteri le scimmie useranno**: conosci solo che ogni cifra è rappresentata da una ripetizione dello stesso carattere
- **NON** usare cicli nè comprehensions
- le scimmie non hanno cognizione dello zero

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[91]:
```
    #0123456789012
s = 'bb bbbbb aaaa'      # 254
#s = 'ccc cc ccc'        # 323
#s = 'vvv rrrr ww'       # 342
#s = 'cccc h jjj'        # 413
#s = '□□□ □□□□□□ □□□□'   # 364   (potresti ottenere *qualunque* strano carattere,␣
↪anche unicode ...)

# scrivi qui
p1 = s.find(' ')
banane = len(s[:p1])*100
p2 = s.find(' ',p1+1)
banane += len(s[p1+1:p2])*10
banane += len(s[p2+1:])*1

print('Le banane sono',banane)
type(banane)
```

```
Le banane sono 254
```

[91]:
```
int
```

[91]:
```
    #0123456789012
s = 'bb bbbbb aaaa'      # 254
#s = 'ccc cc ccc'        # 323
#s = 'vvv rrrr ww'       # 342
#s = 'cccc h jjj'        # 413
#s = '□□□ □□□□□□ □□□□'   # 364   (potresti ottenere *qualunque* strano carattere,␣
↪anche unicode ...)

# scrivi qui
```

```
Le banane sono 254
```

[91]:
```
int
```

4.9.18 Metodo replace

`str.replace` prende due stringhe, e restituisce una NUOVA stringa ricavata sostituendo nella stringa su cui è chiamata *tutte* le occorrenze del primo parametero stringa con il secondo.

Esempio:

```
[92]: "il treno percorre".replace('re', 'RO')
[92]: 'il tROno percorRO'
```

```
[93]: "alberello bello".replace('llo', 'llini')
[93]: 'alberellini bellini'
```

```
[94]: "parlare e brindare".replace('ARE', 'iamo')  # se cerchiamo maiuscole qua non le␣
      ↪troveremo
[94]: 'parlare e brindare'
```

```
[95]: "PARLARE E BRINDARE".replace('ARE', 'IAMO')  # ma qua sì
[95]: 'PARLIAMO E BRINDIAMO'
```

Come sempre per le stringhe, `replace` NON modifica la stringa su cui è chiamato:

```
[96]: x = "sulla panca"
```

```
[97]: y = x.replace('panca', 'panca la capra campa')
```

```
[98]: y
[98]: 'sulla panca la capra campa'
```

```
[99]: x  # IMPORTANTE: x è ancora associato alla vecchia stringa !
[99]: 'sulla panca'
```

E' possile specificare fino a quante occorrenze verranno sostituite:

```
[100]: 'apollo fece una palla di pelle di pollo'.replace('ll', 'Zz', 3)
[100]: 'apoZzo fece una paZza di peZze di pollo'
```

Non abusare replace

ATTENZIONE: `replace` **viene spesso usato in modo errato / inefficiente**

Chiediti sempre:

1. la stringa su cui operare potrebbe contenere duplicati? Ricordati che verranno *tutti* sostituiti!
2. `replace` compie una ricerca in tutta la stringa, che può essere inefficiente: è proprio necessaria, o sappiamo già la posizione dove è presente il testo da sostituire?

Esercizio - Non aprite quella porta

DOMANDA Hai una biblioteca di libri, con etichette come `G-Il padrino`, `R-I promessi sposi` o `'H-Non aprite quella porta'` composte da una un carattere che identifica il tipo (`G` giallo, `R` romanzo, `H` horror) seguito da `-` e dal titolo. Dato un `libro`, vuoi stampare l'etichetta completa, due punti e poi il titolo, come `'Giallo: Il padrino'`. Guarda i seguenti frammenti di codice, e per ciascuno prova a scrivere delle etichette tra quelle proposte **o creane altre** che darebbero risultati errati (se esistono)

```
libro = 'G-Il padrino'
libro = 'R-I promessi sposi'
libro = 'H-Non aprite quella porta'
```

1. ```
   libro.replace('G', 'Giallo: ').replace('R', 'Romanzo: ')
   ```
2. ```
   libro[0].replace('G', 'Giallo: ')  \
           .replace('H', 'HORROR: ')  \
           .replace('R', 'Romanzo: ') + libro[2:]
   ```
3. ```
   libro.replace('G-', 'Giallo: ').replace('R-', 'Romanzo: ')
   ```
4. ```
   libro.replace('G-', 'Giallo: ',1).replace('R-', 'Romanzo: ',1)
   ```
5. ```
   libro[0:2].replace('G-', 'Giallo: ').replace('R-', 'Romanzo: ') + libro[2:]
   ```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a><div class="jupman-sol jupman-sol-code" style="display:none">

[109]:
```
# SOLUZIONE



#1
libro = 'R-Guerra e Pace'
print(libro.replace('G', 'Giallo: ').replace('R', 'Romanzo: '))

#2
libro = 'H-Non aprite quella porta'
print(libro[0].replace('G', 'Giallo: ').replace('H', 'HORROR: ').replace('R',
↪'Romanzo: ') + libro[2:])

#3
libro = 'R-G-Force'
print(libro.replace('G-', 'Giallo: ').replace('R-', 'Romanzo: '))

#4
libro = 'G-L-A-R-A-P-I-N-A'
print(libro.replace('G-', 'Giallo: ',1).replace('R-', 'Romanzo: ',1))

#5 questo è abbastanza blindato, SE assumiamo che le categorie siano fisse e NON␣
↪contengano trattini
# per es una categoria malefica potrebbe essere C- espanso a C-R-O-S-T-A-C-E-I  (che␣
↪conterrebbe R-)
#print(libro[0:2].replace('G-', 'Giallo: ').replace('R-', 'Romanzo: ').replace('H-',
↪'Horror: ') + libro[2:])
```
```

```
Romanzo: -Giallo: uerra e Pace
HORomanzo: Romanzo: ORomanzo: : Non aprite quella porta
Romanzo: Giallo: Force
Giallo: L-A-Romanzo: A-P-I-N-A
```

[109]:

```

```

Esercizio - Il Regno degli Stringardi

La landa dei Caratterinzi è dominata col pugno di ferro dal Ducato degli Stringardi. I borghi comandati degli Stringardi si distinguono per essere monodimensionali e possono essere rappresentati come una stringa: offrono ospitalità ai duchi `'d'`, ai signori `'s'`, ai vassalli `'v'`, e alla plebe `'p'`. Al fine di separare le varie cerchie sociali da disdicevoli commistioni sono state erette delle mura `|mm|`. Purtroppo, il Ducato è sotto attacco dalla tribù dei beceri Replaceri, che con loro tecniche barbare e poco avvedute rischiano di radere al suolo la città. Per difendersi, gli Stringardi decidono di innalzare le mura, trasformandole da `|mm|` a `|MM|`.

- **NON** usare cicli nè list comprehension
- **NON** usare liste (quindi niente split)

Stringardi I: Innalza tutte le mura

Esempio - dati:

```
borgo = 'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
```

dopo il tuo codice, deve risultare:

```
>>> borgo
'ppp|MM|vvvvvv|MM|sss|MM|dd|MM|sssss|MM|vvvvvv|MM|pppppp'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[102]:

```
borgo =       'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
# risultato: 'ppp|MM|vvvvvv|MM|sss|MM|dd|MM|sssss|MM|vvvvvv|MM|pppppp'

# scrivi qui

borgo = borgo.replace('|mm|','|MM|')
print(borgo)
```

```
ppp|MM|vvvvvv|MM|sss|MM|dd|MM|sssss|MM|vvvvvv|MM|pppppp
```

[102]:

```
borgo =       'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
# risultato: 'ppp|MM|vvvvvv|MM|sss|MM|dd|MM|sssss|MM|vvvvvv|MM|pppppp'

# scrivi qui
```

(continues on next page)

(continued from previous page)

Stringardi II: Mura esterne

Purtroppo la plebaglia non lavora abbastanza e non vi sono fondi per innalzare tutte le mura: innalza **solo le mura all'esterno**

- **NON** usare `if`, cicli nè list comprehension
- **NON** usare liste (quindi niente split)

Esempio - dati:

```
borgo = 'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
```

dopo il tuo codice, deve risultare:

```
>>> borgo
'ppp|MM|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|MM|pppppp'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[103]:

```
borgo =       'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
#risultato:   'ppp|MM|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|MM|pppppp'
#borgo =      '|mm|vvvvvv|mm||mm|ddddd|mm|ssvvv|mm|pp'
#risultato:   '|MM|vvvvvv|mm||mm|ddddd|mm|ssvvv|MM|pp'


# scrivi qui

i = borgo.find('|mm|')
borgo = borgo[:i] + '|MM|' + borgo[i+4:]
i = borgo.rfind('|mm|')
borgo = borgo[:i] + '|MM|' + borgo[i+4:]

print(borgo)
```

```
ppp|MM|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|MM|pppppp
```

[103]:

```
borgo =       'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
#risultato:   'ppp|MM|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|MM|pppppp'
#borgo =      '|mm|vvvvvv|mm||mm|ddddd|mm|ssvvv|mm|pp'
#risultato:   '|MM|vvvvvv|mm||mm|ddddd|mm|ssvvv|MM|pp'


# scrivi qui
```

Stringardi III: Potere al popolo

Una minaccia ancora più grande affligge gli Stringardi: *la democrazia*. A seguito del dilagare di questo male oscuro, in alcune città si sono sviluppate fazioni di destra e sinistra, che tendono a privilegiare solo alcune parti della città. Se il sentimento dominante di una città è di sinistra, tutte le abitazioni a sinistra del duca sono privilegiate a suon di dobloni, altrimenti se di destra sono privilegiate quelle a destra del duca. Quando una casa è privilegiata la lettera corrispondente viene resa in maiuscolo.

- assumi che almeno un blocco con `d` sia sempre presente, e che sia unico
- **NON** usare `if`, cicli nè list comprehension
- **NON** usare liste (quindi niente split)

3.1) privilegia solo le case di sinistra

```
borgo = 'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
```

dopo il tuo codice, deve risultare:

```
>>> borgo
'PPP|mm|VVVVVV|mm|SSS|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[104]:
```
borgo =      'PPP|mm|VVVVVV|mm|SSS|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
# risultato: 'PPP|mm|VVVVVV|mm|SSS|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
#borgo =     '|p|ppp||p|pp|mm|vvv|vvvv|mm|sssss|mm|ddd|mm|ssss|ss|mm|vvvvvv|mm|'
# risultato: '|P|PPP||P|PP|mm|VVV|VVVV|mm|SSSSS|mm|ddd|mm|ssss|ss|mm|vvvvvv|mm|'

# scrivi qui

dpos = borgo.find('d')
borgo = borgo[:dpos].replace('p','P').replace('v','V').replace('s','S') + borgo[dpos:]

borgo
```

[104]:
```
'PPP|mm|VVVVVV|mm|SSS|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
```

[104]:
```
borgo =      'PPP|mm|VVVVVV|mm|SSS|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
# risultato: 'PPP|mm|VVVVVV|mm|SSS|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
#borgo =     '|p|ppp||p|pp|mm|vvv|vvvv|mm|sssss|mm|ddd|mm|ssss|ss|mm|vvvvvv|mm|'
# risultato: '|P|PPP||P|PP|mm|VVV|VVVV|mm|SSSSS|mm|ddd|mm|ssss|ss|mm|vvvvvv|mm|'

# scrivi qui
```

3.2) privilegia solo le case di destra

Esempio - dati:

```
borgo = 'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
```

dopo il tuo codice, deve risultare:

```
>>> borgo
'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|SSSSS|mm|VVVVVV|mm|PPPPPP'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[105]:
```
borgo =      'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
#risultato: 'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|SSSSS|mm|VVVVVV|mm|PPPPPP'
#borgo =     '|p|ppp||p|pp|mm|vvv|vvvv|mm|sssss|mm|ddd|mm|ssss|ss|mm|vvvvvv|p|pp|mm|'
#risultato: '|p|ppp||p|pp|mm|vvv|vvvv|mm|sssss|mm|ddd|mm|SSSS|SS|mm|VVVVVV|P|PP|mm|'

# scrivi qui
dpos = borgo.rfind('d')
borgo = borgo[:dpos] + borgo[dpos:].replace('p','P').replace('v','V').replace('s','S')

borgo
```

[105]:
```
'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|SSSSS|mm|VVVVVV|mm|PPPPPP'
```

[105]:
```
borgo =      'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|sssss|mm|vvvvvv|mm|pppppp'
#risultato: 'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|SSSSS|mm|VVVVVV|mm|PPPPPP'
#borgo =     '|p|ppp||p|pp|mm|vvv|vvvv|mm|sssss|mm|ddd|mm|ssss|ss|mm|vvvvvv|p|pp|mm|'
#risultato: '|p|ppp||p|pp|mm|vvv|vvvv|mm|sssss|mm|ddd|mm|SSSS|SS|mm|VVVVVV|P|PP|mm|'

# scrivi qui
```

Stringardi IV: Lotte di potere

Nel tempo la famiglia ducale si è espansa e purtroppo sono nate faide spietate. A seconda della popolosità del borgo di sinistra o destra, un corrispondonete numero dei membri della famiglia più a sinistra nel palazzo ducale o più a destra ricevono supporto per attuare i loro giochi di potere. Un membro del palazzo ducale che riceve supporto diventa in maiuscolo. Ogni lettera `'p'`, `'v'` o `'s'` contribuisce supporto (ma non i muri). I membri della famiglia ducale che non sono raggiunti da nessun supporto vengono trucidati dai parenti, e sostituiti con una Latin Cross Unicode[142] ✝

- assumi che almeno un blocco con `d` sia sempre presente, e che sia unico
- assumi che per ogni casa di sinistra/destra, vi sia *almeno un* duca di sinistra/destra
- **NON** usare `if`, cicli nè list comprehension
- **NON** usare liste (quindi niente split)

Esempio - dato:

```
borgo = 'ppp|mm|vv|mm|v|s|mm|dddddddddddddddddddddddd|mm|ss|mm|vvvvv|mm|pppp'
```

Dopo il tuo codice, deve stampare:

[142] https://www.compart.com/en/unicode/U+271D

```
Membri della famiglia reale: 24
                di sinistra: 7
                  di destra: 11

Dopo la lotta fratricida, il nuovo borgo è:

ppp|mm|vv|mm|v|s|mm|DDDDDDD??????DDDDDDDDDDD|mm|ss|mm|vvvvv|mm|pppp
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[106]:
```
borgo =     'ppp|mm|vv|mm|v|s|mm|dddddddddddddddddddddddd|mm|ss|mm|vvvvv|mm|pppp'
#risultato: 'ppp|mm|vv|mm|v|s|mm|DDDDDDD??????DDDDDDDDDDD|mm|ss|mm|vvvvv|mm|pppp' ␣
↪tot:24 sx:7 dx:11
#borgo =    'ppp|mm|ppp|mm|vv|mm|ss|mm|dddddddddddddddddddd|mm|ss|mm|mm|s|v|mm|p|p|'
#risultato: 'ppp|mm|ppp|mm|vv|mm|ss|mm|DDDDDDDDDD????DDDDDD|mm|ss|mm|mm|s|v|mm|p|p|'␣
↪tot:20 sx:10 dx:6

# scrivi qui

d = borgo.count('d')

print('Membri della famiglia reale:', d)

dpos_sx = borgo.find('d')
c_sx = borgo.count('p', 0, dpos_sx) + borgo.count('v', 0, dpos_sx) + borgo.count('s',␣
↪0, dpos_sx)
print('                di sinistra:', c_sx)


dpos_dx = borgo.rfind('d')
c_dx = borgo.count('p', dpos_dx) + borgo.count('v', dpos_dx) + borgo.count('s', dpos_
↪dx)
print('                  di destra:', c_dx)

borgo = borgo[:dpos_sx] + 'D'*c_sx + '?'*(d-c_sx-c_dx) + 'D'*c_dx + borgo[dpos_dx+1:]

print()
print('Dopo la lotta fratricida, il nuovo borgo è:')
print()
print(borgo)
```

```
Membri della famiglia reale: 24
                di sinistra: 7
                  di destra: 11

Dopo la lotta fratricida, il nuovo borgo è:

ppp|mm|vv|mm|v|s|mm|DDDDDDD??????DDDDDDDDDDD|mm|ss|mm|vvvvv|mm|pppp
```

[106]:
```
borgo =     'ppp|mm|vv|mm|v|s|mm|dddddddddddddddddddddddd|mm|ss|mm|vvvvv|mm|pppp'
#risultato: 'ppp|mm|vv|mm|v|s|mm|DDDDDDD??????DDDDDDDDDDD|mm|ss|mm|vvvvv|mm|pppp' ␣
↪tot:24 sx:7 dx:11
#borgo =    'ppp|mm|ppp|mm|vv|mm|ss|mm|dddddddddddddddddddd|mm|ss|mm|mm|s|v|mm|p|p|'
#risultato: 'ppp|mm|ppp|mm|vv|mm|ss|mm|DDDDDDDDDD????DDDDDD|mm|ss|mm|mm|s|v|mm|p|p|'␣
↪tot:20 sx:10 dx:6
```

(continues on next page)

(continued from previous page)

```
# scrivi qui
```

4.9.19 Altri esercizi

DOMANDA: Per ciascuna espressione di seguito, cerca di indovinare il risultato.

1. `'gUrP'.lower() == 'GuRp'.lower()`
2. `'NaNo'.lower() != 'nAnO'.upper()`
3. `'O' + 'ortaggio'.replace('o','\t \n     ').strip() + 'O'`
4. `'DaDo'.replace('D','b') in 'barbados'`

4.9.20 Prosegui

Trovi ulteriori esercizi nel foglio Stringhe 4[143]

4.10 Stringhe 4 - iterazione e funzioni

4.10.1 Scarica zip esercizi

Naviga file online[144]

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
strings
    strings1.ipynb
    strings1-sol.ipynb
    strings2.ipynb
    strings2-sol.ipynb
    strings3.ipynb
    strings3-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `strings1.ipynb`

[143] https://it.softpython.org/strings/strings4-sol.html
[144] https://github.com/DavidLeoni/softpython-it/tree/master/strings

- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.10.2 Esercizi con le funzioni

ATTENZIONE: Gli esercizi seguenti richiedono di conoscere:

Tutti gli esercizi sulle stringhe precedenti

Controllo di flusso[145]

Funzioni[146]

Se sei alle prime armi con la programmazione, ti conviene saltarli e ripassare in seguito

lung

⊛ a. Scrivi una funzione `lung1(stringa)` in cui data una stringa, RITORNI quanto è lunga la stringa. Usa `len` Per esempio, con la stringa `"ciao"`, la vostra funzione dovrebbe ritornare `4` mentre con `"hi"` dovrebbe ritornare `2`

```
>>> x = lung1("ciao")
>>> x
4
```

⊛ b. Scrivi una funzione `lung2` che come prima calcola la lunghezza della stringa, ma SENZA usare `len` (usa un ciclo `for`)

```
>>> y = lung2("mondo")
>>> y
5
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[2]:
```
# scrivi qui

# versione con len, più veloce perchè python assieme ad una stringa mantiene sempre␣
↪in memoria
# il numero della lunghezza immediatamente disponibile

def lung1(stringa):
    return len(stringa)
```

(continues on next page)

[145] https://it.softpython.org/#control-flow
[146] https://it.softpython.org/functions/functions-sol.html

(continued from previous page)

```
# versione con contatore, più lenta
def lung2(parola):
    contatore = 0
    for lettera in parola:
        contatore = contatore + 1
    return contatore
```

[2]:
```
# scrivi qui
```

contin

⊛ Scrivi la funzione `contin(parola, stringa)`, che RITORNA `True` se `parola` contiene la `stringa` indicata, altrimenti RITORNA `False`

- Usa l'operatore `in`

```
>>> x = contin('carpenteria', 'ent')
>>> x
True
>>> y = contin('carpenteria', 'zent')
>>> y
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[3]:
```
# scrivi qui

def contin(parola, stringa):
    return stringa in parola
```

[3]:
```
# scrivi qui
```

invertilet

⊛ Scrivi la funzione `invertilet(primo, secondo)` che prende in input due stringhe di lunghezza maggiore di 3, e RESTITUISCE una nuova stringa in cui le parole sono concatenate e separate da uno spazio, le ultime lettere delle due parole sono invertite. Questo significa che passando in input 'ciao' e 'world', la funzione dovrebbe restituire 'ciad worlo'

Se le stringhe non sono di lunghezza adeguata, il programma STAMPA *errore!*

SUGGERIMENTO: usa *le slice*

```
>>> x = invertilet('hi','mondo')
'errore!'
>>> x
None
>>> x = invertilet('cirippo', 'bla')
'errore!'
>>> x
None
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[4]:
```
# scrivi qui

def invertilet(primo,secondo):
    if len(primo) <= 3 or len(secondo) <=3:
        print("errore!")
    else:
        return primo[:-1] + secondo[-1] + " " + secondo[:-1] + primo[-1]
```

[4]:
```
# scrivi qui
```

nspazio

⊛ Scrivi la funzione `nspazio` che data una stringa in input, RITORNA una nuova stringa in cui l'n-esimo carattere è uno spazio. Per esempio, data la stringa 'largamente' e il carattere all'indice 5, il programma deve RITORNARE `'larga ente'`. Nota: se il numero dovesse essere troppo grande (i.e., la parola ha 6 caratteri e chiedo di rimuovere il numero 9), il programma STAMPA *errore!*

```
>>> x = nspazio('largamente', 5)
>>> x
'larga ente'

>>> x = nspazio('ciao', 9)
errore!
>>> x
None

>>> x = nspazio('ciao', 4)
errore!
>>> x
None
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[5]:
```
# scrivi qui
```

(continues on next page)

(continued from previous page)

```
def nspazio(parola, indice):
    if indice >= len(parola):
        print("errore!")
    return parola[:indice]+' '+parola[indice+1:]

#nspazio("largamente", 5)
```

[5]:
```
# scrivi qui
```

inifin

⊛ Scrivi un programma in Python prende una stringa, e se la stringa ha una lunghezza maggiore di 4, il programma STAMPA le prime e le ultime due lettere. Altrimenti, nel caso in cui la lunghezza della stringa sia minore di 4, STAMPA `voglio almeno 4 caratteri`. Per esempio, passando alla funzione `"ciaomondo"`, la funzione dovrebbe stampare `"cido"`. Passando `"ciao"` dovrebbe stampare `ciao` e passando `"hi"` dovrebbe stampare `voglio almeno 4 caratteri`

```
>>> inifin('ciaomondo')
cido

>>> inifin('hi')
Voglio almeno 4 caratteri
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[6]:
```
# scrivi qui

def inifin(stringa):
    if len(stringa) >= 4:
        print(stringa[:2] + stringa[-2:])
    else:
        print("Voglio almeno 4 caratteri")
```

[6]:
```
# scrivi qui
```

scambia

Scrivere una funzione che data una stringa, inverte il primo e l'ultimo carattere, e STAMPA il risultato.

Quindi, data la stringa "mondo", il programma stamperà "oondm"

```
>>> scambia('mondo')
oondm
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[7]:
```
# scrivi qui

def scambia(stringa):
    print(stringa[-1] + stringa[1:-1] + stringa[0])
```

[7]:
```
# scrivi qui
```

4.10.3 Esercizi con eccezioni e test

ATTENZIONE: Gli esercizi seguenti richiedono di conoscere:

Controllo di flusso[147]

Funzioni[148]

e anche: Eccezioni e test con assert[149]

halet

⊛ RITORNA `True` se parola contiene lettera, `False` altrimenti

- usare ciclo `while`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[8]:
```
def halet(parola, lettera):

    indice = 0      # inizializziamo indice
    while indice < len(parola):
        if parola[indice] == lettera:
            return True    # abbiamo trovato il carattere, possiamo interrompere la␣
↪ricerca
        indice += 1    # è come scrivere indice = indice + 1
```

(continues on next page)

[147] https://it.softpython.org/#control-flow
[148] https://it.softpython.org/functions/functions-sol.html
[149] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
    # se arriviamo DOPO il while, c'è una sola ragione:
    # non abbiamo trovato nulla, quindi dobbiamo ritornare False
    return False



# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert halet("ciao", 'a')
assert not halet("ciao", 'A')
assert halet("ciao", 'c')
assert not halet("", 'a')
assert not halet("ciao", 'z')
# FINE TEST
```

[8]:
```
def halet(parola, lettera):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert halet("ciao", 'a')
assert not halet("ciao", 'A')
assert halet("ciao", 'c')
assert not halet("", 'a')
assert not halet("ciao", 'z')
# FINE TEST
```

conta

⊛ RITORNA il numero di occorrenze di lettera in parola

NOTA: NON VOGLIO UNA STAMPA, VOGLIO CHE *RITORNI* IL VALORE!

- Usare ciclo for in

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[9]:
```
def conta(parola, lettera):

    occorrenze = 0
    for carattere in parola:
        # print("carattere corrente = ", carattere)     # le print di debug sono␣
↪ammesse
        if carattere == lettera:
            # print("trovata occorrenza !")     # le print di debug sono ammesse
            occorrenze += 1
```

(continues on next page)

(continued from previous page)

```
    return occorrenze     # L'IMPORTANTE E' _RITORANRE_ IL VALORE COME DA TESTO DELL
↪'ESERCIZIO !!!!!


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert conta("ciao", "z") == 0
assert conta("ciao", "c") == 1
assert conta("babbo", "b") == 3
assert conta("", "b") == 0
assert conta("ciao", "C") == 0
# FINE TEST
```

[9]:

```
def conta(parola, lettera):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert conta("ciao", "z") == 0
assert conta("ciao", "c") == 1
assert conta("babbo", "b") == 3
assert conta("", "b") == 0
assert conta("ciao", "C") == 0
# FINE TEST
```

contiene_minuscola

⊛ Esercizio ripreso dall' Esercizio 4 del libro Pensare in Python Capitolo Stringhe leggere in fondo qua: https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2009.html

- RITORNA True se la parola contiene almeno una lettera minuscola
- RITORNA False altrimenti
- Usare ciclo `while`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[10]:

```
def contiene_minuscola(s):

    i = 0
    while i < len(s):
        if s[i] == s[i].lower():
            return True
        i += 1
    return False
```

(continues on next page)

(continued from previous page)

```
# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert contiene_minuscola("David")
assert contiene_minuscola("daviD")
assert not contiene_minuscola("DAVID")
assert not contiene_minuscola("")
assert contiene_minuscola("a")
assert not contiene_minuscola("A")
```

[10]:
```
def contiene_minuscola(s):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert contiene_minuscola("David")
assert contiene_minuscola("daviD")
assert not contiene_minuscola("DAVID")
assert not contiene_minuscola("")
assert contiene_minuscola("a")
assert not contiene_minuscola("A")
```

dialetto

⊛⊛ Esiste un dialetto in cui tutte le "a" devono per forza essere precedute da una "g". Nel caso una parola dovesse contenere "a" *non* preceduta da una "g", possiamo dire con certezza che questa parola non fa parte di quel dialetto. Scrivere una funzione che data una parola, RITORNI `True` se la parola rispetta le regole del dialetto, `False` altrimenti.

```
>>> dialetto("ammot")
False
>>> print(dialetto("paganog")
False
>>> print(dialetto("pgaganog")
True
>>> print(dialetto("ciao")
False
>>> dialetto("cigao")
True
>>> dialetto("zogava")
False
>>> dialetto("zogavga")
True
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a><div class="jupman-sol jupman-sol-code"

style="display:none">

[11]:

```
def dialetto(parola):

    n = 0
    for i in range(0,len(parola)):
        if parola[i] == "a":
            if i == 0 or parola[i - 1] != "g":
                return False
    return True


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert dialetto("a") == False
assert dialetto("ab") == False
assert dialetto("ag") == False
assert dialetto("ag") == False
assert dialetto("ga") == True
assert dialetto("gga") == True
assert dialetto("gag") == True
assert dialetto("gaa") == False
assert dialetto("gaga") == True
assert dialetto("gabga") == True
assert dialetto("gabgac") == True
assert dialetto("gabbgac") == True
assert dialetto("gabbgagag") == True
# FINE TEST
```

[11]:

```
def dialetto(parola):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert dialetto("a") == False
assert dialetto("ab") == False
assert dialetto("ag") == False
assert dialetto("ag") == False
assert dialetto("ga") == True
assert dialetto("gga") == True
assert dialetto("gag") == True
assert dialetto("gaa") == False
assert dialetto("gaga") == True
assert dialetto("gabga") == True
assert dialetto("gabgac") == True
assert dialetto("gabbgac") == True
assert dialetto("gabbgagag") == True
# FINE TEST
```

contavoc

⊛⊛ Data una stringa, scrivere una funzione che conti il numero di vocali. Se il numero di vocali è pari RITORNA il numero di vocali, altrimenti solleva l'eccezione `ValueError`

```
>>> conta_vocali("asso")
2
>>> conta_vocali("ciao")
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-058310342431> in <module>()
     16 contavoc("arco")
---> 19 contavoc("ciao")

ValueError: Vocali dispari !
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
def contavoc(parola):

    n_vocali = 0

    vocali = ["a","e","i","o","u"]

    for lettera in parola:
        if lettera.lower() in vocali:
            n_vocali = n_vocali + 1

    if n_vocali % 2 == 0:
        return n_vocali
    else:
        raise ValueError("Vocali dispari !")


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert contavoc("arco") == 2
assert contavoc("scaturire") == 4

try:
    contavoc("ciao")     # con questa stringa ci attendiamo che sollevi l'eccezione␣
↪ValueError
    raise Exception("Non dovrei arrivare fin qui !")
except ValueError:       # se solleva l'eccezione ValueError,si sta comportando come␣
↪previsto e non facciamo niente
    pass

try:
    contavoc("aiuola")   # con questa stringa ci attendiamo che sollevi l'eccezione␣
↪ValueError
    raise Exception("Non dovrei arrivare fin qui !")
except ValueError:       # se solleva l'eccezione ValueError,si sta comportando come␣
↪previsto e non facciamo niente
```

(continues on next page)

(continued from previous page)

```
    pass
```

[12]:

```
def contavoc(parola):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert contavoc("arco") == 2
assert contavoc("scaturire") == 4

try:
    contavoc("ciao")     # con questa stringa ci attendiamo che sollevi l'eccezione␣
↪ValueError
    raise Exception("Non dovrei arrivare fin qui !")
except ValueError:       # se solleva l'eccezione ValueError,si sta comportando come␣
↪previsto e non facciamo niente
    pass

try:
    contavoc("aiuola")   # con questa stringa ci attendiamo che sollevi l'eccezione␣
↪ValueError
    raise Exception("Non dovrei arrivare fin qui !")
except ValueError:       # se solleva l'eccezione ValueError,si sta comportando come␣
↪previsto e non facciamo niente
    pass
```

palindroma

⊛⊛⊛ Una parola è palindroma quando è esattamente la stessa se letta al contrario

Scrivi una funzione che RITORNA `True` se una parola è palindroma, `False` altrimenti

- assumi che la stringa vuota sia palindroma

Esempio:

```
>>> x = palindroma('radar')
>>> x
True
>>> x = palindroma('scatola')
>>> x
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[13]:
```
def palindroma(parola):

    for i in range(len(parola) // 2):
        if parola[i] != parola[len(parola)- i - 1]:
            return False

    return True   # nota che è FUORI dal for: superati tutti i controlli,
                  # possiamo concludere che la parola è palindroma


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert palindroma('') == True    # assumiamo che la stringa vuota sia palindroma
assert palindroma('a') == True
assert palindroma('aa') == True
assert palindroma('ab') == False
assert palindroma('aba') == True
assert palindroma('bab') == True
assert palindroma('bba') == False
assert palindroma('abb') == False
assert palindroma('abba') == True
assert palindroma('baab') == True
assert palindroma('abbb') == False
assert palindroma('bbba') == False
assert palindroma('radar') == True
assert palindroma('scatola') == False

# FINE TEST
```

[13]:
```
def palindroma(parola):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert palindroma('') == True    # assumiamo che la stringa vuota sia palindroma
assert palindroma('a') == True
assert palindroma('aa') == True
assert palindroma('ab') == False
assert palindroma('aba') == True
assert palindroma('bab') == True
assert palindroma('bba') == False
assert palindroma('abb') == False
assert palindroma('abba') == True
assert palindroma('baab') == True
assert palindroma('abbb') == False
assert palindroma('bbba') == False
assert palindroma('radar') == True
assert palindroma('scatola') == False

# FINE TEST
```

[ ]:

4.11 Stringhe 5 - Challenges

4.11.1 Scarica zip esercizi

Naviga file online[150]

Proponiamo ora degli esercizi senza soluzione, accetti la sfida?

4.11.2 Sfida: Censura al tiramisù

Ci è stato inviato un testo illegale, immorale e che fa pure ingrassare. Per tutelare la pubblica salute abbiamo l'arduo compito di censurarlo e renderlo consono al buon costume.

1) E' necessario sostituire OGNI occorrenza delle due parole malefiche con una più salubre censura, trasformando il testo in modo che la print di testo_censurato non contenga nessuna parola incriminata.

Esempio:

```
testo= "Grazie all'uso di Cera Splendent tutto brilla!"
parola_malefica = "splendent"
censura = "******"
testo_censurato: "Grazie all'uso di Cera ****** tutto brilla!"
```

2) Per scrivere il nostro rapporto di censori dobbiamo rendere conto di quante parole abbiamo censurato. Stampare alla fine il numero (Ovviamente in maniera automatica) Esempio:

```
"Censurata 1 parola"
```

[1]:
```
testo = """Ed è proprio da questa portentosa crema che nasce la crema al mascarpone
base del tiramisù, prima classificata al premio Crema dell'Anno insieme
alla nutella. Il dolce italiano per eccellenza, quello più famoso e amato,
ma soprattutto che ha dato vita a tantissime altre versioni, anche tiramisù senza␣
↪uova!
Poi il Tiramisù alle fragole o quello alla Nutella, giusto per citarne un paio!
Senza contare le rivisitazioni più raffinate come la crostata morbida o la torta al␣
↪tiramisù.
"""

parola_malefica1 = "Tiramisù"
parola_malefica2 = "nutella"

censura = "******"

testo_censurato = ''

# scrivi qui
```

[150] https://github.com/DavidLeoni/softpython-it

```
Ed è proprio da questa portentosa crema che nasce la crema al mascarpone
base del ******, prima classificata al premio Crema dell'Anno insieme
alla ******. Il dolce italiano per eccellenza, quello più famoso e amato,
ma soprattutto che ha dato vita a tantissime altre versioni, anche ****** senza uova!
Poi il ****** alle fragole o quello alla ******, giusto per citarne un paio!
Senza contare le rivisitazioni più raffinate come la crostata morbida o la torta al␣
↪******.

6
```

4.12 Liste 1 - Introduzione

4.12.1 Scarica zip esercizi

Naviga file online[151]

Una lista in Python è una sequenza di elementi eterogenei **mutabile**, in cui possiamo mettere gli oggetti che vogliamo. L'ordine in cui li mettiamo viene preservato.

4.12.2 Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
lists
    lists1.ipynb
    lists1-sol.ipynb
    lists2.ipynb
    lists2-sol.ipynb
    lists3.ipynb
    lists3-sol.ipynb
    lists4.ipynb
    lists4-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `lists1.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

[151] https://github.com/DavidLeoni/softpython-it/tree/master/lists

4.12.3 Creare liste

Possiamo creare una lista specificando tra parentesi quadre gli elementi che contiene, e separandoli da una virgola.

Per esempio, in questa lista inseriamo i numeri `7`, `4` e `9`:

[2]:
```
[7,4,9]
```
[2]:
```
[7, 4, 9]
```

Come tutti gli oggetti in Python, possiamo associarli ad una variabile, in questo caso ce ne inventiamo una chiamata `lista`:

[3]:
```
lista = [7,4,9]
```

[4]:
```
lista
```
[4]:
```
[7, 4, 9]
```

Vediamo meglio che succede in memoria, e compariamo la rappresentazione delle stringhe con quella delle liste:

[5]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[6]:
```
stringa = "prova"

lista = [7,4,9]

jupman.pytut()
```
[6]:
```
<IPython.core.display.HTML object>
```

Notiamo subito una differenza rilevante. La stringa è rimasta nella regione azzurra dove appaiono le associazioni tra variabili e valori, invece dalla variabile `lista` parte una freccia che punta ad una nuova regione gialla di memoria, che si crea non appena l'esecuzione raggiunge la riga che definisce la lista.

In seguito approfondiremo meglio le conseguenze di ciò.

In una lista gli stessi elementi possono apparire più volte

[7]:
```
numeri = [1,2,3,1,3]
```

[8]:
```
numeri
```
[8]:
```
[1, 2, 3, 1, 3]
```

In una lista possiamo mettere qualunque elemento, per es. le stringhe:

[9]:
```
frutti = ["mela", "pera", "pesca", "fragola", "ciliegia"]
```

[10]:
```
frutti
```
[10]:
```
['mela', 'pera', 'pesca', 'fragola', 'ciliegia']
```

Possiamo anche mischiare i tipi di oggetti contenuti in una lista, per esempio possiamo avere interi e stringhe:

[11]:
```
misto = ["tavolo", 4 ,"sedia", 8, 5, 1, "sedia"]
```

In Python Tutor apparirà così:

[12]:
```
misto = ["tavolo", 5 , 4, "sedia", 8, "sedia"]

jupman.pytut()
```

[12]:
```
<IPython.core.display.HTML object>
```

Per comodità possiamo anche scrivere la lista su più righe (gli spazi in questo caso non contano, ricordati solo di terminare le righe con delle virgole `,`)

[13]:
```
misto = ["tavolo",
         5 ,
         4,
         "sedia",
         8,
         "sedia"]
```

ESERCIZIO: prova a scrivere la lista qua sopra SENZA mettere una virgola dopo il 5, che errore appare?

[14]:
```
# scrivi qui
```

Una lista può anche contenere altre liste:

[15]:
```
tabella = [  ['a','b','c'],  ['d','e','f']  ]
```

Tipicamente, quando abbiamo strutture come questa, conviene disporle su più righe (non è obbligatorio ma conviene per chiarezza):

[16]:
```
tabella = [                 # inizio listona esterna
            ['a','b','c'],    # lista interna 1
            ['d','e','f']     # lista interna 2
          ]                 # fine listona esterna
```

[17]:
```
tabella
```

[17]:
```
[['a', 'b', 'c'], ['d', 'e', 'f']]
```

Vediamo come viene mostrata in Python Tutor:

[18]:
```
tabella = [
            ['a','b','c'],
            ['d','e','f']
          ]

jupman.pytut()
```

[18]:
```
<IPython.core.display.HTML object>
```

Come detto in precedenza, in una lista possiamo mettere gli elementi che vogliamo, quindi possiamo mischiare liste di dimensioni diverse, stringhe, numeri, etc:

[19]:
```
ditutto = [
                ['ciao',3,'mondo'],
                'una stringa',
                [9,5,6,7,3,4],
                8,
          ]
```

[20]:
```
print(ditutto)
```

```
[['ciao', 3, 'mondo'], 'una stringa', [9, 5, 6, 7, 3, 4], 8]
```

Vediamo anche come appare in Python Tutor:

[21]:
```
ditutto = [
                ['ciao',3,'mondo'],
                'una stringa',
                [9,5,6,7,3,4],
                8,
          ]

jupman.pytut()
```

[21]:
```
<IPython.core.display.HTML object>
```

Lista vuota

Ci sono due modi per creare una lista vuota.

1) con parentesi quadre:

[22]:
```
lista_vuota = []
```

[23]:
```
lista_vuota
```

[23]:
```
[]
```

2) Oppure con `list()`:

[24]:
```
altra_lista_vuota = list()
```

[25]:
```
altra_lista_vuota
```

[25]:
```
[]
```

ATTENZIONE: Quando crei una lista vuota (indipendentemente dalla notazione usata), in memoria viene allocata una NUOVA regione di memoria per accogliere la lista

Vediamo meglio cosa vuol dire con Python Tutor:

[26]:
```
a = []
b = []

jupman.pytut()
```

[26]:
```
<IPython.core.display.HTML object>
```

Nota che sono apparse due frecce che puntano a regioni di memoria **differenti**. Lo stesso sarebbe accaduto inizializzando le liste con degli elementi:

[27]:
```
la = [8,6,7]
lb = [9,5,6,4]

jupman.pytut()
```

[27]:
```
<IPython.core.display.HTML object>
```

E avremmo avuto due liste in regioni di memoria diverse anche mettendo elementi identici dentro le liste:

[28]:
```
la = [8,6,7]
lb = [8,6,7]

jupman.pytut()
```

[28]:
```
<IPython.core.display.HTML object>
```

Le cose si complicano quando cominciamo ad usare operazioni di assegnazione:

[29]:
```
la = [8,6,7]
```

[30]:
```
lb = [9,5,6,4]
```

[31]:
```
lb = la
```

Scrivendo `lb = la`, abbiamo detto a Python di 'dimenticare' l'assegnazione precedente di `lb` a `[9,5,6,4]`, e di associare invece `lb` alla stesso valore già associato ad `la`, cioè `[8,6,7]`. Quindi, nella memoria vedremo una freccia che parte da `lb` ed arriva a `[8,6,7]`, e la regione di memoria dove stava la lista `[9,5,6,4]` verrà rimossa (non è più associata ad alcuna variabile). Guardiamo che succede con Python Tutor:

[32]:
```
la = [8,6,7]
lb = [9,5,6,4]
lb = la

jupman.pytut()
```

[32]:
```
<IPython.core.display.HTML object>
```

ESERCIZIO: Prova a scambiare le liste associate alle variabili `la` ed `lb` usando solo assegnazioni e **senza creare nuove liste**. Se vuoi, puoi sovrascrivere una terza variabile `lc`. Verifica che succede con Python Tutor.

- il tuo codice deve poter funzionare con qualunque valore di `la`, `lb` ed `lc`

Esempio - dati:

```
la = [9,6,1]
lb = [2,3,4,3,5]
lc = None
```

Dopo il tuo codice, deve risultare:

```
>>> print(la)
[2,3,4,3,5]
>>> print(lb)
[9,6,1]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[33]:
```
la = [9,6,1]
lb = [2,3,4,3,5]
lc = None

# scrivi qui

lc = la
la = lb
lb = lc

#print(la)
#print(lb)
```

[33]:
```
la = [9,6,1]
lb = [2,3,4,3,5]
lc = None

# scrivi qui
```

DOMANDA: Guarda questi due pezzi di codice. Per ciascun caso, prova a pensare come possono essere rappresentati in memoria e verifica poi con Python Tutor.

- che differenza ci potrà essere?
- quante celle di memoria verranno allocate in totale ?
- quante frecce vedrai ?

```
# primo caso
lb = [
       [8,6,7],
       [8,6,7],
       [8,6,7],
       [8,6,7],
     ]
```

```
# secondo caso
la = [8,6,7]
lb = [
       la,
       la,
       la,
       la
     ]
```

[34]:
```
# primo caso
lb = [
       [8,6,7],
       [8,6,7],
       [8,6,7],
       [8,6,7],
     ]
jupman.pytut()
```

[34]:
```
<IPython.core.display.HTML object>
```

[35]:
```
# secondo caso

la = [8,6,7]
lb = [
       la,
       la,
       la,
       la
     ]
jupman.pytut()
```

[35]:
```
<IPython.core.display.HTML object>
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Nel primo caso, abbiamo una 'listona' associata alla variabile `lb` che contiene 4 sottoliste ciascuna da 3 elementi. Ogni sottolista viene creata come nuova, quindi in totale in memoria abbiamo 4 celle della listona `lb` + (4 sottoliste * 3 celle ciascuna) = 16 celle

Nel secondo caso invece abbiamo sempre la 'listona' associata alla variabile `lb` da 4 celle, ma al suo interno contiene dei puntatori alla stessa identica lista `la`. Quindi il numero totale di celle occupate è 4 celle della listona `lb` + (1 sottolista * 3 celle) = 7 celle

Esercizio - creare liste 1

Date due variabili

```
la = [4,3]
lb = [9,6,7]
```

Scrivi del codice che stampi la lista `[[4, 3], [[9, 6, 7], [4, 3], [9, 6, 7]], [4, 3]]`

- **NON scrivere numeri**, usa solo liste di variabili

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[36]:
```
la = [4,3]
lb = [9,6,7]

# scrivi qui
print([
    la,
    [lb,la,lb],
    la
])
```

```
[[4, 3], [[9, 6, 7], [4, 3], [9, 6, 7]], [4, 3]]
```

[36]:
```
la = [4,3]
lb = [9,6,7]

# scrivi qui
```

```
[[4, 3], [[9, 6, 7], [4, 3], [9, 6, 7]], [4, 3]]
```

Esercizio - creare liste 2

Inserire dei valori nelle liste `la`, `lb` tali per cui

```
print([[la,la],[lb,la]])
```

stampi

```
[[[8, 4], [8, 4]], [[4, 8, 4], [8, 4]]]
```

- **inserire solo dei NUMERI**
- Osservare in Python Tutor come vengono raffigurate le frecce

[37]:
```
la = []  # inserisci dei numeri
lb = []  # inserisci dei numeri

print([[la,la],[lb,la]])
```

```
[[[], []], [[], []]]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[38]:
```
# SOLUZIONE

la = [8,4]
lb = [4,8,4]

#print([[la,la],[lb,la]])
```

[38]:

Esercizio - creare liste 3

Inserire dei valori come elementi delle liste `la`, `lb` e `lc` tali per cui

```
print([[lb,lb,[lc,la]],lc])
```

stampi

```
[[[8, [7, 7]], [8, [7, 7]], [[8, 7], [8, 5]]], [8, 7]]
```

- **inseriri solo NUMERI oppure NUOVE LISTE DI NUMERI**

- Osservare in Python Tutor come vengono raffigurate le frecce

[39]:
```
la = []  # inserisci elementi  (numeri o liste di numeri)
lb = []  # inserisci elementi  (numeri o liste di numeri)
lc = []  # inserisci elementi  (numeri o liste di numeri)

print([[lb,lb,[lc,la]],lc])
```

```
[[[], [], [[], []]], []]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[40]:
```
# SOLUZIONE

la = [8,5]
lb = [8,[7,7]]
lc = [8,7]

#print([[lb,lb,[lc,la]],lc])
```

[40]:
```
```

Esercizio - creare liste 4

Inserire dei valori nelle liste `la`, `lb` tali per cui

```
print([[la,lc,la], lb])
```

stampi

```
[[[3, 2], [[3, 2], [8, [3, 2]]], [3, 2]], [8, [3, 2]]]
```

- **inserire solo NUMERI oppure VARIABILI** `la`,`lb` o `lc`
- Osservare in Python tutor come vengono raffigurate le frecce

[41]:
```
la = []  # inserisci numeri o variabili la, lb, lc
lb = []  # inserisci numeri o variabili la, lb, lc
lc = []  # inserisci numeri o variabili la, lb, lc

print([[la,lc,la], lb])
```

```
[[[], [], []], []]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[42]:
```
# SOLUZIONE
la = [3,2]
lb = [8,la]
lc = [la,lb]
```

(continues on next page)

(continued from previous page)

```
#print([[la,lc,la], lb])
```

[42]:

```

```

4.12.4 Convertire sequenze in liste

`list` può anche servire per convertire una qualsiasi sequenza in una NUOVA lista. Un tipo di sequenza che già abbiamo visto sono le stringhe, quindi proviamo a vedere cosa succede se usiamo `list` come fosse una funzione e gli passiamo come parametro una stringa:

[43]:
```
list("treno")
```

[43]:
```
['t', 'r', 'e', 'n', 'o']
```

Abbiamo ottenuto una lista in cui ogni elemento è costituito da un carattere della stringa originale.

Se invece chiamiamo `list` su un'altra lista cosa succede?

[44]:
```
list( [7,9,5,6] )
```

[44]:
```
[7, 9, 5, 6]
```

Apparentemente, niente di particolare, otteniamo una lista con gli stessi elementi di partenza. Ma è proprio la stessa lista? Guardiamo meglio con Python Tutor:

[45]:
```
la = [7,9,5,6]

lb = list( la )

jupman.pytut()
```

[45]:
```
<IPython.core.display.HTML object>
```

Notiamo che è stata creata una NUOVA regione di memoria con gli stessi elementi di `la`.

Esercizio - gulp

Data una stringa con caratteri misti minuscoli e maiuscoli, scrivere del codice che crea una lista contente come primo elemento una lista coi caratteri della stringa tutti minuscoli e come secondo elemento una lista contenente tutti i caratteri maiuscoli

- il tuo codice deve funzionare con qualunque stringa
- se non ricordi i metodi delle stringhe, guarda qui[152]

Esempio - dato

```
s = 'GuLp'
```

il tuo codice dovrà stampare

[152] https://it.softpython.org/strings/strings3-sol.html

```
[['g', 'u', 'l', 'p'], ['G', 'U', 'L', 'P']]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[46]:
```
s = 'GuLp'

# scrivi qui
print([list(s.lower()), list(s.upper())])
```

```
[['g', 'u', 'l', 'p'], ['G', 'U', 'L', 'P']]
```

[46]:
```
s = 'GuLp'

# scrivi qui
```

```
[['g', 'u', 'l', 'p'], ['G', 'U', 'L', 'P']]
```

DOMANDA: Questo codice:

- produce un errore o assegna qualcosa ad `x` ?
- Dopo la sua esecuzione, quante liste rimangono in memoria?
- Possiamo accorciarlo?

```
s = "maratona"
x = list(list(list(list(s))))
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Il codice assegna alla variabile `x` la lista `['m', 'a', 'r', 'a', 't', 'o', 'n', 'a']`. La prima volta `list(s)` genera la NUOVA lista `['m', 'a', 'r', 'a', 't', 'o', 'n', 'a']`. Le successive chiamate a `list` prendono in input la lista appena generata `['m', 'a', 'r', 'a', 't', 'o', 'n', 'a']` e continuano a creare NUOVE liste con contenuto identico. Dato che però nessuna lista prodotta eccetto l'ultima viene assegnata ad una variabile, quelle intermedie alla fine dell'esecuzione vengono di fatto eliminate. Possiamo quindi tranquillamente accorciare il codice scrivendo

```
s = "maratona"
x = list(s)
```

DOMANDA: Questo codice:

- produce un errore o assegna qualcosa ad `x` ?
- Dopo la sua esecuzione, quante liste rimangono in memoria?

```
s = "catena"
a = list(s)
b = list(a)
```

(continues on next page)

(continued from previous page)

```
c = b
x = list(c)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Rimangono in memoria 3 liste ognuna contenente 6 celle. Questa volta le liste permangono in memoria perchè sono associate alle variabili `a`,`b` e `c`. Abbiamo 3 e non 4 liste perchè con l'istruzione `c = b` la variabile `c` viene associata alla stessa identica regione di memoria associata alla variabile `b`

Esercizio - garaga

Date

```
sa = "ga"
sb = "ra"
la = ['ga']
lb = list(la)
```

- Assegnare ad `lc` una lista costruita in modo che una volta stampata produca

```
>>> print(lc)
```

```
[['g', 'a', 'r', 'a'], ['ga'], ['ga'], ['r', 'a', 'g', 'a']]
```

- **in Python Tutor, TUTTE le frecce dovranno puntare ad una regione di memoria diversa**

[47]:
```
sa = "ga"
sb = "ra"
la = ['ga']
lb = list(la)

# inserire del codice nella lista
lc = []

print(lc)
jupman.pytut()
```

```
[]
```

[47]:
```
<IPython.core.display.HTML object>
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[48]:
```
# SOLUZIONE

sa = "ga"
sb = "ra"
la = ['ga']
lb = list(la)
lc = [list(sa + sb), list(la), list(lb), list(sb + sa) ]
```

(continues on next page)

(continued from previous page)

```
print(lc)
jupman.pytut()
```

```
[['g', 'a', 'r', 'a'], ['ga'], ['ga'], ['r', 'a', 'g', 'a']]
```

[48]:
```
<IPython.core.display.HTML object>
```

[48]:

```
[['g', 'a', 'r', 'a'], ['ga'], ['ga'], ['r', 'a', 'g', 'a']]
```

[48]:
```
<IPython.core.display.HTML object>
```

4.12.5 Continua

Prosegui con il foglio Liste 2 - Operatori[153]

[ ]:

4.13 Liste 2 - operatori

4.13.1 Scarica zip esercizi

Naviga file online[154]

Per manipolare le liste vi sono diversi operatori. I seguenti si comportano come quelli visti per le stringhe.

Operatore	Risultato	Significato
`len`(lst)	`int`	Ritorna la lunghezza di una lista
list`[`int`]`	obj	Legge/scrive un elemento all'indice specificato
list`[`int`:`int`]`	`list`	Estrae una sotto-lista - ritorna una NUOVA lista
obj `in` list	`bool`	Controlla se un elemento è presente in una lista
list + list	`list`	Concatena due liste - ritorna una NUOVA lista
`max`(lst)	`int`	Data una lista di numeri, ritorna il massimo
`min`(lst)	`int`	Data una lista di numeri, ritorna il minimo
`sum`(lst)	`int`	Data una lista di numeri, li somma tutti
list `*` int	`list`	Replica la lista - ritorna una NUOVA lista
`==`,`!=`	`bool`	Controlla se due liste sono uguali o differenti

[153] https://it.softpython.org/lists/lists2-sol.html
[154] https://github.com/DavidLeoni/softpython-it/tree/master/lists

4.13.2 Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
lists
    lists1.ipynb
    lists1-sol.ipynb
    lists2.ipynb
    lists2-sol.ipynb
    lists3.ipynb
    lists3-sol.ipynb
    lists4.ipynb
    lists4-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `lists2.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.13.3 Lunghezza di una lista

Una lista è una sequenza, e come per tutte le sequenze per ottenere la lunghezza si può usare la funzione `len`:

[2]:
```
a = [7,5,8]
```

[3]:
```
len(a)
```

[3]:
```
3
```

[4]:
```
b = [8,3,6,4,7]
```

[5]:
```
len(b)
```

[5]:
```
5
```

Se una lista contiene altre liste, contano come singoli elementi:

[6]:
```
mista = [
            [4,5,1],
            [8,6],
            [7,6,0,8],
        ]
```

[7]:
```
len(mista)
```

[7]:
```
3
```

ATTENZIONE: NON puoi usare ``len`` come se fosse un metodo

Per es questo NON funziona: `[3,4,2].len()`

ESERCIZIO: Prova a scrivere `[3,4,2].len()` qua sotto, che errore appare?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[8]:
```
# scrivi qui

#[3,4,2].len()
```

[8]:
```
# scrivi qui
```

ESERCIZIO: Prova a scrivere `[3,4,2].len` SENZA le doppie tonde alla fine, che errore appare?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[9]:
```
# scrivi qui

#[3,4,2].len
```

[9]:
```
# scrivi qui
```

DOMANDA: Se `x` è una lista qualunque, scrivendo

`len(len(x))`

cosa otteniamo?

1. la lunghezza della lista
2. un errore
3. qualcos'altro

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 2: `len` vuole come argomento *una sequenza* e restituisce *un numero*, quindi la chiamata interna a `len(x)` produce un numero che viene dato al `len` esterno e a quel punto Python si lamenterà che ha ricevuto un numero invece di una sequenza. Prova a verificare che errore appare scrivendo `len(len(x))` qua sotto

DOMANDA: Guarda questa espressione, senza eseguirla. Cosa produce?

```
[len([]), len([len(['a','b'])])]
```

1. un errore (quale?)
2. un numero (quale?)
3. una lista (quale?)

Prova a scrivere a mano il risultato, e poi confrontalo con quello ottenuto eseguendo il codice in una cella.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 3: la lista `[0, 1]`

DOMANDA: Guarda questa espressione, senza eseguirla. Cosa produce?

```
len([[[],[]],[],[[[]]],[[],[]]])
```

1. un errore (quale?)
2. un numero (quale?)
3. una lista (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 2. produce il numero 4

DOMANDA: Cosa produce la seguente espressione?

```
[[((len('ababb')))],len(["argg",('b'),("c")]), len([len("bc")])]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: `[[5], 3, 1]`

4.13.4 Leggere un elemento

Come per le stringhe, possiamo accedere ad un elemento di una lista mettendo l'indice della posizione a cui vogliamo accedere tra parentesi quadre:

[10]:
```
    # 0   1   2   3
la = [77, 69, 95, 57]
```

Come per tutte le sequenze, le posizioni iniziano da ``0``:

[11]:
```
la[0]
```
[11]:
```
77
```

[12]:
```
la[1]
```
[12]:
```
69
```

[13]:
```
la[2]
```
[13]:
```
95
```

[14]:
```
la[3]
```
[14]:
```
57
```

Come per le stringhe, se sforiamo con l'indice otteniamo un errore:

```
la[4]

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-134-09bfed834fa2> in <module>
----> 1 la[4]

IndexError: list index out of range
```

Di nuovo come per le stringhe possiamo ottenere l'ultimo elemento usando un indice negativo:

[15]:
```
    # 0   1   2   3
la = [77, 69, 95, 57]
```

[16]:
```
la[-1]
```
[16]:
```
57
```

[17]:
```
la[-2]
```
[17]:
```
95
```

[18]:
```
la[-3]
```
[18]:
```
69
```

[19]:
```
la[-4]
```
[19]:
```
77
```

Se esageriamo ed andiamo oltre la lunghezza della lista, otteniamo un errore:

```
la[-5]

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
```

(continues on next page)

(continued from previous page)

```
<ipython-input-169-f77280923dce> in <module>
----> 1 la[-5]

IndexError: list index out of range
```

DOMANDA: se `x` è una lista qualunque, scrivendo

```
x[0]
```

cosa otteniamo?

1. il primo elemento della lista
2. sempre un errore
3. a volte un elemento a volte un errore a seconda della lista

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 3: se la lista è vuota Python non troverà l'elemento e ci darà un errore. Quale? Prova a scrivere nella cella sotto `[][0]` e vedi cosa succede.

DOMANDA: se `x` è una lista qualunque, scrivendo

```
x[len(x)]
```

cosa otteniamo?

1. un elemento della lista
2. sempre un errore
3. a volte un elemento a volte un errore a seconda della lista

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 2. sempre un errore: `len(x)` sarà sempre un numero uguale all'ultimo indice disponibile della lista + 1

4.13.5 Scrivere un elemento

Visto che le liste sono MUTABILI, dato un oggetto lista possiamo cambiare il contenuto di una qualunque cella al suo interno.

Supponiamo per esempio di voler cambiare la cella all'indice `2` della lista `la`, cambiandolo da `6` a `5`:

[20]:
```
     #0  1  2  3
la = [7, 9, 6, 8]
```

Possiamo scrivere così:

[21]:
```
la[2] = 5
```

[22]:
```
la
```

[22]:
```
[7, 9, 5, 8]
```

Guardiamo meglio che succede in Python Tutor:

[23]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[24]:
```
#     0  1  2  3
la = [7, 9, 6, 8]
la[2] = 5

jupman.pytut()
```

[24]:
```
<IPython.core.display.HTML object>
```

Come vedi, in questo non vengono create regioni di memoria nuove, semplicemente si va a sovrascrivere una cella esistente.

4.13.6 Mutare liste condivise

ATTENZIONE: L'ARGOMENTO CHE SEGUE E' CAUSA DEL 90% DEGLI ERRORI DI PROGRAMMAZIONE !!!

LEGGI BENE !!!

Cosa succede quando associamo a due variabili lo stesso identico oggetto mutabile, come per esempio una lista, e poi usando una delle due variabili mutiamo l'oggetto?

Guardiamo un esempio - per prima cosa, alla variabile `la` associamo la lista `[7,9,6]`:

[25]:
```
la = [7,9,6]
```

Adesso definiamo una nuova variabile `lb`, e associamogli come valore *lo stesso valore* già associato alla variabile `la`. Nota bene: qua NON stiamo creando nuove liste !

[26]:
```
lb = la
```

[27]:
```
print(la)  # la è sempre lo stesso
```

```
[7, 9, 6]
```

[28]:
```
print(lb)  # lb è la stessa lista associata a la
```

```
[7, 9, 6]
```

Proviamo a modificare una cella di `lb`, mettendo `5` nella cella all'indice `0`:

[29]:
```
lb[0] = 5
```

Se proviamo a stampare le variabili `la` ed `lb`, Python andrà a guardare i valori associati a ciascuna variabile. Dato che il valore è la stessa identica lista (che risiede nella stessa identica regione di memoria), in entrambi i casi vedrai la modifica appena fatta !

[30]:
```
print(la)
```

```
[5, 9, 6]
```

[31]:
```
print(lb)
```

```
[5, 9, 6]
```

Guardiamo meglio che succede in Python Tutor:

[32]:
```
la = [7,9,6]
lb = la
lb[0] = 5
print('la è', la)
print('lb è', lb)

jupman.pytut()
```

```
la è [5, 9, 6]
lb è [5, 9, 6]
```

[32]:
```
<IPython.core.display.HTML object>
```

Guardiamo la differenza se invece creiamo esplicitamente una lista uguale a `la`.

In questo caso avremo due regioni di memoria distinte e la NON sarà modificato:

[33]:
```
la = [7,9,6]
lb = [7,9,6]
lb[0] = 5
print('la è', la)
print('lb è', lb)

jupman.pytut()
```

```
la è [7, 9, 6]
lb è [5, 9, 6]
```

[33]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Dopo l'esecuzione di questo codice, cosa verrà stampato? Quante liste ci saranno effettivamente in memoria?

Prova a disegnare **SU CARTA** quello che succede in memoria, e poi confronta con il risultato in Python Tutor!

```
la = [8,7,7]
lb = [9,6,7,5]
lc = lb
la = lb
print('la è', la)
print('lb è', lb)
print('lc è', lc)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Viene stampato

```
la è [9, 6, 7, 5]
lb è [9, 6, 7, 5]
lc è [9, 6, 7, 5]
```

perchè

```
la = [8,7,7]
lb = [9,6,7,5]
# la variabile lc viene associata alla stessa identica lista di lb.
lc = lb
# la variabile la viene associata alla stessa identica lista di lb.
# La lista precedentemente associata a la viene persa
la = lb
```

[34]:
```
la = [8,7,7]
lb = [9,6,7,5]
lc = lb
la = lb
#print('la è', la)
#print('lb è', lb)
#print('lc è', lc)
jupman.pytut()
```

[34]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Guarda il codice seguente. Dopo la sua esecuzione, cosa produrrà la stampa di `la`, `lb` ed `lc` ?

Prova a disegnare **SU CARTA** quello che succede in memoria, e poi confronta con il risultato in Python Tutor!

```
la = [7,8,5]
lb = [6,7]
lc = lb
lb = la
lc[0] = 9
print('la è', la)
print('lb è', lb)
print('lc è', lc)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La stampa produrrà

```
la è [7, 8, 5]
lb è [7, 8, 5]
lc è [9, 7]
```

perchè :

```
la = [7,8,5]
lb = [6,7]
# la variabile lc viene assegnata alla stessa lista di lb [6,7]
lc = lb
# la variabile lb viene associata alla stessa lista di la [7,8,5].
# Questo non cambia l'assegnazione di lc, che resta associato a [6,7] !
lb = la
# Modifica il primo elemento della lista associata a lc che da [6,7] diventa [9,7]
lc[0] = 9
print('la è', la)
print('lb è', lb)
print('lc è', lc)
```

[35]:
```
la = [7,8,5]
lb = [6,7]
lc = lb
lb = la
lc[0] = 9
#print('la è', la)
#print('lb è', lb)
#print('lc è', lc)

jupman.pytut()
```

[35]:
```
<IPython.core.display.HTML object>
```

4.13.7 Slice

Possiamo estrarre delle sequenze dalle liste usando le slice. Una slice si produce aggiungendo delle parentesi quadre alla destra della lista e mettendo nelle quadre l'indice di partenza (INCLUSO), seguito da due punti `:`, seguito dall'indice di fine (ESCLUSO). Funziona esattamente come per le stringhe: in quel caso la quadra produceva una nuova stringa, in questo si produce una NUOVA lista. Vediamo un esempio:

[36]:
```
     #0  1  2  3  4  5  6  7  8  9
la = [43,35,82,75,93,12,43,28,54,65]
```

[37]:
```
la[3:7]
```

[37]:
```
[75, 93, 12, 43]
```

Abbiamo estratto una NUOVA lista `[75, 93, 12, 43]` a partire della lista `la` partendo dall'indice `3` INCLUSO fino all'indice `7` ESCLUSO. Possiamo vedere che la lista originale si è preservata:

[38]:
```
la
```

[38]:
```
[43, 35, 82, 75, 93, 12, 43, 28, 54, 65]
```

Verifichiamo cosa succede con Python Tutor, assegnando la nuova lista ad una variabile `lb`:

[39]:
```
#     0  1  2  3  4  5  6  7  8  9
la = [43,35,82,75,93,12,43,28,54,65]
lb = la[3:7]

jupman.pytut()
```

[39]:
```
<IPython.core.display.HTML object>
```

Noterai che compare una NUOVA regione di memoria, associata alla variabile `lb`.

Slice - limiti

Quando operiamo con le slice dobbiamo stare attenti ai limiti degli indici. Vediamo come si comportano:

[40]:
```
#0  1  2  3  4
[58,97,76,87,99][0:3]  # da indice 0 *incluso* a 3 *escluso*
```

[40]:
```
[58, 97, 76]
```

[41]:
```
#0  1  2  3  4
[58,97,76,87,99][0:4]  # da indice 0 *incluso* a 4 *escluso*
```

[41]:
```
[58, 97, 76, 87]
```

[42]:
```
#0  1  2  3  4
[58,97,76,87,99][0:5]  # da indice 0 *incluso* a 5 *escluso*
```

[42]:
```
[58, 97, 76, 87, 99]
```

[43]:
```
#0  1  2  3  4
[58,97,76,87,99][0:6]   # se andiamo oltre la lunghezza della lista Python non si␣
↪arrabbia
```

[43]:
```
[58, 97, 76, 87, 99]
```

[44]:
```
#0  1  2  3  4
[58,97,76,87,99][8:12]   # anche se partiamo da indici inesistenti Python non si␣
↪arrabbia
```

[44]:
```
[]
```

DOMANDA: Questa espressione

```
[][0:8]
```

1. produce un risultato (quale?)
2. produce un errore (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: data una lista vuota, stiamo cercando di ottenere una sottolista che va dall'indice 0 INCLUSO all'indice 8 ESCLUSO. Come abbiamo visto prima, se sforiamo Pyhton non si arrabbia, quindi semplicemente non trovando elementi ci ritorna una lista vuota.

DOMANDA: Questa espressione

```
[][3:8]
```

1. produce un risultato (quale?)

2. produce un errore (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: data una lista vuota, stiamo cercando di ottenere una sottolista che va dall'indice 3 INCLUSO all'indice 8 ESCLUSO. Come abbiamo visto prima, se sforiamo anche dal limite sinistro Pyhton non si arrabbia, quindi semplicemente non trovando elementi ci ritorna una lista vuota.

DOMANDA: se `x` è una lista qualsiasi (anche vuota), questa espressione cosa fa? Può dare errore? o ritorna qualcosa di utile?

```
x[0:len(x)]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Ritorna sempre una NUOVA copia dell'intera lista, perchè parte da indice 0 *INCLUSO* e finisce a indice `len(x)` *ESCLUSO*

Funziona anche con la lista vuota, perchè `[][0:len([])]` equivale a `[][0:0]` cioè sottolista da 0 *incluso* a 0 *escluso*, quindi non prendiamo nessun carattere e non sforiamo i limiti della lista. In realtà, come abbiamo visto prima, anche se sforassimo Python non avrebbe problemi.

Slice - omissione limiti

Volendo, è possibile omettere l'indice di partenza, in tal caso Python supporrà sia 0:

[45]:
```
#0  1  2  3  4  5  6  7  8  9
[98,67,85,77,65,99,67,55,79][:3]
```

[45]:
```
[98, 67, 85]
```

E' anche possibile omettere l'indice di fine, in tal caso Python estrarrà fino alla fine della lista:

[46]:
```
#0  1  2  3  4  5  6  7  8  9
[98,67,85,77,65,99,67,55,79][3:]
```

[46]:
```
[77, 65, 99, 67, 55, 79]
```

Omettendo entrambi gli indici si ottiene l'intera lista:

[47]:
```
#0  1  2  3  4  5  6  7  8  9
[98,67,85,77,65,99,67,55,79][:]
```

[47]:
```
[98, 67, 85, 77, 65, 99, 67, 55, 79]
```

DOMANDA: Cosa stamperà questo codice ? `la` sarà modificato oppure no?

```
la = [7,8,9]
lb = la[:]
lb[0] = 6
print('la =',la)
print('lb =',lb)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: `lb = la[:]` crea una NUOVA lista contente tutti gli elementi che stanno in `la`. Quindi quando scriviamo `lb[0] = 6` stiamo modificando solo la regione di memoria associat a `lb`. Se osservi in Python Tutor, vedrai che `la` e `lb` puntano a regioni di memoria differenti:

[48]:
```
la = [7,8,9]
lb = la[:]
lb[0] = 6
#print('la =',la)
#print('lb =',lb)

jupman.pytut()
```

[48]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Per ciascuna delle espressioni seguenti, prova a indovinare quale valore produce, o se da errore.

1. `[][]`
2. `[][:]`
3. `[3][:]`
4. `[:][]`
5. `[9,7,8,6][1:1]`
6. `[9,7,8,6][1:2]`
7. `[9,7,8,6][2:3][0]`

Slice - limiti negativi

Volendo è anche possibile impostare limiti negativi, per quanto non sia sempre molto intuitivo.

[49]:
```
#0  1  2  3  4  5  6
[73,48,19,57,64,15,92][3:0]   # da indice 3 a indici positivi <= 3 non produce nulla
```

[49]:
```
[]
```

[50]:
```
#0  1  2  3  4  5  6
[73,48,19,57,64,15,92][3:1]   # da indice 3 a indici positivi <= 3 non produce nulla
```

[50]:
```
[]
```

[51]:
```
#0  1  2  3  4  5  6
[73,48,19,57,64,15,92][3:2]  # da indice 3 a indici positivi <= 3 non produce nulla
```

[51]:
```
[]
```

[52]:
```
#0  1  2  3  4  5  6
[73,48,19,57,64,15,92][3:3]  # da indice 3 a indici positivi <= 3 non produce nulla
```

[52]:
```
[]
```

Vediamo cosa succede con indici negativi:

[53]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][3:-1]
```

[53]:
```
[57, 64, 15]
```

[54]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][3:-2]
```

[54]:
```
[57, 64]
```

[55]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][3:-3]
```

[55]:
```
[57]
```

[56]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][3:-4]
```

[56]:
```
[]
```

[57]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][3:-5]
```

[57]:
```
[]
```

E' anche possibile partire da un indice negativo e arrivare ad uno positivo: finchè il primo indice marca una posizione antecedente a quella del secondo indice, viene restituito qualcosa:

[58]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][-7:3]
```

[58]:
```
[73, 48, 19]
```

[59]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][-6:3]
```

[59]:
```
[48, 19]
```

[60]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][-5:3]
```

[60]:
```
[19]
```

[61]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][-4:3]
```

[61]:
```
[]
```

[62]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][-3:3]
```

[62]:
```
[]
```

[63]:
```
# 0  1  2  3  4  5  6
#-7 -6 -5 -4 -3 -2 -1
[73,48,19,57,64,15,92][-2:3]
```

[63]:
```
[]
```

DOMANDA: Per ciascuna delle espressioni seguenti, prova a indovinare quale valore produce, o se da errore.

1. `[9,7,8,6][0:-2]`
2. `[0:-2][9,7,8,6]`
3. `[5,7,9][1:-1]`
4. `[][-13:-17]`
5. `[9,7,8,6][-4:-1]`
6. `[9,7,8,6][-5:-1]`
7. `[9,7,8,6,10,32][-3:1]`
8. `[9,7,8,6,10,32][-3:5]`

Slice - modifica

Supponiamo di avere la lista

[64]:
```
     #0  1  2  3  4  5  6  7
la = [12,23,35,41,74,65,34,22]
```

e di voler cambiare le celle di `la` dall'indice `3` INCLUSO all'indice `6` ESCLUSO in modo che contengano i numeri presi dalla lista `[98,96,97]`. Possiamo farlo con questa notazione speciale che ci consente di scrivere una slice *alla sinistra* dell'operatore =:

[65]:
```
la[3:6] = [98,96,97]
```

[66]:
```
la
```

[66]:
```
[12, 23, 35, 98, 96, 97, 34, 22]
```

In questo esempio leggermente più complesso verifichiamo in Python Tutor che viene effettivamente modificata la regione di memoria originale:

[67]:
```
#      0  1  2  3  4  5  6  7
la = [12,23,35,41,74,65,34,22]
lb = la
lb[3:6] = [98,96,97]

jupman.pytut()
```

[67]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Guarda il seguente codice - cosa produce?

```
la = [9,6,5,8,2]
la[1:4] = [4,7,0]
print(la)
```

1. modifica `la` (come?)
2. un errore (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 1 - MODIFICA `la` così:

```
# 0  1  2  3  4
[ 9, 4, 7, 0, 2]
```

quindi dall'indice `1` INCLUSO all'indice `4` ESCLUSO

DOMANDA: Guarda il seguente codice. Cosa produce?

```
la = [7,6,8,4,2,4,2,3,1]
i = 3
lb = la[0:i]
la[i:2*i] = lb
print(la)
```

1. modifica `la` (come?)
2. un errore (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 1 - modifica `la` copiando le prime `i` celle nelle successive.

4.13.8 Liste di stringhe

Abbiamo detto che in una lista possiamo mettere qualunque oggetto, per esempio delle stringhe

```
[68]: verdure = ['pomodori', 'verze', 'carote', 'cavoli']
```

Proviamo ad estrarre una verdura scrivendo questa espressione:

```
[69]: verdure[2]
[69]: 'carote'
```

Ora, l'espressione precedente ci ha prodotto il risultato `'carote'`, che sappiamo essere una stringa. Questo ci suggerisce che possiamo usare l'espressione esattamente come se fosse una stringa.

Supponiamo che vogliamo ottenere il primo carattere della stringa `'carote'`, avendo direttamente la stringa scriveremmo così:

```
[70]: 'carote'[0]
[70]: 'c'
```

Ma se la stringa è racchiusa nella lista di prima, potremmo fare direttamente così:

```
[71]: verdure[2][0]
[71]: 'c'
```

Esercizio - sigle

Data una lista con sigle di esattamente 4 capoluoghi in minuscolo, scrivere del codice che crea una NUOVA lista contente le stesse sigle in caratteri tutti maiuscoli.

- il tuo codice deve funzionare con qualunque lista di 4 capoluoghi
- suggerimento: se non ti ricordi il metodo giusto, guarda qua[155]

Esempio 1 - dato:

```
sigle = ['tn','mi','to','ro']
```

il tuo codice deve stampare

```
['TN', 'MI', 'TO', 'RO']
```

Esempio 2 - dato:

```
sigle = ['pa','ge','ve', 'aq']
```

il tuo codice deve stampare:

```
['PA', 'GE', 'VE', 'AQ']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[155] https://it.softpython.org/strings/strings3-sol.html

[72]:
```
sigle = ['tn','mi','to','ro']

# scrivi qui

#print([sigle[0].upper(), sigle[1].upper(), sigle[2].upper(), sigle[3].upper()])
```

[72]:
```
sigle = ['tn','mi','to','ro']

# scrivi qui
```

Esercizio - giochi

Data una lista `giochi` di esattamente 3 stringhe, scrivere del codice che MODIFICA la lista in modo che contenga solo i primi caratteri di ciascuna stringa

- Il tuo codice deve funzionare con qualunque lista di esattamente 3 stringhe

Esempio - data:

```
giochi = ["Monopoli",
          "RISIKO",
          "Tombola"]
```

Dopo l'esecuzione del tuo codice, deve risultare:

```
>>> print(giochi)
["M","R","T"]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[73]:
```
giochi = ["Monopoli",
          "RISIKO",
          "Tombola"]


# scrivi qui

giochi = [ giochi[0][0], giochi[1][0], giochi[2][0] ]
#print(giochi)
```

[73]:
```
giochi = ["Monopoli",
          "RISIKO",
          "Tombola"]


# scrivi qui
```

4.13.9 Liste di liste

NOTA: Parleremo molto più nel dettaglio delle liste di liste nel tutorial Matrici - liste di liste[156], questa è solo un'introduzione sommaria.

Le considerazioni viste per le liste di stringhe valgono anche per una lista di liste:

```
[74]: coppie = [          # lista esterna
            [67,95],     # lista interna ad indice 0
            [60,59],     # lista interna ad indice 1
            [86,75],     # lista interna ad indice 2
            [96,90],     # lista interna ad indice 3
            [88,87],     # lista interna ad indice 4
         ]
```

Se vogliamo estrarre il numero `90`, dobbiamo prima estrarre la sottolista all'indice `3`:

```
[75]: coppie[3]   # NOTA: il risultato di questa espressione è una lista
[75]: [96, 90]
```

e quindi dalla sottolista estratta (che ha solo due elementi) possiamo recuperare il numero all'indice `0`:

```
[76]: coppie[3][0]
[76]: 96
```

e all'indice `1`:

```
[77]: coppie[3][1]
[77]: 90
```

Esercizio - coppie

1. Scrivi il codice per recuperare e stampare il numero `86`, il `67` e l'`87`
2. Date una riga con indice `i` e una colonna `j`, stampare il numero alla riga `i` e colonna `j` moltiplicato per il numero alla riga successiva e stessa colonna

Dopo il tuo codice, dovresti veder stampato:

```
punto 1: 86 67 87

punto 2: 7830
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

```
[78]: coppie = [           # lista esterna
            [67,95],     # lista interna ad indice 0
            [60,59],     # lista interna ad indice 1
            [86,75],     # lista interna ad indice 2
            [96,90],     # lista interna ad indice 3
            [88,87],     # lista interna ad indice 4
```

(continues on next page)

[156] https://it.softpython.org/matrices-lists/matrices-lists-sol.html

(continued from previous page)

```
        ]

i = 3
j = 1

# scrivi qui

#print("punto 1:", coppie[2][0], coppie[0][0], coppie[4][1])
#print()
#print("punto 2:", coppie[i][j]*coppie[i+1][j])
```

[78]:
```
coppie = [            # lista esterna
            [67,95],     # lista interna ad indice 0
            [60,59],     # lista interna ad indice 1
            [86,75],     # lista interna ad indice 2
            [96,90],     # lista interna ad indice 3
            [88,87],     # lista interna ad indice 4
        ]

i = 3
j = 1

# scrivi qui
```

Esercizio - nonunif

Data una lista `nonunif` di sottoliste di lunghezza arbitraria, e una riga ad indice `i`, scrivere del codice che MODIFICA le sottoliste di `nonunif` alla riga `i` e alla successiva in modo che l'ultimo elemento di entrambe le liste diventi `99`.

- il tuo codice deve funzionare con qualunque `nonunif` e qualunque `i`

Esempio 1 - date:

```
nonunif = [                    # lista esterna
            [67,95],           # lista interna ad indice 0
            [60,23,23,13,59],  # lista interna ad indice 1
            [86,75],           # lista interna ad indice 2
            [96,90,92],        # lista interna ad indice 3
            [88,87],           # lista interna ad indice 4
        ]

i = 1
```

dopo il tuo codice, scrivendo (usiamo `pprint` perchè così la stampa avverrà su più linee)

```
from pprint import pprint
pprint(nonunif,width=30)
```

deve stampare:

```
[[67, 95],
 [60, 23, 23, 13, 99],
 [86, 99],
 [96, 90, 92],
 [88, 87]]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[79]:
```
nonunif = [                     # lista esterna
            [67,95],            # lista interna ad indice 0
            [60,23,23,13,59],   # lista interna ad indice 1
            [86,75],            # lista interna ad indice 2
            [96,90,92],         # lista interna ad indice 3
            [88,87],            # lista interna ad indice 4
         ]

i = 1

# scrivi qui

nonunif[i][-1] = 99
nonunif[i+1][-1] = 99
#from pprint import pprint
#pprint(nonunif,width=30)
```

[79]:
```
nonunif = [                     # lista esterna
            [67,95],            # lista interna ad indice 0
            [60,23,23,13,59],   # lista interna ad indice 1
            [86,75],            # lista interna ad indice 2
            [96,90,92],         # lista interna ad indice 3
            [88,87],            # lista interna ad indice 4
         ]

i = 1

# scrivi qui
```

4.13.10 Operatore in

Per verificare se un oggetto è contenuto in una lista, possiamo usare l'operatore `in`.

Nota che il risultato di questa espressione è un booleano:

[80]:
```
9 in [6,8,9,7]
```

[80]:
```
True
```

[81]:
```
5 in [6,8,9,7]
```

[81]:
```
False
```

[82]:
```
"mela" in ["anguria","mela","banana"]
```
[82]:
```
True
```

[83]:
```
"carota" in ["anguria","banana","mela"]
```
[83]:
```
False
```

DOMANDA: Questa espressione cosa restituisce? `True` o `False`?

```
True in [ 5 in [6,7,5],
          2 in [8,1]
        ]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Restituisce `True` perchè

```
[ 5 in [6,7,5],
  2 in [8,1]
]
```

rappresenta una lista di due elementi. Ogni elemento è una espressione con `in`, che viene valutata. Nel primo caso, `5 in [6,7,5]` risulta `True`, nel secondo `2 in [8,1]` risulta `False`. Quindi la lista finale diventa `[True, False]` e scrivendo `True in [True, False]` si ottiene `True`

not in

Possiamo scrivere la verifica di **non** appartenza in due forme:

Forma 1:

[84]:
```
"carota" not in ["anguria","banana","mela"]
```
[84]:
```
True
```

[85]:
```
"anguria" not in ["anguria","banana","mela"]
```
[85]:
```
False
```

Forma 2:

[86]:
```
not "carota" in ["anguria","banana","mela"]
```
[86]:
```
True
```

[87]:
```
not "anguria" in ["anguria","banana","mela"]
```
[87]:
```
False
```

DOMANDA: Data qualunque elmento `x` e lista `y` , la seguente espressione cosa restituisce?

```
x in y and not x in y
```

1. `False`

2. `True`
3. `False` oppure `True` a seconda dei valori di `x` e `y`
4. un errore

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 1. Restituisce `False`, perchè internamente python 'Parentesizza' l'espressione così:

```
(x in y) and (not x in y)
```

e un elemento non può contemporaneamente essere e non essere nella stessa lista.

DOMANDA: Per ciascuna delle seguenti espressioni, provare a indovinare il risultato

1. `3 in [3]`
2. `[4,5] in [1,2,3,4,5]`
3. `[4,5] in [[1,2,3],[4,5]]`
4. `[4,5] in [[1,2,3,4],[5,6]]`
5. `'a' in ['prova'[-1]]`
6. `'ghe' in 'paghe'[1:4]`
7. `[] in [[[]]]`
8. `[] in [[]]`
9. `[] in ["[]"]`

DOMANDA: Per ciascuna delle seguenti espressioni, indipendentemente dal valore di `x`, dire se risulta sempre `True` :

1. `x in x`
2. `x in [x]`
3. `x not in []`
4. `x in [[x]]`
5. `x in [[x][0]]`
6. `(x and y) in [x,y]`
7. `x in [x,y] and y in [x,y]`

Esercizio - ortaggi

Date le lista `ortaggi` di esattamente 5 stringhe e la lista di stringhe `frutta`, MODIFICARE la variabile `ortaggi` in modo che in ogni cella vi sia `True` se l'ortaggio è un frutto oppure `False` altrimenti.

- il tuo codice deve funzionare con qualsiasi lista di 5 stringhe `ortaggi` e qualsiasi lista `frutta`

Esempio - dati:

```
ortaggi = ["carota",
           "cavolfiore",
           "mela",
           "melanzana",
           "anguria"]

frutta = ["anguria","banana","mela",]
```

dopo l'esecuzione del tuo codice deve stampare:

```
>>> print(ortaggi)
[False, False, True, False, True]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[88]:
```
ortaggi = ["carota",
           "cavolfiore",
           "mela",
           "melanzana",
           "anguria"]

frutta = ["anguria","banana","mela",]

# scrivi qui

ortaggi = [ortaggi[0] in frutta,
           ortaggi[1] in frutta,
           ortaggi[2] in frutta,
           ortaggi[3] in frutta,
           ortaggi[4] in frutta,
]

#print(ortaggi)
```

[88]:
```
ortaggi = ["carota",
           "cavolfiore",
           "mela",
           "melanzana",
           "anguria"]

frutta = ["anguria","banana","mela",]

# scrivi qui
```

4.13.11 Concatenazione di liste con +

Date due liste `la` e `lb`, possiamo concatenarle con l'operatore + che produce una NUOVA lista:

[89]:
```
la = [77,66,88]
lb = [99,55]

la + lb
```

[89]:
```
[77, 66, 88, 99, 55]
```

Nota bene che l'operatore + produce una NUOVA lista, quindi `la` ed `lb` sono rimaste immutate:

[90]:
```
print(la)
```

```
[77, 66, 88]
```

[91]:
```
print(lb)
```

```
[99, 55]
```

Vediamo meglio con Python Tutor:

[92]:
```
la = [77,66,88]
lb = [99,55]
lc = la + lb

print(la)
print(lb)
print(lc)

jupman.pytut()
```

```
[77, 66, 88]
[99, 55]
[77, 66, 88, 99, 55]
```

[92]:
```
<IPython.core.display.HTML object>
```

Esercizio - concatenazione

Scrivere del codice che date due liste `la` e `lb`, mette nella lista `lc` gli ultimi due elementi di `la` e i primi due elementi di `lb`

Esempio - date:

```
la = [18,26,30,45,55]
lb = [16,26,37,45]
```

dopo il tuo codice deve risultare:

```
>>> print(la)
[18, 26, 30, 45, 55]
>>> print(lb)
[16, 26, 37, 45]
>>> print(lc)
[45, 55, 37, 45]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[93]:
```
la = [18,26,30,45,55]
lb = [16,26,37,45]

# scrivi qui
lc = la[-2:] + lb[2:]
#print(la)
#print(lb)
#print(lc)
```

[93]:
```
la = [18,26,30,45,55]
lb = [16,26,37,45]

# scrivi qui
```

DOMANDA: Per ciascuna delle seguenti espressioni, provare a indovinare il risultato

1. `[6,7,8] + [9]`
2. `[6,7,8] + []`
3. `[] + [6,7,8]`
4. `[] + []`
5. `[] + [[]]`
6. `[[]]+[]`
7. `[[]]+[[]]`
8. `([6] + [8])[0]`
9. `([6] + [8])[1]`
10. `([6] + [8])[2:]`
11. `len([4,2,5])+len([3,1,2])`
12. `len([4,2,5] + [3,1,2])`
13. `[5,4,3] + "3,1"`
14. `[5,4,3] + "[3,1]"`

15. ```
"[5,4,3]" + "[3,1]"
```

16. ```
["4","1","7"] + ["3","1"]
```

17. ```
list('coca') + ['c','o','l','a']
```

### 4.13.12 min e max

Una lista è una sequenza di elementi, e come tale la possiamo passare alla funzione `min` o `max` per trovare rispettivamente il minimo o massimo elemento della lista.

```
[94]: min([4,5,3,7,8,6])
[94]: 3
```

```
[95]: max([4,5,3,7,8,6])
[95]: 8
```

ATTENZIONE: MAI CHIAMARE VARIABILI `min` O `max`

Facendolo, finiresti per sovrascrivere le funzioni e al momento di chiamarle otterresti errori molto strani !

Notare che è anche possibile passare direttamente a `min` e `max` gli elementi da comparare senza includerli in una lista:

```
[96]: min(4,5,3,7,8,6)
[96]: 3
```

```
[97]: max(4,5,3,7,8,6)
[97]: 8
```

Ma se ne passiamo uno solo, senza includerlo in una lista, otterremo un errore:

```
min(4)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-156-bb3db472b52e> in <module>
----> 1 min(4)

TypeError: 'int' object is not iterable
```

L'errore ci comunica che quando passiamo un solo argomento, Python si aspetta che sia una lista:

```
[98]: min([4])
[98]: 4
```

A `min` e `max` possiamo anche passare stringhe, e ci verrà tornato il carattere alfabeticamente rispettivamente minore o maggiore :

```
[99]: min("sportello")
```
```

[99]: 'e'

[100]:
```
max("sportello")
```

[100]: 't'

Se passiamo una lista di stringhe, otterremo la stringa minore o maggiore in ordine lessicografico (per intenderci, quello dell'elenco telefonico):

[101]:
```
min(['il','marinaio','cammina','per','le','vie','del','porto'])
```

[101]: 'cammina'

[102]:
```
max(['il','marinaio','cammina','per','le','vie','del','porto'])
```

[102]: 'vie'

DOMANDA: Per ciascuna delle seguenti espressioni, prova a indovinare il risultato (o se da un errore)

1. `max(7)`
2. `max([7])`
3. `max([5,4,6,2])`
4. `max([min([7,3])])`
5. `max([])`
6. `max(2,9,3)`
7. `max([3,2,5] + [9,2,3])`
8. `max(max([3,2,5], max([9,2,3]))`
9. `max(min(3,6), min(8,2))`
10. `min(max(3,6), max(8,2))`
11. `max(['a','b','d','c'])`
12. `max(['barca', 'dado', 'aloa', 'cerchio'])`
13. `min(['prova','','z','v'])`
14. `max(['martello'[-1],'cacciavite'[-1],'brugola'[-1]])`
15. `min(['martello'[-1],'cacciavite'[-1],'brugola'[-1]])`

4.13.13 sum

Con `sum` possiamo sommare tutti gli elementi di una lista:

```
[103]: sum([1,2,3])
[103]: 6
```

```
[104]: sum([1.0, 2.0, 0.14])
[104]: 3.14
```

ATTENZIONE: MAI CHIAMARE VARIABILI `sum`

Facendolo, finiresti per sovrascrivere la funzione e al momento di chiamarle otterresti errori molto strani !

DOMANDA: Per ciascuna delle seguenti espressioni, prova a indovinare il risultato (o se da un errore)

1. `sum[3,1,2]`
2. `sum(1,2,3)`
3.
```
la = [1,2,3]
sum(la) > max(la)
```
4.
```
la = [1,2,3]
sum(la) > max(la)*len(la)
```
5.
```
la = [4,2,6,4,7]
lb = [max(la), min(la), max(la)]
print(max(lb) != max(la))
```

Esercizio - bilancia

Data una lista di `n` numeri `bilancia` con `n` pari, scrivere del codice che stampa `True` se la somma di tutti i primi `n/2` numeri è uguale alla somma di tutti i successivi

- il tuo codice deve funzionare per *qualunque* lista di numeri

Esempio 1 - dati:

```
bilancia = [4,3,7,1,5,8]
```

dopo il tuo codice, deve stampare

```
True
```

Esempio 2 - dati:

```
bilancia = [4,3,3,1,9,8]
```

dopo il tuo codice, deve stampare

```
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[105]:
```
bilancia = [4,3,7,1,5,8]
#bilancia = [4,3,3,1,9,8]

# scrivi qui
n = len(bilancia)
sum(bilancia[:n//2]) == sum(bilancia[n//2:])
```

[105]:
```
True
```

[105]:
```
bilancia = [4,3,7,1,5,8]
#bilancia = [4,3,3,1,9,8]

# scrivi qui
```

[105]:
```
True
```

4.13.14 Moltiplicazione per liste

Per replicare gli elementi di una lista, è possibile usare l'operatore `*` che produce una NUOVA lista:

[106]:
```
[7,6,8] * 2
```

[106]:
```
[7, 6, 8, 7, 6, 8]
```

[107]:
```
[7,6,8] * 3
```

[107]:
```
[7, 6, 8, 7, 6, 8, 7, 6, 8]
```

Notare che viene prodotta una NUOVA lista, e quella originale non viene modificata:

[108]:
```
la = [7,6,8]
```

[109]:
```
lb = [7,6,8] * 3
```

[110]:
```
la   # originale
```

[110]:
```
[7, 6, 8]
```

[111]:
```
lb   # risultato dell'espressione
```

[111]:
```
[7, 6, 8, 7, 6, 8, 7, 6, 8]
```

Possiamo moltiplicare una lista di stringhe:

[112]:
```
la = ["un", "mondo", "di", "parole"]
```

[113]:
```
lb = la * 2
```

[114]:
```
print(la)
```

```
['un', 'mondo', 'di', 'parole']
```

[115]:
```
print(lb)
```

```
['un', 'mondo', 'di', 'parole', 'un', 'mondo', 'di', 'parole']
```

Finchè moltiplichiamo liste che contengono elementi immutabili come numeri o stringhe, non si presentano particolari problemi:

[116]:
```
la = ["un", "mondo", "di", "parole"]
lb = la * 2

jupman.pytut()
```

[116]:
```
<IPython.core.display.HTML object>
```

La questione diventa molto più complicata quando moltiplichiamo liste che contengono oggetti mutabili come altre liste. Vediamo un esempio:

[117]:
```
la = [5,6]
lb = [7,8,9]
lc = [la,lb] * 2
```

[118]:
```
print(la)
```

```
[5, 6]
```

[119]:
```
print(lb)
```

```
[7, 8, 9]
```

[120]:
```
print(lc)
```

```
[[5, 6], [7, 8, 9], [5, 6], [7, 8, 9]]
```

Stampando, vediamo che le liste `la` e `lb` si ripresentano all'interno di `lc` - ma esattamente, come? Le `print` possono ingannare sull'effettivo stato della memoria - per indagare meglio conviene usare Python Tutor:

[121]:
```
la = [5,6]
lb = [7,8,9]
lc = [la,lb] * 2

jupman.pytut()
```

[121]:
```
<IPython.core.display.HTML object>
```

Arggh ! Vedrai apparire una giungla di frecce ! Questo perchè quando scriviamo `[la, lb]` creiamo una lista con due *riferimenti* alle liste `[5,6]` e `[7,8,9]`, e l'operatore `*` mentre duplica copia *i riferimenti*.

Per il momento ci fermiamo qui, vedremo meglio le implicazioni in seguito nel tutorial matrici - liste di liste[157].

[157] https://it.softpython.org/matrices-lists/matrices-lists-sol.html

4.13.15 Uguaglianza

Possiamo verificare se due liste sono uguali con l'operatore di uguaglianza ==, che date due liste ritorna `True` se contengono elementi uguali oppure `False` altrimenti.

```
[122]: [4,3,6] == [4,3,6]
[122]: True
```

```
[123]: [4,3,6] == [4,3]
[123]: False
```

```
[124]: [4,3,6] == [4,3,6, 'ciao']
[124]: False
```

```
[125]: [4,3,6] == [2,2,8]
[125]: False
```

Possiamo verificare l'uguaglianza di liste con elementi eterogenei:

```
[126]: ['mele', 3, ['ciliegie', 2], 6] == ['mele', 3, ['ciliegie', 2], 6]
[126]: True
```

```
[127]: ['banane', 3,['ciliegie', 2], 6] == ['mele', 3, ['ciliegie', 2], 6]
[127]: False
```

Per verificare la disuguaglianza, possiamo usare l'operatore `!=`:

```
[128]: [2,2,8] != [2,2,8]
[128]: False
```

```
[129]: [4,6,0] != [2,2,8]
[129]: True
```

```
[130]: [4,6,0] != [4,6,0,2]
[130]: True
```

DOMANDA: Per ciascuna delle seguenti espressioni, indovinare se è `True`, `False` o produce un errore

1. `[2,3,1] != [2,3,1]`
2. `[4,8,12] == [2*2,4*2,6*2]`
3. `[][:] == []`
4. `[[]] == [] + []`
5. `[[],[]] == [] + []`

6. `[[[]]] == [[[]+[]]]`

7. `[7,8][:] == [7,9-1]`

8. `[7][0] == [[7]][0]`

9. `[9] == [9][0]`

10. `[max(7,9)] == [max([7]),max([9])]`

11. `['a','b','c'] == ['A','B','C']`

12. `['a','b'] != ['a','b','c']`

13. `["ciao"] != ["CIAO".lower()]`

14. `[True in [True]] != [False]`

4.13.16 Prosegui

Trovi ulteriori esercizi nel foglio Liste 3[158]

```
[ ]:
```

4.14 Liste 3 - Metodi

4.14.1 Scarica zip esercizi

Naviga file online[159]

Le liste sono oggetti di tipo `list` e possiedono dei metodi che permettono di operare su di essi:

Metodo	Ritorna	Descrizione
list.append(obj)	`None`	Aggiunge un nuovo elemento alla fine della lista
list.extend(list)	`None`	Aggiunge diversi nuovi elementi alla fine della lista
list.insert(int,obj)	`None`	Aggiunge un nuovo elemento a qualche posizione data
list.remove(obj)	`None`	Rimuove la prima occorrenza di un elemento
list.pop()	obj	Rimuove e ritorna l'elemento all'ultima posizione
list.pop(int)	obj	Dato un indice, rimuove e ritorna l'elemento a quella posizione
list.reverse()	`None`	Inverte l'ordine degli elementi
list.sort	`None`	Ordina gli elementi
list.index(obj)	`int`	Trova la prima occorrenza di un elemento e ne ritorna la posizione
list.count(obj)	`int`	Conta le occorrenze di un elemento

[158] https://it.softpython.org/lists/lists3-sol.html
[159] https://github.com/DavidLeoni/softpython-it/tree/master/lists

ATTENZIONE: I METODI DELLE LISTE *MODIFICANO* LA LISTA SU CUI VENGONO CHIAMATI !

Quando chiami un metodo di una lista (l'oggetto a sinistra del punto `.`), MODIFICHI la lista stessa (diversamente dai metodi sulle stringhe che generano sempre una nuova stringa senza cambiare l'originale)

ATTENZIONE: I METODI DELLE LISTE *NON* RITORNANO NULLA!

Quasi sempre ritornano l'oggetto `None` (diversamente da quelli delle stringhe che ritornano sempre una nuova stringa)

4.14.2 Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
lists
    lists1.ipynb
    lists1-sol.ipynb
    lists2.ipynb
    lists2-sol.ipynb
    lists3.ipynb
    lists3-sol.ipynb
    lists4.ipynb
    lists4-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `lists3.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.14.3 Metodo append

Possiamo MODIFICARE una lista aggiungendo un elemento alla volta usando il metodo `append`.

Supponiamo di partire da una lista vuota:

[2]:
```
la = []
```

Se vogliamo aggiungere come elemento il numero `57`, possiamo scrivere così:

[3]:
```
la.append(57)
```

Notiamo che la lista che abbiamo creato inizialmente risulta MODIFICATA:

[4]:
```
la
```

[4]:
```
[57]
```

> **ATTENZIONE:** `la.append(57)` **non ha restituito NULLA !!!!**
>
> Guarda bene l'output della cella con l'istruzione `la.append(57)`, noterai che non c'è proprio niente. Questo perchè lo scopo di `append` è MODIFICARE la lista su cui viene chiamato, NON generare nuove liste.

Aggiungiamo un'altro numero *alla fine* della lista:

[5]:
```
la.append(96)
```

[6]:
```
la
```

[6]:
```
[57, 96]
```

[7]:
```
la.append(74)
```

[8]:
```
la
```

[8]:
```
[57, 96, 74]
```

Riguardiamoci cosa è successo in Python Tutor:

[9]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[10]:
```
la = []
la.append(57)
la.append(96)
la.append(74)

jupman.pytut()
```

[10]:
```
<IPython.core.display.HTML object>
```

Nota come ad espandersi sia sempre la stessa zona di memoria gialla associata alla variabile `la`.

Abbiamo detto che il metodo `append` non ritorna nulla, cerchiamo di specificare meglio. Nella tabella dei metodi, è presente una colonna chiamata *Ritorna*. Se vai a vedere, per quasi tutti i metodi incluso `append` è indicato che viene ritornato `None`.

None è l'oggetto più noioso di Python, perchè letteralmente significa niente. Cosa si può fare con niente? Ben poco, così poco che Jupyter quando si ritrova come risultato un oggetto None non lo stampa nemmeno. Proviamo a inserire direttamente None in una cella, vedrai che non verrà riportato nell'output della cella:

[11]:
```
None
```

Un modo per forzare la stampa è usare il comando print:

[12]:
```
print(None)
```

```
None
```

ESERCIZIO: Qual'è il tipo dell'oggetto None? Scoprilo usando la funzione type

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[13]:
```
# scrivi qui

#type(None)
```

[13]:
```
# scrivi qui
```

Proviamo a ripetere cosa succede con append. Se chiami il metodo append su una lista, append MODIFICA silenziosamente la lista, e RITORNA come risultato di essere stato chiamato l'oggetto None. Dato che Jupyter ritiene questo oggetto non interessante, non lo stampa nemmeno come risultato.

Cerchiamo di esplicitare meglio questo misterioso None. Se è vero che append lo produce come risultato di essere chiamato, vuol dire che possiamo associare questo risultato a qualche variabile. Proviamo ad associarlo alla variabile x:

[14]:
```
la = []
x = la.append(78)
```

Ora, se tutto è andato come abbiamo scritto, append dovrebbe aver modificato la lista:

[15]:
```
la
```

[15]:
```
[78]
```

e alla variabile x dovrebbe essere associato None. Quindi, se chiediamo a Jupyter di mostrare il valore associato ad x e se quel valore è None, non dovremmo vedere nulla:

[16]:
```
x
```

notiamo che non c'è nessun output nella cella, pare che siamo davvero in presenza di None. Forziamo la stampa con il comando print:

[17]:
```
print(x)
```

```
None
```

Eccolo ! Probabilmente sarai un po' confuso da tutto ciò, quindi proviamo a rivedere bene che succede in Python Tutor:

[18]:
```
la = []
x = la.append(78)
print("la è", la)
print("x è ", x)

jupman.pytut()
```

```
la è [78]
x è  None
```

[18]:
```
<IPython.core.display.HTML object>
```

Qual'è il succo di tutto questo discorso?

RIUSARE IL RISULTATO DI CHIAMATE AI METODI DELLE LISTE E' QUASI SEMPRE UN ERRORE !!!!

Dato che chiamare i metodi delle lista ci ritorna `None`, che è un oggetto 'inutile', tentare di riusarlo produrrà quasi sicuramente un errore

ESERCIZIO: Costruisci una lista aggiungendo un elemento alla volta con il metodo `append`. Aggiungi gli elementi `77`, `"prova"`, `[60,93]` con tre chiamate ad `append`, ed infine stampa la lista.

Dopo il tuo codice, dovresti vedere `[77, 'prova', [60, 93]]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
la = []

# scrivi qui
la.append(77)
la.append("prova")
la.append([60, 93])

#print(la)
```

[19]:
```
la = []

# scrivi qui
```

DOMANDA: Il codice seguente:

```
la = []
la.append(85,70,94)
```

1. produce un errore (quale?)
2. modifica la lista (come?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 1: `append` accetta un solo argomento, passandone di più produrrà un errore, prova ad eseguire il codice in una cella per vedere quale.

DOMANDA: Il codice seguente

```
la = []
la.append(87).append(96)
```

1. produce un errore
2. aggiunge a `la` i numeri `87` e `96`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 1: produce un errore, perchè come abbiamo detto la chiamata ad `la.append(87)` MODIFICA la lista `la` su cui è chiamato e ritorna il valore `None`. Se su `None` proviamo a chiamare `.append(96)`, visto che `None` non è una lista otterremo un messaggio di errore. Sincerati di questo usando Python Tutor.

DOMANDA: torniamo brevemente alle stringhe. Guarda il codice seguente (se non ti ricordi cosa fanno i metodi delle stringhe guarda qua[160]):

```
sa = '    trento    '
sb = sa.strip().capitalize()
print(sb)
```

1. produce un errore (quale?)
2. cambia `sa` (come?)
3. stampa qualcosa (cosa?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 3: stampa `Trento`. Diversamente dalle liste, le stringhe sono sequenze *immutabili*: significa che quando chiami un metodo sulle stringhe sei sicuro che RESTITUIRA' una stringa NUOVA. Quindi la prima chiamata a `sa.strip()` RESTITUISCE la stringa senza spazi all'inizio e alla fine `'trento'`, e su questa stringa viene chiamato il metodo `capitalize()` che rende il primo carattere maiuscolo.

Se questo non ti è chiaro, prova ad eseguire il codice seguente in Python Tutor. E' equivalente a quello dell'esempio ma esplicita il passaggio assegnando alla variabile extra `x`. il risultato della chiamata a `sa.strip()`

```
sa = '    trento    '
x = sa.strip()
sb = x.capitalize()
print(sb)
```

DOMANDA: Guarda questo codice. Stamperà qualcosa alla fine? O produrrà un errore?

[160] https://it.softpython.org/strings/strings3-sol.html#Metodi

```
la = []
lb = []
la.append(lb)

lb.append(98)
lb.append(77)

print(la)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Stamperà `[[98, 77]]`, perchè abbiamo messo `lb` dentro `la`.

Anche se con la prima `append` abbiamo inserito `lb` come primo elemento di `la`, dopo è perfettamente lecito continuare a modificare `lb` chiamando `lb.append(98)`.

Prova ad eseguire il codice in Python Tutor, e guarda le frecce.

Esercizio - accrescere una lista 1

Data la lista `la` di *dimensione fissa 7*, scrivi del codice per crescere la lista vuota `lb` così che contenga *solo* gli elementi di `la` a indici pari (0, 2, 4, …).

- Il tuo codice dovrebbe funzionare per qualunque lista `la` di dimensione fissa 7

```
#   0 1 2 3 4 5 6
la=[8,4,3,5,7,3,5]
lb=[]
```

Dopo il tuo codice, dovresti ottenere:

```
>>> print(lb)
[8,3,7,5]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[20]:
```
#   0 1 2 3 4 5 6
la=[8,4,3,5,7,3,5]
lb=[]

# scrivi qui
lb.append(la[0])
lb.append(la[2])
lb.append(la[4])
lb.append(la[6])
print(lb)
```

```
[8, 3, 7, 5]
```

[20]:
```
#   0 1 2 3 4 5 6
la=[8,4,3,5,7,3,5]
lb=[]

# scrivi qui
```

```
[8, 3, 7, 5]
```

4.14.4 Metodo extend

Prima con `append` abbiamo visto come accrescere una lista *un elemento alla volta*.

E se volessimo aggiungere in un colpo solo parecchi elementi, magari presi da un'altra lista? Come potremmo fare?

Dovremmo usare il metodo `extend`, che MODIFICA la lista su cui è chiamato aggiungendo tutti gli elementi trovati nella sequenza presa in input.

[21]:
```
la = [78,60,59]
```

[22]:
```
lb = [68,97,67,98]
```

[23]:
```
la.extend(lb)
```

[24]:
```
la
```

[24]:
```
[78, 60, 59, 68, 97, 67, 98]
```

[25]:
```
lb
```

[25]:
```
[68, 97, 67, 98]
```

Nell'esempio qua sopra, `extend` è chiamato sulla variabile `la`, e come parametro gli abbiamo passato `lb`

> **ATTENZIONE**: `la` è MODIFICATA, invece la sequenza che gli abbiamo passato tra le parentesi tonde no (`lb` nell'esempio)!

DOMANDA: l'esecuzione del metodo `extend` ritorna qualcosa? Cosa vedi nell'output della cella `la.extend(lb)` ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: `extend`, come quasi tutti i metodi delle liste, non ritorna nulla, o meglio, ritorna il poco utile oggetto `None`, che non viene nemmeno stampato da Jupyter.

Verifichiamo meglio cosa è successo con Python Tutor:

[26]:
```
la = [78,60,59]
lb = [68,97,67,98]
la.extend(lb)

jupman.pytut()
```

[26]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Guarda questo codice. Quali saranno i valori associati alle variabili `la` ,`lb` e `x` dopo la sua esecuzione?

```
la = [34,79,54]
lb = [86,45]
x = la.extend(lb)

print('la è ', la)
print('lb è ', lb)
print('x è ', x)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Verrà stampato questo:

```
la è  [34, 79, 54, 86, 45]
lb è  [86, 45]
x è  None
```

`la` è stata MODIFICATA aggiungendo tutti gli elementi di `lb`.

La chiamata ad `extend`, come quasi tutti i metodi delle liste, ha ritornano l'oggetto `None` che è stato associato alla variabile `x`. Cerca di capire bene cosa è successo usando Python Tutor.

Estendere con sequenze

Abbiamo detto che tra le parentesi tonde `extend` può prendere una sequenza generica, non solo liste. Questo vuol dire che possiamo anche passargli una stringa. Per esempio:

[27]:
```
la = [78,65,87]

s = "ciao"

la.extend(s)
```

[28]:
```
la
```

[28]:
```
[78, 65, 87, 'c', 'i', 'a', 'o']
```

Dato che stringa è una sequenza di caratteri, `extend` ha preso ciascuno di questi elementi e li ha aggiunti a `la`

DOMANDA: il valore associato alla variabile `s` è stato modificato?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: assolutamente impossibile, perchè a) `extend` modifica solo la lista su cui è chiamato e b) le stringhe sono comunque immutabili.

DOMANDA: Il codice seguente:

```
la = [78,65]
la.extend(68,85,87)
```

1. produce un errore (quale?)
2. modifica `la` (come?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 1: produce un errore, perchè ad `extend` dobbiamo passare UN parametro solo, che deve essere *una sequenza*. Qua invece stiamo passando tanti parametri. Un'alternativa potrebbe essere costruire una lista così:

```
la = [78,65]
la.extend([68,85,87])
```

DOMANDA: Se questo codice viene eseguito, che succede?

```
sa = "ciao"
sb = "mondo"
sa.extend(sb)
```

1. `sa` viene modificata (come?)
2. otteniamo un errore (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 2: otteniamo un errore, perchè `extend` è un metodo esclusivo delle liste. Appartiene solo alle liste perchè MODIFICA l'oggetto su cui è chiamato - dato che le stringhe sono oggetti immutabili, non avrebbe senso estenderle.

DOMANDA: Se questo codice viene eseguito, che succede?

```
la = [1,2,3]
lb = [4,5]
lc = [6,7,8]

la.extend(lb).extend(lc)
```

1. `la` diventa `[1,2,3,4,5,6,7,8]`
2. un errore (quale?)
3. `la` diventa `[1,2,3,4,5]` e un errore(quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 3: `la` diventa `[1,2,3,4,5]` e subito dopo otteniamo un errore, perchè la chiamata a `la.extend(lb)` MODIFICA `la` a `[1,2,3,4,5]` e RITORNA il valore `None`. A quel punto, Python cerca di chiamare il metodo `extend` sull'oggetto `None`, ma non essendo una lista, ci becchiamo questo errore (**per convincerti, verifica il tutto il Python Tutor** !!!)

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-45-0a08a154ada4> in <module>
      3 lc = [6,7,8]
      4
----> 5 la.extend(lb).extend(lc)

AttributeError: 'NoneType' object has no attribute 'extend'
```

Esercizio: accrescere una lista 2

Date due *liste* `la` ed `lb` e un elemento `x`, scrivi del codice che MODIFICA `la` in modo che `la` contenga alla fine l'elemento `x` seguito da tutti gli elementi di `lb`

- **NOTA 1**: il tuo codice dovrebbe funzionare con qualunque `la` ed `lb`
- **NOTA 2**: `id` è una funzione di Python che associa ad ogni regione di memoria un identificativo numerico univoco. Se provi a stampare `id(la)` *prima* di modificare `la` e `id(la)` dopo, dovresti ottenere *esattamente* lo stesso id. Se ne ottieni uno diverso, significa che hai generato una lista interamente nuova. In ogni caso, verifica che funzioni in Python Tutor.

```
la = [5,9,2,4]
lb = [7,1,3]
x = 8
```

Dovresti ottenere:

```
>>> print(la)
[5,9,2,4,8,7,1,3]
>>> print(lb)
[7,1,3]
>>> print(x)
8
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[29]:
```
la = [5,9,2,4]
lb = [7,1,3]
x = 8

# scrivi qui
la.append(x)
la.extend(lb)
#print(la)
#print(lb)
#print(x)
```

[29]:
```
la = [5,9,2,4]
lb = [7,1,3]
x = 8

# scrivi qui
```

Esercizio - zslice

Scrivi del codice che date due liste `la` (di almeno 3 elementi) e `lb`, MODIFICA `lb` in modo che vi siano aggiunti i primi 3 elementi di `la` seguiti dagli ultimi 3 elementi di `la`

- il tuo codice deve funzionare con qualsiasi lista
- usa `extend` e le slice

```
la = ['a','b','c','d','e','f','g','h','i','l','m','n','o']
lb = ['z']
```

Dovresti ottenere:

```
>>> print(la)
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'l', 'm', 'n', 'o']
>>> print(lb)
['z', 'a', 'b', 'c', 'm', 'n', 'o']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[30]:
```
la = ['a','b','c','d','e','f','g','h','i','l','m','n','o']
lb = ['z']

# scrivi qui

lb.extend(la[:3])  # una slice genera una lista
lb.extend(la[-3:])

#print(la)
#print(lb)
```

[30]:
```
la = ['a','b','c','d','e','f','g','h','i','l','m','n','o']
lb = ['z']

# scrivi qui
```

Esercizio - vedunazeblag

Scrivi del codice che data una lista di tre stringhe `parole` e una lista vuota `la`, riempie `la` con tutti i primi 3 caratteri di ogni stringa in `parole`.

- il tuo codice deve funzionare con qualsiasi lista di 3 stringhe
- usa le slice

Esempio - data:

```
parole = ["vedo", "una", "zebra", "laggiù"]
la = []
```

il tuo codice deve mostrare

```
>>> print(t)
['v', 'e', 'd', 'u', 'n', 'a', 'z', 'e', 'b', 'l', 'a', 'g']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[31]:
```
parole = ["vedo", "una", "zebra", "laggiù"]

la = []

# scrivi qui
la.extend(parole[0][:3])
la.extend(parole[1][:3])
la.extend(parole[2][:3])
la.extend(parole[3][:3])
print(la)
```

```
['v', 'e', 'd', 'u', 'n', 'a', 'z', 'e', 'b', 'l', 'a', 'g']
```

[31]:
```
parole = ["vedo", "una", "zebra", "laggiù"]

la = []

# scrivi qui
```

```
['v', 'e', 'd', 'u', 'n', 'a', 'z', 'e', 'b', 'l', 'a', 'g']
```

4.14.5 join - costruire stringhe da liste

Data una stringa che funge da separatore, e una sequenza come per esempio una lista `la` contenente solo stringhe, è possibile concatenarle in una sola stringa (nuova) con il metodo `join`:

[32]:
```
la = ['Quando', 'fuori', 'piove']

'SEPARATORE'.join(la)
```

[32]:
```
'QuandoSEPARATOREfuoriSEPARATOREpiove'
```

Come separatore possiamo mettere qualunque carattere, come uno spazio:

[33]:
```
' '.join(la)
```

[33]:
```
'Quando fuori piove'
```

Nota che la lista originale non viene modificata:

[34]:
```
la
```

[34]:
```
['Quando', 'fuori', 'piove']
```

DOMANDA: Questo codice cosa produce?

```
''.join(['a','b','c']).upper()
```

1. un errore (quale?)
2. una stringa (quale?)
3. una lista (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 2: produce la stringa `'ABC'`: prima prende tutti i caratteri dalla lista `['a','b','c']` e li unisce separandoli con lo spazio vuoto `''` formando `'abc'`, poi questa stringa viene resa tutta maiuscola con `upper()`.

DOMANDA: Questa codice cosa produce?

```
'A'.join('porto')
```

1. una stringa (quale?)
2. un errore (quale?)
3. una lista (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 1: produce la stringa `'pAoArAtAo'` - abbiamo detto che join prende come input una sequenza, quindi non siamo vincolati a passargli liste ma possiamo anche passare direttamente una stringa, che è una sequenza di caratteri. `join` intervallerà quindi ogni carattere della stringa con il separatore che forniamo prima del punto.

DOMANDA: Questo codice cosa produce?

```
'\''.join('mmmm')
```

1. un errore (quale?)
2. una stringa (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 2: `\'` è una sequenza di escape che rappresenta il carattere singolo apice `'`, quindi otterremo `m'm'm'm`

DOMANDA: Data una stringa qualsiasi `s` e una lista di stringhe qualsiasi `la` di almeno due elementi, il seguente codice darà sempre lo stesso risultato - quale ? (pensaci, e se non sai rispondere prova a mettere dei valori a caso di `s` e `la`)

```
len(s) <= len(s.join(la))
```

1. un errore (quale?)
2. una stringa (quale?)
3. altro (cosa?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 3: il codice produrrà sempre il booleano `True` perchè `s.join(la)` produce una stringa contenente tutte le stringhe in `la` intervallate dalla stringa `s`. Quindi la lunghezza di questa stringa sarà sempre uguale o superiore alla lunghezza di `s`: comparando le due lunghezze con l'operatore `<=` otterremo sempre il booleano `True`.

Esempio

```
s = "ab"
la = ['uief','cb','sd']
len(s) <= len(s.join(la))
```

ESERCIZIO - dub dab dib dob

Scrivi del codice che data una lista di stringhe `la`, associa alla variabile `s` una stringa con le stringhe concatenate separate da virgole e uno spazio

Esempio:

Data

`la = ['dub', 'dab','dib', 'dob']`

dopo il tuo codice, dovresti ottenere questi risultati:

```
>>> print(s)
dub, dab, dib, dob
>>> len(s)
18
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[35]:
```
la = ['dub', 'dab','dib', 'dob']

# scrivi qui
```

(continues on next page)

(continued from previous page)

```
s = ', '.join(la)

#print(s)
#len(s)
```

[35]:
```
la = ['dub', 'dab','dib', 'dob']

# scrivi qui
```

Esercizio - ghirigori

Data una lista di stringhe `la` e una lista di tre separatori `seps`, scrivi del codice che stampa gli elementi di `la` separati dal primo separatore, seguiti dal secondo separatore, seguiti dagli elementi di `la` separati dal terzo separatore.

- il tuo codice deve funzionare con qualunque lista `la` e `seps`

Esempio: dati

```
la = ['ghi','ri','go','ri']
seps = [',','_','+']
```

Dopo il tuo codice, deve stampare:

```
ghi,ri,go,ri_ghi+ri+go+ri
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[36]:
```
la = ['ghi','ri','go','ri']
seps = [',','_','+']

# scrivi qui

#print(seps[0].join(la) + seps[1] + seps[2].join(la))
```

[36]:
```
la = ['ghi','ri','go','ri']
seps = [',','_','+']

# scrivi qui
```

Esercizio - welldone

Data la lista

```
la = ["walnut", "eggplant", "lemon", "lime", "date", "onion", "nectarine", "endive" ]:
```

1. Crea un'altra lista (chiamala `nuova`) contenente il primo carattere di ogni elemento di `la`
2. Aggiungi uno spazio a `nuova` all aposizione 4 e attacca un punto esclamativo (`'!'`) alla fine
3. Stampa la lista
4. Stampa il contenuto della lista unendo tutti gli elementi con uno spazio vuoto (per es usa il metodo `join`: `"".join(nuova)`)

Dovresti ottenere:

```
['w', 'e', 'l', 'l', ' ', 'd', 'o', 'n', 'e', '!']

 well done!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[37]:
```
la = ["walnut", "eggplant", "lemon", "lime", "date", "onion", "nectarine", "endive" ]

# scrivi qui

nuova = []
nuova.append(la[0][0])
nuova.append(la[1][0])
nuova.append(la[2][0])
nuova.append(la[3][0])
nuova.append(la[4][0])
nuova.append(la[5][0])
nuova.append(la[6][0])
nuova.append(la[7][0])

nuova.insert(4," ")
nuova.append("!")

print(nuova)
print("\n", "".join(nuova))
```

```
['w', 'e', 'l', 'l', ' ', 'd', 'o', 'n', 'e', '!']

 well done!
```

[37]:
```
la = ["walnut", "eggplant", "lemon", "lime", "date", "onion", "nectarine", "endive" ]

# scrivi qui
```

```
['w', 'e', 'l', 'l', ' ', 'd', 'o', 'n', 'e', '!']

 well done!
```

4.14.6 Metodo insert

`insert` MODIFICA la lista inserendo un elemento ad uno specifico indice - tutti gli elementi a partire da quell'indice vengono spostati in avanti di una posizione

[38]:
```
     #0 1 2 3
la = [6,7,8,9]
```

[39]:
```
la.insert(2,55)  # inserisce il numero 55 all'indice 2
```

[40]:
```
la
```

[40]:
```
[6, 7, 55, 8, 9]
```

[41]:
```
la.insert(0,77)  # inserisce il numero 77 all'indice 0
```

[42]:
```
la
```

[42]:
```
[77, 6, 7, 55, 8, 9]
```

Possiamo inserire dopo la fine:

[43]:
```
la.insert(6,88)  # inserisce il numero 88 all'indice 6
```

[44]:
```
la
```

[44]:
```
[77, 6, 7, 55, 8, 9, 88]
```

Nota che se sforiamo con l'indice, l'elemento viene comunque messo alla fine e non vengono create celle vuote:

[45]:
```
la.insert(1000,99)  # in questo caso inserisce il numero 99 all'indice 7
```

DOMANDA: Data una lista qualsiasi `x`, questo codice cosa produce? Possiamo riscriverlo in un'altra maniera?

```
x.insert(len(x),66)
```

1. produce una nuova lista (quale?)
2. modifica `x` (come?)
3. un errore

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 2 - il codice MODIFICA la lista `x` aggiungendo alla lista l'elemento `66` alla fine. Il codice è pertanto equivalente al codice

```
x.append(66)
```

DOMANDA: Il seguente codice, cosa produce?

```
la = [3,4,5,6]
la.insert(0,[1,2])
print(la)
```

1. stampa [1,2,3,4,5,6]
2. un errore (quale?)
3. qualcos'altro (cosa?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 3 - il codice inserisce come zeresimo elemento di `la` la lista `[1,2]`. La stampa produrrà quindi `[[1, 2], 3, 4, 5, 6]`

DOMANDA: Il seguente codice cosa produce?

```
la = [4,5,6]
la.insert(0,1,2,3)
print(la)
```

1. stampa `[1,2,3,4,5,6]`
2. un errore (quale?)
3. qualcos'altro (cosa?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 2 - un errore, a `insert` possiamo passare solo 2 parametri, l'indice di inserimento e il singolo oggetto da inserire

DOMANDA: Il seguente codice cosa produce?

```
la = [4,5,6]
lb = la.insert(0,3)
lc = lb.insert(0,2)
ld = lc.insert(0,1)
print(ld)
```

1. stampa `[1,2,3,4,5,6]`
2. un errore (quale?)
3. qualcos'altro (cosa?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 2 - un errore: come quasi tutti i metodi delle liste, `insert` ritorna `None`, quindi scrivendo `lb = la.insert(0,3)` andiamo ad associare `None` a `lb`, e quando nella riga successiva Python incontra `lc = lb.insert(0,2)` e prova ad eseguire `None.insert(0,2)` si lamenterà perchè `None` non essendo una lista non ha il metodo `insert`.

Esercizio - insertando

Data la lista:

`la = [7,6,8,5,6]`

scrivi del codice che la MODIFICA usando solo chiamate a `insert`. Dopo il tuo codice, `la` deve apparire così:

```
>>> print(la)
[7, 77, 99, 6, 8, 88, 5, 6, 55]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[46]:
```
la = [7,6,8,5,6]

# scrivi qui

la.insert(3,88)
la.insert(1,99)
la.insert(1,77)
la.insert(len(la),55)

#print(la)
```

[46]:
```
la = [7,6,8,5,6]

# scrivi qui
```

ATTENZIONE: chiamare `insert` è molto più lento di `append` !!

Una chiamata ad `insert` riscrive tutte le celle successive a quella dell'inserimento, mentre invece `append` aggiunge una cella e basta. Dato che il computer è veloce, molto spesso non ci si accorge della differenza, ma quando possibile, e specialemente se devi scrivere programmi che operano su grandi quantità di dati, prova a scrivere il codice usando `append` invece di `insert`.

Esercizio - barzoletta

Data la stringa:

```
sa = 'barzoletta'
```

scrivi del codice che crea una NUOVA stringa `sb` cambiando la stringa originale in modo che risulti:

```
>>> print(sb)
'barzelletta'
```

- **USA** il metodo `insert` e riassegnazione di celle

- **NOTA**: non puoi usarle su una stringa, perchè è IMMUTABILE - dovrai quindi prima convertire la stringa in una lista

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[47]:
```
sa = 'barzoletta'

# scrivi qui

la = list(sa)
la[4] = 'e'
la.insert(5,'l')
sb = ''.join(la)
#print(sb)
```

[47]:
```
sa = 'barzoletta'

# scrivi qui
```

Esercizio - insappend

Questo codice prende come input una lista vuota `la` e una lista di numeri `lb`. Cerca di capire cosa fa, e riscrivilo usando degli `append`.

[48]:
```
la = []
lb = [7,6,9,8]
la.insert(0,lb[0]*2)
la.insert(0,lb[1]*2)
la.insert(0,lb[2]*2)
la.insert(0,lb[3]*2)
print(la)
```

```
[16, 18, 12, 14]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[49]:
```
la = []
lb = [7,6,9,8]

# scrivi qui
la.append(lb[-1]*2)
la.append(lb[-2]*2)
la.append(lb[-3]*2)
la.append(lb[-4]*2)
#print(la)
```

[49]:
```
la = []
lb = [7,6,9,8]

# scrivi qui
```

4.14.7 Metodo remove

`remove` prende come parametro un oggetto, cerca la PRIMA cella che contiene quell'oggetto e la elimina:

[50]:
```
#     0 1 2 3 4 5
la = [6,7,9,5,9,8]   # il 9 è alla cella con indice 2 e 4
```

[51]:
```
la.remove(9)   # cerca la prima cella contenente il numero 9
```

[52]:
```
la
```

[52]:
```
[6, 7, 5, 9, 8]
```

Come si può vedere, la cella che era all'indice 2 e che conteneva la PRIMA occorrenza di 9 è stata eliminata. La cella contenente la SECONDA occorrenza di 9 invece è ancora lì.

Se si cerca di rimuovere un oggetto non presente, si riceve un errore:

```
la.remove(666)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-121-5d04a71f9d33> in <module>
----> 1 la.remove(666)

ValueError: list.remove(x): x not in list
```

Esercizio - nob

Scrivi del codice che rimuove dalla lista `la` tutti i numeri contenuti nella lista di 3 elementi `lb`

- il tuo codice deve funzionare con qualsiasi lista `la` e `lb` di 3 elementi
- puoi assumere che `la` contenga esattamente DUE occorrenze di tutti gli elementi di `lb` (più eventuali altri numeri)

Esempio - dati:

```
lb = [8,7,4]
la = [7,8,11,8,7,4,5,4]
```

dopo il tuo codice deve risultare

```
>>> print(la)
[11, 5]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[53]:
```
lb = [8,7,4]
la = [7,8,11,8,7,4,5,4]

# scrivi qui

la.remove(lb[0])
la.remove(lb[0])
la.remove(lb[1])
la.remove(lb[1])
la.remove(lb[2])
la.remove(lb[2])
#print(la)
```

[53]:
```
lb = [8,7,4]
la = [7,8,11,8,7,4,5,4]

# scrivi qui
```

4.14.8 Metodo pop

Il metodo `pop` se chiamato senza argomenti rimuove l'ultimo elemento (MODIFICANDO la lista) e lo ritorna:

[54]:
```
cesta = ['melone','fragola', 'anguria']
```

[55]:
```
cesta.pop()
```

[55]:
```
'anguria'
```

[56]:
```
cesta
```

[56]:
```
['melone', 'fragola']
```

[57]:
```
cesta.pop()
```

[57]:
```
'fragola'
```

[58]:
```
cesta
```

[58]:
```
['melone']
```

Visto che l'ultimo elemento è *ritornato* dalla `pop`, possiamo assegnarlo ad una variabile:

[59]:
```
frutto = cesta.pop()
```

Nota che non vediamo più nessun risultato stampato perchè l'elemento ritornato è stato assegnato alla variabile `frutto`:

[60]:
```
frutto
```

[60]:
```
'melone'
```

Constatiamo anche che `cesta` è stata MODIFICATA:

[61]:
```
cesta
```

[61]:
```
[]
```

Chiamare ulteriormente `pop` su una lista vuota genera un'errore:

```
cesta.pop()
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-67-086f38c9fbc0> in <module>()
----> 1 cesta.pop()

IndexError: pop from empty list
```

[ ]:
```

```

Opzionalmente, per rimuovere un elemento ad una specifica posizione possiamo passare a `pop` un indice da 0 INCLUSO alla lunghezza della lista ESCLUSA:

[62]:
```
#           0            1              2         3
attrezzi = ['martello', 'cacciavite', 'pinza', 'martello']
```

[63]:
```
attrezzi.pop(2)
```

[63]:
```
'pinza'
```

[64]:
```
attrezzi
```

[64]:
```
['martello', 'cacciavite', 'martello']
```

DOMANDA: Guarda i frammenti di codice seguenti, e per ciascuno cerca di indovinare che risultato produce (o se risulta in un errore).

1.
```
la = ['a']
print(la.pop())
print(la.pop())
```

2.
```
la = [4,3,2,1]
print(la.pop(4))
print(la)
```

3.
```
la = [1,2,3,4]
print(la.pop(3))
print(la)
```

4.
```
la = [1,2,3,4]
print(la.pop(-1))
print(la)
```

5.
```
s = 'grezzo'
print(s.pop())
print(s)
```

6.
```
la = ['molto', 'grezzo']
print(la.pop())
print(la)
```

7.
```
la = ['a', ['a']]
print(la.pop())
print(la)
```

Esercizio - popcorn

Data una lista `corn` di esattamente 4 caratteri, scrivi del codice che trasferisce in ordine inverso tutti i caratteri da `corn` ad un'altra lista `scatola` che inizialmente è vuota.

- **NON** usare metodi come `reverse` o funzioni come `reversed`
- Il tuo codice deve funzionare con *qualsiasi* lista `corn` di 4 elementi

Esempio - date:

```
corn = ['t','o','r','o']
scatola = []
```

dopo il tuo codice, deve risultare:

```
>>> print(corn)
[]
>>> print(scatola)
['o','r','o','t']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[65]:
```
corn = ['t','o','r','o']
scatola = []

# scrivi qui

scatola.append(corn.pop())
scatola.append(corn.pop())
scatola.append(corn.pop())
scatola.append(corn.pop())
print(scatola)
```

```
['o', 'r', 'o', 't']
```

[65]:
```
corn = ['t','o','r','o']
scatola = []

# scrivi qui
```

```
['o', 'r', 'o', 't']
```

Esercizio - zonzo

Data una lista `la` contenente dei caratteri, e una lista `lb` contente esattamente due posizioni *ordinate in modo crescente* , scrivi del codice che elimina da `la` i caratteri alle posizioni specificate in `lb`.

- **ATTENZIONE**: chiamando la `pop` la prima volta MODIFICHERAI `la`, quindi l'indice del secondo elemento da eliminare andrà opportunamente aggiustato !
- **NON** creare nuove liste, quindi niente righe che iniziano con `la =`
- Il tuo codice deve funzionare per *qualsiasi* `la`, e *qualsiasi* `lb` da due elementi

Esempio - dati:

```
#       0   1   2   3   4
la = ['z','o','n','z','o']
lb = [2,4]
```

alla posizione 2 in `la` troviamo la `n` e alla 4 la `o`, quindi dopo il tuo codice dovrà risultare:

```
>>> print(la)
['z', 'o', 'z']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[66]:
```
#       0   1   2   3   4
la = ['z','o','n','z','o']
lb = [2,4]

# scrivi qui
la.pop(lb[0])
la.pop(lb[1]-1)
#print(la)
```

[66]:
```
'o'
```

[66]:
```
#       0   1   2   3   4
la = ['z','o','n','z','o']
lb = [2,4]

# scrivi qui
```

[66]:
```
'o'
```

4.14.9 Metodo reverse

Il metodo `reverse` MODIFICA la lista su cui è chiamato invertendo l'ordine degli elementi.

Vediamo un esempio:

[67]:
```
la = [7,6,8,4]
```

[68]:
```
la.reverse()
```

[69]:
```
la
```

[69]:
```
[4, 8, 6, 7]
```

ATTENZIONE: `reverse` **NON RITORNA NULLA!**

Per essere precisi, ritorna `None`

[70]:
```
lb = [7,6,8,4]
```

[71]:
```
x = lb.reverse()
```

[72]:
```
print(x)
```

```
None
```

[73]:
```
print(lb)
```

```
[4, 8, 6, 7]
```

DOMANDA: Il codice seguente che effetto produce?

```
s = "transatlantico"
s.reverse()
print(s)
```

1. un errore (quale?)
2. stampa la stringa rovesciata

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: `.reverse()` è un metodo presente SOLO nelle LISTE, quindi usandolo sulle stringhe stringhe si otterrà un errore. E c'è da attenderselo, visto che `reverse` MODIFICA l'oggetto su cui è chiamato e perchè le stringhe sono *immutabili* nessun metodo delle stringhe può modificare la stringa su cui è chiamato.

DOMANDA: Se `x` è una lista qualsiasi, che effetto produce il codice seguente?

```
x.reverse().reverse()
```

1. cambia la lista (come?)
2. non cambia la lista

3. genera un errore (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 3 - genera un errore, perchè `reverse()` ritorna `None` e `None` non essendo una lista non ha il metodo `reverse()`.

Esercizio - come va?

Scrivi del codice che date due liste `la` e `lb`, MODIFICA `la` aggiungendogli tutti gli elementi di `lb` e rovesciando poi l'intera lista.

- il tuo codice deve funzionare per qualunque `la` e `lb`
- **NON** modificare `lb`

Esempio - dati:

```
la = ['c','o','m','e']
lb = ['v','a','?']
```

Dopo il tuo codice, deve stampare:

```
>>> print('la=',la)
la= ['?', 'a', 'v', 'e', 'm', 'o', 'c']
>>> print('lb=',lb)
lb= ['v', 'a', '?']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[74]:
```
la = ['c','o','m','e']
lb = ['v','a','?']

# scrivi qui
la.extend(lb)
la.reverse()
#print('la=',la)
#print('lb=',lb)
```

[74]:
```
la = ['c','o','m','e']
lb = ['v','a','?']

# scrivi qui
```

4.14.10 Esercizio - cose preziose

Date due liste `la` e `lb`, scrivi del codice che STAMPA una lista con gli elementi di `la` e `lb` in ordine invertito

- **NON** modificare `la` e **NON** modificare `lb`
- il tuo codice deve funzionare per qualsiasi lista `la` e `lb`

Esempio - dati

```
la = ['c', 'o', 's', 'e']
lb = ['p', 'r', 'e', 'z', 'i', 'o', 's', 'e']
```

dopo il tuo codice deve stampare

```
['e', 's', 'o', 'i', 'z', 'e', 'r', 'p', 'e', 's', 'o', 'c']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[75]:
```
la = ['c', 'o', 's', 'e']
lb = ['p', 'r', 'e', 'z', 'i', 'o', 's', 'e']

# scrivi qui
lc = la + lb  # il + crea una NUOVA lista
lc.reverse()
print(lc)
```

```
['e', 's', 'o', 'i', 'z', 'e', 'r', 'p', 'e', 's', 'o', 'c']
```

[75]:
```
la = ['c', 'o', 's', 'e']
lb = ['p', 'r', 'e', 'z', 'i', 'o', 's', 'e']

# scrivi qui
```

```
['e', 's', 'o', 'i', 'z', 'e', 'r', 'p', 'e', 's', 'o', 'c']
```

4.14.11 Esercizio - potenze

Il codice seguente usa degli `insert` che come già detto non sono molto efficienti. Cerca di capire cosa fa, e riscrivilo usando solo `append` e `reverse`

- il tuo codice deve funzionare per qualsiasi valore di `x`

[76]:
```
x = 2
la = [x]
la.insert(0,la[0]*2)
la.insert(0,la[0]*2)
la.insert(0,la[0]*2)
la.insert(0,la[0]*2)
print(la)
```

```
[32, 16, 8, 4, 2]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[77]:
```
x = 2
la = [x]

# scrivi qui
la.append(la[-1]*2)
la.append(la[-1]*2)
la.append(la[-1]*2)
la.append(la[-1]*2)
la.reverse()
#print(la)
```

[77]:
```
x = 2
la = [x]

# scrivi qui
```

4.14.12 Metodo sort

Se una lista contiene elementi omogenei, è possibile ordinarla rapidamente con il metodo `sort`, che MODIFICA la lista su cui viene chiamato:

[78]:
```
la = [8,6,7,9]
```

[79]:
```
la.sort()  # NOTA: sort non ritorna nulla !!!
```

[80]:
```
la
```

[80]:
```
[6, 7, 8, 9]
```

Anche le stringhe sono ordinabili:

[81]:
```
lb = ['Boccaccio', 'Alighieri', 'Manzoni', 'Leopardi']
```

[82]:
```
lb.sort()
```

[83]:
```
lb
```

[83]:
```
['Alighieri', 'Boccaccio', 'Leopardi', 'Manzoni']
```

Una lista con elementi non comparabili tra loro non è ordinabile, e Python si lamenterà:

[84]:
```
lc = [3,4,'cavoli',7,'patate']
```

```
>>> lc.sort()

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

(continues on next page)

(continued from previous page)

```
<ipython-input-288-0cabfae30939> in <module>
----> 1 lc.sort()

TypeError: '<' not supported between instances of 'str' and 'int'
```

Esercizio - moltelinee

Dato la seguente stringa di testo:

```
"""Questa è una stringa
di testo su
diverse linee che non dice niente. """
```

1. stampala
2. stampa quante linee, parole e caratteri contiene
3. metti in ordine alfabetico le parole e stampale le prime e ultime in ordine lessicografico

Dovresti ottenere:

```
Questa è una stringa
di testo su
diverse linee che non dice niente.

Lines: 3 words: 13 chars: 67

['Q', 'u', 'e', 's', 't', 'a', ' ', 'è', ' ', 'u', 'n', 'a', ' ', 's', 't', 'r', 'i',
↪'n', 'g', 'a', '\n', 'd', 'i', ' ', 't', 'e', 's', 't', 'o', ' ', 's', 'u', '\n', 'd
↪', 'i', 'v', 'e', 'r', 's', 'e', ' ', 'l', 'i', 'n', 'e', 'e', ' ', 'c', 'h', 'e',
↪' ', 'n', 'o', 'n', ' ', 'd', 'i', 'c', 'e', ' ', 'n', 'i', 'e', 'n', 't', 'e', '.']
67

First word:  Questa
Last word: è
['Questa', 'che', 'di', 'dice', 'diverse', 'linee', 'niente.', 'non', 'stringa', 'su',
↪ 'testo', 'una', 'è']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[85]:
```
s = """Questa è una stringa
di testo su
diverse linee che non dice niente."""


# scrivi qui

# 1) stampa
print(s)
print("")

# 2) stampa le linee, parole e caratteri
lines = s.split('\n')
```

(continues on next page)

(continued from previous page)

```
# NOTA: le parole sono separate da uno spazio o da un ritorno a capo (newline)

words = lines[0].split(' ') + lines[1].split(' ') + lines[2].split(' ')
num_chars = len(s)
print("Lines:", len(lines), "words:", len(words), "chars:", num_chars)

# modo alternativo per numero di caratteri
print("")
characters = list(s)
num_chars2 = len(characters)
print(characters)
print(num_chars2)

# 3. ordina alfabeticamente le parole e stampa la prima e ultima in ordine␣
↪lessicografico

words.sort() # NOTA: non ritorna NIENTE !!!!
print("")
print("First word: ", words[0])
print("Last word:", words[-1])
print(words)
```

```
Questa è una stringa
di testo su
diverse linee che non dice niente.

Lines: 3 words: 13 chars: 67

['Q', 'u', 'e', 's', 't', 'a', ' ', 'è', ' ', 'u', 'n', 'a', ' ', 's', 't', 'r', 'i',
↪'n', 'g', 'a', '\n', 'd', 'i', ' ', 't', 'e', 's', 't', 'o', ' ', 's', 'u', '\n', 'd
↪', 'i', 'v', 'e', 'r', 's', 'e', ' ', 'l', 'i', 'n', 'e', 'e', ' ', 'c', 'h', 'e',
↪' ', 'n', 'o', 'n', ' ', 'd', 'i', 'c', 'e', ' ', 'n', 'i', 'e', 'n', 't', 'e', '.']
67

First word:  Questa
Last word: è
['Questa', 'che', 'di', 'dice', 'diverse', 'linee', 'niente.', 'non', 'stringa', 'su',
↪ 'testo', 'una', 'è']
```

[85]:
```
s = """Questa è una stringa
di testo su
diverse linee che non dice niente."""


# scrivi qui
```

```
Questa è una stringa
di testo su
diverse linee che non dice niente.

Lines: 3 words: 13 chars: 67

['Q', 'u', 'e', 's', 't', 'a', ' ', 'è', ' ', 'u', 'n', 'a', ' ', 's', 't', 'r', 'i',
↪'n', 'g', 'a', '\n', 'd', 'i', ' ', 't', 'e', 's', 't', 'o', ' ', 's', 'u', '\n', 'd
↪', 'i', 'v', 'e', 'r', 's', 'e', ' ', 'l', 'i', 'n', 'e', 'e', ' ', 'c', 'h', 'e',
↪' ', 'n', 'o', 'n', ' ', 'd', 'i', 'c', 'e', ' ', 'n', 'i', 'e', 'n', 't', 'e', '.']
```

(continues on next page)

(continued from previous page)

```
67

First word:  Questa
Last word: è
['Questa', 'che', 'di', 'dice', 'diverse', 'linee', 'niente.', 'non', 'stringa', 'su',
↪ 'testo', 'una', 'è']
```

Esercizio - numlist

Data la lista `la = [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]`

1. trova il min, max e valore mediano (SUGGERIMENTO: ordinala ed estrai i giusti valori)
2. crea una lista solo con gli elementi a indici pari (per es [10, 72, 11, ..], nota che ".." indica che lista non è completa !) e ricalcola i valori di min, max e mediana
3. rifai lo stesso con gli elementi ad indici dispari (per es [60, 118,..])

Dovresti ottenere:

```
la: [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]
pari: [10, 72, 11, 56, 120]
dispari: [60, 118, 71, 89, 175]

ordinati:   [10, 11, 56, 60, 71, 72, 89, 118, 120, 175]
ordinati pari:   [10, 11, 56, 72, 120]
ordinati dispari:   [60, 71, 89, 118, 175]

la: Min:  10  Max. 175  Median:  72
pari: Min:  10  Max. 120  Median:  56
dispari: Min:  60  Max. 175  Median:  89
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[86]:
```
la = [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]

# scrivi qui

pari = la[0::2]     # prendiamo solo elementi ad indici pari
dispari = la[1::2] # prendiamo solo elementi ad indici dispari

print("originale:" , la)
print("pari:", pari)
print("dispari:", dispari)

la.sort()
pari.sort()
dispari.sort()

print()
print("ordinata:  " , la)
print("ordinata pari:  " , pari)
print("ordinata dispari:  " , dispari)
print()
```

(continues on next page)

(continued from previous page)

```
print("originale: Min: ", la[0], " Max." , la[-1], " Median: ", la[len(la) // 2])
print("pari: Min: ", pari[0], " Max." , pari[-1], " Median: ", pari[len(pari) // 2])
print("dispari: Min: ", dispari[0], " Max." , dispari[-1], " Median: ",␣
↪dispari[len(dispari) // 2])
```

```
originale: [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]
pari: [10, 72, 11, 56, 120]
dispari: [60, 118, 71, 89, 175]

ordinata:   [10, 11, 56, 60, 71, 72, 89, 118, 120, 175]
ordinata pari:   [10, 11, 56, 72, 120]
ordinata dispari:   [60, 71, 89, 118, 175]

originale: Min:  10  Max. 175  Median:  72
pari: Min:  10  Max. 120  Median:  56
dispari: Min:  60  Max. 175  Median:  89
```

[86]:
```
la = [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]

# scrivi qui
```

```
originale: [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]
pari: [10, 72, 11, 56, 120]
dispari: [60, 118, 71, 89, 175]

ordinata:   [10, 11, 56, 60, 71, 72, 89, 118, 120, 175]
ordinata pari:   [10, 11, 56, 72, 120]
ordinata dispari:   [60, 71, 89, 118, 175]

originale: Min:  10  Max. 175  Median:  72
pari: Min:  10  Max. 120  Median:  56
dispari: Min:  60  Max. 175  Median:  89
```

4.14.13 Metodo index

Il metodo `index` ci permette di trovare l'indice della PRIMA occorrenza di un elemento.

[87]:
```
#      0   1   2   3   4
la = ['p','a','e','s','e']
```

[88]:
```
la.index('p')
```

[88]:
```
0
```

[89]:
```
la.index('a')
```

[89]:
```
1
```

[90]:
```
la.index('e')  # troviamo la PRIMA occorrenza
```

[90]:
```
2
```

Se l'elemento che cerchiamo non è presente, otterremo un errore:

```
>>> la.index('z')

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-303-32d9c064ebe0> in <module>
----> 1 la.index('z')

ValueError: 'z' is not in list
```

4.14.14 Metodo count

Si può ottenere il numero di occorrenze di un certo elemento in una lista usando il metodo `count`.

[91]:
```
la = ['c', 'o', 'r', 'r', 'o', 'b', 'o', 'r', 'a', 'r', 'e']
```

[92]:
```
la.count('c')
```

[92]:
```
1
```

[93]:
```
la.count('o')
```

[93]:
```
3
```

[94]:
```
la.count('r')
```

[94]:
```
4
```

4.14.15 Prosegui

Trovi ulteriori esercizi nel foglio Liste 4

[ ]:
```

```

4.15 Liste 4 - iterazione e funzioni

4.15.1 Scarica zip esercizi

Naviga file online[161]

In questo foglio trovi esercizi su iterazione su liste e come usarle quando sono usate come argomento di funzioni.

ATTENZIONE: Gli esercizi seguenti richiedono di conoscere:

Liste 1[162], Liste 2[163] e Liste 3[164]

Controllo di flusso[165]

Funzioni[166]

[161] https://github.com/DavidLeoni/softpython-it/tree/master/lists

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
lists
    lists1.ipynb
    lists1-sol.ipynb
    lists2.ipynb
    lists2-sol.ipynb
    lists3.ipynb
    lists3-sol.ipynb
    lists4.ipynb
    lists4-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `lists4.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.15.2 Esercizi con le funzioni

stamparole

⊛ Scrive una funzione `stamparole` che STAMPI tutte le parole in una frase

```
>>> stamparole("ciao come stai?")
```

```
ciao
come
stai?
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[162] https://it.softpython.org/lists/lists1-sol.html
[163] https://it.softpython.org/lists/lists2-sol.html
[164] https://it.softpython.org/lists/lists3-sol.html
[165] https://it.softpython.org/control-flow/control-flow-sol.html
[166] https://it.softpython.org/functions/functions-sol.html

[16]:
```
# scrivi qui

frase = "ciao come stai?"

def stamparole(f):

    lista = f.split()
    for parola in lista:
        print(parola)

stamparole(frase)
```

```
ciao
come
stai?
```

[16]:
```
# scrivi qui
```

```
ciao
come
stai?
```

stampari

⊛ Scrivere una funzione `stampari(lista)` che STAMPI i numeri pari di una lista di numeri

```
>>> stampari([1,2,3,4,5,6])

2
4
6
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[17]:
```
# scrivi qui

def stampari(lista):

    for numero in lista:
        if numero % 2 == 0:
            print(numero)

numeri = [1,2,3,4,5,6]
stampari(numeri)
```

```
2
4
6
```

[17]:
```
# scrivi qui
```

```
2
4
6
```

cerca26

⊛ Scrivere una funzione che RITORNA True se il numero 26 è contenuto in una lista di numeri

```
>>> cerca26( [1,26,143,431,53,6] )
True
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
# scrivi qui

def cerca26(lista):
    return (26 in numeri)

numeri = [1,26,143,431,53,6]
cerca26(numeri)
```

[18]:
```
True
```

[18]:
```
# scrivi qui
```

[18]:
```
True
```

stamprisec

⊛ Scrivere una funzione `stamprisec(stringa)` che STAMPI la prima e la seconda parola di una frase

- per ottenere una lista di parole usare il metodo delle stringhe `.split()`

```
>>> stamprisec("ciao come stai?")
```

```
ciao come
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
# scrivi qui

def stamprisec(stringa):
```

(continues on next page)

(continued from previous page)

```
    lista = frase.split()
    print(lista[0], lista[1])

frase = "ciao come stai?"
stamprisec(frase)
```

```
ciao come
```

[19]:
```
# scrivi qui
```

```
ciao come
```

trepari

⊛ Scrivi una funzione che STAMPI "si" se i primi tre elementi di una lista sono numeri pari. Altrimenti, la funzione deve STAMPARE "no". Nel caso in cui la lista contenga meno di tre elementi, STAMPARE "non va bene"

```
>>> trepari([6,4,8,4,5])
True
>>> trepari([2,5,6,3,4,5])
False
>>> trepari([4])
non va bene
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[20]:
```
# scrivi qui

def trepari(lista):
    if len(lista) >= 3:
        print(lista[0] % 2 == 0 and lista[1] % 2 == 0 and lista[2] % 2 == 0)
    else:
        print("non va bene")

trepari([6,4,8,4,5])
trepari([2,5,6,3,4,5])
trepari([4])
```

```
True
False
non va bene
```

[20]:
```
# scrivi qui
```

```
True
False
non va bene
```

separa_ip

⊛ Un indirizzo IP è una stringa in cui ci sono quattro sequenze di numeri (di lunghezza massima 3) separati ".". Per esempio, `192.168.19.34` e `255.31.1.0` sono indirizzi IP. Scrivere una funzione che dato un indirizzo IP in input, STAMPI i numeri che compongono l'indirizzo IP (nota: non vale utilizzare il metodo `.replace()`)

```
>>> separa_ip("192.168.0.1")

192
168
0
1
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[21]:
```
# scrivi qui

def separa_ip(stringa):
    separata = stringa.split(".")
    for elemento in separata:
        print(elemento)


separa_ip("192.168.0.1")
```

```
192
168
0
1
```

[21]:
```
# scrivi qui
```

```
192
168
0
1
```

media

⊛ Data una lista di numeri interi, scrivi una funzione `media(lista)` che RITORNI la media aritmetica dei numeri che contiene. Se la lista passata alla funzione dovesse essere vuota, RITORNARE 0

```
>>> x = media([3,4,2,3])   # ( 10/4 => 2.5)
>>> x
2.5
>>> y = media([])
>>> y
0
>>> z = media([ 30, 28 , 20, 29 ])
>>> z
26.75
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
# scrivi qui

def media(lista):

    if len(lista) == 0:
        return 0
    else:
        totale = 0
        for elemento in lista:
            totale = totale + elemento

        return(totale / len(lista))

x = media([])
print(x)
media([30,28,20,29])
```

```
0
```

[22]:
```
26.75
```

[22]:
```
# scrivi qui
```

```
0
```

[22]:
```
26.75
```

4.15.3 Verifica comprensione

ATTENZIONE

Gli esercizi che seguono contengono dei test con gli *assert*. Per capire come svolgerli, leggi prima Gestione errori e testing[167]

contiene

⊛ RITORNA `True` se `elem` è presente in lista, altrimenti RITORNA `False`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
def contiene(lista, elem):

    return elem in lista
```

(continues on next page)

[167] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert contiene([],'a') == False
assert contiene(['a'],'a') == True
assert contiene(['a','b','c'],'b') == True
assert contiene(['a','b','c'],'z') == False
# FINE TEST
```

[23]:
```
def contiene(lista, elem):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert contiene([],'a') == False
assert contiene(['a'],'a') == True
assert contiene(['a','b','c'],'b') == True
assert contiene(['a','b','c'],'z') == False
# FINE TEST
```

primi

⊛ RITORNA una lista con i primi numeri da `0` incluso a `n` escluso.

- Per esempio, `primi(3)` deve ritornare `[0,1,2]`
- Se `n` < 0, ritorna la lista vuota

Ingredienti:

- variabile lista da ritornare
- variabile contatore
- ciclo `while` (volendo ci sono anche altri modi)
- `return`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[24]:
```
def primi(n):

    lista = []
    contatore = 0
    while contatore < n:
        lista.append(contatore)
        contatore += 1
```

(continues on next page)

(continued from previous page)

```
    return lista




# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert primi(-1) == []
assert primi(-2) == []
assert primi(0) == []
assert primi(1) == [0]
assert primi(2) == [0,1]
assert primi(3) == [0,1,2]
# FINE TEST
```

[24]:
```
def primi(n):
    raise Exception('TODO IMPLEMENT ME !')



# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert primi(-1) == []
assert primi(-2) == []
assert primi(0) == []
assert primi(1) == [0]
assert primi(2) == [0,1]
assert primi(3) == [0,1,2]
# FINE TEST
```

primul

⊛ RITORNA `True` se il primo elemento di lista è uguale all'ultimo, altrimenti RITORNA `False`

NOTA: Si può assumere che lista contenga sempre almeno un elemento.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[25]:
```
def primul(lista):

    return lista[0] == lista[-1]

    # nota: la comparazione   lista[0] == lista[-1]   è una ESPRESSIONE che genera un␣
↪booleano,
    #       in questo caso True se il primo carattere è uguale all'ultimo e False␣
↪altrimenti,
```

(continues on next page)

(continued from previous page)

```
    #       quindi possiamo ritornare direttamente il risultato dell'espressione




# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert primul(['a']) == True
assert primul(['a','a']) == True
assert primul(['a','b']) == False
assert primul(['a','b','a']) == True
assert primul(['a','b','c','a']) == True
assert primul(['a','b','c','d']) == False
# FINE TEST
```

[25]:

```
def primul(lista):
    raise Exception('TODO IMPLEMENT ME !')



# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert primul(['a']) == True
assert primul(['a','a']) == True
assert primul(['a','b']) == False
assert primul(['a','b','a']) == True
assert primul(['a','b','c','a']) == True
assert primul(['a','b','c','d']) == False
# FINE TEST
```

duplica

⊛ RITORNA una NUOVA lista, in cui ciascun elemento della lista in ingresso è duplicato. Per esempio,

```
duplica(['ciao','mondo','python'])
```

deve ritornare

```
['ciao','ciao','mondo','mondo','python','python']
```

Ingredienti: - variabile per nuova lista - ciclo for - return

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[26]:
```
def duplica(lista):


    ret = []
    for elemento in lista:
        ret.append(elemento)
        ret.append(elemento)
    return ret




# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert duplica([]) ==  []
assert duplica(['a']) == ['a','a']
assert duplica(['a','b']) == ['a','a','b','b']
assert duplica(['a','b','c']) == ['a','a','b','b','c','c']
assert duplica(['a','a']) == ['a','a','a','a']
assert duplica(['a','a','b','b']) == ['a','a','a','a','b','b','b','b']
# FINE TEST
```

[26]:
```
def duplica(lista):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert duplica([]) ==  []
assert duplica(['a']) == ['a','a']
assert duplica(['a','b']) == ['a','a','b','b']
assert duplica(['a','b','c']) == ['a','a','b','b','c','c']
assert duplica(['a','a']) == ['a','a','a','a']
assert duplica(['a','a','b','b']) == ['a','a','a','a','b','b','b','b']
# FINE TEST
```

hadup

⊛⊛ RESTITUISCE `True` se `lista` contiene l'elemento `el` più di una volta, altrimenti RESTITUISCE `False`.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[27]:
```
def hadup(el, lista):


    contatore = 0
```

(continues on next page)

(continued from previous page)

```
    for x in lista:
        if x == el:
            contatore += 1
            if contatore > 1:
                return True
    return False


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert hadup("a", []) == False
assert hadup("a", ["a"]) == False
assert hadup("a", ["a", "a"]) == True
assert hadup("a", ["a", "a", "a"]) == True
assert hadup("a", ["b", "a", "a"]) == True
assert hadup("a", ["b", "a", "a", "a"]) == True
assert hadup("b", ["b", "a", "a", "a"]) == False
assert hadup("b", ["b", "a", "b", "a"]) == True
# FINE TEST
```

[27]:
```
def hadup(el, lista):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert hadup("a", []) == False
assert hadup("a", ["a"]) == False
assert hadup("a", ["a", "a"]) == True
assert hadup("a", ["a", "a", "a"]) == True
assert hadup("a", ["b", "a", "a"]) == True
assert hadup("a", ["b", "a", "a", "a"]) == True
assert hadup("b", ["b", "a", "a", "a"]) == False
assert hadup("b", ["b", "a", "b", "a"]) == True
# FINE TEST
```

ord3

⊛⊛ RITORNA `True` se la lista fornita ha i primi tre elementi ordinati in modo crescente, `False` altrimenti

- se `lista` ha meno di 3 elementi, ritorna `False`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
def ord3(lista):

    if len(lista) >= 3:
```

(continues on next page)

(continued from previous page)

```
        return lista[0] <= lista[1] and lista[1] <= lista[2]
    else:
        return False


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert ord3([5]) == False
assert ord3([4,7]) == False
assert ord3([4,6,9]) == True
assert ord3([4,9,7]) == False
assert ord3([9,5,7]) == False
assert ord3([4,8,9,1,5]) == True   # primi 3 elementi crescenti
assert ord3([9,4,8,10,13]) == False  # primi 3 elementi NON crescenti
```

[28]:
```
def ord3(lista):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert ord3([5]) == False
assert ord3([4,7]) == False
assert ord3([4,6,9]) == True
assert ord3([4,9,7]) == False
assert ord3([9,5,7]) == False
assert ord3([4,8,9,1,5]) == True   # primi 3 elementi crescenti
assert ord3([9,4,8,10,13]) == False  # primi 3 elementi NON crescenti
```

filtrab

⊛⊛ Prende in input una lista di caratteri, e RITORNA una NUOVA lista contenente solo i caratteri `'a'` e `'b'` trovati scorrendo la lista originale

Esempio:

```
filtrab(['c','a','c','d','b','a','c','a','b','e'])
```

deve ritornare

```
['a','b','a','a','b']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[29]:
```
def filtrab(lista):

    ret = []
    for el in lista:
```

(continues on next page)

(continued from previous page)

```
        if el == 'a' or el == 'b':
            ret.append(el)
    return ret



# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert filtrab([]) == []
assert filtrab(['a']) == ['a']
assert filtrab(['b']) == ['b']
assert filtrab(['a','b']) == ['a','b']
assert filtrab(['a','b','c']) == ['a','b']
assert filtrab(['a','c','b']) == ['a','b']
assert filtrab(['c','a','b']) == ['a','b']
assert filtrab(['c','a','c','d','b','a','c','a','b','e']) == ['a','b','a','a','b']

l = ['a','c','b']
assert filtrab(l) == ['a','b'] # verifica che sia ritornata una NUOVA lista
assert l == ['a','c','b']      # verifica che la lista originale non sia stata␣
↪modificata

# FINE TEST
```

[29]:
```
def filtrab(lista):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert filtrab([]) == []
assert filtrab(['a']) == ['a']
assert filtrab(['b']) == ['b']
assert filtrab(['a','b']) == ['a','b']
assert filtrab(['a','b','c']) == ['a','b']
assert filtrab(['a','c','b']) == ['a','b']
assert filtrab(['c','a','b']) == ['a','b']
assert filtrab(['c','a','c','d','b','a','c','a','b','e']) == ['a','b','a','a','b']

l = ['a','c','b']
assert filtrab(l) == ['a','b'] # verifica che sia ritornata una NUOVA lista
assert l == ['a','c','b']      # verifica che la lista originale non sia stata␣
↪modificata

# FINE TEST
```

collina

⊛⊛ RITORNA una lista in cui all'inizio sono presenti i primi numeri da uno a `n` in crescendo, e dopo `n` decrescono fino a `1`. NOTA: `n` è contenuto una sola volta.

Per esempio,

```
collina(4)
```

deve ritornare

```
[1,2,3,4,3,2,1]
```

Ingredienti: - variabile per la lista da ritornare - due cicli for di seguito e funzioni range oppure due while di seguito

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[30]:
```
def collina(n):


    ret = []
    for i in range(1,n):
        ret.append(i)
    for i in range(n,0,-1):
        ret.append(i)
    return ret


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert collina(0) == []
assert collina(1) == [1]
assert collina(2) == [1,2,1]
assert collina(3) == [1,2,3,2,1]
assert collina(4) == [1,2,3,4,3,2,1]
assert collina(5) == [1,2,3,4,5,4,3,2,1]
# FINE TEST
```

[30]:
```
def collina(n):

    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

assert collina(0) == []
assert collina(1) == [1]
assert collina(2) == [1,2,1]
assert collina(3) == [1,2,3,2,1]
assert collina(4) == [1,2,3,4,3,2,1]
```

(continues on next page)

(continued from previous page)

```
assert collina(5) == [1,2,3,4,5,4,3,2,1]
# FINE TEST
```

vetta

⊛⊛ Supponiamo che in una lista vengano salvate le altitudini di una strada di montagna prendendo una misura ogni 3 km (assumiamo che la strada sia costantemente in salita). Ad un certo punto, si arriverà alla vetta della montagna dove si misurerà l'altezza sul livello del mare. Chiaramente, esiste anche una strada per scendere (costantemente in discesa) e anche qui verrà misurata l'altitudine ogni 3 km.

Un esempio di misurazione è `[100, 400, 800, 1220, 1600, 1400, 1000, 300, 40]`

- Scrivere una funzione che RITORNI il *valore* della lista che corrisponde alla misurazione presa in vetta
- se la lista contiene meno di tre elementi, lanciare eccezione `ValueError`

```
>>> vetta([100,400, 800, 1220, 1600, 1400, 1000, 300, 40])
1600
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[31]:

```
def vetta(lista):

    if len(lista) < 3:
        raise ValueError("Lista vuota !")
    if len(lista) == 1:
        return lista[0]

    for i in range(len(lista)):
        if lista[i] > lista[i+1]:
            return lista[i]

    return lista[-i]  # strada senza discesa




# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

try:
    vetta([])         # con questa lista anomala ci attendiamo che sollevi l
↪'eccezione ValueError
    raise Exception("Non sarei dovuto arrivare fin qua !")
except ValueError:    # se solleva l'eccezione,si sta comportando come previsto e non␣
↪facciamo niente
    pass
assert vetta([5,40,7]) == 40
assert vetta([5,30,4]) == 30
assert vetta([5,70,70, 4]) == 70
```

(continues on next page)

(continued from previous page)

```
assert vetta([5,10,80,25,2]) == 80
assert vetta([100,400, 800, 1220, 1600, 1400, 1000, 300, 40]) == 1600
```

[31]:
```
def vetta(lista):
    raise Exception('TODO IMPLEMENT ME !')



# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError

try:
    vetta([])         # con questa lista anomala ci attendiamo che sollevi l
↪'eccezione ValueError
    raise Exception("Non sarei dovuto arrivare fin qua !")
except ValueError:    # se solleva l'eccezione,si sta comportando come previsto e non␣
↪facciamo niente
    pass
assert vetta([5,40,7]) == 40
assert vetta([5,30,4]) == 30
assert vetta([5,70,70, 4]) == 70
assert vetta([5,10,80,25,2]) == 80
assert vetta([100,400, 800, 1220, 1600, 1400, 1000, 300, 40]) == 1600
```

pari

⊛⊛ RITORNA una lista contenente gli elementi dalla lista di input alle posizioni pari, cominciando dalla zero che è considerato pari

- si può assumere che la lista di input contenga sempre un numero pari di elementi
- Suggerimento: ricordati che range può prendere tre parametri.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[32]:
```
def pari(lista):

    ret = []
    for i in range(0,len(lista),2):
        ret.append(lista[i])
    return ret




# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError
assert pari([]) == []
assert pari(['a','b']) == ['a']
```

(continues on next page)

(continued from previous page)

```
assert pari(['a','b','c','d']) == ['a', 'c']
assert pari(['a','b','a','c']) == ['a', 'a']
assert pari(['a','b','c','d','e','f']) == ['a', 'c','e']
# FINE TEST
```

[32]:
```
def pari(lista):
    raise Exception('TODO IMPLEMENT ME !')



# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError
assert pari([]) == []
assert pari(['a','b']) == ['a']
assert pari(['a','b','c','d']) == ['a', 'c']
assert pari(['a','b','a','c']) == ['a', 'a']
assert pari(['a','b','c','d','e','f']) == ['a', 'c','e']
# FINE TEST
```

mix

⊛⊛ RITORNA una nuova lista in cui gli elementi sono presi alternativamente da lista1 e da lista2. - si può assumere che `lista1` e `lista2` contengano lo stesso numero di elementi

Esempio:

```
mix(['a', 'b','c'], ['x', 'y','z'])
```

deve dare

```
['a', 'x', 'b','y', 'c','z']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[33]:
```

def mix(lista1, lista2):

    ret = []
    for i in range(len(lista1)):
        ret.append(lista1[i])
        ret.append(lista2[i])
    return ret



# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError
assert mix([], []) == []
assert mix(['a'], ['x']) == ['a', 'x']
```

(continues on next page)

(continued from previous page)

```
assert mix(['a'], ['a']) == ['a', 'a']
assert mix(['a', 'b'], ['x', 'y']) == ['a', 'x', 'b','y']
assert mix(['a', 'b','c'], ['x', 'y','z']) == ['a', 'x', 'b','y', 'c','z']
# FINE TEST
```

[33]:

```
def mix(lista1, lista2):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError
assert mix([], []) == []
assert mix(['a'], ['x']) == ['a', 'x']
assert mix(['a'], ['a']) == ['a', 'a']
assert mix(['a', 'b'], ['x', 'y']) == ['a', 'x', 'b','y']
assert mix(['a', 'b','c'], ['x', 'y','z']) == ['a', 'x', 'b','y', 'c','z']
# FINE TEST
```

nostop

⊛⊛ Quando si analizza una frase, può essere utile processarla per rimuovere parole molto comuni, come per esempio gli articoli e le preposizioni: `"un libro su Python"` si può semplificare in `"libro Python"`

Le parole 'poco utili' vengono chiamate *stopwords*. Questo processo è per esempio eseguito dai motori di ricerca per ridurre la complessità della stringa di input fornita dall'utente.

Implementa una funzione che prende una stringa e RITORNA la stringa di input senza le stopwords

SUGGERIMENTO 1: le stringhe in Python sono *immutabili* ! Per rimuovere le parole devi creare una *nuova* stringa a partire dalla stringa di partenza.

SUGGERIMENTO 2: crea una lista di parole così:

```
lista = stringa.split(" ")
```

SUGGERIMENTO 3: opera le opportune trasformazioni su lista, e poi costruisci la stringa da restituire con `" ".join(lista)`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[34]:

```
def nostop(stringa, stopwords):

    lista = stringa.split(" ")
    for s in stopwords:
        if s in lista:
            lista.remove(s)
    return " ".join(lista)
```

(continues on next page)

(continued from previous page)

```
# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError
assert nostop("un", ["un"]) == ""
assert nostop("un", []) == "un"
assert nostop("", []) == ""
assert nostop("", ["un"]) == ""
assert nostop("un libro", ["un"]) == "libro"
assert nostop("un libro su Python", ["un","su"]) == "libro Python"
assert nostop("un libro su Python per principianti", ["un","uno","il","su","per"]) ==
↪"libro Python principianti"
```

[34]:

```
def nostop(stringa, stopwords):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError
assert nostop("un", ["un"]) == ""
assert nostop("un", []) == "un"
assert nostop("", []) == ""
assert nostop("", ["un"]) == ""
assert nostop("un libro", ["un"]) == "libro"
assert nostop("un libro su Python", ["un","su"]) == "libro Python"
assert nostop("un libro su Python per principianti", ["un","uno","il","su","per"]) ==
↪"libro Python principianti"
```

4.16 Tuple

4.16.1 Scarica zip esercizi

Naviga file online[168]

Una tupla in Python è una sequenza *immutabile* di elementi eterogenei che ammette duplicati, perciò possiamo mettere gli oggetti che vogliamo, anche di tipi diversi, e anche ripetuti.

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
tuples
    tuples.ipynb
    tuples-sol.ipynb
    jupman.py
```

[168] https://github.com/DavidLeoni/softpython-it/tree/master/tuples

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `tuples.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.16.2 Creare tuple

Le tuple si creano con le parentesi tonde `()` e separando gli elementi da virgole `,`.

Qualche esempio:

[2]:
```
numeri = (6,7,5,7,7,9)
```

[3]:
```
print(numeri)
```

```
(6, 7, 5, 7, 7, 9)
```

Tuple di un solo elemento: Puoi creare una tupla di un solo elemento **aggiungendo una virgola dopo l'elemento**:

[4]:
```
tuplina = (4,)  # occhio alla virgola !!!
```

Verifichiamo che il tipo sia quello atteso:

[5]:
```
type(tuplina)
```

[5]:
```
tuple
```

Per vedere la differenza, scriviamo qua sotto `(4)` senza virgola e verifichiamo il tipo dell'oggetto ottenuto

[6]:
```
farlocca = (4)
```

[7]:
```
type(farlocca)
```

[7]:
```
int
```

Vediamo che `farlocca` è un `int`, perchè il `4` è stato valutato come un'espressione dentro delle tonde e il risultato è il contenuto delle tonde

Tupla vuota

Possiamo anche creare una tupla vuota:

[8]:
```
vuota = ()
```

[9]:
```
print(vuota)
```

```
()
```

[10]:
```
type(vuota)
```

[10]:
```
tuple
```

Tuple senza parentesi

Quando assegnamo dei valori a delle variabili, (e *solo* quando assegnamo valori a delle variabili) è possibile usare una notazione del genere, in cui a sinistra dell' = mettiamo nomi di variabili e a destra una sequenza di valori:

[11]:
```
a,b,c = 1, 2, 3
```

[12]:
```
a
```

[12]:
```
1
```

[13]:
```
b
```

[13]:
```
2
```

[14]:
```
c
```

[14]:
```
3
```

Se ci chiediamo cosa sia quel `1,2,3`, possiamo provare a mettere a sinistra solo una variabile:

[15]:
```
# ATTENZIONE: MEGLIO EVITARE !
x = 1,2,3
```

[16]:
```
type(x)
```

[16]:
```
tuple
```

Vediamo che Python ha considerato quel `1,2,3` come una tupla. Tipicamente, non scriverai mai assegnazioni con meno variabili che valori da metterci, ma se per caso ti capita, probabilmente ti troverai con qualche tupla indesiderata!

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se danno errore):

1.
```
z,w = 5,6
print(type(z))
print(type(w))
```

2.
```
a,b = 5,6
a,b = b,a
print('a=',a)
print('b=',b)
```

3.
```
z = 5,
print(type(z))
```

4.
```
z = ,
print(type(z))
```

Elementi eterogenei

In una tupla possiamo mettere elementi di tipo diverso, come numeri e stringhe:

[17]:
```
roba = (4, "carta", 5, 2,"scatole", 7)
```

[18]:
```
roba
```

[18]:
```
(4, 'carta', 5, 2, 'scatole', 7)
```

[19]:
```
type(roba)
```

[19]:
```
tuple
```

Possiamo anche inserire anche altre tuple:

[20]:
```
insalata = ( ("cavoli", 3), ("pomodori",9), ("verze",4) )
```

[21]:
```
insalata
```

[21]:
```
(('cavoli', 3), ('pomodori', 9), ('verze', 4))
```

[22]:
```
type(insalata)
```

[22]:
```
tuple
```

E anche liste:

[23]:
```
misto = ( ["quando", "fuori", "piove"], ["scrivo", "programmi"], [7,3,9] )
```

ATTENZIONE agli oggetti mutabili dentro le tuple!

Inserire oggetti *mutabili* come le liste dentro le tuple potrebbe causare problemi in alcune situazioni - per es. se poi vuoi usare la tupla come elemento di un insieme o chiave di un dizionario (lo vedremo meglio nei rispettivi tutorial)

Vediamo come gli esempi precedenti vengono rappresentati in Python Tutor:

[24]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[25]:
```
roba = (4, "carta", 5, 2,"scatole", 7)
insalata = ( ("cavoli", 3), ("pomodori",9), ("verze",4) )
misto = ( ["quando", "fuori", "piove"], ["scrivo", "programmi"], [7,3,9] )

jupman.pytut()
```

[25]:
```
<IPython.core.display.HTML object>
```

Creare tuple da sequenze

Puoi creare una tupla da una qualsiasi sequenza, per esempio da una lista:

[26]:
```
tuple( [8,2,5] )
```

[26]:
```
(8, 2, 5)
```

Oppure una stringa (che è una sequenza di caratteri):

[27]:
```
tuple("abc")
```

[27]:
```
('a', 'b', 'c')
```

Creare sequenze da tuple

Visto che la tupla è una sequenza, è anche possibile generare liste a partire da tuple:

[28]:
```
list( (3,4,2,3)  )
```

[28]:
```
[3, 4, 2, 3]
```

DOMANDA: Ha senso creare una tupla da un'altra tupla così? Possiamo riscrivere il codice in modo più conciso?

[29]:
```
x = (4,2,5)
y = tuple(x)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: visto che una tupla è IMMUTABILE, una volta che abbiamo creato in memoria l'oggetto `(4,2,5)` siamo sicuri che nessuno lo modificherà, quindi non occorre creare una nuova tupla e possiamo scrivere direttamente

```
x = (4,2,5)
y = x
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `(1.2,3.4)`
2. `(1;2;3;4)`
3. `(1,2;3,4)`
4. `(1,2,3,4)`
5. `(())`

6. `type(())`

7. `((),)`

8. `tuple([('a'),('b'),('c')])`

9. `tuple(tuple(('z','u','m')))`

10. `str(('a','b','c'))`

11. `"".join(('a','b','c'))`

4.16.3 Operatori

I seguenti operatori funzionano sulle tuple e si comportano esattamente come per le liste:

Operatore	Risultato	Significato
`len`(tuple)	`int`	Ritorna la lunghezza di una tupla
tuple[int]	obj	Legge un elemento all'indice specificato
tuple[int:int]	`tuple`	Estrae una sotto-tupla - ritorna una NUOVA tupla
tuple + tuple	`tuple`	Concatena due tuple - ritorna una NUOVA tupla
obj `in` tuple	`bool`	Controlla se un elemento è presente in una tupla
tuple * int	`tuple`	Replica la tupla - ritorna una NUOVA tupla
==,!=	`bool`	Controlla se due tuple sono uguali o differenti

len

La funzione `len` ritorna la lunghezza della tupla:

```
[30]: len( (4,2,3) )
[30]: 3
```

```
[31]: len( (7,) )
[31]: 1
```

```
[32]: len( () )
[32]: 0
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare il risultato (o se produce un errore)

1. `len(3,2,4)`

2. `len((3,2,4))`

3. `len(('a',))`

4. `len(('a,'))`
5. `len((((),(),()))`
6. `len(len((1,2,3,4)))`
7. `len([('d','a','c','d'),(('ab')),[('a','b','c')]])`

[ ]:

Leggere un elemento

Come per stringhe e liste possiamo leggere un elemento ad una certa posizione con le quadre:

[33]:
```
#         0  1  2  3
tupla = (10,11,12,13)
```

[34]:
```
tupla[0]
```
[34]: 10

[35]:
```
tupla[1]
```
[35]: 11

[36]:
```
tupla[2]
```
[36]: 12

[37]:
```
tupla[3]
```
[37]: 13

Anche per le tuple possiamo usare indici negativi:

[38]:
```
tupla[-1]
```
[38]: 13

DOMANDA: Guarda le espressioni seguenti e per ciascuna cerca di indovinare il risultato o se produce un errore:

1. `(1,2,3)[0]`
2. `(1,2,3)[3]`
3. `(1,2,3)0`
4. `()[0]`
5. `(())[0]`
6. `type((())[0])`

7. ```
   ('a,')[0]
   ```

8. ```
   ('a',)[0]
   ```

9. ```
   (1,2,3)[-0]
   ```

10. ```
    (1,2,3)[-1]
    ```

11. ```
    (1,2,3)[-3]
    ```

[ ]:

Esercizio: animali

Data la stringa `animali = "gatto siamese,cane,canarino,porcellino,coniglio,criceto"`

1. convertila in una lista
2. crea una tupla di tuple dove ogni tupla ha due informazioni: il nome dell'animale e la lunghezza del nome, per esempio ((“cane”,3), (“criceto”,7))
3. stampa la tupla

Dovresti ottenere

```
gatto siamese,cane,canarino,porcellino,coniglio,criceto

(('gatto siamese', 13), ('cane', 4), ('canarino', 8), ('porcellino', 10), ('coniglio',
↪ 8), ('criceto', 7))
```

- puoi assumere che `animali` contenga sempre esattamente 6 animali

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a><div class="jupman-sol jupman-sol-code" style="display:none">

[39]:
```
animali = "gatto siamese,cane,canarino,porcellino,coniglio,criceto"

# scrivi qui
lista = animali.split(',')

#print(animali)
print()

tuple_animali = ( (lista[0], len(lista[0])),
                  (lista[1], len(lista[1])),
                  (lista[2], len(lista[2])),
                  (lista[3], len(lista[3])),
                  (lista[4], len(lista[4])),
                  (lista[5], len(lista[5])))

#print(tuple_animali)
```

</div>
```

[39]:
```
animali = "gatto siamese,cane,canarino,porcellino,coniglio,criceto"

# scrivi qui
```

Slice

Come per stringhe e liste possiamo ricavare sottosequenze da una tupla usando le *slice*, cioè scrivendo alla destra della tupla delle quadre con dentro un indice di partenza INCLUSO, due punti `:` e un indice di fine ESCLUSO:

[40]:
```
tupla = (10,11,12,13,14,15,16,17,18,19)
```

[41]:
```
tupla[2:6]  # da indice 2 INCLUSO a 6 ESCLUSO
```

[41]:
```
(12, 13, 14, 15)
```

Si può specificare di raccogliere elementi saltandone tra uno e l'altro aggiungendo il numero di elementi da saltare come un terzo numero tra le quadre, per esempio:

[42]:
```
tupla = (10,11,12,13,14,15,16,17)
```

[43]:
```
tupla[0:8:5]
```

[43]:
```
(10, 15)
```

[44]:
```
tupla[0:8:2]
```

[44]:
```
(10, 12, 14, 16)
```

[45]:
```
tupla[1:8:1]
```

[45]:
```
(11, 12, 13, 14, 15, 16, 17)
```

ATTENZIONE: ricordati che le slice producono una NUOVA tupla !

DOMANDA: Guarda il codice seguente e per ciascuna espressione prova a indovinare il risultato o se produce un errore:

1. `(7,6,8,9,5)(1:3)`
2. `(7,6,8,9,5)[1:3]`
3. `(10,11,12,13,14,15,16)[3:100]`
4. `(10,11,12,13,14,15,16)[-3:5]`
5. `(1,0,1,0,1,0)[::2]`
6. `(1,2,3)[::1]`

7. `(1,0,1,0,1,0)[1::2]`

8. `tuple("cartolina")[0::2]`

9. `(4,5,6,3,4,7)[0:::2]`

[ ]:

Concatenazione

E' possibile concatenare due tuple usando l'operatore +, che crea una NUOVA tupla

[46]:
```
t = (1,2,3) + (4,5,6,7,8)
```

[47]:
```
t
```

[47]:
```
(1, 2, 3, 4, 5, 6, 7, 8)
```

[48]:
```
type(t)
```

[48]:
```
tuple
```

Verifichiamo che le tuple originali non vengano modificate:

[49]:
```
x = (1,2,3)
y = (4,5,6,7,8)
```

[50]:
```
t = x + y
```

[51]:
```
t
```

[51]:
```
(1, 2, 3, 4, 5, 6, 7, 8)
```

[52]:
```
x
```

[52]:
```
(1, 2, 3)
```

[53]:
```
y
```

[53]:
```
(4, 5, 6, 7, 8)
```

Guardiamo come vengono rappresentate in Python Tutor:

[54]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[55]:
```
x = (1,2,3)
y = (4,5,6,7,8)
t = x + y
print(t)
print(x)
print(y)
```

(continues on next page)

(continued from previous page)

```
jupman.pytut()
```

```
(1, 2, 3, 4, 5, 6, 7, 8)
(1, 2, 3)
(4, 5, 6, 7, 8)
```

[55]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Guarda il codice seguente e per ciascuna espressione cerca di indovinare il risultato, o se produce un errore.

1. `()+()`
2. `type(()+())`
3. `len(()+())`
4. `()+[]`
5. `[]+()`
6. `(2,3,4) + tuple([5,6,7])`
7. `"razzo"+('p','a','z','z','o')`

Appartenenza

Come per tutte le sequenze, se vogliamo verificare se un elemento è contenuto in una tupla possiamo usare l'operatore `in` che ci ritorna un valore booleano:

[56]:
```
'a' in ('c','a','s','c','o')
```

[56]:
```
True
```

[57]:
```
'z' in ('c','a','s','c','o')
```

[57]:
```
False
```

not in

Per verificare se qualcosa **non** appartiene ad una tupla, possiamo usare due forme:

not in - forma 1:

[58]:
```
"carota" not in ("anguria","banana","mela")
```

[58]:
```
True
```

[59]:
```
"anguria" not in ("anguria","banana","mela")
```

[59]:
```
False
```

not in - forma 2

[60]:
```
not "carota" in ("anguria","banana","mela")
```
[60]:
```
True
```

[61]:
```
not "anguria" in ("anguria","banana","mela")
```
[61]:
```
False
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `3 in (1.0, 2.0,3.0)`
2. `3.0 in (1,2,3)`
3. `3 not in (3)`
4. `3 not in (3,)`
5. `6 not in ()`
6. `0 in (0)[0]`
7. `[] in ()`
8. `() in []`
9. `not [] in ()`
10. `() in ()`
11. `() in (())`
12. `() in (),)`
13. `'ciao' in ('c','i','a','o')`

[ ]:
```

```

Moltiplicazione

Per replicare gli elementi di una tupla, è possibile usare l'operatore `*` che produce una NUOVA tupla:

[62]:
```
(7,8,5) * 3
```
[62]:
```
(7, 8, 5, 7, 8, 5, 7, 8, 5)
```

[63]:
```
(7,8,5) * 1
```
[63]:
```
(7, 8, 5)
```

```
[64]: (7,8,5) * 0
```

```
[64]: ()
```

DOMANDA: Il seguente codice cosa stamperà?

```
x = (5,6,7)
y = x * 3
print('x=',x)
print('y=',y)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Stamperà

```
x = (5, 6, 7)
y = (5, 6, 7, 5, 6, 7, 5, 6, 7)
```

perchè la moltiplicazione genera una NUOVA tupla che viene associata a `y`. La tupla associata a `x` rimane invariata.

DOMANDA: Guarda il codice seguente e per ciascuna espressione cerca di indovinare il risultato, o se produce un errore.

1. `(5,6,7)*(3.0)`
2. `(5,6,7)*(3,0)`
3. `(5,6,7)*(3)`
4. `(5,6,7)*3`
5. `(4,2,3)*int(3.0)`
6. `(1,2)*[3][0]`
7. `(1,2)*(3,4)[-1]`
8. `[(9,8)]*4`
9. `(1+2,3+4)*5`
10. `(1+2,)*4`
11. `(1+2)*4`
12. `(1,2,3)*0`
13. `(7)*0`
14. `(7,)*0`

[ ]:

Esercizio - welcome

Data una tupla `x` contenente esattamente 3 interi, e una tupla `y` contenente esattamente 3 tuple di caratteri, scrivere del codice che crea una tupla `z` contenente ogni tupla di `y` replicata per il corrispondente intero in `x`.

Esempio - dati

```
x = (2,4,3)
y = (('w','e','l','c'),('o',),('m','e'))
```

dopo il tuo codice dovrà stampare

```
>>> print(z)
('w', 'e', 'l', 'c', 'w', 'e', 'l', 'c', 'o', 'o', 'o', 'o', 'm', 'e', 'm', 'e', 'm',
↪'e')
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[65]:
```
x = (2,4,3)
y = (('w','e','l','c'),('o',),('m','e'))

# scrivi qui
z = y[0]*x[0] + y[1]*x[1] + y[2]*x[2]

#print(z)
```

[65]:
```
x = (2,4,3)
y = (('w','e','l','c'),('o',),('m','e'))

# scrivi qui
```

4.16.4 Scrivere un elemento

Le tuple sono *immutabili,* quindi provare per esempio a scrivere un assegnazione per mettere il numero `12` nella cella all'indice `3` provoca un errore:

```
#         0 1 2 3 4
tupla = (5,8,7,9,11)
tupla[3] = 666

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-118-83949b0c81e2> in <module>
      1 tupla = (5,8,7,9)
----> 2 tupla[1] = 666
```

(continues on next page)

(continued from previous page)

```
TypeError: 'tuple' object does not support item assignment
```

Quello che possiamo fare è creare una NUOVA tupla componendola da sequenze prese dall'originale

[66]:
```
#         0  1  2  3  4  5  6
tupla = (17,54,34,87,26,95,34)
```

[67]:
```
tupla =  tupla[0:3] + (12,) + tupla[4:]
```

[68]:
```
tupla
```

[68]:
```
(17, 54, 34, 12, 26, 95, 34)
```

ATTENZIONE: `append`, `extend`, `insert`, `sort` **NON FUNZIONANO CON LE TUPLE !**

Tutti i metodi che hai usato per modificare le liste *non* funzioneranno con le tuple.

Esercizio - errando

Prova a scrivere qua sotto `(1,2,3).append(4)` e guarda che errore appare:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[69]:
```
# scrivi qui
#(1,2,3).append(4)
```

[69]:
```
# scrivi qui
```

Esercizio: abde

Data una tupla `x`, salva nella variabile `y` un'altra tupla contenente:

- all'inizio gli stessi elementi di `x` *eccetto* l'ultimo
- alla fine gli elementi `'d'` e `'e'` .
- Il tuo codice dovrebbe funzionare con *qualunque* tupla `x`

Esempio

```
x = ('a','b','c')
```

dopo il tuo codice, dovresti veder stampato:

```
x = ('a', 'b', 'c')
y = ('a', 'b', 'd', 'e')
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[70]:
```
x = ('a','b','c')

# scrivi qui
y = x[:-1] + ('d','e')

#print('x=',x)
#print('y=',y)
```

[70]:
```
x = ('a','b','c')

# scrivi qui
```

Esercizio - carismatico

Data una tupla `t` avente caratteri intervallati maiuscolo / minuscolo, scrivere del codice che modifica l'assegnazione di `t` in modo che `t` sia uguale ad una tupla avente tutti i caratteri maiuscoli per primi e tutti i caratteri minuscoli per ultimi.

Esempio - data

```
t = ('C', 'a', 'R', 'i', 'S', 'm', 'A', 't', 'I', 'c', 'O')
```

dopo il tuo codice deve risultare

```
>>> print(t)
('C', 'R', 'S', 'A', 'I', 'O', 'a', 'i', 'm', 't', 'c')
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[71]:
```
t = ('C', 'a', 'R', 'i', 'S', 'm', 'A', 't', 'I', 'c', 'O')

# scrivi qui
t = t[::2] + t[1::2]
#print(t)
```

[71]:
```
t = ('C', 'a', 'R', 'i', 'S', 'm', 'A', 't', 'I', 'c', 'O')

# scrivi qui
```

Esercizio - ordinare

Data una tupla `x` di numeri disordinati, scrivere del codice che cambia l'assegnazione di `x` in modo che abbia assegnata una tupla con i numeri ordinati.

- il tuo codice deve funzionare per *qualsiasi* tupla `x`
- **SUGGERIMENTO**: Come già scritto, le tuple NON hanno il metodo `sort` (perchè le muterebbe), ma le liste ce l'hanno….

Esempio - dati:

```
x = (3,4,2,5,5,5,2,3)
```

dopo il tuo codice deve risultare

```
>>> print(x)
(2, 2, 3, 3, 4, 5, 5, 5)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[72]:
```
x = (3,4,2,5,5,5,2,3)

# scrivi qui
y = list(x)
y.sort()
x = tuple(y)
#print(x)
```

[72]:
```
x = (3,4,2,5,5,5,2,3)

# scrivi qui
```

4.16.5 Metodi

Le tuple sono oggetti di tipo `tuple` e possiedono dei metodi che permettono di operare su di esse:

Metodo	Ritorna	Descrizione
`tuple.index(obj)`	`int`	Trova la prima occorrenza di un elemento e ne ritorna la posizione
`tuple.count(obj)`	`int`	Conta le occorrenze di un elemento

Metodo index

Il metodo `index` ci permette di trovare l'indice della PRIMA occorrenza di un elemento.

[73]:
```
tupla = ('b','a','r','a','t','t','o')
```

[74]:
```
tupla.index('b')
```

[74]:
```
0
```

[75]:
```
tupla.index('a')
```

[75]:
```
1
```

[76]:
```
tupla.index('t')
```

[76]:
```
4
```

Se l'elemento che cerchiamo non è presente, otterremo un errore:

```
>>> tupla.index('z')

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-318-96cf33478b69> in <module>
----> 1 tupla.index('z')

ValueError: tuple.index(x): x not in tuple
```

DOMANDA: Guarda il codice seguente e per ciascuna espressione prova a indovinare il risultato o se produce un errore:

1. `(3,4,2).index(4)`
2. `(3,4,---1).index(-1)`
3. `(2.2,.2,2,).index(2)`
4. `(3,4,2).index(len([3,8,2,9]))`
5. `(6,6,6).index(666)`
6. `(4,2,3).index(3).index(3)`
7. `tuple("GUG").index("g")`
8. `(tuple("ci") + ("a","o")).index('a')`
9. `(()).index(())`
10. `((),).index(())`

Metodo count

Si può ottenere il numero di occorrenze di un certo elemento in una lista usando il metodo `count`:

[77]:
```
t = ('g', 'u', 'a', 'r', 'a', 'n', 't', 'i', 'g', 'i', 'a')
```

[78]:
```
t.count('g')
```

[78]:
```
2
```

[79]:
```
t.count('a')
```

[79]:
```
3
```

Se un elemento non è presente viene ritornato `0`:

[80]:
```
t.count('z')
```

[80]:
```
0
```

Esercizio: frutti

Data la stringa `s = "mela|pera|mela|ciliegia|pera|mela|pera|pera|ciliegia|pera|fragola"`

Inserisci gli elementi separati da `"|"` in una lista.

1. Quanti elementi deve avere la lista?
2. Sapendo che la lista creata al punto precedente ha solo quattro elementi distinti (es `"mela"`, `"pera"`, `"ciliegia"`, e `"fragola"`), crea un'altra lista dove ogni elemento è una tupla contenente il nome del frutto e la sua moltiplicità (cioè il numero di volte che compare nella lista originale). Esempio:

```
conteggi = [("mela", 3), ("pera",5), ...]
```

Qua puoi scrivere codice che funziona data una specifica costante, quindi non hai bisogno di cicli.

3. Stampa il contenuto di ciascuna tupla in una linea separata (es: prima linea: `"mela"` è presente 3 volte)

Dovresti ottenere:

```
['mela', 'pera', 'mela', 'ciliegia', 'pera', 'mela', 'pera', 'pera', 'ciliegia', 'pera
↪', 'fragola']
[('mela', 3), ('pera', 5), ('ciliegia', 2), ('fragola', 1)]

mela  è presente  3  volte
pera  è presente  5  volte
ciliegia  è presente  2  volte
fragola  è presente  1  volte
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[81]:
```
s = "mela|pera|mela|ciliegia|pera|mela|pera|pera|ciliegia|pera|fragola"

# scrivi qui
```

(continues on next page)

(continued from previous page)

```
parole = s.split("|")
#print(parole)

tmele = ("mela", parole.count("mela"))
tpere = ("pera", parole.count("pera"))
tciliegie = ("ciliegia", parole.count("ciliegia"))
tfragole = ("fragola", parole.count("fragola"))
conteggi =[tmele, tpere, tciliegie, tfragole]

#print(conteggi)
#print()
#print(tmele[0], " è presente ", tmele[1], " volte")
#print(tpere[0], " è presente ", tpere[1], " volte")
#print(tciliegie[0], " è presente ", tciliegie[1], " volte")
#print(tfragole[0], " è presente ", tfragole[1], " volte")
```

```
[81]: s = "mela|pera|mela|ciliegia|pera|mela|pera|pera|ciliegia|pera|fragola"

# scrivi qui
```

4.16.6 Esercizi con le funzioni

ATTENZIONE: Gli esercizi seguenti richiedono di conoscere:

Controllo di flusso[169]

Funzioni[170]

appaia

⊛⊛ Scrivere una funzione `appaia` che data una tupla `t`, RITORNA una lista avente come elementi delle tuple da due elementi presi a due a due da `t`.

- se la tupla `t` ha un numero dispari di elementi, l'ultima tupla nella lista da ritornare sarà costituita da un solo elemento

Esempio :

```
>>> appaia( ('c', 'a', 'p', 'i', 'r', 'e') )    # lunghezza pari
[('c', 'a'), ('p', 'i'), ('r', 'e')]
>>> appaia( ('t','a','p','p','e','t','o') )     # lunghezza dispari
[('t', 'a'), ('p', 'p'), ('e', 't'), ('o',)]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[169] https://it.softpython.org/control-flow/control-flow-sol.html
[170] https://it.softpython.org/functions/functions-sol.html

[82]:
```
# scrivi qui
def appaia(t):
    ret = []
    i = 0
    while i < len(t)-1:
        ret.append((t[i],t[i+1]))
        i += 2
    if i == len(t)-1:
        ret.append((t[-1],))
    return ret
```

[82]:
```
# scrivi qui
```

giunto

⊛⊛ Scrivi una funzione che date due tuple di caratteri `ta` e `tb` aventi ciascuna caratteri diversi (possono essere anche vuote), se la tupla `ta` termina con lo stesso carattere con cui `tb` inizia, RITORNA la concatenazione di `ta` e `tb` SENZA caratteri duplicati, altrimenti RITORNA una tupla vuota.

Esempio:

```
>>> giunto(('a','b','c'), ('c','d','e'))
('a', 'b', 'c', 'd', 'e')
>>> giunto(('a','b'), ('b','c','d'))
('a', 'b', 'c', 'd')
>>> giunto((),('e','f','g'))
()
>>> giunto(('a',),('e','f','g'))
()
>>>  f(('a','b','c'),())
()
>>> f(('a','b','c'),('d','e'))
()
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[83]:
```
# scrivi qui

def giunto(ta,tb):
    if len(ta) > 0 and len(tb) > 0:
        if ta[-1] == tb[0]:
            return ta[:-1] + tb

    return ()
```

[83]:
```
# scrivi qui
```

4.16.7 Verifica comprensione

ATTENZIONE

Gli esercizi che seguono contengono dei test con gli *assert*. Per capire come svolgerli, leggi prima Gestione errori e testing[171]

doppie

⊛⊛ Prende in input una lista con `n` numeri interi, e RITORNA una NUOVA lista che contiene `n` tuple ciascuna da due elementi. Ogni tupla contiene un numero preso dalla corrispondente posizione della lista di partenza, e il suo doppio.

Per esempio:

```
doppie([ 5, 3, 8])
```

deve dare la nuova lista

```
[(5,10), (3,6), (8,16)]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[84]:
```
def doppie(lista):

    ret = []
    for elemento in lista:
        ret.append((elemento, elemento * 2))
    return ret


# INIZIO TEST - NON TOCCARE !
assert doppie([]) == []
assert doppie([3]) == [(3,6)]
assert doppie([2,7]) == [(2,4),(7,14)]
assert doppie([5,3,8]) == [(5,10), (3,6), (8,16)]

# verifica che la lista originale non venga cambiata
la = [6]
lb = doppie(la)
assert la == [6]
assert lb == [(6,12)]
# FINE TEST
```

[84]:
```
def doppie(lista):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
assert doppie([]) == []
assert doppie([3]) == [(3,6)]
assert doppie([2,7]) == [(2,4),(7,14)]
```

(continues on next page)

[171] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
assert doppie([5,3,8]) == [(5,10), (3,6), (8,16)]

# verifica che la lista originale non venga cambiata
la = [6]
lb = doppie(la)
assert la == [6]
assert lb == [(6,12)]
# FINE TEST
```

nasty

⊛⊛⊛ Date due tuple `ta` di caratteri e `tb` di numeri interi positivi, scrivere una funzione `nasty` che RITORNA una tupla avente stringhe da due caratteri: il primo carattere è preso da `ta`, e il secondo carattere è un numero preso dalla posizione corrispondente in `tb`. Le stringhe vengono ripetute per un numero di volte pari a quel numero.

```
>>> nasty(('u','r','g'), (4,2,3))
('u4', 'u4', 'u4', 'u4', 'r2', 'r2', 'g3', 'g3', 'g3')

>>> nasty(('g','a','s','p'), (2,4,1,3))
('g2', 'g2', 'a4', 'a4', 'a4', 'a4', 's1', 'p3', 'p3', 'p3')
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[85]:
```
# scrivi qui

def nasty(ta, tb):

    i = 0
    ret = []
    while i < len(tb):
        s = ta[i]+str(tb[i])
        ret.extend( (s,) * tb[i] )
        i += 1
    return tuple(ret)


# INIZIO TEST - NON TOCCARE !
assert nasty(('a',), (3,)) == ('a3','a3','a3')
assert nasty(('a','b'), (3,1)) == ('a3','a3','a3','b1')
assert nasty(('u','r','g'), (4,2,3)) == ('u4', 'u4', 'u4', 'u4', 'r2', 'r2', 'g3', 'g3
↪', 'g3')
assert nasty(('g','a','s','p'), (2,4,1,3)) == ('g2', 'g2', 'a4', 'a4', 'a4', 'a4', 's1
↪', 'p3', 'p3', 'p3')
# FINE TEST
```

[85]:
```
# scrivi qui
```

[ ]:
```
```

4.17 Insiemi

4.17.1 Scarica zip esercizi

Naviga file online[172]

Un insieme è una collezione *mutabile senza ordine* di elementi *immutabili* e *distinti* (cioè senza duplicati). Il tipo di dati in Python per rappresentare gli insiemi si chiama `set`.

Che fare

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `sets.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.17.2 Creare un insieme

Possiamo creare un insieme usando le parentesi graffe, e separando gli elementi da virgole `,`

Proviamo un insieme di caratteri:

[2]:
```
s = {'b','a','d','c'}
```

[3]:
```
type(s)
```

[3]:
```
set
```

ATTENZIONE: GLI INSIEMI *NON* SONO ORDINATI !!!

NON CREDERE A QUELLO CHE VEDI !!

Proviamo a stampare l'insieme:

[4]:
```
print(s)
```

```
{'b', 'd', 'c', 'a'}
```

[172] https://github.com/DavidLeoni/softpython-it/tree/master/sets

Come vedi, l'ordine in cui è stata effettuata la stampa è diverso da quello con cui abbiamo costruito l'insieme. A seconda della versione di Python che stai usando, sul tuo computer potrebbe essere diverso ancora!!

Questo perchè l'ordine negli insiemi NON è garantito: l'unica cosa che conta è se un elemento appartiene ad un insieme oppure no.

Come ulteriore dimostrazione, possiamo chiedere a Jupyter di mostrarci il contenuto dell'insieme, scrivendo solo la variabile `s` SENZA la `print`:

```
[5]: s
[5]: {'a', 'b', 'c', 'd'}
```

Adesso appare in ordine alfabetico ! Succede così perchè Jupyter quando mostra le variabili le stampa implicitamente non con la `print` ma con la pprint[173] (*pretty* print), che SOLO per gli insiemi ci fa la cortesia di ordinare il risultato prima di stamparlo. Possiamo ringraziare, ma non lasciamo che ci confonda !!

Indice degli elementi: visto che gli insiemi non hanno ordine, chiedere a Python di estrarre un elemento ad una certa posizione non avrebbe senso. Quindi, diversamente da stringhe, liste e tuple, con gli insiemi NON è possibile ricavare un elemento a partire da un indice:

```
s[0]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-352-c9c96910e542> in <module>
----> 1 s[0]

TypeError: 'set' object is not subscriptable
```

Abbiamo detto che un insieme ha solo elementi *distinti*, cioè senza duplicati - che succede se proviamo a metterli comunque ?

```
[6]: s = {6,7,5,9,5,5,7}
```

```
[7]: s
[7]: {5, 6, 7, 9}
```

Notiamo che Python ha silenziosamente rimosso i duplicati.

Convertire sequenze in insiemi

Come per liste e stringhe, possiamo creare un `set` a partire da un'altra sequenza:

```
[8]: set('acacia') # da stringa
[8]: {'a', 'c', 'i'}
```

```
[9]: set( [1,2,3,1,2,1,2,1,3,1] ) # da lista
[9]: {1, 2, 3}
```

```
[10]: set( (4,6,1,5,1,4,1,5,4,5) ) # da tupla
[10]: {1, 4, 5, 6}
```

[173] https://docs.python.org/3/library/pprint.html

Di nuovo, notiamo come nell'insieme creato non siano presenti duplicati.

RICORDATI: Gli insiemi sono utili per rimuovere duplicati da una sequenza

Elementi mutabili e hash

Rivediamo la definizione di insieme data all'inizio:

> Un insieme è una collezione *mutabile senza ordine* di elementi *immutabili* e *distinti*

Finora abbiamo creato l'insieme solo usando elementi *immutabili* come numeri e stringhe.

Cosa succede se mettiamo degli elementi mutabili, come liste?

```
>>> s = { [1,2,3], [4,5] }

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-40-a6c538692ccb> in <module>
----> 1 s = { [1,2,3], [4,5]  }

TypeError: unhashable type: 'list'
```

Otteniamo `TypeError: unhashable type: 'list'`, che letteralmente significa che Python non è riuscito a calcolare lo spezzatino (*hash*) della lista. Cosa sarà mai questa particolare pietanza??

Cos'è lo hash? Lo *hash* di un oggetto è un numero che Python può associargli, per esempio puoi vedere lo `hash` di un oggetto con l'omonima funzione:

```
[11]: hash( "Questa è una bella giornata" )  # stringa
[11]: -1325950510056206031
```

```
[12]: hash( 111112222233333334444444555555555 )   # numero
[12]: 651300278308214397
```

Immagina che lo *hash* sia una specie di etichetta con queste caratteristiche:

- è troppo breve per descrivere completamente l'oggetto a cui è associata (tradotto: data solo un'etichetta hash, *non* puoi ricostruire l'oggetto che rappresenta)
- è abbastanza lunga per identificare *quasi* univocamente l'oggetto…
- … anche se al mondo *potrebbero* esistere oggetti diversi hanno però associata esattamente la stessa etichetta

Cosa c'entra con i nostri insiemi? Lo *hash* ha vari utilizzi, ma tipicamente Python lo usa per ritrovare velocemente un'oggetto in collezioni basate sugli hash, come gli insiemi e i dizionari. Quanto velocemente? Parecchio: anche con insiemi enormi, otteniamo una risposta sempre in un tempo costante e brevissimo! In altre parole, la velocità di risposta *non* dipende dalla dimensione dell'insieme (salvo casi patologici).

Questa velocità è consentita dal fatto che dato un oggetto da cercare, Python è in grado di ricavare velocemente la sua etichetta *hash:* poi con l'etichetta in mano, riesce a individuare nel magazzino della memoria molto in fretta se vi sono oggetti che hanno la stessa etichetta. Se vengono trovati, saranno quasi sicuramente molto pochi, e basterà quindi confrontarli con quello cercato.

Gli oggetti *immutabili* hanno sempre lo stessa etichetta hash da quando sono creati fino alla fine del programma. Quelli *mutabili* invece no: ogni volta che li cambiamo, viene anche automaticamente cambiato l'*hash*. Immaginati un supermercato dove i commessi dispongono gli alimentari in base all'etichetta separando per esempio il caffè nello scaffale

per la prima colazione e la varechina nello scaffale dei detersivi. Se sei un cliente e vuoi il caffè, guardi i cartelli e ti dirigi subito verso lo scaffale della prima colazione. Immagina cosa succederebbe se un mago malvagio potesse trasmutare gli oggetti già collocati negli scaffali in altri oggetti, quindi per esempio il caffè in varechina (assumiamo che al momento della trasmutazione oltre al caffè cambi anche l'etichetta *hash*). Sicuramente porterebbe tanta confusione, e se non si sta attenti, anche un gran mal di pancia.

Quindi per offrirti il vantaggio della ricerca rapida evitando situazioni disastrose, Python ti impone di collocare nell'insieme solo oggetti con *hash* stabile, cioè gli oggetti *immutabili*.

DOMANDA: Possiamo inserire una tupla dentro un insieme? Prova a verificare la tua supposizione con un esempio di codice.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Sì, le tuple sono *immutabili*, quindi hanno uno *hash* corrispondente stabile per tutta la durata del programma, per es questo è un set di tuple: `{(1,2), (3,4,5)}`

Insieme vuoto

ATTENZIONE: Se scrivi `{}` otterrai un dizionario, NON un insieme !!!

Per creare un insieme vuoto dobbiamo chiamare la funzione `set()`:

[13]:
```
s = set()
```

[14]:
```
s
```

[14]:
```
set()
```

ESERCIZIO: prova a scrivere nella cella qua sotto `{}` e guarda il tipo dell'oggetto ottenuto con `type`

[15]:
```
# scrivi qui
```

DOMANDA: Possiamo inserire un insieme dentro un'altro insieme? Guarda bene la definizione di insieme, poi verifica le tuo supposizioni provando a scrivere del codice per creare un insieme che abbia dentro un'altro insieme.

ATTENZIONE: Per fare la verifica, NON usare la funzione `set`, usa solo creazione con parentesi graffe

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Un insieme è *mutabile*, pertanto *non* possiamo inserirlo come elemento di un altro insieme (la sua etichetta *hash* potrebbe variare nel tempo). Scrivendo `{{1,2,3}}` otterrai un errore.

DOMANDA: Se scriviamo una cosa del genere, cosa otterremo? (attento !)

```
set(set(['a','b']))
```

1. un insieme con dentro `'a'` e `'b'`
2. un insieme con dentro un insieme contenente gli elementi `'a'` e `'b'`
3. un errore (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: la 1:

- all'interno abbiamo l'espressione `set(['a','b'])` che genera l'insieme `{'a','b'}`
- all'esterno abbiamo l'espressione `set( set(['a','b']) )` che si vede passare questo insieme appena creato, quindi la possiamo riscrivere come `set({'a','b'})`
- Dato che la `set` quando usata come funzione si attende una sequenza, e un insieme *è* una sequenza, la `set` esterna preleva tutti gli elementi che trova all'interno della sequenza `{'a','b'}` che gli abbiamo passato, e genera un nuovo insieme con dentro `'a'` e `'b'`.

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `{'oh','la','la'}`
2. `set([3,4,2,3,2,2,2,-1])`
3. `{(1,2),(2,3)}`
4. `set('aba')`
5. `str({'a'})`
6. `{1;2;3}`
7. `set(  1,2,3  )`
8. `set( {1,2,3} )`
9. `set( [1,2,3] )`
10. `set( (1,2,3) )`
11. `set(  "abc"  )`
12. `set(  "1232"  )`
13. `set( [ {1,2,3,2} ] )`

14. `set( [ [1,2,3,2] ] )`

15. `set( [ (1,2,3,2) ] )`

16. `set( [ "abcb"   ] )`

17. `set( [ "1232"   ] )`

18. `set((1,2,3,2))`

19. `set([(),()])`

20. `set([])`

21. `set(list(set()))`

Esercizio: dedup

Scrivi del codice breve per creare una lista `lb` che contiene tutti gli elementi dalla lista `la` senza duplicati e ordinati alfabeticamente.

- NON DEVE cambiare la lista originale `la`
- NON usare cicli
- il tuo codice dovrebbe funzionare con qualunque `la`

```
la = ['c','a','b','c','d','b','e']
```

dopo il tuo codice, dovresti ottenere:

```
>>> print(la)
['c', 'a', 'b', 'c', 'd', 'b', 'e']
>>> print(lb)
['a', 'b', 'c', 'd', 'e']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[16]:
```
la = ['c','a','b','c','d','b','e']

# scrivi qui

lb = list(set(la))
lb.sort()
#lb = list(sorted(set(la)))  # alternativa, NOTA: sorted genera una NUOVA sequenza

print("la =",la)
print("lb =",lb)
```

```
la = ['c', 'a', 'b', 'c', 'd', 'b', 'e']
lb = ['a', 'b', 'c', 'd', 'e']
```

[16]:
```
la = ['c','a','b','c','d','b','e']

# scrivi qui
```

```
la = ['c', 'a', 'b', 'c', 'd', 'b', 'e']
lb = ['a', 'b', 'c', 'd', 'e']
```

Frozenset

INFO: questo argomento è opzionale ai fini della comprensione del libro**

In Python esistono anche insiemi *immutabili* che si chiamano `frozenset`. Qui ci limitamo a ricordare che i `frozenset` essendo *immutabili* hanno un'etichetta *hash* associata e possono essere inseriti come elementi di altri insiemi. Per il resto rimandiamo alla documentazione ufficiale[174].

4.17.3 Operatori

Operatore	Risultato	Descrizione
`len`(set)	`int`	il numero di elementi nel set
el `in` set	`bool`	verifica se elemento è contenuto nel set
set | set	`set`	unione, crea un NUOVO set
set & set	`set`	intersezione, crea un NUOVO set
set − set	`set`	differenza, crea un NUOVO set
set ^ set	`set`	differenza simmetrica, crea un NUOVO set
==, !=	`bool`	Controlla se due insiemi sono uguali o differenti

len

[17]:
```
len( {'a','b','c'}  )
```

[17]:
```
3
```

[18]:
```
len( set() )
```

[18]:
```
0
```

[174] https://docs.python.org/3/library/stdtypes.html#frozenset

Esercizio - distinte

Data una stringa `parola`, scrivere del codice che

- stampa le lettere distinte presenti in `parola` ordinate alfabeticamente (senza le quadre!), assieme al loro numero
- stampa il numero di lettere duplicate trovate in totale

Esempio 1 - data:

```
parola = "ababbbbcdd"
```

dopo il tuo codice deve stampare

```
parola     : ababbbbcdd
4 distinte : a,b,c,d
6 duplicate
```

Esempio 2 - data:

```
parola = "cccccaaabbbb"
```

dopo il tuo codice deve stampare

```
parola     : cccccaaabbbb
3 distinte : a,b,c
9 duplicate
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
# scrivi qui
parola = "ababbbbcdd"
#parola = "cccccaaabbbb"
s = set(parola)
print("parola     :", parola)
la = list(s)
la.sort()
print(len(s), 'distinte :', ",".join(la))
#print(len(s), 'distinte :', list(sorted(s)))  # ALTERNATIVA CON SORTED
print(len(parola) - len(s), 'duplicate')
```

```
parola     : ababbbbcdd
4 distinte : a,b,c,d
6 duplicate
```

[19]:
```
# scrivi qui
```

```
parola     : ababbbbcdd
4 distinte : a,b,c,d
6 duplicate
```

Unione

L'operatore di unione `|` (detto *pipe*) produce un NUOVO insieme contenente tutti gli elementi del primo e del secondo insieme.

[20]:
```
{'a','b','c'} | {'b','c','d','e'}
```

[20]:
```
{'a', 'b', 'c', 'd', 'e'}
```

Notiamo che non ci sono elementi duplicati

ESERCIZIO: E se usiamo il +? Prova a scrivere in una cella `{'a','b'} + {'c','d','e'}`. Cosa succede?

[21]:
```
# scrivi qui
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `{'a','d','b'}|{'a','b','c'}`
2. `{'a'}|{'a'}`
3. `{'a'|'b'}`
4. `{1|2|3}`
5. `{'a'|'b'|'a'}`
6. `{{'a'}|{'b'}|{'a'}}`
7. `[1,2,3] | [3,4]`
8. `(1,2,3) | (3,4)`
9. `"abc" | "cd"`
10. `{'a'} | set(['a','b'])`
11. `set(".".join('pacca'))`
12. `'{a}'|'{b}'|'{a}'`

13. `set((1,2,3))|set([len([4,5])])`

14. `{()}|{()}`

15. `{'|'}|{'|'}`

DOMANDA: Dati insiemi `x` e `y` qualunque, questa espressione

```
len(x | y) <= len(x) + len(y)
```

produce:

1. un errore (quale?)
2. sempre `True`
3. sempre `False`
4. a volte `True` a volte `False` a seconda dei valori di `x` e `y`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 2: il numero degli elementi dell'unione sarà sempre inferiore o uguale alla somma del numero degli elementi di ogni singolo set che andiamo ad unire, pertanto dalla comparazione con <= otterremo sempre `True`.

Esercizio: tuttotranne 1

Scrivi del codice che crea un set `s4` che contiene tutti gli elementi di `s1` ed `s2` ma non contiene gli elementi di `s3`.

- Il tuo codice dovrebbe funzionare con *qualunque* insieme `s1`, `s2`, `s3`

Esempio - dati

```
s1 = set(['a','b','c','d','e'])
s2 = set(['b','c','f','g'])
s3 = set(['b','f'])
```

Dopo il tuo codice dovresti ottenere

```
>>> print(s4)
{'d', 'a', 'c', 'g', 'e'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
s1 = set(['a','b','c','d','e'])
s2 = set(['b','c','f','g'])
s3 = set(['b','f'])

# scrivi qui
s4 = (s1 | s2) - s3
#print(s4)
```

[22]:
```
s1 = set(['a','b','c','d','e'])
s2 = set(['b','c','f','g'])
s3 = set(['b','f'])

# scrivi qui
```

Intersezione

L'operatore di intersezione `&` produce un NUOVO insieme contenente tutti gli elementi in comune del primo e secondo insieme

[23]:
```
{'a','b','c'} & {'b','c','d','e'}
```

[23]:
```
{'b', 'c'}
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `{0}&{0,1}`
2. `{0,1}&{0}`
3. `set("capra") & set("campa")`
4. `set("cba") & set("dcb")`
5. `{len([1,2,3]),4} & {len([5,6,7])}`
6. `{1,2}&{1,2}`
7. `{0,1}&{}`
8. `{0,1}&set()`
9. `set([1,2,3,4,5][::2]) & set([1,2,3,4,5][2::2])`
10. `{(),)}&{()}`
11. `{(())}&{()}`

Differenza

L'operatore di differenza − produce un NUOVO insieme contenente tutti gli elementi del primo insieme eccetto quelli del secondo:

```
[24]: {'a','b','c','d'} - {'b','c','e','f','g'}
```

```
[24]: {'a', 'd'}
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `{3,4,2}-2`
2. `{1,2,3}-{3,4}`
3. `'{"a"}-{"a"}'`
4. `{1,2,3}--{3,4}`
5. `{1,2,3}-(-{3,4})`
6. `set("chiodo") - set("chiave")`
7. `set("prova") - set("prova".capitalize())`
8. `set("BarbA") - set("BARBA".lower())`
9. `set([(1,2),(3,4),(5,6)]) - set([(2,3),(4,5)])`
10. `set([(1,2),(3,4),(5,6)]) - set([(3,4),(5,6)])`
11. `{1,2,3} - set()`
12. `set() - {1,2,3}`

DOMANDA: Dati due insiemi qualunque `x` e `y`, il seguente codice cosa produce? Un errore? E' semplificabile?

```
(x & y) | (x-y)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Stiamo unendo gli elementi in comune tra `x` e `y`, con gli elementi presenti in `x` ma non in `y`. Pertanto, stiamo prendendo tutti gli elementi di `x`, quindi l'espressione può essere semplificata scrivendo semplicemente

```
x
```

Differenza simmetrica

La differenza simmetrica di due insiemi è la loro unione meno la loro intersezione, cioè tutti gli elementi tranne quelli in comune

In Python si può esprimere direttamente con l'operatore ^:

[25]:
```
{'a','b','c'} ^ {'b','c','d','e'}
```

[25]:
```
{'a', 'd', 'e'}
```

Verifichiamo che il risultato corrisponda alla definizione:

[26]:
```
s1 = {'a','b','c'}
s2 = {'b','c','d','e'}

(s1 | s2) - (s1 & s2)
```

[26]:
```
{'a', 'd', 'e'}
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `{'p','e','p','p','o'} ^ {'p','a','p','p','e'}`
2. `{'ab','cd'} ^ {'ba','dc'}`
3. `set('brodino') ^ set('bordo')`
4. `set((1,2,5,3,2,3,1)) ^ set((1,4,3,2))`

DOMANDA: Dati 3 insiemi `A`, `B`, `C`, qual'è l'espressione per ottenere la parte in azzurro?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

```
(A & B) | (A & C) | (B & C)
```

DOMANDA: Se usiamo i seguenti valori nell'esercizio precedente, l'insieme che indica la parte in blu cosa conterrebbe?

```
A = {'a','ab','ac','abc'}
B = {'b','ab','bc','abc'}
C = {'c','ac','bc','abc'}
```

Una volta fatta la supposizione, prova ad eseguire la formula che hai trovato nell'esercizio precedente con i valori forniti e confronta i risultati con la soluzione.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Se la formula è corretta dovresti ottenere

```
{'abc', 'ac', 'bc', 'ab'}
```

Appartenenza

Come per tutte le sequenze, se vogliamo verificare se un elemento è contenuto in un insieme possiamo usare l'operatore `in` che ci ritorna un valore booleano:

[27]:
```
'a' in {'m','e','n','t','a'}
```

[27]:
```
True
```

[28]:
```
'z' in {'m','e','n','t','a'}
```

[28]:
```
False
```

> `in` **NEGLI INSIEMI E' UN'OPERAZIONE MOLTO VELOCE**
>
> La velocità dell'operatore `in` NON dipende dalla dimensione dell'insieme

Questo è una differenza sostanziale rispetto alle altre sequenze già viste: se provi a cercare un elemento con `in` su stringhe, liste o tuple, Python potrebbe dover scorrere *tutta* la lista se per sfortuna l'elemento da cercare è alla fine (o non c'è proprio).

not in

Per verificare se qualcosa **non** appartiene ad una sequenza, possiamo usare due forme:

not in - forma 1:

```
[29]: "carota" not in {"anguria","banana","mela"}
[29]: True
```

```
[30]: "anguria" not in {"anguria","banana","mela"}
[30]: False
```

not in - forma 2

```
[31]: not "carota" in {"anguria","banana","mela"}
[31]: True
```

```
[32]: not "anguria" in {"anguria","banana","mela"}
[32]: False
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `2*10 in {10,20,30,40}`
2. `'four' in {'f','o','u','r'}`
3. `'aa' in set('aa')`
4. `'a' in set(['a','a'])`
5. `'c' in (set('parco') - set('cassa'))`
6. `'cc' in (set('pacca') & set('zucca'))`
7. `[3 in {3,4}, 6 in {3,4} ]`
8. `4 in set([1,2,3]*4)`
9. `2 in {len('3.4'.split('.'))}`

10. ```
    4 not in {1,2,3}
    ```

11. ```
    '3' not in {1,2,3}
    ```

12. ```
    not 'a' in {'b','c'}
    ```

13. ```
    not {} in set([])
    ```

14. ```
    {not 'a' in {'a'}}
    ```

15. ```
    4 not in set((4,))
    ```

16. ```
    () not in set([()])
    ```

DOMANDA: le seguenti espressioni sono simili. Cosa hanno in comune? Qual'è la differenza con l'ultima (oltre al fatto che è su un insieme)?

1. ```
   'e' in 'abcde'
   ```

2. ```
   'abcde'.find('e') >= 0
   ```

3. ```
   'abcde'.count('e') > 0
   ```

4. ```
   'e' in ['a','b','c','d','e']
   ```

5. ```
   ['a','b','c','d','e'].count('e') > 0
   ```

6. ```
   'e' in ('a','b','c','d','e')
   ```

7. ```
   ('a','b','c','d','e').count('e') > 0
   ```

8. ```
   'e' in {'a','b','c','d','e'}
   ```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a><div class="jupman-sol jupman-sol-question" style="display:none">

RISPOSTA: Tutte le espressioni sopra riportate ritornano un booleano che è `True` se l'elemento `'e'` è presente nella sequenza.

Tutte le operazioni di ricerca e/o conteggio (`in`, `find`, `index`, `count`) su stringhe, liste e tuple impiegano un tempo di ricerca che alla peggio come in questo caso può essere pari alla dimensione della sequenza (`'e'` à alla fine).

Gli insiemi invece (espressione 8.), visto che sono basati sugli *hash*, consentono una ricerca immediata, indipendentemente dalla dimensione dell'insieme o posizione degli elementi (quindi non ha nessuna importanza se abbiamo creato l'insieme con `'e'` alla fine).

Per fare ricerche performanti è preferibile usare sequenze basate su hash, come insiemi o dizionari !

</div>
   ```

4.17.4 Uguaglianza

Possiamo verificare se due insiemi sono uguali con l'operatore di uguaglianza ==, che dati due insiemi ritorna `True` se contengono elementi uguali oppure `False` altrimenti:

```
[33]: {4,3,6} == {4,3,6}
[33]: True
```

```
[34]: {4,3,6} == {4,3}
[34]: False
```

```
[35]: {4,3,6} == {4,3,6, 'ciao'}
[35]: False
```

Attento alla rimozione dei duplicati !

```
[36]: {2,8} == {2,2,8}
[36]: True
```

Per verificare la disuguaglianza, possiamo usare l'operatore `!=`:

```
[37]: {2,5} != {2,5}
[37]: False
```

```
[38]: {4,6,0} != {2,8}
[38]: True
```

```
[39]: {4,6,0} != {4,6,0,2}
[39]: True
```

Attenti ai duplicati e all'ordine!

```
[40]: {0,1} != {1,0,0,0,0,0,0,0}
[40]: False
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `{2 == 2, 3 == 3}`
2. `{1,2,3,2,1} == {1,1,2,2,3,3}`
3. `{'aa'} == {'a'}`
4. `set('aa') == {'a'}`
5. `[{1,2,3}] == {[1,2,3]}`

6. `set({1,2,3}) == {1,2,3}`

7. `set((1,2,3)) == {(1,2,3)}`

8. `{'aa'} != {'a', 'aa'}`

9. `{set() != set()}`

10. `set('scarpa') == set('capras')`

11. `set('papa') != set('pappa')`

12. `set('pappa') != set('reale')`

13. `{(),()} == {(())}`

14. `{(),()} != {(()), (())}`

15. `[set()] == [set(),set()]`

16. `(set('gosh') | set('posh')) == (set('shopping') - set('in'))`

4.17.5 Metodi simili agli operatori

Vi sono metodi analoghi agli operatori `|`, `&`, `-`, `^` che creano un **NUOVO** set.

NOTA: diversamente dagli operatori, questi metodi accettano come parametro una *qualsiasi* sequenza, non solo insiemi:

Metodo	Risultato	Descrizione	Operatore analogo
`set.union(seq)`	`set`	unione, crea un NUOVO set	`|`
`set.intersection(seq)`	`set`	intersezione, crea un NUOVO set	`&`
`set.difference(seq)`	`set`	differenza, crea un NUOVO set	`-`
`set.symmetric_difference(seq)`	`set`	differenza simmetrica, crea un NUOVO set	`^`

Metodi che **MODIFICANO** il primo insieme su cui sono chiamati (e ritornano `None`!):

Metodo	Risultato	Descrizione
`setA.update(setB)`	`None`	unione, MODIFICA `setA`
`setA.intersection_update(setB)`	`None`	intersezione, MODIFICA `setA`
`setA.difference_update(setB)`	`None`	differenza, MODIFICA `setA`
`setA.symmetric_difference_update(setB)`	`None`	differenza simmetrica, MODIFICA `setA`

union

Guarderemo solo `union`/`update`, gli altri si comportano in modo analoghi

Con `union` dato un insieme e una generica sequenza (quindi non necessariamente un insieme) possiamo creare un NUOVO insieme:

[41]:
```
sa = {'g','a','r','a'}
```

[42]:
```
la = ['a','g','r','a','r','i','o']
```

[43]:
```
sb = sa.union(la)
```

[44]:
```
sb
```

[44]:
```
{'a', 'g', 'i', 'o', 'r'}
```

ESERCIZIO: con `union` possiamo usare sequenze arbitrarie, invece con gli operatori no. Prova a scrivere `{1,2,3} | [2,3,4]` e guarda cosa succede.

[45]:
```
# scrivi qui
```

Possiamo verificare che `union` crei un nuovo insieme con Python Tutor:

[46]:
```
sa = {'g','a','r','a'}
la = ['a','g','r','a','r','i','o']
sb = sa.union(la)

jupman.pytut()
```

[46]:
```
<IPython.core.display.HTML object>
```

update

Se vogliamo invece MODIFICARE il primo insieme, possiamo utilizzare i metodi che terminano con la parola `update`:

[47]:
```
sa = {'g','a','r','a'}
```

[48]:
```
la = ['a','g','r','a','r','i','o']
```

[49]:
```
sa.update(la)
```

[50]:
```
print(sa)
```

```
{'r', 'i', 'o', 'g', 'a'}
```

DOMANDA: che cosa ha ritornato la chiamata ad `update`?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Dal momento che Jupyter non ha mostrato nulla, significa che implicitamente la chiamata al metodo `update` ha ritornato l'oggetto `None`.

Guardiamo che è successo con Python Tutor - per evidenziare cosa è stato ritornato da update aggiungiamo anche un `x =`:

[51]:
```
sa = {'g','a','r','a'}
la = ['a','g','r','a','r','i','o']
x = sa.update(la)
print(sa)
print(x)

jupman.pytut()
```

```
{'r', 'i', 'o', 'g', 'a'}
None
```

[51]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Guarda i seguenti pezzi di codice, e per ciascuno cerca di indovinare quale risultato producono (o se danno errore):

1. `set('case').intersection('sebo') == 'se'`

2. `set('naso').difference('caso')`

3.
```
s = {1,2,3}
s.intersection_update([2,3,4])
print(s)
```

4.
```
s = {1,2,3}
s = s & [2,3,4]
```

5.
```
s = set('cartone')
s = s.intersection('parto')
print(s)
```

6.
```
sa = set("mastice")
sb = sa.difference("mastro").difference("collo")
print(sa)
print(sb)
```

7.
```
sa = set("mastice")
sb = sa.difference_update("mastro").difference_update("collo")
print(sa)
print(sb)
```

[ ]:
```

```

Esercizio - tuttotranne 2

Dati i set `s1`, `s2` e `s3`, scrivere del codice che MODIFICA `s1` in modo che contenga anche gli elementi di `s2` ma non contenga gli elementi di `s3`.

- Il tuo codice dovrebbe funzionare con *qualunque* insieme `s1`, `s2`, `s3`
- **NON** creare nuovi set

Esempio - dati

```
s1 = set(['a','b','c','d','e'])
s2 = set(['b','c','f','g'])
s3 = set(['b','f'])
```

Dopo il tuo codice dovresti ottenere

```
>>> print(s1)
{'a', 'g', 'e', 'd', 'c'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[52]:
```
s1 = set(['a','b','c','d','e'])
s2 = set(['b','c','f','g'])
s3 = set(['b','f'])

# scrivi qui
s1.update(s2)
s1.difference_update(s3)
print(s1)
```

```
{'e', 'd', 'c', 'g', 'a'}
```

[52]:
```
s1 = set(['a','b','c','d','e'])
s2 = set(['b','c','f','g'])
s3 = set(['b','f'])

# scrivi qui
```

```
{'e', 'd', 'c', 'g', 'a'}
```

4.17.6 Altri metodi

Metodo	Risultato	Descrizione
`set.add(el)`	`None`	aggiunge l'elemento specificato - se già presente non fa nulla)
`set.remove(el)`	`None`	rimuove l'elemento specificato - se non presente solleva errore
`set.discard(el)`	`None`	rimuove l'elemento specificato - se non presente non fa nulla
`set.pop()`	obj	rimuove un elemento arbitrario dall'insieme e lo ritorna
`set.clear()`	`None`	rimuove tutti gli elementi
`setA.issubset(setB)`	`bool`	verifica se `setA` è un sottoinsieme di `setB`
`setA.issuperset(setB)`	`bool`	verifica se `setA` contiene tutti gli elementi di `setB`
`setA.isdisjoint(setB)`	`bool`	verifica se `setA` non ha elementi in comune con `setB`

add

Dato un insieme, possiamo aggiungergli un elemento con il metodo `.add`:

[53]:
```
s = {3,7,4}
```

[54]:
```
s.add(5)
```

[55]:
```
s
```

[55]:
```
{3, 4, 5, 7}
```

Se aggiungiamo lo stesso elemento due volte, non accade nulla:

[56]:
```
s.add(5)
```

[57]:
```
s
```

[57]:
```
{3, 4, 5, 7}
```

DOMANDA: Se scriviamo questo codice, che risultato otteniamo?

```
s = {'a','b'}
s.add({'c','d','e'})
print(s)
```

1. stampa `{'a','b','c','d','e'}`
2. stampa `{{'a','b','c','d','e'}}`
3. stampa `{'a','b',{'c','d','e'}}`
4. un errore (quale?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 4 - produce `TypeError: unhashable type: 'set'` : stiamo cercando di inserire un insieme come elemento di un altro insieme, ma gli insiemi sono *mutabili* e pertanto la loro etichetta *hash* (che permette a Python di trovarli velocemente) potrebbe variare nel tempo

DOMANDA: Guarda il codice seguente, che risultato produce?

```
x = {'a','b'}
y = set(x)
x.add('c')
print('x=',x)
print('y=',y)
```

1. un errore (quale?)
2. `x` e `y` saranno uguali (come?)
3. `x` e `y` saranno diversi (come?)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 3. Stamperà

```
x= {'c', 'a', 'b'}
y= {'a', 'b'}
```

perchè `y=set(x)` crea un NUOVO insieme copiando gli elementi dalla sequenza di input `x`. Possiamo verificarlo con Python Tutor:

[58]:
```
x = {'a','b'}
y = set(x)
x.add('c')

jupman.pytut()
```

[58]:
```
<IPython.core.display.HTML object>
```

remove

Il metodo `remove` toglie un elemento specificato dall'insieme. Se non esiste, produce un errore:

[59]:
```
s = {'a','b','c'}
```

[60]:
```
s.remove('b')
```

[61]:
```
s
```

[61]:
```
{'a', 'c'}
```

[62]:
```
s.remove('c')
```

[63]:
```
s
```

[63]:
```
{'a'}
```

```
s.remove('z')

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
```

(continues on next page)

(continued from previous page)

```
<ipython-input-266-a9e7a977e50c> in <module>
----> 1 s.remove('z')

KeyError: 'z'
```

Esercizio - bababiba

Data una stringa `scritta` di esattamente 4 sillabe da due caratteri ciascuna, creare un insieme `s` che contenga delle tuple con 2 caratteri ciascuna. Ogni tupla deve rappresentare una sillaba presa da `scritta`.

- per aggiungere elementi all'insieme, usa solo `add`
- il tuo codice deve funzionare con qualunque `scritta` da 4 bisillabe

Esempio 1 - data:

```
scritta = "bababiba"
```

dopo il tuo codice, deve risultare:

```
>>> print(s)
{('b', 'a'), ('b', 'i')}
```

Esempio 2 - data:

```
scritta = "rubareru"
```

dopo il tuo codice, deve risultare:

```
>>> print(s)
{('r', 'u'), ('b', 'a'), ('r', 'e')}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[64]:
```
scritta = "bababiba"
#scritta = "rubareru"

# scrivi qui

s = set()
s.add(tuple(scritta[:2]))
s.add(tuple(scritta[2:4]))
s.add(tuple(scritta[4:6]))
s.add(tuple(scritta[6:8]))
#print(s)
```

[64]:
```
scritta = "bababiba"
#scritta = "rubareru"

# scrivi qui
```

discard

Il metodo `discard` toglie un elemento specificato dall'insieme. Se non esiste, non fa nulla:

[65]:
```
s = {'a','b','c'}
```

[66]:
```
s.discard('a')
```

[67]:
```
s
```

[67]:
```
{'b', 'c'}
```

[68]:
```
s.discard('c')
```

[69]:
```
s
```

[69]:
```
{'b'}
```

[70]:
```
s.discard('z')
```

[71]:
```
s
```

[71]:
```
{'b'}
```

Esercizio - spazzatura

⊛⊛ Un impianto di processamento rifiuti riceve un carico di `spazzatura`, che rappresentiamo come insieme di stringhe:

```
spazzatura = {'alcheni','verdura','mercurio','carta'}
```

Per rimuovere gli elementi contaminanti che *potrebbero* essere presenti (NOTA: non sempre sono presenti !), l'impianto ha esattamente 3 `filtri` (come lista di stringhe) che applicherà in serie alla spazzatura:

```
filtri = ['cadmio','mercurio','alcheni']
```

Per ogni filtro applicato, si vuole vedere lo stato della `spazzatura` processata, per vedere se il filtro ha effettivamente rimosso il contaminante (se presente)

Alla fine, si vuole anche stampare tutti e *soli* i contaminanti che sono stati effettivamente rimossi (mettili come insieme nella variabile `separati`)

- **NON** usare comandi `if`
- **NON** serve usare cicli (il numero di filtri è fisso a 3, puoi usare copia e incolla di codice)
- Il tuo codice deve funzionare per *qualsiasi* lista `filtri` di 3 elementi e *qualsiasi* insieme `spazzatura`

Esempio - dati:

```
filtri = ['cadmio','mercurio','alcheni']
spazzatura = {'alcheni','verdura','mercurio','carta'}
```

Dopo il tuo codice, deve mostrare:

```
spazzatura iniziale: {'carta', 'verdura', 'mercurio', 'alcheni'}
Applico filtro per cadmio : {'carta', 'verdura', 'mercurio', 'alcheni'}
Applico filtro per mercurio : {'carta', 'verdura', 'alcheni'}
Applico filtro per alcheni : {'carta', 'verdura'}

Contaminanti separati: {'mercurio', 'alcheni'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[72]:
```
filtri = ['cadmio','mercurio','alcheni']
spazzatura = {'alcheni','verdura','mercurio','carta'}
separati = spazzatura.intersection(filtri) # crea un NUOVO insieme

# scrivi qui
s = "Applico filtro per"
print("spazzatura iniziale:", spazzatura)
spazzatura.discard(filtri[0])
print(s,filtri[0],":", spazzatura)
spazzatura.discard(filtri[1])
print(s,filtri[1],":", spazzatura)
spazzatura.discard(filtri[2])
print(s,filtri[2],":", spazzatura)
print("")


print("Contaminanti separati:", separati)
```

```
spazzatura iniziale: {'alcheni', 'verdura', 'carta', 'mercurio'}
Applico filtro per cadmio : {'alcheni', 'verdura', 'carta', 'mercurio'}
Applico filtro per mercurio : {'alcheni', 'verdura', 'carta'}
Applico filtro per alcheni : {'verdura', 'carta'}

Contaminanti separati: {'alcheni', 'mercurio'}
```

[72]:
```
filtri = ['cadmio','mercurio','alcheni']
spazzatura = {'alcheni','verdura','mercurio','carta'}
separati = spazzatura.intersection(filtri) # crea un NUOVO insieme

# scrivi qui



```

```
spazzatura iniziale: {'alcheni', 'verdura', 'carta', 'mercurio'}
Applico filtro per cadmio : {'alcheni', 'verdura', 'carta', 'mercurio'}
Applico filtro per mercurio : {'alcheni', 'verdura', 'carta'}
Applico filtro per alcheni : {'verdura', 'carta'}

Contaminanti separati: {'alcheni', 'mercurio'}
```

4.17.7 issubset

Per verificare se tutti gli elementi di un insieme `sa` sono contenuti in un altro insieme `sb` possiamo scrivere `sa.issubset(sb)`. Esempi:

```
[73]: {2,4}.issubset({1,2,3,4})
[73]: True
```

```
[74]: {3,5}.issubset({1,2,3,4})
[74]: False
```

ATTENZIONE: l'insieme vuoto è sempre considerato un sottoinsieme di un qualsiasi insieme

```
[75]: set().issubset({3,4,2,5})
[75]: True
```

issuperset

Per verificare se un insieme `sa`contiene tutti gli elementi di un altro insieme `sb` possiamo scrivere `sa.issuperset(sb)`. Esempi:

```
[76]: {1,2,3,4,5}.issuperset({1,3,5})
[76]: True
```

```
[77]: {1,2,3,4,5}.issuperset({2,4})
[77]: True
```

```
[78]: {1,2,3,4,5}.issuperset({1,3,5,7,9})
[78]: False
```

ATTENZIONE: l'insieme vuoto è sempre considerato un sottoinsieme di un qualsiasi insieme

```
[79]: {1,2,3,4,5}.issuperset({})
[79]: True
```

isdisjoint

Un insieme è disgiunto da un altro se non ha alcun elemento in comune, per verificarlo possiamo usare il metodo `isdisjoint`:

```
[80]: {1,3,5}.isdisjoint({2,4})
[80]: True
```

```
[81]: {1,3,5}.isdisjoint({2,3,4})
```

```
[81]: False
```

DOMANDA: Dato un insieme qualsiasi `x`, cosa produce la seguente espressione?

```
x.isdisjoint(x)
```

1. un errore (quale?)
2. sempre `True`
3. sempre `False`
4. `True` o `False` a seconda del valore di `x`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La 4, `True` o `False` a seconda del valore di `x`.

Probabilmente avrai pensato che l'espressione restituisca sempre `False`, dopotutto, come potrebbe un insieme a essere disgiunto da sè stesso? In effetti l'espressione ritorna sempre `False` *eccetto* nel caso particolare dell'insieme vuoto:

```
x = set()
x.isdisjoint(x)
```

in cui ritorna `True`.

MORALE: CONTROLLA SEMPRE L'INSIEME VUOTO !

Per questo e per tanti altri metodi l'insieme vuoto causa spesso comportamenti non sempre intuitivi, quindi ti invitiamo sempre a provare caso per caso.

4.17.8 Esercizio - matrioska

⊛⊛ Data una lista `insiemi` di esattamente 4 insiemi, la definiamo *a matrioska* se ogni insieme contiene tutti gli elementi del precedente insieme (più eventualmente altri). Scrivi del codice che STAMPA `True` se la sequenza è a matrioska, altrimenti STAMPA `False`.

- **NON** usare `if`
- il tuo codice deve funzionare per *qualunque* sequenza di esattamente 4 insiemi
- **SUGGERIMENTO**: puoi creare una lista di 3 booleani che verificano se un set è contenuto nel successivo…

Esempio 1 - data :

```
insiemi = [{'a','b'},
           {'a','b','c'},
           {'a','b','c','d','e'},
           {'a','b','c','d','e','f','g','h','i'}]
```

dopo il tuo codice, deve stampare:

```
La sequenza è a matrioska? True
```

Esempio 2 - data :

```
insiemi = [{'a','b'},
           {'a','b','c'},
           {'a','e','d'},
           {'a','b','d','e'}]
```

dopo il tuo codice, deve stampare:

```
La sequenza è a matrioska? False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[82]:
```
insiemi = [{'a','b'},
           {'a','b','c'},
           {'a','b','c','d','e'},
           {'a','b','c','d','e','f','g','h','i'}]


#insiemi = [{'a','b'},
#            {'a','b','c'},
#            {'a','e','d'},
#            {'a','b','d','e'}]


# scrivi qui

controlli = [ insiemi[0].issubset(insiemi[1]),
              insiemi[1].issubset(insiemi[2]),
              insiemi[2].issubset(insiemi[3]) ]

#print("La sequenza è a matrioska?", controlli.count(True) == 3)
```

[82]:
```
insiemi = [{'a','b'},
           {'a','b','c'},
           {'a','b','c','d','e'},
           {'a','b','c','d','e','f','g','h','i'}]


#insiemi = [{'a','b'},
#            {'a','b','c'},
#            {'a','e','d'},
#            {'a','b','d','e'}]


# scrivi qui
```

4.17.9 Esercizi con le funzioni

ATTENZIONE: Gli esercizi seguenti richiedono di conoscere:

Controllo di flusso[175]

Funzioni[176]

sillabe

Scrivere una funzione `sillabe` che data una stringa `parola` costituita solo da bisillabe e un insieme `trovate` passato alla funzione, trova tutte le bisillabe distinte e le mette nell'insieme `trovate`.

- **NOTA**: la funzione `sillabe` NON ritorna nulla !

Esempio 1:

```
>>> trovate = set()
>>> sillabe("banana", trovate)
>>> print(trovate)
{'an', 'ba'}
```

Esempio 2:

```
>>> trovate = set()
>>> sillabe("bonobo", trovate)
>>> print(trovate)
{'bo', 'on'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[83]:
```
# scrivi qui
def sillabe(parola, t):
    for i in range(len(parola)//2):
        t.add(parola[i:i+2])

trovate = set()
sillabe("banana", trovate)
#print(trovate)

trovate = set()
sillabe("bonobo", trovate)
#print(trovate)
```

[83]:
```
# scrivi qui
```

[175] https://it.softpython.org/#control-flow
[176] https://it.softpython.org/functions/functions-sol.html

distingui

⊛⊛ Scrivere una funzione `distingui` che data una lista `listona` contenente sottoliste di due caratteri ciascuna, RITORNA una NUOVA LISTA contenente tutte le sottoliste *distinte* (ignorando quindi le sottoliste duplicate)

- la lista ritornata deve avere gli elementi *nello stesso ordine* in cui li si è trovati in `listona`
- per sapere velocemente se una sottolista è già stata incontrata o meno, **usare un insieme**
- **NON** effettuare ricerche in liste (quindi niente `count`, `index`, `in` in liste - sono lenti) !
- **NON** effettuare rimozioni da liste (quindi niente `remove` da liste - è lenta) !
- **SUGGERIMENTO**: le liste sono *mutabili* , possiamo metterle in un insieme? Se non è possibile, come possiamo fare?

Esempio:

```
>>> listona= [ ['d','d'],['a','b'],['d','d'],['c','a'],['c','a'],['d','d'],['a','b'] ]
>>> print(distingui( listona))
[['d', 'd'], ['a', 'b'], ['c', 'a']]
#NOTA: la variabile listona NON deve essere modificata:
>>> print(listona)
[ ['d','d'],['a','b'],['d','d'],['c','a'],['c','a'],['d','d'],['a','b'] ]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[84]:
```
# scrivi qui

def distingui(lona):
    s = set()
    ret = []

    for sottolista in lona:
        # Negli insiemi non possiamo mettere le liste perchè sono mutabili,
        # ma possiamo mettere le tuple
        tup = tuple(sottolista)

        #  Verificare se un elemento appartiene ad un insieme è molto veloce:
        #  è indipendente dalla dimensione dell'insieme!

        if tup not in s:
            ret.append(sottolista)
            #  Aggiungere un elemento ad un insieme è molto veloce:
            #  è indipendente dalla dimensione dell'insieme!
            s.add(tup)

    return ret

listona = [ ['d','d'],['a','b'],['d','d'],['c','a'],['c','a'],['d','d'],['a','b'] ]
#print('distinte:', distingui(listona))
#print('listona:', listona)
```

[84]:
```
# scrivi qui
```

4.17.10 Verifica comprensione

ATTENZIONE

Gli esercizi che seguono contengono dei test con gli *assert*. Per capire come svolgerli, leggi prima Gestione errori e testing[177]

Esercizio - intersectron

Dati una lista `insiemi` contenente un numero arbitrario di insiemi, RITORNA un NUOVO insieme che contiene gli elementi comuni a tutti gli insiemi.

Per risolvere l'esercizio, si può intersecare un insieme alla volta con un ciclo `for` (lento) oppure la tecnica descritta qui[178] (breve ed efficace).

- prova a risolvere in **entrambi** i modi
- **ATTENTO** alla lista vuota !
- il tuo codice deve funzionare con un **qualsiasi** numeri di insiemi (l'immagine è solo un esempio)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[85]:
```
def inter_for(insiemi):



    if len(insiemi) == 0:
        return set()

    primo = True

    for el in insiemi:
        if primo:
            ret = set(el)
            primo = False
        else:
            ret.intersection_update(el)
```

(continues on next page)

[177] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html
[178] https://stackoverflow.com/a/2541814

(continued from previous page)

```
    return ret


# INIZIO TEST - NON TOCCARE !
assert inter_for([]) == set()
assert inter_for([set(),set()]) == set()
assert inter_for([set(),set(),set()]) == set()
assert inter_for([{'a'},{'a'},{'a'}]) == {'a'}
assert inter_for([{'a','b'},{'b'},{'b'}]) == {'b'}
assert inter_for([{'a'},{'a','b'},{'a'}]) == {'a'}
assert inter_for([{'c'},{'c'},{'c','b'}]) == {'c'}
assert inter_for([{'a','b'},{'a','b'},{'a','b'}]) == {'a','b'}
assert inter_for([{'a','b','c'},{'a','b','c','d'},{'b','c','d'}, {'b','c'}]) == {'b',
↪'c'}
# verifica che non abbiamo modificato gli insiemi di input
s = {'a','b'}
assert inter_for([s,{'b','c'}]) == {'b'}
assert s == {'a','b'}
# FINE TEST
```

[85]:
```
def inter_for(insiemi):

    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
assert inter_for([]) == set()
assert inter_for([set(),set()]) == set()
assert inter_for([set(),set(),set()]) == set()
assert inter_for([{'a'},{'a'},{'a'}]) == {'a'}
assert inter_for([{'a','b'},{'b'},{'b'}]) == {'b'}
assert inter_for([{'a'},{'a','b'},{'a'}]) == {'a'}
assert inter_for([{'c'},{'c'},{'c','b'}]) == {'c'}
assert inter_for([{'a','b'},{'a','b'},{'a','b'}]) == {'a','b'}
assert inter_for([{'a','b','c'},{'a','b','c','d'},{'b','c','d'}, {'b','c'}]) == {'b',
↪'c'}
# verifica che non abbiamo modificato gli insiemi di input
s = {'a','b'}
assert inter_for([s,{'b','c'}]) == {'b'}
assert s == {'a','b'}
# FINE TEST
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[86]:
```
def inter_fast(insiemi):



    if len(insiemi) == 0:
        return set()

    return set.intersection(*insiemi)
```

(continues on next page)

(continued from previous page)

```
# INIZIO TEST - NON TOCCARE !
assert inter_fast([]) == set()
assert inter_fast([set(),set()]) == set()
assert inter_fast([set(),set(),set()]) == set()
assert inter_fast([{'a'},{'a'},{'a'}]) == {'a'}
assert inter_fast([{'a','b'},{'b'},{'b'}]) == {'b'}
assert inter_fast([{'a'},{'a','b'},{'a'}]) == {'a'}
assert inter_fast([{'c'},{'c'},{'c','b'}]) == {'c'}
assert inter_fast([{'a','b'},{'a','b'},{'a','b'}]) == {'a','b'}
assert inter_fast([{'a','b','c'},{'a','b','c','d'},{'b','c','d'}, {'b','c'}]) == {'b',
↪'c'}
# verifica che non abbiamo modificato gli insiemi di input
s = {'a','b'}
assert inter_fast([s,{'b','c'}]) == {'b'}
assert s == {'a','b'}
# FINE TEST
```

[86]:
```
def inter_fast(insiemi):

    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
assert inter_fast([]) == set()
assert inter_fast([set(),set()]) == set()
assert inter_fast([set(),set(),set()]) == set()
assert inter_fast([{'a'},{'a'},{'a'}]) == {'a'}
assert inter_fast([{'a','b'},{'b'},{'b'}]) == {'b'}
assert inter_fast([{'a'},{'a','b'},{'a'}]) == {'a'}
assert inter_fast([{'c'},{'c'},{'c','b'}]) == {'c'}
assert inter_fast([{'a','b'},{'a','b'},{'a','b'}]) == {'a','b'}
assert inter_fast([{'a','b','c'},{'a','b','c','d'},{'b','c','d'}, {'b','c'}]) == {'b',
↪'c'}
# verifica che non abbiamo modificato gli insiemi di input
s = {'a','b'}
assert inter_fast([s,{'b','c'}]) == {'b'}
assert s == {'a','b'}
# FINE TEST
```

4.18 Dizionari 1 - Introduzione

4.18.1 Scarica zip esercizi

Naviga file online[179]

I dizionari sono dei contenitori mutabili che ci consentono di associare velocemente voci dette *chiavi* a dei *valori*

- Le *chiavi* sono immutabili, non hanno ordine e non vi possono essere duplicati
- I *valori* possono essere duplicati

Data una chiave, possiamo reperire velocemente il valore corrispondente.

[179] https://github.com/DavidLeoni/softpython-it/tree/master/dictionaries

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
dictionaries
    dictionaries1.ipynb
    dictionaries1-sol.ipynb
    dictionaries2.ipynb
    dictionaries2-sol.ipynb
    dictionaries3.ipynb
    dictionaries3-sol.ipynb
    dictionaries4.ipynb
    dictionaries4-sol.ipynb
    dictionaries5.ipynb
    dictionaries5-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `dictionaries1.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.18.2 Creare un dizionario

Nella vita di tutti i giorni quando pensiamo ad un dizionario tipicamente ci riferiamo ad un libro che data una voce (per es. `'sedia'`), ci permette **rapidamente** di trovare la relativa descrizione (es. `'un mobile per sedersi'`).

In Python abbiamo una struttura dati chiamata `dict` che ci consente di rappresentare comodamente i dizionari.

Riprendendo l'esempio di prima, potremmo creare un `dict` con diverse voci così:

[2]:
```
{'sedia':'un mobile per sedersi',
 'armadio':'un mobile a ripiani',
 'lampadario': 'un apparecchio di illuminazione'
}
```

[2]:
```
{'armadio': 'un mobile a ripiani',
 'lampadario': 'un apparecchio di illuminazione',
 'sedia': 'un mobile per sedersi'}
```

Chiariamo un momento i nomi:

> I dizionari sono dei contenitori mutabili che ci consentono di associare velocemente **voci dette chiavi** a dei **valori**.

La definizione ci dice che le voci in Python vengono chiamate *chiavi* (nell esempio `'sedia'`, `'armadio'` etc), mentre quelle che nell'esempio sono descrizioni (`'un mobile per sedersi'`) in Python le chiamiamo *valori*.

Quando creaiamo un dizionario, scriviamo prima una graffa `{`, poi la facciamo seguire da una serie di coppie chiave `:` valore, ciascuna seguita da una virgola `,` (tranne l'ultima, in cui la virgola è opzionale). Alla fine chiudiamo con una graffa `}`.

Mettere spazi o ritorni a capo all'interno **è opzionale**. Quindi possiamo scrivere anche così:

[3]:
```
{
 'sedia'     : 'un mobile per sedersi',
 'armadio'   : 'un mobile a ripiani',
 'lampadario': 'un apparecchio di illuminazione'
}
```

[3]:
```
{'armadio': 'un mobile a ripiani',
 'lampadario': 'un apparecchio di illuminazione',
 'sedia': 'un mobile per sedersi'}
```

O anche tutto su una riga:

[4]:
```
{'sedia':'un mobile per sedersi','armadio':'un mobile a ripiani','lampadario':'un␣
↪apparecchio di illuminazione'}
```

[4]:
```
{'armadio': 'un mobile a ripiani',
 'lampadario': 'un apparecchio di illuminazione',
 'sedia': 'un mobile per sedersi'}
```

Nota che se usiamo parole brevi Python probabilmente stamperà il dizionario comunque su una riga:

[5]:
```
{'barca': 'remo',
 'auto': 'ruota',
 'aereo': 'ala'}
```

[5]:
```
{'aereo': 'ala', 'auto': 'ruota', 'barca': 'remo'}
```

Mettere una virgola anche dopo l'ultima coppia non provoca errori:

[6]:
```
{
 'barca': 'remo',
 'auto': 'ruota',
 'aereo': 'ala',  # nota virgola 'extra'
}
```

[6]:
```
{'aereo': 'ala', 'auto': 'ruota', 'barca': 'remo'}
```

Vediamo come viene rappresentato un dizionario in Python Tutor - per agevolarci, lo assegnamo alla variabile `diz`

[7]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[8]:
```

diz = {
 'sedia'     : 'un mobile per sedersi',
 'armadio'   : 'un mobile a ripiani',
 'lampadario': 'un apparecchio di illuminazione'
}
print(diz)
```

(continues on next page)

(continued from previous page)

```
jupman.pytut()
```

```
{'armadio': 'un mobile a ripiani', 'lampadario': 'un apparecchio di illuminazione',
↪'sedia': 'un mobile per sedersi'}
```

[8]:
```
<IPython.core.display.HTML object>
```

Notiamo che una volta eseguito, appare una freccia da `diz` che punta ad una regione di memoria arancione/gialla. Le chiavi hanno sfondo arancione, mentre i corrispondenti valori hanno sfondo giallo. Dalle frecce e colori, si può intuire che quando si parla di assegnazioni di variabili, i dizionari si comportano come altre strutture dati mutabili, come le liste e gli insiemi.

DOMANDA: Guarda il codice seguente, e prova ad immaginare cosa succederà durante l'esecuzione - alla fine, come sarà organizzata la memoria? Cosa sarà stampato? Dove andranno le frecce?

[9]:
```
da = {
 'sedia'     : 'un mobile per sedersi',
 'armadio'   : 'un mobile a ripiani',
 'lampadario': 'un apparecchio di illuminazione'
}

db = {
 'barca': 'remo',
 'auto': 'ruota',
 'aereo': 'ala'
}
dc = db
db = da
da = dc
dc = db
#print(da)
#print(db)
#print(dc)

jupman.pytut()
```

[9]:
```
<IPython.core.display.HTML object>
```

Le chiavi

Cerchiamo di capire meglio quali chiavi possiamo mettere riguardando la definizione:

> I dizionari sono dei contenitori mutabili che ci consentono di associare velocemente voci dette chiavi a dei valori
>
> - **Le chiavi sono immutabili, non hanno ordine e non vi possono essere duplicati**
> - I valori possono essere duplicati

DOMANDA: guarda bene le parole in grassetto - ti viene in mente una struttura dati già vista che ha quelle caratteristiche?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Le chiavi dei dizionari per molti aspetti si comportano come elementi di un insieme.

Hai letto bene il foglio sugli insiemi[180]**?**

Prima di proseguire, assicurati di aver capito bene la sezione Elementi mutabili e hash[181]

Le chiavi sono immutabili

DOMANDA: La definizione non impone di usare per forza stringhe come chiavi, possiamo usare anche altri tipi. Ma possiamo usare tutti quelli che vogliamo?

Per ciascuno degli esempi seguenti, prova a dire se il dizionario si può creare o se otterremo un errore (quale?). Controlla come sono rappresentati in Python Tutor.

1. interi

```
{
  4 : 'gatti',
  3 : 'cani'
}
```

2. float

```
{
  4.0 : 'gatti',
  3.0 : 'cani'
}
```

3. stringhe

```
{
  'a' : 'gatti',
  'b' : 'cani'
}
```

4. liste

```
{
    [1,2] : 'zam',
    [3,4] : 'zum'
}
```

5. tuple

```
{
    (1,2) : 'zam',
    (4,3) : 'zum'
}
```

6. insiemi

[180] https://it.softpython.org/sets/sets-sol.html
[181] https://it.softpython.org/sets/sets-sol.html#Elementi-mutabili-e-hash

```
{
    {1,2} : 'zam',
    {3,4} : 'zum'
}
```

7. altri dizionari (guarda la prima parte della definizione !)

```
{
    {'a':'x','b':'y'} : 'zam',
    {'c':'w','d':'z'} : 'zum'
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: interi, float, stringhe e tuple sono IMMUTABILI e quindi possiamo usarli come chiavi (vedere definizione). Invece, liste e insiemi (e altri dizionari) sono MUTABILI, e quindi non potremo usarli come chiavi. Se proviamo ad usare un elemento MUTABILE come una lista come se fosse la chiave di un dizionario, Python si offenderà, segnalandoci che l'oggetto non è *hashable* (esattamente come si offenderebbe se cercassimo di inserirlo come elemento di un insieme)

```
>>> {  [1,2]:'zam',
       [3,4]:'zum'}

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-c3c2d6cc97b8> in <module>
      1 {  [1,2]:'zam',
----> 2    [3,4]:'zum'}

TypeError: unhashable type: 'list'
```

Le chiavi non hanno ordine

In un dizionario che possiamo trovare nella vita reale, le voci sono sempre ordinate secondo qualche criterio, (tipicamente in ordine alfabetico)

Con Python invece dobbiamo considerare un'importante differenza:

- Le chiavi sono immutabili, **non hanno ordine** e non vi possono essere duplicati

Quando diciamo che una collezione 'non ha ordine', significa che l'ordine degli elementi che vediamo quando li inseriamo o stampiamo non ha alcuna importanza nel determinare se una collezione è uguale ad un'altra. Nel caso dei dizionari significa che cambiando l'ordine in cui sono specificate le coppie, otteniamo dizionari che Python considera uguali.

Per esempio, i seguenti dizionari sono da considerarsi tutti uguali:

[10]:
```
{
 'navi' :'porto',
 'aerei': 'aereoporto',
 'treni': 'stazione'
}
```

[10]:
```
{'aerei': 'aereoporto', 'navi': 'porto', 'treni': 'stazione'}
```

[11]:
```
{
 'aerei': 'aereoporto',
 'navi' :'porto',
 'treni': 'stazione'
}
```

[11]:
```
{'aerei': 'aereoporto', 'navi': 'porto', 'treni': 'stazione'}
```

[12]:
```
{
 'treni': 'stazione',
 'navi' : 'porto',
 'aerei': 'aereoporto',
}
```

[12]:
```
{'aerei': 'aereoporto', 'navi': 'porto', 'treni': 'stazione'}
```

Stampare un dizionario: avrai notato che Jupyter stampa sempre le chiavi nello stesso ordine alfabetico. Questa è una cortesia che ci fa Jupyter, ma non lasciarti ingannare! Se proviamo una stampa nativa con `print` otterremo un risultato diverso!

[13]:
```
print({
 'navi' :'porto',
 'aerei': 'aereoporto',
 'treni': 'stazione'
})
```

```
{'navi': 'porto', 'aerei': 'aereoporto', 'treni': 'stazione'}
```

Duplicati delle chiavi

> Le chiavi sono immutabili, non hanno ordine e **non vi possono essere duplicati**

Possiamo chiederci come Python gestisca i duplicati nelle chiavi. Proviamo di proposito a creare una coppia duplicata:

[14]:
```
{
 'sedia'     : 'un mobile per sedersi',
 'sedia'     : 'un mobile per sedersi',
 'lampadario': 'un apparecchio di illuminazione'
}
```

[14]:
```
{'lampadario': 'un apparecchio di illuminazione',
 'sedia': 'un mobile per sedersi'}
```

Notiamo che Python non si è lamentato e ha silenziosamente scartato il duplicato. E se provassimo ad inserire una coppia con stessa chiave ma valore diverso?

[15]:
```
{
 'sedia'     : 'un mobile per sedersi',
 'sedia'     : 'appoggio con schienale',
 'lampadario': 'un apparecchio di illuminazione'
}
```

[15]:
```
{'lampadario': 'un apparecchio di illuminazione',
 'sedia': 'appoggio con schienale'}
```

Notiamo che Python ha mantenuto solo l'ultima coppia.

I valori

Riguardiamo la definizione:

> I dizionari sono dei contenitori mutabili che ci consentono di associare velocemente voci dette chiavi a dei valori.
>
> - Le chiavi sono immutabili, non hanno ordine e non vi possono essere duplicati
> - **I valori possono essere duplicati**

Pare che per i valori vi siano meno vincoli rispetto alle chiavi

DOMANDA: Per ciascuno degli esempi seguenti, prova a dire se il dizionario si può creare o se otterremo un errore (quale?). Controlla come sono rappresentati in Python Tutor.

1. interi

```
{
    'a':3,
    'b':4
}
```

2. interi duplicati

```
{
    'a':3,
    'b':3
}
```

3. float

```
{
    'a':3.0,
    'b':4.0
}
```

4. stringhe

```
{
    'a' : 'ghiaccio',
    'b' : 'fuoco'
}
```

5. liste

```
{
    'a' : ['t','w'],
    'b' : ['x'],
    'c' : ['y','z','k']
}
```

6. liste duplicate

```
{
    'a' : ['x','y','z'],
    'b' : ['x','y','z']
}
```

7. liste contenenti duplicati

```
{
    'a' : ['x','y','y'],
    'b' : ['z','y','z']
}
```

8. tuple

```
{
    'a': (6,9,7),
    'b': (8,1,7,4)
}
```

9. insiemi

```
{
    'a' : {6,5,6},
    'b' : {2,4,1,5}
}
```

10. dizionari

```
{
    'a': {
            'x':3,
            'y':9
        },
    'b': {
            'x':3,
            'y':9,
            'z':10
        },
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Per i valori possiamo tranquillamente mettere quello che vogliamo, Python non si offenderà. In particolare nota come chiavi diverse possano avere lo stesso valore.

Dizionario vuoto

Possiamo creare un dizionario vuoto scrivendo `{}`:

ATTENZIONE: DA NON CONFONDERSI CON L'INSIEME VUOTO[182] !!

[16]:
```
{}
```

[16]:
```
{}
```

[182] https://it.softpython.org/sets/sets-sol.html#Insieme-vuoto

[17]:
```
type({})
```

[17]:
```
dict
```

Un dizionario è una collezione, e come già visto (con liste, tuple e insiemi), possiamo creare un dizionario vuoto scrivendo il suo tipo `dict` seguito da parentesi tonde:

[18]:
```
dict()
```

[18]:
```
{}
```

Vediamo come viene rappresentato in Python Tutor:

[19]:
```

diz = dict()

jupman.pytut()
```

[19]:
```
<IPython.core.display.HTML object>
```

Chiavi e valori eterogenei

Finora abbiamo sempre usato chiavi tutte dello stesso tipo e valori tutti dello stesso tipo, ma non è obbligatorio. Basta che i tipi delle chiavi siano immutabili:

[20]:
```
{
    "a": 3,
    "b": ["una", "lista"],
     7 : ("questa","è","una","tupla")
}
```

[20]:
```
{7: ('questa', 'è', 'una', 'tupla'), 'a': 3, 'b': ['una', 'lista']}
```

NOTA: Sebbene mischiare tipi sia possibile, è sconsigliabile!

Buttare nel dizionario tipi diversi spesso porta sfortuna, nel senso che aumenta la probabilità di incorrere in bug.

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1.
```
{'a':'b'        ,
 'c':'d'

 }
```

2.
```
{'a b':'c',
 'c d':'e f'}
```

3.
```
{'a' = 'c',
 'b' = 'd'}
```

4.
```
{'a':'b':
 'c':'d'}
```

5.
```
{
    "1":[2,3],
```

(continues on next page)

(continued from previous page)

```
    "2,3":1,
}
```

6.
```
type({'a:b,c:d'})
```

7.
```
{'a':'b';
 'c':'d'}
```

8.
```
{'a:b',
 'c:d'}
```

9.
```
{5,2:
 4,5}
```

10.
```
{1:2,
 1:3}
```

11.
```
{2:1,
 3:1}
```

12.
```
{'a':'b',
 'c':'d',}
```

13.
```
type({'a','b',
      'c','d'})
```

14.
```
{'a':'b',
 'c':'d',
 'e','f'}
```

15.
```
{{}: 2}
```

16.
```
{(1,2):[3,4]}
```

17.
```
{[1,2]:(3,4)}
```

18.
```
{'[1,2]':(3,4)}
```

19.
```
{{1,2}:(3,4)}
```

20.
```
{len({1,2}):(3,4)}
```

21.
```
{5:{'a':'b'}}
```

22.
```
{"a":{1:2}}
```

23.
```
{"a":{[1]:2}}
```

24.
```
{"a":{1:[2]}}
```

25.
```
{["a":{1:[2]}]}
```

26.
```
set([{2:4}])
```

4.18.3 Dizionario da sequenza di coppie

Possiamo ottenere un dizionario specificando come parametro della funzione `dict` una sequenza di coppie chiave/valore, per esempio potremmo passare una lista di tuple:

[21]:
```
dict(  [
         ('farina',500),
         ('uova',2),
         ('zucchero',200),
       ])
```

[21]:
```
{'farina': 500, 'uova': 2, 'zucchero': 200}
```

Possiamo usare anche altre sequenze, l'importante è che le sottosequenze siano tutte da due elementi. Qua per esempio abbiamo una tupla di liste:

[22]:
```
dict(  (
         ['farina',500],
         ['uova',2],
         ['zucchero',200],
       ))
```

[22]:
```
{'farina': 500, 'uova': 2, 'zucchero': 200}
```

Se una sottosequenza ha un numero di elementi diverso da due, otteniamo questo errore:

```
>>> dict(  (
         ['farina',500],
         ['uova','marce', 3,],
         ['zucchero',200],
       ))

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-88-563d301b4aef> in <module>
      2          ['farina',500],
      3          ['uova','marce', 3,],
----> 4          ['zucchero',200],
      5        ))

ValueError: dictionary update sequence element #1 has length 3; 2 is required
```

DOMANDA: Compara i seguenti codici. Fanno la stessa cosa? Se sì, quale preferiresti?

```
dict( {
        ('a',5),
        ('b',8),
        ('c',3)
      } )
```

```
dict( (
        {'a',5},
        {'b',8},
        {'c',3}
      )
    )
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: I due codici NON producono lo stesso risultato, e dobbiamo decisamente preferire il primo.

Sul nostro computer, abbiamo ottenuto questo:

ATTENZIONE: sul tuo computer potresti aver ottenuto risultati diversi!

```
# primo
>>> dict( {
            ('a',5),
            ('b',8),
            ('c',3)} )

{'b': 8, 'a': 5, 'c': 3}
```

```
# secondo
>>> dict( (
            {'a',5},
            {'b',8},
            {'c',3}) )

{'a': 5, 8: 'b', 3: 'c'}
```

Nel primo caso siamo partiti da un insieme di tuple: dato che è un insieme, gli elementi al suo interno sono memorizzati in un ordine che *non* possiamo predire. Quando Python controlla le tuple all'interno, per ognuna ricava una coppia chiave/valore. Ora, dalla definizione di dizionario sappiamo che anche le chiavi di un dizionario sono memorizzate senza un ordine preciso. Quindi, non ha nessuna importanza inserire chiavi in un ordine piuttosto che un'altro, la cosa importante è mantenere la distinzione chiave/valore. Nella stampa del dizionario vediamo le stesse coppie che gli abbiamo specificato, solo in ordine diverso. Le coppie giuste sono state create grazie al fatto che le tuple invece *sono* ordinate.

Nel secondo caso siamo invece partiti da una tupla di insiemi, quindi Python ha visitato gli elementi della tupla nello stesso ordine che vediamo, purtroppo specificando le coppie come insiemi non possiamo più essere sicuri dell'ordine in cui Python ha letto gli elementi all'interno degli insiemi. Sul nostro computer, con il primo insieme siamo stati fortunati e Python ha letto prima `a` e poi `5`, invece con gli altri insiemi ha letto prima il numero e poi il carattere! E sul tuo computer potresti aver ottenuto risultati completamente diversi !

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. ```
   dict('abcd')
   ```
2. ```
   dict(['ab','cd'])
   ```

3. ```
   dict(['a1','c2'])
   ```

4. ```
   dict([])
   ```

5. ```
   dict(())
   ```

6. ```
   dict(('  ',))    # nasty
   ```

4.18.4 Dizionario da parametri con nome

Come ulteriore metodo di creazione, possiamo specificare le chiavi come se fossero parametri con nome:

[23]:
```
dict(a=5,b=6)
```

[23]:
```
{'a': 5, 'b': 6}
```

ATTENZIONE: in questo caso le chiavi sono soggette alle stesse restrittive regole dei nomi di parametri di funzione!

Per esempio, usando le graffe questo dizionario è perfettamente legittimo:

[24]:
```
{'a b' : 2,
 'c d' : 6}
```

[24]:
```
{'a b': 2, 'c d': 6}
```

Ma se proviamo a crearlo usando `a b` come argomento di `dict`, avremo dei problemi:

```
>>> dict(a b=2, c d=6)

  File "<ipython-input-97-444f8661585a>", line 1
    dict(a b=2, c d=6)
           ^
SyntaxError: invalid syntax
```

Avremo dei problemi anche usando stringhe:

```
>>> dict('a b'=2,'c d'=6)

  File "<ipython-input-98-45aafbb56e81>", line 1
    dict('a b'=2,'c d'=6)
        ^
SyntaxError: keyword can't be an expression
```

E attenzione a furbate tipo usare variabili, non otterremo il risultato desiderato:

[25]:
```
ka = 'a b'
kc = 'c d'

dict(ka=2,kc=6)
```

[25]:
```
{'ka': 2, 'kc': 6}
```

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. ```
   dict(3=5,2=8)
   ```
2. ```
   dict('costi'=9,'ricavi'=15)
   ```
3. ```
   dict(_costi=9,_ricavi=15)
   ```
4. ```
   dict(33trentini=5)
   ```
5. ```
   dict(trentini33=5)
   ```
6. ```
   dict(trentini_33=5)
   ```
7. ```
   dict(trentini-33=5)
   ```
8. ```
   dict(costi=1=2,ricavi=3=3)
   ```
9. ```
   dict(costi=1==2,ricavi=3==3)
   ```
10. ```
    v1 = 6
    v2 = 8
    dict(k1=v1,k2=v2)
    ```

4.18.5 Copiare un dizionario

Ci sono due modi di copiare un dizionario, si può fare una copia *superficiale* (*shallow copy*) oppure una *copia in profondità* (*deep copy*).

Copia superficiale

E' possibile creare una copia superficiale (*shallow copy*) di un dizionario passando alla funzione `dict` un'altro dizionario:

[26]:
```
da = {'x':3,
      'y':5,
      'z':1}
```

[27]:
```
db = dict(da)
```

[28]:
```
print(da)
```
```
{'x': 3, 'y': 5, 'z': 1}
```

[29]:
```
print(db)
```
```
{'x': 3, 'y': 5, 'z': 1}
```

In Python Tutor vedremo apparire due regioni di memoria separate:

[30]:
```
da = {'x':3,
      'y':5,
      'z':1}
db = dict(da)

jupman.pytut()
```

[30]:
```
<IPython.core.display.HTML object>
```

DOMANDA: possiamo scrivere così? Rispetto all'esempio precedente, otterremo risultati diversi?

[31]:
```
da = {'x':3,
      'y':5,
      'z':1}
db = dict(dict(da))

jupman.pytut()
```

[31]:
```
<IPython.core.display.HTML object>
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Il codice produce gli stessi risultati dell'esempio precedente, per quanto non sia efficiente (verrà creato un dizionario temporaneo extra con la `dict` interna e immediatamente buttato via)

Valori mutabili: Nell'esempio abbiamo usato valori interi, che sono *immutabili*. E se provassimo dei valori *mutabili* come delle liste, che succederebbe?

[32]:
```
da = {'x':['a','b','c'],
      'y':['d'],
      'z':['e','f']}
db = dict(da)

jupman.pytut()
```

[32]:
```
<IPython.core.display.HTML object>
```

Se provi ad eseguire Python Tutor, vedrai un'esplosione di frecce che vanno dal nuovo dizionario `db` ai valori di `da` (che sono liste). Niente paura ! Nel prossimo foglio spiegheremo meglio il significato, per il momento tieni nota che **con la copia superficiale di valori mutabili il nuovo dizionario avrà in comune regioni di memoria con il dizionario originale.**

Copia in profondità

Quando ci sono regioni di memoria mutabili condivise come nel caso qua sopra, è più probabile introdurre bug.

Per aver regioni di memoria completamente separate, possiamo usare la *copia in profondità* (*deep copy*).

Per usarla, dobbiamo prima dire a Python che intendiamo usare funzioni presenti nel modulo `copy`, poi potremo usare la funzione `deepcopy`, a cui passeremo il dizionario da copiare:

[33]:
```
import copy

da = {'x':['a','b','c'],
```

(continues on next page)

(continued from previous page)

```
      'y':['d'],
      'z':['e','f']}
db = copy.deepcopy(da)

jupman.pytut()
```

[33]:
```
<IPython.core.display.HTML object>
```

Se esegui il codice in Python Tutor, noterai come partendo con la freccia da `db`, finiremo in una regione di memoria arancione/gialla completamente nuova che non condivide nulla con la regione di memoria puntata da `da`.

DOMANDA: Guarda il codice seguente - dopo la sua esecuzione, vedrai frecce che da `db` arrivano ad elementi di `da`?

[34]:
```
da = {'x': {1,2,3},
      'y': {4,5}}
db = dict(da)
jupman.pytut()
```

[34]:
```
<IPython.core.display.HTML object>
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La risposta è sì, perchè come valori del dizionario abbiamo usato insiemi che sono mutabili.

4.18.6 Prosegui

Prosegui con Dizionari 2[183]

[ ]:
```

```

4.19 Dizionari 2 - operatori

4.19.1 Scarica zip esercizi

Naviga file online[184]

Per manipolare i dizionari vi sono diversi operatori:

Operatore	Ritorna	Descrizione
`len`(dict)	`int`	Ritorna il numero di chiavi
dict`[`chiave`]`	obj	Ritorna il valore associato alla chiave
dict`[`chiave`]` = valore		Aggiunge o modifica il valore associato ad una chiave
`del` dict`[`chiave`]`		Rimuove la coppia chiave/valore
chiave `in` dict	`bool`	Ritorna `True` se chiave è presente nel dizionario
`==`, `!=`	`bool`	Controlla se due dizionari sono uguali o differenti

[183] https://it.softpython.org/dictionaries/dictionaries2-sol.html
[184] https://github.com/DavidLeoni/softpython-it/tree/master/dictionaries

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
dictionaries
    dictionaries1.ipynb
    dictionaries1-sol.ipynb
    dictionaries2.ipynb
    dictionaries2-sol.ipynb
    dictionaries3.ipynb
    dictionaries3-sol.ipynb
    dictionaries4.ipynb
    dictionaries4-sol.ipynb
    dictionaries5.ipynb
    dictionaries5-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `dictionaries2.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.19.2 len

E' possibile ottenere il numero di associazioni chiave/valore presente in un dizionario usando la funzione `len`:

[2]:
```
len({'a':5,
     'b':9,
     'c':7
})
```

[2]:
```
3
```

[3]:
```
len({3:8,
     1:3
})
```

[3]:
```
2
```

[4]:
```
len({})
```

[4]:
```
0
```

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. `len(dict())`
2. `len({'a':{}})`
3. `len({(1,2):{3},(4,5):{6},(7,8):{9}})`
4. `len({1:2,1:2,2:4,2:4,3:6,3:6})`
5. `len({1:2,',':3,',':4,})`
6. `len(len({3:4,5:6}))`

4.19.3 Leggere di un valore

In fondo alla definizione dei dizionari, è riportato

> **Data una chiave, possiamo reperire velocemente il valore corrispondente.**

Come possiamo specificare la chiave di ricerca? Basta usare le quadre `[ ]`, un po' come abbiamo già fatto per le liste:

[5]:
```
diz = {  'sedia'     : 'un mobile per sedersi',
         'armadio'   : 'un mobile a ripiani',
         'lampadario': 'un apparecchio di illuminazione'
}
```

[6]:
```
diz['sedia']
```

[6]:
```
'un mobile per sedersi'
```

[7]:
```
diz['lampadario']
```

[7]:
```
'un apparecchio di illuminazione'
```

ATTENZIONE Quello che mettiamo tra parentesi quadre **deve** essere una chiave presente nel dizionario

Se mettiamo chiavi non presenti, otterremo un errore:

```
>>> diz['tavolo']

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-19-ee891f51417b> in <module>
----> 1 diz['tavolo']

KeyError: 'tavolo'
```

Disordine veloce

Quando diamo una chiave a Python, quanto è veloce a reperire il valore corrispondente? Tanto, così veloce che la velocità *non dipende dalla dimensione del dizionario*. Che sia piccolo o enorme, data una chiave troverà il valore associato in circa il medesimo tempo.

Quando abbiamo in mano un dizionario nella vita reale, tipicamente abbiamo una voce da cercare e lo sfogliamo finchè troviamo la voce ordinata: il fatto che le voci siano ordinate è quello che ci consente di trovare rapidamente la voce.

Potremmo aspettarci lo stesso anche in Python, invece guardando la definizione troviamo una notevole differenza:

> I dizionari sono dei contenitori mutabili che ci consentono di associare velocemente voci dette chiavi a dei valori
>
> - Le chiavi sono immutabili, **non hanno ordine** e non vi possono essere duplicati
> - I valori possono essere duplicati

Se le chiavi *non* sono ordinate, come fa Python ad essere veloce a reperire i valori? La rapidità nasce dal fatto che Python memorizza le chiavi con un sistema con basato sugli *hash* simile al meccanismo impiegato per gli insiemi[185]. Il prezzo da pagare per noi è l'imposizione di dover usare chiavi di tipo *immutabile*.

DOMANDA: Se volessimo stampare il valore `'un apparecchio di illuminazione'` che vediamo in fondo al dizionario, senza sapere che corrisponde a `'lampadario'`, avrebbe senso scrivere qualcosa del genere ?:

```
arredo = {  'sedia'     : 'un mobile per sedersi',
            'armadio'   : 'un mobile a ripiani',
            'lampadario': 'un apparecchio di illuminazione'
}

print( arredo[2] )
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Assolutamente NO. Le coppie chiave/valore nel dizionario *non* sono ordinate, pertanto non ha nessun senso procurarsi un valore ad una certa posizione.

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

```
kabbalah = {
    1 : 'Progresso',
    3 : 'Amore',
    5 : 'Creazione'
}
```

- `kabbalah[0]`
- `kabbalah[1]`
- `kabbalah[2]`
- `kabbalah[3]`

[185] https://it.softpython.org/sets/sets-sol.html#Elementi-mutabili-e-hash

- `kabbalah[4]`
- `kabbalah[5]`
- `kabbalah[-1]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Nel dizionario abbiamo delle chiavi che sono dei numeri interi. Pertanto possiamo usare dei numeri tra le quadre, ma indicheranno *chiavi*, *non posizioni*.

Le uniche espressioni che produrranno risultati sono quelle per cui il numero specificato tra quadre è effettivamente presente tra le chiavi:

```
>>> kabbalah[1]
'Progresso'
>>> kabbalah[3]
'Amore'
>>> kabbalah[5]
'Creazione'
```

Tutte le altre daranno `KeyError`, per es:

```
>>> kabbalah[2]

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-29-de66b9721e9b> in <module>
      5 }
      6
----> 7 kabbalah[2]

KeyError: 2
```

DOMANDA: Guarda le seguenti espressioni, e per ciascuna cerca di indovinare quale risultato producono (o se danno errore):

1. `{'a':4,'b':5}('a')`
2. `{1:2,2:3,3:4}[2]`
3. `{'a':1,'b':2}['c']`
4. `{'a':1,'b':2}[a]`
5. `{'a':1,'b':2}[1]`
6. `{'a':1,'b':2,'c':3}['c']`
7. `{'a':1,'b':2,'c':3}[len(['a','b','c'])]`

8.
```
{(3,4):(1,2)}[(1,2)]
```

9.
```
{(1,2):(3,4)}[(1,2)]
```

10.
```
{[1,2]:[3,4]}[[1,2]]
```

11.
```
{'a','b','c'}['a']
```

12.
```
{'a:b','c:d'}['c']
```

13.
```
{'a':4,'b':5}{'a'}
```

14.
```
d1 = {'a':'b'}
d2 = {'b':'c'}
print(d1[d2['c']])
```

15.
```
d1 = {'a':'b'}
d2 = {'b':'c'}
print(d2[d1['a']])
```

16.
```
{}[]
```

17.
```
{[]:3}[[]]
```

18.
```
{1:7}['1']
```

19.
```
{'':7}"[]"
```

20.
```
{'':7}[""]
```

21.
```
{"":7}['']
```

22.
```
{'"':()}['']
```

23.
```
{():7}[()]
```

24.
```
{(()):7}[()]
```

25.
```
{(()):7}[((),)]
```

Esercizio - z7

⊛ Dato un dizionario `diz1` con chiavi `'b'` e `'c'` associate a numeri, crea un dizionario `diz2` che abbia una chiave `'z'` associata alla somma dei valori delle chiavi `'b'` e `'c'` di `diz1`

- il tuo codice deve funzionare per *qualunque* `diz1` con chiavi `'b'` e `'c'`

Esempio - dato:

```
diz1 = {'a':6, 'b':2,'c':5}
```

Dopo il tuo codice, deve risultare:

```
>>> print(diz2)
{'z': 7}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[8]:
```
diz1 = {'a':6, 'b':2,'c':5}

# scrivi qui

diz2 = {'z' : diz1['b'] + diz1['c']}

print(diz2)
```

```
{'z': 7}
```

[8]:
```
diz1 = {'a':6, 'b':2,'c':5}

# scrivi qui
```

```
{'z': 7}
```

4.19.4 Scrivere nel dizionario

Possiamo scrivere in un dizionario?

> **I dizionari sono dei contenitori mutabili** che ci consentono di associare velocemente voci dette chiavi a dei valori

La definizione parla di mutabilità, quindi una volta creati, possiamo successivamente modificarli.

I dizionari sono collezioni di coppie chiave/valore, e tra le modifiche possibili troviamo:

1. aggiunta di una coppia chiave/valore
2. associare una chiave esistente ad un valore diverso
3. rimuovere una coppia chiave/valore

Scrivere - aggiunta chiave/valore

Supponiamo di aver creato il nostro dizionario `arredo`

[9]:
```
arredo = {  'sedia'     : 'un mobile per sedersi',
            'armadio'   : 'un mobile a ripiani',
            'lampadario': 'un apparecchio di illuminazione'
}
```

e vogliamo in seguito aggiungere una definizione per `'divano'`. Possiamo riusare la variabile `arredo` seguita da quadre con dentro la chiave che vogliamo aggiungere `['divano']` e dopo le quadre metteremo un segno di uguale =

[10]:
```
arredo['divano'] = 'mobile per rilassarsi'
```

Nota che Jupyter non ha mostrato risultati, perchè l'operazione precedente è un *comando* di assegnamento (solo le *espressioni* generano risultati).

Ma qualcosa comunque internamente nella memoria è successo, lo possiamo verificare stampando `arredo`:

[11]:
```
arredo
```

[11]:
```
{'armadio': 'un mobile a ripiani',
 'divano': 'mobile per rilassarsi',
 'lampadario': 'un apparecchio di illuminazione',
 'sedia': 'un mobile per sedersi'}
```

Notiamo che il dizionario associato alla variabile `arredo` è stato MODIFICATO con l'aggiunta del divano.

Quando aggiungiamo una coppia chiave/valore, possiamo usare tipi eterogenei:

[12]:
```
bidone = {'bla':3,
          4 : 'boh',
          (7,9) : ['spaz','zatura']
         }
```

[13]:
```
bidone[5.0] = 'un float'
```

[14]:
```
bidone
```

[14]:
```
{(7, 9): ['spaz', 'zatura'], 4: 'boh', 5.0: 'un float', 'bla': 3}
```

E siamo soggetti agli stessi vincoli sulle chiavi che abbiamo durante la creazione, quindi possiamo solo usare chiavi *immutabili*. Se proviamo ad immettere un tipo *mutabile* come per es. una lista, otteniamo un errore:

```
>>> bidone[ ['una', 'lista']  ] = 8

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-51-195ac9c21bcd> in <module>
----> 1 bidone[ ['una', 'lista']  ] = 8

TypeError: unhashable type: 'list'
```

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
diz = {1:'a'}
diz[2] = 'a'
print(diz)
```

2.
```
diz = {}
print(len(diz))
diz['a'] = 'b'
print(len(diz))
```

3.
```
diz1 = {'a':3, 'b':4}
diz2 = diz1
diz1['a'] = 5
print(diz1)
print(diz2)
```

4.
```
diz1 = {'a':3, 'b':4}
diz2 = dict(diz1)
diz1['a'] = 5
print(diz1)
print(diz2)
```

5.
```
la = ['a','c']
diz = {'a':3,
       'b':4,
       'c':5}
diz['d'] = diz[la[0]] + diz[la[1]]
print(diz)
```

6.
```
diz = {}
diz[()]: ''
diz[('a',)]: 'A'
diz[('a','b')]: 'AB'
print(diz)
```

7.
```
la = [5,8,6,9]
diz = {}
diz[la[0]]=la[2]
diz[la[2]]=la[0]
print(diz)
```

8.
```
diz = {}
diz[(4,5,6)[2]] = 'c'
diz[(4,5,6)[1]] = 'b'
diz[(4,5,6)[0]] = 'a'
print(diz)
```

9.
```
diz1 = {
    'a' : 'x',
    'b' : 'x',
    'c' : 'y',
    'd' : 'y',
}

diz2 = {}
diz2[diz1['a']] = 'a'
diz2[diz1['b']] = 'b'
diz2[diz1['c']] = 'c'
diz2[diz1['d']] = 'd'
print(diz2)
```

Scrivere - riassociare chiave

Supponiamo di voler cambiare la definizione di `lampadario`:

[15]:
```
arredo = {  'sedia'     : 'un mobile per sedersi',
            'armadio'   : 'un mobile a ripiani',
            'lampadario': 'un apparecchio di illuminazione'
}
```

[16]:
```
arredo['lampadario'] = 'un apparecchio di illuminazione appeso al soffitto'
```

[17]:
```
arredo
```

[17]:
```
{'armadio': 'un mobile a ripiani',
 'lampadario': 'un apparecchio di illuminazione appeso al soffitto',
 'sedia': 'un mobile per sedersi'}
```

Esercizio - officina

⊛ MODIFICA il dizionario `officina`:

1. poni il valore della chiave `'bulloni'` uguale al valore della chiave `'tenaglie'`
2. incrementa il valore della chiave `ruote` di 1

- il tuo codice deve funzionare per qualunque numero associato alle chiavi
- **NON** creare nuovi dizionari, quindi niente linee che cominciano con `officina = {`

Esempio - dati:

```
officina = {'ruote':3,
            'bulloni':2,
            'tenaglie':5}
```

dopo il tuo codice, devi ottenere:

```
>>> print(officina)
{'ruote': 4, 'bulloni': 5, 'tenaglie': 5}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
officina = {'ruote':3,
            'bulloni':2,
            'tenaglie':5}

# scrivi qui

officina['ruote'] = officina['ruote'] + 1
officina['bulloni'] = officina['tenaglie']
#print(officina)
```

[18]:
```
officina = {'ruote':3,
            'bulloni':2,
            'tenaglie':5}

# scrivi qui
```

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
diz = {'a':'b'}
diz['a'] = 'a'
print(diz)
```

2.
```
diz = {'1':'2'}
diz[1] = diz[1] + 5    # nasty
print(diz)
```

3.
```
diz = {1:2}
diz[1] = diz[1] + 5
print(diz)
```

4.
```
d1 = {1:2}
d2 = {2:3}
d1[1] = d2[d1[1]]
print(d1)
```

Scrivere - cancellare

Per cancellare una coppia chiave/valore esiste il comando speciale `del`. Prendiamo un dizionario:

[19]:
```
cucina = { 'pentole' : 3,
           'padelle': 7,
           'forchette' : 20
}
```

Se vogliamo eliminare la coppia `'padelle' : 7`, scriveremo `del` seguito dal nome del dizionario e la chiave da eliminare tra quadre:

[20]:
```
del cucina['padelle']
```

[21]:
```
cucina
```

[21]:
```
{'forchette': 20, 'pentole': 3}
```

Cercare di cancellare una chiave inesistente produrrà un errore:

```
>>> del cucina['spinterogeno']

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-34-c0d541348698> in <module>
----> 1 del cucina['spinterogeno']
```

(continues on next page)

(continued from previous page)

```
KeyError: 'spinterogeno'
```

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. ```
   diz = {'a':'b'}
   del diz['b']
   print(diz)
   ```

2. ```
   diz = {'a':'b', 'c':'d'}
   del diz['a']
   print(diz)
   ```

3. ```
   diz = {'a':'b', 'c':'d'}
   del diz['a']
   del diz['a']
   print(diz)
   ```

4. ```
   diz = {'a':'b'}
   new_diz = del diz['a']
   print(diz)
   print(new_diz)
   ```

5. ```
   diz1 = {'a':'b', 'c':'d'}
   diz2 = diz1
   del diz1['a']
   print(diz1)
   print(diz2)
   ```

6. ```
   diz1 = {'a':'b', 'c':'d'}
   diz2 = dict(diz1)
   del diz1['a']
   print(diz1)
   print(diz2)
   ```

7. ```
   diz = {'a':'b'}
   del diz['c']
   print(diz)
   ```

8. ```
   diz = {'a':'b'}
   diz.del('a')
   print(diz)
   ```

9. ```
   diz = {'a':'b'}
   diz['a'] = None
   print(diz)
   ```
   ```

Esercizio - scrivania

Dato un dizionario `scrivania`:

```
scrivania = {
            'carta':5,
            'matite':2,
            'penne':3
}
```

scrivi del codice che lo MODIFICA in modo che dopo l'esecuzione del tuo codice, il dizionario appaia così:

```
>>> print(scrivania)
{'carta': 4,'matite': 2, 'temperino': 1}
```

- **NON** scrivere linee che iniziano con `scrivania =` (questo creerebbe un nuovo dizionario, invece noi vogliamo modificare quello esistente)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
scrivania = {
            'carta':5,
            'matite':2,
            'penne':3
}

# scrivi qui
scrivania['carta'] = 4
del scrivania['penne']
scrivania['temperino'] = 1
#print(scrivania)
```

[22]:
```
scrivania = {
            'carta':5,
            'matite':2,
            'penne':3
}

# scrivi qui
```

Esercizio - giardino

Hai un dizionario `giardino` che associa nomi di oggetti presenti alla loro quantità. Ti vengono fornite:

- una lista `da_togliere` contenente i nomi di esattamente 2 oggetti da eliminare
- un dizionario `da_aggiungere` contenente 2 nomi di fiori associati alla loro quantità da aggiungere

MODIFICA il dizionario `giardino` secondo le quantità indicate da `da_togliere` (**cancellando** le chiavi) e `da_aggiungere` (**incrementando** i valori corrispondenti)

- assumi che `giardino` contenga sempre gli oggetti indicati in `da_togliere` e `da_aggiungere`

- assumi che `da_aggiungere` contenga sempre e solo `tulipani` e `rose`

Esempio:

```
da_togliere = ['erbacce', 'cartacce']
da_aggiungere = { 'tulipani':4,
                  'rose' : 2
}

giardino = { 'ortensie':3,
             'tulipani':7,
             'erbacce' : 10,
             'rose' : 5,
             'cartacce' : 6,
}
```

dopo il tuo codice, deve stampare

```
>>> print(giardino)
{'ortensie': 3, 'tulipani': 11, 'rose': 7}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
da_togliere = ['erbacce', 'cartacce']
da_aggiungere = {
    'tulipani':4,
    'rose' : 2
}

giardino = { 'ortensie':3,
             'tulipani':7,
             'erbacce' : 10,
             'rose' : 5,
             'cartacce' : 6,
}

# scrivi qui

del giardino[da_togliere[0]]
del giardino[da_togliere[1]]
giardino['rose'] = giardino['rose'] + da_aggiungere['rose']
giardino['tulipani'] = giardino['tulipani'] + da_aggiungere['tulipani']
#print(giardino)
```

[23]:
```
da_togliere = ['erbacce', 'cartacce']
da_aggiungere = {
    'tulipani':4,
    'rose' : 2
}

giardino = { 'ortensie':3,
             'tulipani':7,
             'erbacce' : 10,
             'rose' : 5,
```

(continues on next page)

(continued from previous page)

```
            'cartacce' : 6,
}

# scrivi qui
```

Esercizio - traduzioni

Dati due dizionari `en_it` e `it_es` di traduzioni inglese-italiano e italiano-spagnolo, scrivi del codice che MODIFICA un terzo dizionario `en_es` mettendoci traduzioni dall'inglese allo spagnolo

- assumi che `en_it` contenga sempre e solo le traduzioni di `hello` e `road`
- assumi che `it_es` contenga sempre e solo le traduzioni di `ciao` e `strada`
- nella soluzione, usa **SOLO** le costanti `'hello'` e `'road'`, le altre che ti servono dovrai recuperarle usando i dizionari
- **NON** creare un nuovo dizionario - quindi niente linee che iniziano con `en_es = {`

Esempio - dati:

```
en_it = {
    'hello' : 'ciao',
    'road' : 'strada'
}

it_es = {
    'ciao' : 'hola',
    'strada' : 'carretera'
}
en_es = {}
```

dopo il tuo codice, dovrà stampare:

```
>>> print(en_es)
{'hello': 'hola', 'road': 'carretera'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[24]:
```
en_it = {
    'hello' : 'ciao',
    'road' : 'strada'
}

it_es = {
    'ciao' : 'hola',
    'strada' : 'carretera'
}

en_es = {}

# scrivi qui
```

(continues on next page)

(continued from previous page)

```
en_es['hello'] = it_es[en_it['hello']]
en_es['road'] = it_es[en_it['road']]
#print(en_es)
```

[24]:
```
en_it = {
    'hello' : 'ciao',
    'road' : 'strada'
}

it_es = {
    'ciao' : 'hola',
    'strada' : 'carretera'
}

en_es = {}

# scrivi qui
```

4.19.5 Appartenenza con `in`

Per verificare se una *chiave* è presente in un dizionario, possiamo usare l'operatore `in`:

[25]:
```
'a' in {'a':5,'b':7}
```

[25]:
```
True
```

[26]:
```
'b' in {'a':5,'b':7}
```

[26]:
```
True
```

[27]:
```
'z' in {'a':5,'b':7}
```

[27]:
```
False
```

ATTENZIONE: `in` cerca nelle *chiavi* , non nei *valori* !

[28]:
```
5 in {'a':5,'b':7}
```

[28]:
```
False
```

Come sempre quando operiamo con chiavi, *non* possiamo cercare un oggetto che sia mutabile, come per esempio le liste:

```
>>> [3,5] in {'a':'c','b':'d'}

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-41-3e3e336117aa> in <module>
----> 1 [3,5] in {'a':'c','b':'d'}

TypeError: unhashable type: 'list'
```

not in

E' possibile verificare la *non* appartenza con l'operatore `not in`:

```
[29]: 'z' not in {'a':5,'b':7}
[29]: True
```

```
[30]: 'a' not in {'a':5,'b':7}
[30]: False
```

Equivalentemente, possiamo usare quest'altra forma:

```
[31]: not 'z' in {'a':5,'b':7}
[31]: True
```

```
[32]: not 'a' in {'a':5,'b':7}
[32]: False
```

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. `('a') in {'a':5}`
2. `('a','b') in {('a','b'):5}`
3. `('a','b',) in {('a','b'):5}`
4. `['a','b'] in {('a','b'):5}`
5. `{3: 'q' in {'q':5}}`
6. `{'q' not in {'q':0} : 'q' in {'q':0}}`
7. `{'a' in 'b'}`
8. `{'a' not in {'b':'a'}}`
9. `len({'a':6,'b':4}) in {1:2}`
10. `'ab' in {('a','b'): 'ab'}`
11. `None in {}`
12. `None in {'None':3}`
13. `None in {None:3}`
14. `not None in {0:None}`

4.19.6 Dizionari di sequenze

Finora abbiamo quasi sempre associato alle chiavi un solo valore. E se volessimo associarne di più? Per esempio, supponiamo di essere una biblioteca e vogliamo associare agli utenti i libri che hanno preso in prestito. Potremmo rappresentare il tutto come un dizionario in cui al nome di ciascun utente si associa una lista con i libri presi in prestito:

[33]:
```
prestiti = {'Marco': ['I Miserabili', 'Ulisse'],
            'Gloria': ['Guerra e pace'],
            'Rita': ['Shining','Dracula','1984']}
```

Vediamo come è rappresentato in Python Tutor:

[34]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[35]:
```
prestiti = {'Marco': ['I Miserabili', 'Ulisse'],
            'Gloria': ['Guerra e pace'],
            'Rita': ['Shining','Dracula','1984']}
jupman.pytut()
```

[35]:
```
<IPython.core.display.HTML object>
```

Se proviamo a scrivere l'espressione

[36]:
```
prestiti['Rita']
```

[36]:
```
['Shining', 'Dracula', '1984']
```

Python ci mostra la lista corrispondente. Quindi Python considera `prestiti['Rita']` a tutti gli effetti come una lista, e come tale la possiamo usare. Per esempio, se volessimo accedere al libro unesimo della lista scriveremmo `[1]` dopo l'espressione

[37]:
```
prestiti['Rita'][1]
```

[37]:
```
'Dracula'
```

Equivalentemente, potremmo anche salvarci un puntatore alla lista assegnando l'espressione ad una variabile:

[38]:
```
lista_rita = prestiti['Rita']
```

[39]:
```
lista_rita
```

[39]:
```
['Shining', 'Dracula', '1984']
```

[40]:
```
lista_rita[1]
```

[40]:
```
'Dracula'
```

Rivediamo il tutto in Python Tutor:

[41]:
```
prestiti = {'Marco':  ['I Miserabili', 'Ulisse'],
            'Gloria': ['Guerra e pace'],
            'Rita':   ['Shining','Dracula','1984']}
lista_rita = prestiti['Rita']
print(lista_rita[1])

jupman.pytut()
```

```
Dracula
```

[41]:
```
<IPython.core.display.HTML object>
```

Se esegui il codice in Python Tutor, noterai come nel momento in cui assegnamo `lista_rita` la lista corrispondente a Rita sembra 'staccarsi' dal dizionario. Questo è un solo effetto grafico causato da Python Tutor, dal punto di vista del dizionario non è cambiato nulla. L'intenzione è mostrare che la lista adesso è *raggiungibile* sia dal dizionario che dalla nuova variabile `lista_rita`.

4.19.7 Esercizio - prestiti

Scrivi del codice per recuperare e stampare:

1. Il primo libro preso in prestito da Gloria (`'Guerra e Pace'`) e l'ultimo preso in prestito da Rita (`'1984'`)
2. Il numero di libri presi in prestito da Rita
3. `True` se tutti tra Marco, Gloria e Rita hanno preso in prestito almeno un libro, `False` altrimenti

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[42]:
```
prestiti = {'Marco':  ['I Miserabili', 'Ulisse'],
            'Gloria': ['Guerra e pace'],
            'Rita':   ['Shining','Dracula','1984']}

# scrivi qui
print("1. Il primo libro preso in prestito da Gloria è", prestiti['Gloria'][0])
print("   L'ultimo libro preso in prestito da Rita è", prestiti['Rita'][-1])
print("2. Rita ha preso in prestito", len(prestiti['Rita']), "libro/i")
res = len(prestiti['Marco']) > 0 and len(prestiti['Gloria']) > 0 and len(prestiti[
↪'Rita']) > 0
print("3. Hanno preso tutti in prestito almeno un libro?", res)
```

```
1. Il primo libro preso in prestito da Gloria è Guerra e pace
   L'ultimo libro preso in prestito da Rita è 1984
2. Rita ha preso in prestito 3 libro/i
3. Hanno preso tutti in prestito almeno un libro? True
```

[42]:
```
prestiti = {'Marco':  ['I Miserabili', 'Ulisse'],
            'Gloria': ['Guerra e pace'],
            'Rita':   ['Shining','Dracula','1984']}

# scrivi qui
```

```
1. Il primo libro preso in prestito da Gloria è Guerra e pace
   L'ultimo libro preso in prestito da Rita è 1984
2. Rita ha preso in prestito 3 libro/i
3. Hanno preso tutti in prestito almeno un libro? True
```

4.19.8 Uguaglianza

Possiamo verificare se due dizionari sono uguali con l'operatore di uguaglianza ==, che dati due dizionari ritorna `True` se contengono coppie chiave/valore uguali oppure `False` altrimenti:

```
[43]: {'a':3, 'b':4} == {'a':3, 'b':4}
[43]: True
```

```
[44]: {'a':3, 'b':4} == {'c':3, 'b':4}
[44]: False
```

```
[45]: {'a':3, 'b':4} == {'a':3, 'b':999}
[45]: False
```

Possiamo verificare l'uguaglianza di dizionari con numero di elementi diverso:

```
[46]: {'a':3, 'b':4} == {'a':3}
[46]: False
```

```
[47]: {'a':3, 'b':4} == {'a':3,'b':3,'c':5}
[47]: False
```

… e con elementi eterogenei:

```
[48]: {'a':3, 'b':4} == {2:('q','p'), 'b':[99,77]}
[48]: False
```

Uguaglianza e ordine

Dalla definizione:

- Le chiavi sono immutabili, **non hanno ordine** e non vi possono essere duplicati

Visto che l'ordine non ha importanza, dizionari creati inserendo le stesse coppie chiavi/valore ma in ordine diverso saranno considerati uguali.

Esempio con creazione diretta:

```
[49]: {'a':5, 'b':7} == {'b':7, 'a':5}
[49]: True
```

Esempio con aggiunta graduale:

```
[50]: diz1 = {}
      diz1['a'] = 5
      diz1['b'] = 7

      diz2 = {}
      diz2['b'] = 7
      diz2['a'] = 5

      print(diz1 == diz2)
```

```
True
```

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
{1:2} == {2:1}
```

2.
```
{1:2,3:4} == {3:4,1:2}
```

3.
```
{'a'.upper():3} == {'a':3}
```

4.
```
{'A'.lower():3} == {'a':3}
```

5.
```
{'a': {1:2} == {3:4}}
```

6.
```
diz1 = {}
diz1[2] = 5
diz1[3] = 7

diz2 = {}
diz2[3] = 7
diz2[2] = 5
print(diz1 == diz2)
```

7.
```
diz1 = {'a':3,'b':8}
diz2 = diz1
diz1['a'] = 7
print(diz1 == diz2)
```

8.
```
diz1 = {}
diz1['a']=3
diz2 = diz1
diz2['a']=4
print(diz1 == diz2)
```

9.
```
diz1 = {}
diz1['a']=3
diz2 = diz1
diz2['a']=4
print(diz1 == diz2)
```

10.
```
diz1 = {'a':3, 'b':4, 'c':5}
diz2 = {'a':3,'c':5}
del diz1['a']
print(diz1 == diz2)
```

11.
```
diz1 = {}
diz2 = {'a':3}
diz1['a'] = 3
diz1['b'] = 5
diz2['b'] = 5
print(diz1 == diz2)
```

Uguaglianza e copie

Quando si duplicano contenitori che contengono oggetti mutabili, se non si presta attenzione si possono ottenere sorprese. Ritorniamo quindi sull'argomento copie di dizionari superficiale e in profondità, questa volta cercando di verificare l'effettiva uguaglianza con Python.

ATTENZIONE: Per comprendere quanto segue, è necessario (ri)guardare bene il foglio dizionari 1 - Copiare un dizionario[186]

DOMANDA: Vediamo un esempio semplice, con una copia 'manuale'. Se esegui il seguente codice in Python Tutor, cosa stamperà? Quante regioni di memoria vedrai?

```
diz1 = {'a':3,
        'b':8}
diz2 = {'a':diz1['a'],
        'b':diz1['b'] }
diz1['a'] = 6

print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)
```

NOTA: tutti i valori (`3` e `8`) sono **immutabili**.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: In questo caso abbiamo creato manualmente un dizionario `diz2` indicando dei valori *immutabili* presi da `diz1`. Pertanto in Python Tutor vedremo due regioni di memoria distinte e una successiva modifica a `diz1` non altererà `diz2`:

[51]:
```
diz1 = {'a':3,
        'b':8}
diz2 = {'a':diz1['a'],
        'b':diz1['b'] }
diz1['a'] = 6

print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)

jupman.pytut()
```

```
uguali? False
diz1= {'b': 8, 'a': 6}
diz2= {'b': 8, 'a': 3}
```

[51]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Se esegui il seguente codice in Python Tutor, cosa stamperà?

1. Che tipo di copia abbiamo fatto? Superficiale? In profondità? (o tutte e due..?)
2. Quante regioni di memoria vedrai?

[186] https://it.softpython.org/dictionaries/dictionaries1-sol.html#Copiare-un-dizionario

```
diz1 = {'a':3,
        'b':8}
diz2 = dict(diz1)
diz1['a'] = 7

print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: `dict` quando usato come funzione esegue una *copia superficiale* (*shallow copy*), cioè copia la struttura del dizionario senza duplicare i valori mutabili. In questo caso specifico, tutti i valori che abbiamo sono interi quindi immutabili, perciò la copia può anche essere considerata una duplicazione completa. Quando assegnamo il valore `7` alla chiave `'a'` in `diz1` stiamo modificando la struttura dati originale, lasciando inalterata la copia `diz2` appena fatta, pertanto `diz1 == diz2` varrà `False`. Verifichiamolo in Python Tutor:

[52]:
```
diz1 = {'a':3,
        'b':8}
diz2 = dict(diz1)
diz1['a'] = 7

print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)

jupman.pytut()
```

```
uguali? False
diz1= {'b': 8, 'a': 7}
diz2= {'b': 8, 'a': 3}
```

[52]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Se esegui il seguente codice in Python Tutor, cosa stamperà?

1. Che tipo di copia abbiamo fatto? Superficiale? In profondità? (o tutte e due..?)
2. Quante regioni di memoria vedrai?

NOTA: i valori sono liste, perciò **mutabili**

```
diz1 = {'a':[1,2],
        'b':[4,5,6]}
diz2 = dict(diz1)
diz1['a'].append(3)

print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Abbiamo usato `dict` come una funzione, quindi abbiamo fatto una *copia superficiale*. In questo caso come

valori abbiamo liste che sono oggetti *mutabili*. Questo vuol dire che la copia superficiale si è limitata a copiare i riferimenti alle liste, ma *non* le liste stesse. Per questa ragione vedrai delle frecce puntare dalla copia del dizionario `diz2` alle regioni di memoria delle liste originali. Questo significa che se cerchi di modificare una lista dopo che la copia è avvenuta (per esempio con il metodo `.append(3)`), di fatto modificherai anche la lista raggiungibile dal dizionario copiato `diz2`. Verifichiamolo in Python Tutor:

[53]:
```
diz1 = {'a':[1,2],
        'b':[4,5,6]}
diz2 = dict(diz1)
diz1['a'].append(3)

print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)

jupman.pytut()
```

```
uguali? True
diz1= {'b': [4, 5, 6], 'a': [1, 2, 3]}
diz2= {'b': [4, 5, 6], 'a': [1, 2, 3]}
```

[53]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Se esegui il seguente codice in Python Tutor, cosa stamperà?

1. Che tipo di copia abbiamo fatto? Superficiale? In profondità? (o tutte e due..?)
2. Quante regioni di memoria vedrai?

NOTA: i valori sono liste, perciò **mutabili**

```
import copy
diz1 = {'a':[1,2],
        'b':[4,5,6]}
diz2 = copy.deepcopy(diz1)
diz1['a'].append(3)

print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Abbiamo usato `copy.deepcopy`, quindi abbiamo fatto una copia *in profondità*. In questo caso come valori abbiamo liste che sono oggetti mutabili. La copia in profondità ha duplicato qualunque oggetto che è riuscita a raggiungere, liste incluse. Quindi in questo caso otteniamo due regioni di memoria completamente distinte. Se dopo aver effettuato la copia cerchiamo di modificare una lista accessibile dall'originale `diz1`, siamo sicuri che non potremo intaccare oggetti raggiungibili da `diz2`. Verifichiamolo in Python Tutor:

[54]:
```
import copy
diz1 = {'a':[1,2],
        'b':[4,5,6]}
diz2 = copy.deepcopy(diz1)
diz1['a'].append(3)
```

(continues on next page)

(continued from previous page)

```
print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)

jupman.pytut()
```

```
uguali? False
diz1= {'b': [4, 5, 6], 'a': [1, 2, 3]}
diz2= {'b': [4, 5, 6], 'a': [1, 2]}
```

[54]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
diz1 = {'a':[4,5],
        'b':[6,7]}
diz2 = dict(diz1)
diz2['a'] = diz1['b']
diz2['b'][0] = 9
print(diz1 == diz2)
print(diz1)
print(diz2)
```

2.
```
da = {'a':['x','y','z']}
db = dict(da)
db['a'] = ['w','t']
dc = dict(db)
print(da)
print(db)
print(dc)
```

3.
```
import copy

la = ['x','y','z']
diz1 = {'a':la,
        'b':la }
diz2 = copy.deepcopy(diz1)
diz2['a'][0] = 'w'
print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)
```

4.19.9 Esercizio - ZOOM DOOM

Scrivi del codice che data una stringa `s` (es `'ZOOM'`), crea un dizionario `diz` ed assegna alle chiavi `'a'`, `'b'` e `'c'` *la stessa identica lista* contenente i caratteri della stringa come elementi (es `['Z','O','O','M']`).

- in Python Tutor dovrai vedere 3 frecce che dalle chiavi puntano *alla stessa identica regione di memoria*
- modificando la lista associata ad una chiave, dovresti vedere la modifica anche nei liste associate alle altre chiavi
- il tuo codice deve funzionare per *qualunque* stringa `s`

Esempio - data:

```
s = 'ZOOM'
```

Dopo il tuo codice, deve risultare:

```
>>> print(diz)
{'a': ['Z', 'O', 'O', 'M']
 'b': ['Z', 'O', 'O', 'M'],
 'c': ['Z', 'O', 'O', 'M'],
}
>>> diz['a'][0] = 'D'
>>> print(diz)
{'a': ['D', 'O', 'O', 'M']
 'b': ['D', 'O', 'O', 'M'],
 'c': ['D', 'O', 'O', 'M'],
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[55]:
```
s = 'ZOOM'

# scrivi qui

zoom = list(s)

diz = {'a':zoom,
       'b':zoom,
       'c':zoom}
print(diz)
diz['a'][0] = 'D'
print(diz)

#jupman.pytut()
```

```
{'b': ['Z', 'O', 'O', 'M'], 'a': ['Z', 'O', 'O', 'M'], 'c': ['Z', 'O', 'O', 'M']}
{'b': ['D', 'O', 'O', 'M'], 'a': ['D', 'O', 'O', 'M'], 'c': ['D', 'O', 'O', 'M']}
```

[55]:
```
s = 'ZOOM'

# scrivi qui
```

```
{'b': ['Z', 'O', 'O', 'M'], 'a': ['Z', 'O', 'O', 'M'], 'c': ['Z', 'O', 'O', 'M']}
{'b': ['D', 'O', 'O', 'M'], 'a': ['D', 'O', 'O', 'M'], 'c': ['D', 'O', 'O', 'M']}
```

4.19.10 Prosegui

Prosegui con Dizionari 3[187]

```
[ ]:
```

4.20 Dizionari 3 - metodi e classi

4.20.1 Scarica zip esercizi

Naviga file online[188]

In questo foglio vedremo i metodi principali per estrarre informazioni e manipolare i dizionari, assieme a delle classi di dizionari speciali

Metodi:

Metodo	Ritorna	Descrizione
`dict.keys()`	`dict_keys`	Ritorna una *vista* di chiavi che sono presenti nel dizionario
`dict.values()`	`dict_values`	Ritorna una *vista* di valori presenti nel dizionario
`dict.items()`	`dict_items`	Ritorna una *vista* di coppie (chiave, valore) presenti nel dizionario
`diz1.update(diz2)`	`None`	MODIFICA il dizionario `diz1` con le coppie chiave / valore trovate in `diz2`

Classi:

Classe	Descrizione
OrderedDict[189]	Dizionario che permette di mantenere l'ordine di inserimento delle chiavi
Counter[190]	Dizionario che permette di calcolare rapidamente istogrammi

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
dictionaries
dictionaries
    dictionaries1.ipynb
    dictionaries1-sol.ipynb
    dictionaries2.ipynb
    dictionaries2-sol.ipynb
    dictionaries3.ipynb
    dictionaries3-sol.ipynb
    dictionaries4.ipynb
    dictionaries4-sol.ipynb
    dictionaries5.ipynb
    dictionaries5-sol.ipynb
    jupman.py    jupman.py
```

[187] https://it.softpython.org/dictionaries/dictionaries3-sol.html
[188] https://github.com/DavidLeoni/softpython-it/tree/master/dictionaries
[189] https://docs.python.org/3/library/collections.html#collections.OrderedDict
[190] https://docs.python.org/3/library/collections.html#collections.Counter

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `dictionaries3.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.20.2 keys

Chiamando il metodo `.keys()` possiamo ottenere tutte le chiavi del dizionario:

[2]:
```
verdure = {'carote':5,
           'pomodori':8,
           'cavoli':3}
```

[3]:
```
verdure.keys()
```

[3]:
```
dict_keys(['cavoli', 'pomodori', 'carote'])
```

ATTENZIONE: LA SEQUENZA RITORNATA E' DI TIPO `dict_keys`

`dict_keys` potrebbe sembrare una lista ma è ben diversa !

In particolare, la sequenza `dict_keys` ritornata è **una vista** sul dizionario originale. In informatica, quando parliamo di *viste* di solito intendiamo collezioni che contengono una parte degli oggetti contenuti in un'altra collezione, *e se la collezione originale viene modificata, si modifica contemporaneamente anche la vista*.

Vediamo cosa vuol dire. Per prima cosa proviamo ad assegnare la sequenza di chiavi ad una variabile:

[4]:
```
chiavi = verdure.keys()
```

Poi modifichiamo il dizionario originale, aggiungendo una associazione:

[5]:
```
verdure['patate'] = 8
```

Se adesso stampiamo `chiavi`, dovremmo vedere la modifica:

[6]:
```
chiavi
```

[6]:
```
dict_keys(['patate', 'cavoli', 'pomodori', 'carote'])
```

Quando riusi la sequenza da `.keys()` poni attenzione ad eventuali modifiche successive al dizionario di partenza

Se vogliamo una versione stabile che sia una specie di 'fotografia' in un dato momento delle chiavi del dizionario, dobbiamo esplicitamente convertirle ad una altra sequenza, come per esempio `list`:

[7]:
```
come_lista = list(verdure.keys())
```

[8]:
```
come_lista
```

[8]:
```
['patate', 'cavoli', 'pomodori', 'carote']
```

[9]:
```
verdure['cetrioli'] = 9
```

[10]:
```
come_lista     # niente cetrioli
```

[10]:
```
['patate', 'cavoli', 'pomodori', 'carote']
```

Rivediamo l'esempio in Python Tutor:

[11]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[12]:
```
verdure = {'carote':5,
           'pomodori':8,
           'cavoli':3}
chiavi = verdure.keys()
verdure['patate'] = 8
come_lista = list(verdure.keys())
verdure['cetrioli'] = 9
print(come_lista)

jupman.pytut()
```

```
['patate', 'cavoli', 'pomodori', 'carote']
```

[12]:
```
<IPython.core.display.HTML object>
```

ATTENZIONE: NON POSSIAMO ACCEDERE AD UA SPECIFICA POSIZIONE DI `dict_keys`

Se ci proviamo, otterremo un errore:

```
>>> verdure = {'carote':5,
               'pomodori':8,
               'cavoli':3}
>>> chiavi = verdure.keys()
>>> chiavi[0]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-90-c888bf602918> in <module>()
----> 1 chiavi[0]

TypeError: 'dict_keys' object does not support indexing
```

ATTENZIONE: NON POSSIAMO MODIFICARE DIRETTAMENTE `dict_keys`

Non vi sono operazioni o metodi che ci permettono di cambiare gli elementi di `dict_keys`, si può solo agire sul dizionario originale.

DOMANDA: Guarda questi frammenti di codice. Per ciascuno, prova a indovinare se può funzionare, e quale risultato produce.

1.
```
diz = {'a':4,
       'b':5}

chiavi = diz.keys()
chiavi.append('c')
```

2.
```
diz = {'a':4,
       'b':5}

chiavi = diz.keys()
chiavi.add('c')
```

3.
```
diz = {'a':4,
       'b':5}

chiavi = diz.keys()
chiavi['c'] = 3
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Nessuno degli esempi può funzionare, dato che non possiamo modificare direttamente oggetti di tipo `dict_keys`. Operatori come le quadre o metodi come l'`.append`, `.add`, etc non sono supportati.

DOMANDA: Guarda questi frammenti di codice. Per ciascuno, prova a indovinare che risultato produce (o se da errore)

1.
```
diz = {'a':1,'b':2}
s = set(diz.keys())
s.add(('c',3))
print(diz)
print(s)
```

2.
```
diz = {'a':3,'b':4}
k = diz.keys()
diz['c'] = 5
print(len(k))
```

3.
```
diz = {'a':'x',
       'b':'y'}
print('a' in diz.keys())
```

4.
```
diz1 = {'a':1,'b':2}
chiavi = diz1.keys()
diz2 = dict(diz1)
```

(continues on next page)

(continued from previous page)

```
diz2['c'] = 3
print('diz1=',diz1)
print('diz2=',diz2)
print('chiavi=',chiavi)
```

5.
```
diz1 = {'a':'b','c':'d'}
diz2 = {'a':'b','b':'c'}
print( set(diz1.keys()) - set(diz2.keys()) )
```

6.
```
diz1 = {'a':'b','c':'d'}
diz2 = {'e':'a','f':'c'}
chiavi = diz1.keys()
del diz1[diz2['e']]
del diz1[diz2['f']]
print(len(chiavi))
```

Esercizio - chiavi disordinate

⊛ STAMPA una LISTA con tutte le chiavi del dizionario.

- **NOTA 1**: NON è necessario che la lista sia ordinata
- **NOTA 2**: per convertire una qualsiasi sequenza a lista, usa la funzione predefinita `list`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[13]:
```
diz = {'c':6, 'b':2,'a':5}

# scrivi qui

list(diz.keys())
```

[13]:
```
['b', 'a', 'c']
```

[13]:
```
diz = {'c':6, 'b':2,'a':5}

# scrivi qui
```

[13]:
```
['b', 'a', 'c']
```

Esercizio - chiavi ordinate

⊛ STAMPA una LISTA con tutte le chiavi del dizionario.

- **NOTA 1**: Adesso E' necessario che la lista sia ordinata
- **NOTA 2**: per convertire una qualsiasi sequenza a lista, usa la funzione predefinita `list`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[14]:
```
diz = {'c':6, 'b':2,'a':5}

# scrivi qui

lista = list(diz.keys())
lista.sort()
print(lista)
```

```
['a', 'b', 'c']
```

[14]:
```
diz = {'c':6, 'b':2,'a':5}

# scrivi qui
```

```
['a', 'b', 'c']
```

Esercizio - chiavistello

Dati i dizionari `diz1` e `diz2`, scrivi del codice che mette in una **lista** `chiavi` tutte le chiavi dei due dizionari, **senza duplicati** e **ordinate alfabeticamente**, e infine stampa la lista.

- il tuo codice deve funzionare per qualunque `diz1` e `diz2`

Esempio - dati:

```
diz1 = {
    'a':5,
    'b':9,
    'e':2,

}
diz2 = {'a':9,
        'c':2,
        'e':2,
        'f':6}
```

dopo il tuo codice, deve risultare:

```
>>> print(chiavi)
['a', 'b', 'c', 'e', 'f']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[15]:
```
diz1 = {
    'a':5,
    'b':9,
    'e':2,

}
diz2 = {'a':9,
        'c':2,
        'e':2,
        'f':6}


# scrivi qui
chiavi = list(set(diz1.keys()) | set(diz2.keys()))
chiavi.sort()
#print(chiavi)
```

[15]:
```
diz1 = {
    'a':5,
    'b':9,
    'e':2,

}
diz2 = {'a':9,
        'c':2,
        'e':2,
        'f':6}


# scrivi qui
```

4.20.3 values

Dato un dizionario, è possibile ottenere tutti i valori chiamando il metodo `.values()`

Supponiamo di avere un dizionario `veicoli` che ad ogni targa di automobile assegna un proprietario:

[16]:
```
veicoli = {
    'AA111AA' : 'Mario',
    'BB222BB' : 'Lidia',
    'CC333CC' : 'Mario',
    'DD444DD' : 'Gino',
    'EE555EE' : 'Gino'
}

proprietari = veicoli.values()
```

ATTENZIONE: LA SEQUENZA RITORNATA E' DI TIPO `dict_values`

`dict_values` può sembrare una lista ma non lo è !

Come nel caso di `dict_keys`, anche `dict_values` è **una vista** sul dizionario originale, quindi aggiungendo un'associazione a `veicoli`:

[17]:
```
veicoli['FF666FF'] = 'Paola'
```

Anche la vista`proprietari` risulterà automaticamente cambiata:

[18]:
```
proprietari
```

[18]:
```
dict_values(['Lidia', 'Mario', 'Gino', 'Gino', 'Mario', 'Paola'])
```

Notiamo anche che essendo *valori* di un dizionario, sono ammessi duplicati.

ATTENZIONE: NON POSSIAMO ACCEDERE AD UNA SPECIFICA POSIZIONE DI `dict_values`

Se ci proviamo, otterremo un errore:

```
>>> proprietari[0]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-90-c888bf602918> in <module>()
----> 1 proprietari[0]

TypeError: 'dict_values' object does not support indexing
```

ATTENZIONE: NON POSSIAMO MODIFICARE DIRETTAMENTE `dict_values`

Non vi sono operazioni o metodi che ci permettono di cambiare gli elementi di `dict_values`, si può solo agire sul dizionario originale.

DOMANDA: Guarda questi frammenti di codice. Per ciascuno, prova a indovinare se può funzionare, e quale risultato produce.

1.
```
diz = {'a':4,
       'b':5}

valori = diz.values()
valori.append(4)
```

2.
```
d = {0:'a',
     1:'b',
     2:'b'}
vs = d.values()
d[2]='c'
print(vs)
```

3.
```
diz = {'a':4,
       'b':5}

valori = diz.values()
valori.add(5)
```

4.
```
diz = {0:1,
       1:2,
       2:3}

diz[list(diz.values())[0]-1]
```

5.
```
diz = {'a':4,
       'b':5}

valori = diz.values()
valori['c'] = 6
```

6.
```
diz = {'a':4,
       'b':5}

valori = diz.values()
valori[6] = 'c'
```

4.20.4 Esercizio - uno a uno

Dato un dizionario `diz`, scrivi del codice che stampa `True` se ad ogni chiave corrisponde un valore *diverso* dai valori corrispondenti a tutte le altre chiavi, altrimenti stampa `False`.

Esempio 1 - dati

```
diz = {'a' : 3,
       'c' : 6,
       'g' : 8}
```

Dopo il tuo codice, deve stampare `True` (perchè 3,6 e 8 sono tutti diversi)

```
True
```

Esempio 2 - dati

```
diz = {'x' : 5,
       'y' : 7,
       'z' : 5}
```

deve stampare

```
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
diz = {'a' : 3,
       'c' : 6,
       'g' : 8}

"""
diz = {'x' : 5,
       'y' : 7,
       'z' : 5}
"""

# scrivi qui

print(len(diz.keys()) == len(set(diz.values())))
```

```
True
```

[19]:
```
diz = {'a' : 3,
       'c' : 6,
       'g' : 8}

"""
diz = {'x' : 5,
       'y' : 7,
       'z' : 5}
"""

# scrivi qui
```

```
True
```

4.20.5 Esercizio - borsa

Dato un dizionario `diz` di associazioni tra caratteri, scrivi del codice che mette nella variabile `borsa` la lista ordinata di tutte le chiavi e i valori.

Esempio - dato

```
diz = {
    'a':'b',
    'b':'f',
    'c':'b',
    'd':'e'
}
```

Dopo il tuo codice, deve stampare

```
>>> print(borsa)
['a', 'b', 'c', 'd', 'e', 'f']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[20]:
```
diz = {
    'a':'b',
    'b':'f',
    'c':'b',
    'd':'e'
}

# scrivi qui

borsa = list(set(diz.keys()) | set(diz.values()))
borsa.sort()

print(borsa)
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```

[20]:
```
diz = {
    'a':'b',
    'b':'f',
    'c':'b',
    'd':'e'
}

# scrivi qui
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```

Esercizio - valori comuni

Dati due dizionari `diz1` e `diz2`, scrivere del codice che STAMPA `True` se hanno *almeno* un valore in comune (senza considerare le chiavi).

Esempio 1 - dati

```
diz1 = {
    'a':4,
    'k':2,
    'm':5
}

diz2 = {
    'b':2,
    'e':4,
    'g':9,
    'h':1
}
```

dopo il tuo codice, deve stampare `True` (perchè hanno i valori `2` e `4` in comune):

```
Hanno valori in comune? True
```

Esempio 2 - dati

```
diz1 = {
    'd':1,
    'e':2,
    'f':6
}

diz2 = {
    'a':3,
    'b':5,
    'c':9,
    'd':7
}
```

dopo il tuo codice, deve stampare:

```
Hanno valori in comune? False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[21]:
```
diz1 = {
    'a':4,
    'k':2,
    'm':5
}

diz2 = {
    'b':2,
    'e':4,
    'g':9,
    'h':1
}

"""
diz1 = {
    'd':1,
    'e':2,
    'f':6
}

diz2 = {
    'a':3,
    'b':5,
    'c':9,
    'd':7
}
"""

# scrivi qui

print('Hanno valori in comune?',len(set(diz1.values()) & set(diz2.values())) > 0)
```

```
Hanno valori in comune? True
```

[21]:
```
diz1 = {
    'a':4,
    'k':2,
    'm':5
}

diz2 = {
    'b':2,
    'e':4,
    'g':9,
    'h':1
}

"""
diz1 = {
    'd':1,
    'e':2,
    'f':6
}

diz2 = {
    'a':3,
    'b':5,
    'c':9,
    'd':7
}
"""

# scrivi qui
```

```
Hanno valori in comune? True
```

Esercizio - piccolo grande

Dato un dizionario `diz` che ha interi come chiavi e valori, stampa `True` se la chiave più piccola è uguale al valore più grande.

Esempio 1 - dato:

```
diz = {
    14:1,
    11:7,
    7:3,
    70:5
}
```

dopo il tuo codice, deve stampare `True` (perchè chiave minima `7` è uguale a valore massimo `7`)

```
True
```

Esempio 2 - dato:

```
diz = {
    12:1,
```

(continues on next page)

(continued from previous page)

```
    11:9,
    7:3,
    2:5,
    9:1
}
```

dopo il tuo codice, deve stampare `False` (perchè chiave minima `2` è diversa da valore massimo `9`):

```
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
diz = {
    14:1,
    11:7,
    7:3,
    70:5
}

"""
diz = {
    12:1,
    11:9,
    7:3,
    2:5,
    9:1
}
"""

# scrivi qui

min(diz.keys()) == max(diz.values())
```

[22]:
```
True
```

[22]:
```
diz = {
    14:1,
    11:7,
    7:3,
    70:5
}

"""
diz = {
    12:1,
    11:9,
    7:3,
    2:5,
    9:1
}
"""

# scrivi qui
```

(continues on next page)

(continued from previous page)

```
[22]: True
```

4.20.6 items

Possiamo ricavare tutte le associazioni chiave/valore come lista di coppie di tipo tupla con il metodo `.items()`. Vediamo un esempio che associa attrazioni da visitare alla città dove si trovano:

```
[23]: vacanza = {'Piazza S.Marco':'Venezia',
           'Fontana di Trevi':'Roma',
           'Uffizi':'Firenze',
           'Colosseo':'Roma',
}
```

```
[24]: vacanza.items()
```

```
[24]: dict_items([('Uffizi', 'Firenze'), ('Piazza S.Marco', 'Venezia'), ('Colosseo', 'Roma
↪'), ('Fontana di Trevi', 'Roma')])
```

In questo caso vediamo che ci è ritornato un oggetto di tipo `dict_items`. Come nei precedenti casi, è **una vista** che **non** possiamo modificare direttamente. Se il dizionario originale viene cambiato, la mutazione si rifletterà sulla vista:

```
[25]: attrazioni = vacanza.items()
```

```
[26]: vacanza['Palazzo Ducale'] = 'Venezia'
```

```
[27]: attrazioni
```

```
[27]: dict_items([('Uffizi', 'Firenze'), ('Piazza S.Marco', 'Venezia'), ('Palazzo Ducale',
↪'Venezia'), ('Colosseo', 'Roma'), ('Fontana di Trevi', 'Roma')])
```

DOMANDA: Guarda questi frammenti di codice. Per ciascuno, prova a indovinare se può funzionare, e quale risultato produce.

1. `{'a':7, 'b':9}.items()[0] = ('c',8)`
2. `dict({'a':7,'b':5}.items())['a']`
3. `len(set({'a':'b', 'a':'B'}.items()))`
4. `{'a':2}.items().find(('a',2))`
5. `{'a':2}.items().index(('a',2))`
6. `list({'a':2}.items()).index(('a',2))`
7.
```
diz1 = {'a':7,
        'b':5}
diz2 = dict(diz1.items())
diz1['a'] = 6
print(diz1 == diz2)
```

8. ```
   ('a','b') in {'a':('a','b'), 'b':('a','b')}.items()
   ```

9. ```
   ('a','b') in list({'a':('a','b'), 'b':('a','b')}.items())[0]
   ```

Esercizio - unione senza update

Dati i dizionari `diz1` e `diz2`, scrivi del codice che crea un NUOVO dizionario `diz3` contenente tutte le coppie chiave/valore da `diz1` e `diz2`.

- si suppone che tutte le coppie chiave/valore siano distinte
- **NON** usare cicli
- **NON** usare `.update()`
- il tuo codice deve funzionare per *qualsiasi* `diz1` e `diz2`

Esempio - dati:

```
diz1 = {'a':4,
        'b':7}
diz2 = {'c':5,
        'd':8,
        'e':2}
```

dopo il tuo codice, deve risultare (l'ordine non è importante):

```
>>> print(diz3)
{'a': 4, 'e': 2, 'd': 8, 'c': 5, 'b': 7}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
diz1 = {'a':4,
        'b':7}
diz2 = {'c':5,
        'd':8,
        'e':2}

# scrivi qui
diz3 = dict(list(diz1.items()) + list(diz2.items()))
#print(diz3)
```

[28]:
```
diz1 = {'a':4,
        'b':7}
diz2 = {'c':5,
        'd':8,
        'e':2}

# scrivi qui
```

4.20.7 update

Avendo un dizionario di partenza, è possibile MODIFICARLO unendone un'altro con il metodo `.update()`:

[29]:
```
diz1 = {'capre':6,
        'cavoli':9,
        'pastori':1}


diz2 = {'capre':12,
        'cavoli':15,
        'panche':3,
        'fieno':7}
```

[30]:
```
diz1.update(diz2)
```

[31]:
```
diz1
```

[31]:
```
{'capre': 12, 'cavoli': 15, 'fieno': 7, 'panche': 3, 'pastori': 1}
```

Notare come le chiavi in comune tra i dizionari come `'capre'` e `'cavoli'` e abbiano valori dal secondo.

Volendo, si può anche passare una sequenza di coppie così:

[32]:
```
diz1.update([('fieno',3),('panche',18), ('stalle',4)])
```

[33]:
```
diz1
```

[33]:
```
{'capre': 12,
 'cavoli': 15,
 'fieno': 3,
 'panche': 18,
 'pastori': 1,
 'stalle': 4}
```

4.20.8 Esercizio - axby

Dato un dizionario `diz` che associa caratteri a caratteri ed una stringa `s` formattata con coppie di caratteri come `ax` separate da punto e virgola `;`, sostituire tutti i valori in `diz` con i corrispondenti valori indicati nella stringa

- il tuo codice deve funzionare per *qualsiasi* dizionario `diz` e lista `s`

Esempio - dati

```
diz = {
    'a':'x',
    'b':'y',
    'c':'z',
    'd':'w'
}
s = "bx;cw;ex"
```

dopo il tuo codice, deve risultare

```
>>> diz
{'a': 'x', 'b': 'x', 'c': 'w', 'd': 'w', 'e': 'x'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[34]:
```
diz = {
    'a':'x',
    'b':'y',
    'c':'z',
    'd':'w'
}
s = "bx;cw;ex"

# scrivi qui

la = s.split(';')
diz.update(la)
diz
```

[34]:
```
{'a': 'x', 'b': 'x', 'c': 'w', 'd': 'w', 'e': 'x'}
```

[34]:
```
diz = {
    'a':'x',
    'b':'y',
    'c':'z',
    'd':'w'
}
s = "bx;cw;ex"

# scrivi qui
```

[34]:
```
{'a': 'x', 'b': 'x', 'c': 'w', 'd': 'w', 'e': 'x'}
```

4.20.9 Classi - OrderedDict

Come abbiamo detto in precedenza, quando stampiamo un dizionario con `print` o lasciamo la visualizzazione a Jupyter, l'ordine il più delle volte non è lo stesso di quello d'inserimento. Affinchè sia predicibile, devi usare un `OrderedDict`

Per poterlo usare, per prima cosa devi importarlo dal modulo delle collezioni:

[35]:
```
from collections import OrderedDict
```

[36]:
```
od = OrderedDict()
```

Un `OrderedDict` appare e si comporta come dizionari regolari:

[37]:
```
od['qualche chiave'] = 5
od['qualche altra chiave'] = 7
od[('una', 'tupla', 'immutabile','come chiave')] = 3
od["un'altra chiave"] = 'adesso una stringa!'
od[123] = 'hello'
```

La visualizzazione con Jupyter mantiene l'ordine:

[38]:
```
od
```

[38]:
```
OrderedDict([('qualche chiave', 5),
             ('qualche altra chiave', 7),
             (('una', 'tupla', 'immutabile', 'come chiave'), 3),
             ("un'altra chiave", 'adesso una stringa!'),
             (123, 'hello')])
```

Così come la `print`:

[39]:
```
print(od)
```

```
OrderedDict([('qualche chiave', 5), ('qualche altra chiave', 7), (('una', 'tupla',
↪'immutabile', 'come chiave'), 3), ("un'altra chiave", 'adesso una stringa!'), (123,
↪'hello')])
```

Vediamo come appare in Python Tutor:

[40]:
```
from collections import OrderedDict
od = OrderedDict()
od['qualche chiave'] = 5
od['qualche altra chiave'] = 7
od[('una', 'tupla', 'immutabile','come chiave')] = 3
od["un'altra chiave"] = 'adesso una stringa!'
od[123] = 'hello'

jupman.pytut()
```

[40]:
```
<IPython.core.display.HTML object>
```

Esercizio: agenda OrderedDict

Scrivi del codice che date tre tuple come le seguenti, STAMPA un `OrderedDict` che associa nomi a numeri di telefono, nell'ordine in cui sono proposti

- Il tuo codice deve funzionare con tuple *qualsiasi*
- Non dimenticarti di importare l'`OrderedDict` dalle `collections`

Esempio:

```
t1 = ('Alice', '143242903')
t2 = ('Bob', '417483437')
t3 = ('Carlo', '423413213')
```

dopo il tuo codice, dovrebbe risultare:

```
OrderedDict([('Alice', '143242903'), ('Bob', '417483437'), ('Charles', '423413213')])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[41]:
```
t1 = ('Alice', '143242903')
t2 = ('Bob', '417483437')
t3 = ('Charles', '423413213')

# scrivi qui
```

(continues on next page)

(continued from previous page)

```
# prima bisogna importare dalle collezioni
from collections import OrderedDict

od = OrderedDict([t1, t2, t3])
#print(od)
```

[41]:
```
t1 = ('Alice', '143242903')
t2 = ('Bob', '417483437')
t3 = ('Charles', '423413213')

# scrivi qui
```

4.20.10 Esercizio - copia di OrderedDict

Dato un `OrderedDict od1` contenente traduzioni Inglese -> Italiano, crea un NUOVO `OrderedDict` chiamato `od2` che contiene le stesse traduzioni come input PIU' la traduzione `'water' : 'acqua'`

- NOTA 1: il tuo codice dovrebbe funzionare con qualsiasi ordered dict di input
- NOTA 2: `od2` DEVE essere associata ad un NUOVO `OrderedDict` !!

Esempio - dato:

```
od1 = OrderedDict()
od1['dog'] = 'cane'
od1['home'] = 'casa'
od1['table'] = 'tavolo'
```

dopo il tuo codice, dovresti ottenere:

```
>>> print(od1)
OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo')])
>>> print(od2)
OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo'), ('water', 'acqua
↪')])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[42]:
```
from collections import OrderedDict

od1 = OrderedDict()
od1['dog'] = 'cane'
od1['home'] = 'casa'
od1['table'] = 'tavolo'

# scrivi qui
od2 = OrderedDict(od1)
od2['water'] = 'acqua'
```

(continues on next page)

(continued from previous page)

```
print("od1=", od1)
print("od2=", od2)
```

```
od1= OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo')])
od2= OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo'), ('water',
↪'acqua')])
```

[42]:
```
from collections import OrderedDict

od1 = OrderedDict()
od1['dog'] = 'cane'
od1['home'] = 'casa'
od1['table'] = 'tavolo'

# scrivi qui
```

```
od1= OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo')])
od2= OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo'), ('water',
↪'acqua')])
```

4.20.11 Classi - Counter

Se abbiamo bisogno di contare quanti elementi diversi si trovano in una sequenza (in altre parole, se dobbiamo calcolare un istogramma delle frequenze), la classe `Counter` ci può tornare molto utile. `Counter` è un tipo di dizionario speciale. Prima di tutto, dobbiamo dichiarare a Python la nostra intenzione di usarlo, recuperandolo dal modulo `collections`:

[43]:
```
from collections import Counter
```

Supponiamo di voler contare quanti elementi diversi ci sono nella lista

```
['c', 'a', 'n', 't', 'a', 'r', 'e', 'l', 'l', 'a', 'n', 'd', 'o']
```

Possiamo inizializzare `Counter` così:

[44]:
```
istogramma = Counter(['c', 'a', 'n', 't', 'a', 'r', 'e', 'l', 'l', 'a', 'n', 'd', 'o
↪'])
```

Se lo stampiamo, vediamo che per primi sono messi gli elementi più frequenti:

[45]:
```
print(istogramma)
```

```
Counter({'a': 3, 'n': 2, 'l': 2, 't': 1, 'e': 1, 'o': 1, 'd': 1, 'c': 1, 'r': 1})
```

ATTENZIONE: SE NON SI USA LA `print` JUPYTER STAMPERA' IN ORDINE ALFABETICO!

[46]:
```
istogramma    # occhio !
```

[46]:
```
Counter({'a': 3,
         'c': 1,
         'd': 1,
         'e': 1,
         'l': 2,
         'n': 2,
         'o': 1,
         'r': 1,
         't': 1})
```

Possiamo ottenere una lista con gli `n` più frequenti col metodo `most_common`, che ritorna una lista di tuple:

[47]:
```
istogramma.most_common(5)
```

[47]:
```
[('a', 3), ('n', 2), ('l', 2), ('t', 1), ('e', 1)]
```

`Counter` si può inizializzare con una sequenza qualsiasi, per esempio con tuple:

[48]:
```
ct = Counter((50,70,40,60,40,50,40,70,50,50,50,60,50,30,50,30,40,50,60,70))
print(ct)
```

```
Counter({50: 8, 40: 4, 60: 3, 70: 3, 30: 2})
```

o stringhe:

[49]:
```
cs = Counter('rabbrividirai')
```

[50]:
```
print(cs)
```

```
Counter({'i': 4, 'r': 3, 'b': 2, 'a': 2, 'v': 1, 'd': 1})
```

Vi sono altri metodi che si possono usare, ma per quelli facciamo riferimento alla documentazione di Python[191]

4.20.12 Esercizio - frequenti

Dato una stringa `s`, scrivi del codice che stampa

- il carattere più frequente
- quello meno frequente
- quante e quali frequenze diverse vi sono
- Il tuo codice deve funzionare con *qualsiasi* stringa `s`
- Ignora la possibilità che vi siano pari merito tra i più/meno frequenti
- ricordati di importare `Counter` da `collections`

Esempio - data:

```
s = 'rattristato'
```

il tuo codice deve stampare:

```
Tra i più frequenti troviamo ('t', 4)
Tra i meno frequenti troviamo ('i', 1)
Vi sono  3 frequenze diverse: {1, 2, 4}
```

[191] https://docs.python.org/3/library/collections.html#collections.Counter

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[51]:
```
s = 'rattristato'

# scrivi qui
from collections import Counter

c = Counter(s)

#print("Tra i più frequenti troviamo", c.most_common()[0])
#print("Tra i meno frequenti troviamo", c.most_common()[-1])
#print("Vi sono ", len(set(c.values())), "frequenze diverse:", set(c.values()))
```

[51]:
```
s = 'rattristato'

# scrivi qui
```

4.20.13 Prosegui

Prosegui con Dizionari 4[192]

[ ]:
```
```

4.21 Dizionari - iterazione e funzioni

4.21.1 Scarica zip esercizi

Naviga file online[193]

In questo foglio troviamo esercizi su iterazione in dizionari, e come usarli quando sono argomento di funzioni.

ATTENZIONE: Gli esercizi seguenti richiedono di conoscere:

Dizionari 1[194], Dizionari 2[195] e Dizionari 3[196]

Controllo di flusso[197]

Funzioni[198]

[192] https://it.softpython.org/dictionaries/dictionaries4-sol.html
[193] https://github.com/DavidLeoni/softpython-it/tree/master/dictionaries
[194] https://it.softpython.org/dictionaries/dictionaries1-sol.html
[195] https://it.softpython.org/dictionaries/dictionaries2-sol.html
[196] https://it.softpython.org/dictionaries/dictionaries3-sol.html
[197] https://it.softpython.org/control-flow/control-flow-sol.html
[198] https://it.softpython.org/functions/functions-sol.html

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
dictionaries
    dictionaries1.ipynb
    dictionaries1-sol.ipynb
    dictionaries2.ipynb
    dictionaries2-sol.ipynb
    dictionaries3.ipynb
    dictionaries3-sol.ipynb
    dictionaries4.ipynb
    dictionaries4-sol.ipynb
    dictionaries5.ipynb
    dictionaries5-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `dictionaries4.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.21.2 Esercizi con le funzioni

stampa_val

⊛ Scrivi la funzione `stampa_val(d,chiave)` che RITORNA il valore associato a `chiave`

```
>>> x = stampa_val({'a':5,'b':2}, 'a')
>>> x
5
>>> y = stampa_val({'a':5,'b':2}, 'b')
>>> y
2
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[1]:
```
# scrivi qui

def stampa_val(d, key):
```

(continues on next page)

(continued from previous page)

```
    return d[key]

#x = stampa_val({'a':5,'b':2}, 'a')
#x
```

[1]:
```
# scrivi qui
```

ha_chiave

Scrivi la funzione `ha_chiave(d,chiave)` che STAMPA `"trovate"` se `diz` contiene la chiave `chiave`, altrimenti STAMPA `"non trovato"`.

```
>>> ha_chiave({'a':5,'b':2}, 'a')
trovato
>>> ha_chiave({'a':5,'b':2}, 'z')
non trovato
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[2]:
```
# scrivi qui

def ha_chiave(d, chiave):
    if chiave in d:
        print("trovato")
    else:
        print("non trovato")


#ha_chiave({'a':5,'b':2}, 'a')
#ha_chiave({'a':5,'b':2}, 'b')
#ha_chiave({'a':5,'b':2}, 'z')
```

[2]:
```
# scrivi qui
```

dim

⊛ Scrivi la funzione `dim(d)` che RITORNA le associazioni chiave valore presenti nel dizionario

```
>>> x = dim({'a':5,'b':2,'c':9})
>>> x
3
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[3]:
```
# scrivi qui

def dim(d):
    return len(d)

#x = dim({'a':5,'b':2,'c':9})
#x
```

[3]:
```
# scrivi qui
```

mazzol

⊛ Dato un dizionario, scrivi una funzione `mazzol` che RITORNA una LISTA ORDINATA con tutte le chiavi, una alla volta.

NOTA: l'ordine delle chiavi in questa lista E' importante !

```
>>> x = mazzol({'a':5,'c':2,'b':9})
>>> x
['a','b','c']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[4]:
```
# scrivi qui

def mazzol(d):
    la = list(d.keys())
    la.sort()  # RICORDA: .sort() NON ritorna nulla !
    return la


#x = mazzol({'a':5,'c':2,'b':9})
#x
```

[4]:
```
# scrivi qui
```

coppie

⊛ Dato un dizionario, scrivi una funzione `coppie` che STAMPA tutte le coppie chiave/valore, una per riga

NOTA: l'ordine di stampa *NON* è importante, è sufficiente stampare tutte le coppie !

```
>>> coppie({'a':5,'b':2,'c':9})
a 5
c 9
b 2
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[5]:
```
# scrivi qui

def coppie(d):
    for chiave in d:
        print(chiave,d[chiave])

#coppie({'a':5,'b':2,'c':9})
```

[5]:
```
# scrivi qui
```

4.21.3 Verifica comprensione

ATTENZIONE

Gli esercizi che seguono contengono dei test con gli *assert*. Per capire come svolgerli, leggi prima Gestione errori e testing[199]

istogramma

⊛⊛ RITORNA un NUOVO dizionario. contenente una chiave per ogni carattere di stringa. A ciascuna chiave sarà associato come valore il numero di occorrenze del relativo carattere in stringa

Ingredienti:

- variabile dizionario da ritornare
- ciclo for (dobbiamo iterare su stringa)

[199] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html

- **NON** usare `Counter`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[6]:
```
def istogramma(stringa):


    diz = {}
    for carattere in stringa:
        if carattere in diz:
            diz[carattere] += 1
        else:
            diz[carattere] = 1
    return diz


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert istogramma("") == {}
assert istogramma("a") == {'a':1}
assert istogramma("aa") == {'a':2}
assert istogramma("aaa") == {'a':3}
assert istogramma("ba") == {'a':1,
                            'b':1}
assert istogramma("aba") == {'a':2,
                             'b':1}
assert istogramma("abc") == {'a':1,
                             'b':1,
                             'c':1}
assert istogramma("acccbb") == {'a':1,
                                'b':2,
                                'c':3}
```

[6]:
```
def istogramma(stringa):

    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert istogramma("") == {}
assert istogramma("a") == {'a':1}
assert istogramma("aa") == {'a':2}
assert istogramma("aaa") == {'a':3}
assert istogramma("ba") == {'a':1,
                            'b':1}
assert istogramma("aba") == {'a':2,
                             'b':1}
assert istogramma("abc") == {'a':1,
                             'b':1,
                             'c':1}
assert istogramma("acccbb") == {'a':1,
```

(continues on next page)

(continued from previous page)

```
                                    'b':2,
                                    'c':3}
```

listifica

⊛⊛ Prende un dizionario `d` come input e RITORNA una LISTA con soli i valori dal dizionario (quindi nessuna chiave)

Per avere un ordine prevedibile, la funzione prende anche come input una lista `ordine` dove vi sono chiavi dal primo dizionario ordinate come le vorremmo vedere nella lista risultante

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[7]:
```
def listifica(d, ordine):

    ret = list()
    for elemento in ordine:
        ret.append (d[elemento])
    return ret


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError
assert listifica({}, []) == []
assert listifica({'ciao':123}, ['ciao']) == [123]
assert listifica({'a':'x','b':'y'}, ['a','b']) == ['x','y']
assert listifica({'a':'x','b':'y'}, ['b','a']) == ['y','x']
assert listifica({'a':'x','b':'y','c':'x'}, ['c','a','b']) == ['x','x','y']
assert listifica({'a':'x','b':'y','c':'x'}, ['b','c','a']) == ['y','x','x']
assert listifica({'a':5,'b':2,'c':9}, ['b','c','a']) == [2,9,5]
assert listifica({6:'x',8:'y',3:'x'}, [6,3,8]) == ['x','x','y']
# FINE TEST
```

[7]:
```
def listifica(d, ordine):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, python non dovrebbe␣
↪lanciare AssertionError
assert listifica({}, []) == []
assert listifica({'ciao':123}, ['ciao']) == [123]
assert listifica({'a':'x','b':'y'}, ['a','b']) == ['x','y']
assert listifica({'a':'x','b':'y'}, ['b','a']) == ['y','x']
assert listifica({'a':'x','b':'y','c':'x'}, ['c','a','b']) == ['x','x','y']
assert listifica({'a':'x','b':'y','c':'x'}, ['b','c','a']) == ['y','x','x']
assert listifica({'a':5,'b':2,'c':9}, ['b','c','a']) == [2,9,5]
assert listifica({6:'x',8:'y',3:'x'}, [6,3,8]) == ['x','x','y']
# FINE TEST
```

tcont

⊛⊛ Prende una lista di tuple. Ogni tupla ha due valori, il primo è un oggetto immutabile e il secondo un numero intero (il conteggio dell'oggetto). RITORNA un dizionario che per ogni oggetto immutabile trovato nelle tuple, associa il conteggio totale trovato.

Per esempi vedere gli assert

[8]:
```
def tcont(lst):
    ret = {}
    for c in lst:
        if c[0] in ret:
            ret[c[0]] += c[1]
        else:
            ret[c[0]] = c[1]
    return ret

# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert tcont([]) == {}
assert tcont([('a',3)]) == {'a':3}
assert tcont([('a',3),('a',4)]) == {'a':7}
assert tcont([('a',3),('b',8), ('a',4)]) == {'a':7, 'b':8}
assert tcont([('a',5), ('c',8), ('b',7), ('a',2), ('a',1), ('c',4)]) == {'a':5+2+1, 'b
↪':7, 'c': 8 + 4}
# FINE TEST
```

⊛⊛ Scrivi una funzione `inter(d1,d2)` che prende due dizionari e RITORNA un SET di chiavi per le quali la coppia è la stessa in entrambi i dizionari

Esempio:

```
>>> a = {'chiave1': 1, 'chiave2': 2 , 'chiave3': 3}
>>> b = {'chiave1': 1 ,'chiave2': 3 , 'chiave3': 3}
>>> inter(a,b)
{'chiave1','chiave3'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[9]:
```
def inter(d1, d2):

    res = set()
    for chiave in d1:
        if chiave in d2:
            if d1[chiave] == d2[chiave]:
                res.add(chiave)
    return res



# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert inter({'key1': 1, 'key2': 2 , 'key3': 3}, {'key1':1 ,'key2':3 , 'key3':3}) == {
↪'key1', 'key3'}
```

(continues on next page)

(continued from previous page)

```
assert inter(dict(), {'key1':1 ,'key2':3 , 'key3':3}) == set()
assert inter({'key1':1 ,'key2':3 , 'key3':3}, dict()) == set()
assert inter(dict(),dict()) == set()
# FINE TEST
```

[9]:

```
def inter(d1, d2):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert inter({'key1': 1, 'key2': 2 , 'key3': 3}, {'key1':1 ,'key2':3 , 'key3':3}) == {
↪'key1', 'key3'}
assert inter(dict(), {'key1':1 ,'key2':3 , 'key3':3}) == set()
assert inter({'key1':1 ,'key2':3 , 'key3':3}, dict()) == set()
assert inter(dict(),dict()) == set()
# FINE TEST
```

valori_unici

⊛⊛ Scrivi una funzione `valori_unici(d)` che RITORNA una lista di valori unici dal dizionario. La lista DEVE essere ordinata alfanumericamente.

Domanda: ci serve ordinata per fini di test. Come mai?

- per ordinare la lista, usa il metodo `.sort()`

Esempio:

```
>>> valori_unici({'a':'y','b':'x','c':'x'})

['x','y']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[10]:

```
def valori_unici(d):

    s = set(d.values())
    ret = list(s)  # possiamo solo ordinare liste (gli insiemi non hanno ordine)
    ret.sort()
    return ret


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert valori_unici({}) == []
assert valori_unici({'a':'y','b':'x','c':'x'}) == ['x','y']
```

(continues on next page)

(continued from previous page)

```
assert valori_unici({'a':4,'b':6,'c':4,'d':8}) == [4,6,8]
# FINE TEST
```

[10]:
```
def valori_unici(d):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert valori_unici({}) == []
assert valori_unici({'a':'y','b':'x','c':'x'}) == ['x','y']
assert valori_unici({'a':4,'b':6,'c':4,'d':8}) == [4,6,8]
# FINE TEST
```

maiuscole

⊛⊛ RITORNA un dizionario che associa ad ogni stringa nella lista fornita la stessa stringa ma con tutte le lettere in maiuscolo

Esempio:

```
>>> `maiuscole(["ciao", "mondo", "come va?"])`
{"ciao":"CIAO", "mondo":"MONDO", "come va?":"COME VA?"}
```

Ingredienti:

```
- ciclo `for`
- metodo `.upper()`
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[11]:
```
def maiuscole(lista):

    diz = {}
    for stringa in lista:
        diz[stringa] = stringa.upper()
    return diz



# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert maiuscole([]) == {}
assert maiuscole(["ciao"]) == {"ciao":"CIAO"}
assert maiuscole(["ciao", "mondo"]) == {"ciao":"CIAO", "mondo":"MONDO"}
assert maiuscole(["ciao", "mondo", "ciao"]) == {"ciao":"CIAO", "mondo":"MONDO"}
```

(continues on next page)

(continued from previous page)

```
assert maiuscole(["ciao", "mondo", "come va?"]) == {"ciao":"CIAO", "mondo":"MONDO",
↪"come va?":"COME VA?"}
# FINE TEST
```

[11]:

```
def maiuscole(lista):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert maiuscole([]) == {}
assert maiuscole(["ciao"]) == {"ciao":"CIAO"}
assert maiuscole(["ciao", "mondo"]) == {"ciao":"CIAO", "mondo":"MONDO"}
assert maiuscole(["ciao", "mondo", "ciao"]) == {"ciao":"CIAO", "mondo":"MONDO"}
assert maiuscole(["ciao", "mondo", "come va?"]) == {"ciao":"CIAO", "mondo":"MONDO",
↪"come va?":"COME VA?"}
# FINE TEST
```

filtraz

⊛⊛ RITORNA un NUOVO dizionario, che contiene solo le chiavi/valori del dizionario diz in ingresso nella cui chiave è presente la lettera 'z'

Esempio:

```
filtraz({'zibibbo':'da bere',
         'mc donald': 'da evitare',
         'liquirizia': 'ze best',
         'burger king': 'zozzerie'
})
```

deve RITORNARE il NUOVO dizionario

```
{
'zibibbo':'da bere',
'liquirizia': 'ze best'
}
```

In altre parole, abbiamo solo tenuto quelle chiavi che contenevano almeno una `'z'`. Se nei valori ci sono `z` non ce ne curiamo.

Ingredienti:

Per vedere se 'z' è nella chiave, usare l'operatore `in` per es

```
'z' in 'zibibbo' == True
'z' in 'mc donald' == False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
def filtraz(diz):


    ret = {}
    for chiave in diz:
        if 'z' in chiave:
            ret[chiave] = diz[chiave]
    return ret




# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert filtraz({}) == {}
assert filtraz({'az':'t'}) == {'az':'t'}
assert filtraz({'zc':'w'}) == {'zc':'w'}
assert filtraz({'b':'h'}) == {}
assert filtraz({'b':'hz'}) == {}
assert filtraz({'az':'t','b':'hz'}) == {'az':'t'}
assert filtraz({'az':'t','b':'hz','zc':'w'}) == {'az':'t', 'zc':'w'}
# FINE TEST
```

[12]:
```
def filtraz(diz):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert filtraz({}) == {}
assert filtraz({'az':'t'}) == {'az':'t'}
assert filtraz({'zc':'w'}) == {'zc':'w'}
assert filtraz({'b':'h'}) == {}
assert filtraz({'b':'hz'}) == {}
assert filtraz({'az':'t','b':'hz'}) == {'az':'t'}
assert filtraz({'az':'t','b':'hz','zc':'w'}) == {'az':'t', 'zc':'w'}
# FINE TEST
```

powers

⊛⊛ RITORNA un dizionario in cui le chiavi sono numeri interi da 1 a `n` inclusi, e i rispettivi valori sono i quadrati delle chiavi

Esempio:

```
powers(3)
```

Ritorna

```
{
 1:1,
 2:4,
```

(continues on next page)

(continued from previous page)

```
 3:9
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[13]:
```
def powers(n):

    d=dict()
    for i in range(1,n+1):
        d[i]=i**2
    return d


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert powers(1) == {1:1}
assert powers(2) == {
                        1:1,
                        2:4
                    }
assert powers(3) == {
                        1:1,
                        2:4,
                        3:9
                    }

assert powers(4) == {
                        1:1,
                        2:4,
                        3:9,
                        4:16
                    }
# FINE TEST
```

[13]:
```
def powers(n):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert powers(1) == {1:1}
assert powers(2) == {
                        1:1,
                        2:4
                    }
assert powers(3) == {
                        1:1,
                        2:4,
```

(continues on next page)

(continued from previous page)

```
                        3:9
                    }

assert powers(4) == {
                        1:1,
                        2:4,
                        3:9,
                        4:16
                    }
# FINE TEST
```

dilist

⊛⊛ Restituisce un dizionario con `n` coppie chiave-valore, dove le chiavi sono numeri interi da `1` a `n` incluso, e ad ogni chiave `i` è associata una lista di numeri da `1` a `i`

NOTA: le chiavi sono *numeri interi*, NON stringhe !!!!!

Esempio:

```
dilist(3)
```

deve dare:

```
{
    1:[1],
    2:[1,2],
    3:[1,2,3]
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[14]:

```
def dilist(n):

    ret = dict()
    for i in range(1,n+1):
        lista = []
        for j in range(1,i+1):
            lista.append(j)
        ret[i] = lista
    return ret


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert dilist(0) == dict()
assert dilist(1) == {
                        1:[1]
                    }
```

(continues on next page)

(continued from previous page)

```
assert dilist(2) == {
                        1:[1],
                        2:[1,2]
                    }
assert dilist(3) == {
                        1:[1],
                        2:[1,2],
                        3:[1,2,3]
                    }
# FINE TEST
```

[14]:

```
def dilist(n):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert dilist(0) == dict()
assert dilist(1) == {
                        1:[1]
                    }
assert dilist(2) == {
                        1:[1],
                        2:[1,2]
                    }
assert dilist(3) == {
                        1:[1],
                        2:[1,2],
                        3:[1,2,3]
                    }
# FINE TEST
```

prefissi

⊛⊛ Scrivere una funzione `prefissi` che dati:

- un dizionario `diz` avente come chiavi province italiane e come valori i loro prefissi telefonici (nota: anche i prefissi sono stringhe!)
- una lista `province` con le province italiane

RITORNA una lista di prefissi corrispondenti alle province della lista passata

Esempio:

```
prefissi({
            'tn':'0461',
            'bz':'0471',
            'mi':'02',
            'to':'011',
            'bo':'051'
          },
          ['tn','to', 'mi'])
```

deve ritornare

```
['0461', '011', '02']
```

SUGGERIMENTI:

- inizializzare lista vuota da ritornare
- scorrere la lista di province e pescarsi i corrispondenti prefissi dal dizionario

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[15]:
```
def prefissi(diz, province):


    ret = []
    for provincia in province:
        ret.append(diz[provincia])

    return ret


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert prefissi({'tn':'0461'}, []) == []
assert prefissi({'tn':'0461'}, ['tn']) == ['0461']
assert prefissi({'tn':'0461', 'bz':'0471'}, ['tn']) == ['0461']
assert prefissi({'tn':'0461', 'bz':'0471'}, ['bz']) == ['0471']
assert prefissi({'tn':'0461', 'bz':'0471'}, ['tn','bz']) == ['0461', '0471']
assert prefissi({'tn':'0461', 'bz':'0471'}, ['bz','tn']) == ['0471', '0461']
assert prefissi({'tn':'0461',
                 'bz':'0471',
                 'mi':'02',
                 'to':'011',
                 'bo':'051'
                },
                ['tn','to', 'mi']) == ['0461', '011', '02']
# FINE TEST
```

[15]:
```
def prefissi(diz, province):

    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert prefissi({'tn':'0461'}, []) == []
assert prefissi({'tn':'0461'}, ['tn']) == ['0461']
assert prefissi({'tn':'0461', 'bz':'0471'}, ['tn']) == ['0461']
assert prefissi({'tn':'0461', 'bz':'0471'}, ['bz']) == ['0471']
assert prefissi({'tn':'0461', 'bz':'0471'}, ['tn','bz']) == ['0461', '0471']
assert prefissi({'tn':'0461', 'bz':'0471'}, ['bz','tn']) == ['0471', '0461']
assert prefissi({'tn':'0461',
                 'bz':'0471',
```

(continues on next page)

(continued from previous page)

```
                 'mi':'02',
                 'to':'011',
                 'bo':'051'
                },
                ['tn','to', 'mi']) == ['0461', '011', '02']
# FINE TEST
```

traduci

⊛⊛ L'italiano è oramai una lingua obsoleta, perciò va soppiantata dall'inglese che è più *trendy*. A tal fine, scrivere una funzione che prende una frase come lista di parole e un dizionario di traduzioni italiano-inglese, e RITORNA una NUOVA lista in cui tutte le parole della lista originale e che sono presenti come chiavi nel dizionario sono sostituite con la corrispondente voce in inglese.

Se non c'è traduzione, viene messa la parola in italiano.

Esempio:

```
lista = ["Oggi", "ho", "una riunione", "dove", "discuteremo", "gli obiettivi", "da",
↪"raggiungere", "per", "soddisfare", "i nostri clienti" ]

inglesismi={
    "una riunione": "un meeting",
    "gli obiettivi" : "i goal",
    "i nostri clienti" : "il nostro target"
}


traduci(lista, inglesismi)
```

deve ritornare

```
["Oggi", "ho", "un meeting", "dove", "discuteremo", "i goal", "da", "raggiungere",
↪"per", "soddisfare", "il nostro target"]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[16]:
```
def traduci(lista, inglesismi):

    ret = []
    for parola in lista:
        if parola in inglesismi:
            ret.append(inglesismi[parola])
        else:
            ret.append(parola)
    return ret



# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert traduci([], {}) == []
```

(continues on next page)

(continued from previous page)

```
assert traduci(['a'], {}) == ['a']
assert traduci(['a'], {'a':'x'}) == ['x']
assert traduci(['a b'], {'a b':'x'}) == ['x']
assert traduci(['a','b'], {'a':'x'}) == ['x','b']
assert traduci(['a','b'], {'a':'x','b':'y'}) == ['x','y']
assert traduci(["Oggi", "ho", "una riunione", "dove", "discuteremo",
"gli obiettivi", "da", "raggiungere", "per", "soddisfare", "i nostri clienti"],
 {
     "una riunione": "un meeting",
     "gli obiettivi" : "i goal",
     "i nostri clienti" : "il nostro target"
}) == ["Oggi", "ho", "un meeting", "dove", "discuteremo", "i goal", "da",
 "raggiungere", "per", "soddisfare", "il nostro target"]
# FINE TEST
```

[16]:
```
def traduci(lista, inglesismi):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST: NON TOCCARE QUESTA PARTE !
# se hai scritto tutto il codice giusto, ed esegui la cella, Python non dovrebbe␣
↪lanciare AssertionError
assert traduci([], {}) == []
assert traduci(['a'], {}) == ['a']
assert traduci(['a'], {'a':'x'}) == ['x']
assert traduci(['a b'], {'a b':'x'}) == ['x']
assert traduci(['a','b'], {'a':'x'}) == ['x','b']
assert traduci(['a','b'], {'a':'x','b':'y'}) == ['x','y']
assert traduci(["Oggi", "ho", "una riunione", "dove", "discuteremo",
"gli obiettivi", "da", "raggiungere", "per", "soddisfare", "i nostri clienti"],
 {
     "una riunione": "un meeting",
     "gli obiettivi" : "i goal",
     "i nostri clienti" : "il nostro target"
}) == ["Oggi", "ho", "un meeting", "dove", "discuteremo", "i goal", "da",
 "raggiungere", "per", "soddisfare", "il nostro target"]
# FINE TEST
```

4.21.4 Prosegui

Prosegui con Dizionari 5[200]

[ ]:
```

```

[200] https://it.softpython.org/dictionaries/dictionaries5-sol.html

4.22 Dizionari - strutture composte

4.22.1 Scarica zip esercizi

Naviga file online[201]

In questo foglio vedremo come gestire una struttura dati più complesse come una liste di dizionari e dizionari di liste, esaminando al contempo il significato di copia in superficie e copia in profondità.

ATTENZIONE: Gli esercizi seguenti richiedono di conoscere:

Dizionari 1[202], Dizionari 2[203], Dizionari 3[204] e Dizionari 4[205]

Controllo di flusso[206]

Funzioni[207]

(possibilmente) Matrici - Liste di liste[208]

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
dictionaries
    dictionaries1.ipynb
    dictionaries1-sol.ipynb
    dictionaries2.ipynb
    dictionaries2-sol.ipynb
    dictionaries3.ipynb
    dictionaries3-sol.ipynb
    dictionaries4.ipynb
    dictionaries4-sol.ipynb
    dictionaries5.ipynb
    dictionaries5-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `dictionaries4.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

[201] https://github.com/DavidLeoni/softpython-it/tree/master/dictionaries
[202] https://it.softpython.org/dictionaries/dictionaries1-sol.html
[203] https://it.softpython.org/dictionaries/dictionaries2-sol.html
[204] https://it.softpython.org/dictionaries/dictionaries3-sol.html
[205] https://it.softpython.org/dictionaries/dictionaries3-sol.html
[206] https://it.softpython.org/control-flow/control-flow-sol.html
[207] https://it.softpython.org/functions/functions-sol.html
[208] https://it.softpython.org/matrices-lists/matrices-lists-sol.html

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.22.2 Impiegati

ATTENZIONE

Gli esercizi che seguono contengono dei test con gli *assert*. Per capire come svolgerli, leggi prima Gestione errori e testing[209]

Supponiamo di avere una lista di dizionari che rappresenta un database di impiegati. Ogni impiegato è rappresentato da un dizionario:

```
{
    "nome":"Mario",
    "cognome": "Rossi",
    "età": 34,
    "azienda": {
                    "nome": "Aringhe Candite Spa",
                    "settore":"Alimentari"
               }
}
```

Il dizionario ha diversi attributi semplici quali `nome`, `cognome`, `età`. L'attributo `azienda` è più complesso, perchè è rappresentato come un altro dizionario: `

```
"azienda": {
        "nome": "Aringhe Candite Spa",
        "settore":"Alimentari"
    }
```

[1]:
```
db_impiegati = [
  {
    "nome":"Mario",
    "cognome": "Rossi",
    "età": 34,
    "azienda": {
                   "nome": "Aringhe Candite Spa",
                   "settore":"Alimentari"
               }
  },
  {
    "nome":"Pippo",
    "cognome": "Rossi",
    "età": 20,
    "azienda": {
                    "nome": "Batworks",
                    "settore":"Abbigliamento"
```

(continues on next page)

[209] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
                }
  },
  {
    "nome":"Paolo",
    "cognome": "Bianchi",
    "età": 25,
    "azienda": {
                    "nome": "Aringhe Candite Spa",
                    "settore":"Alimentari"
                }
  }

]
```

estrai_impiegati

⊛⊛ RITORNA i nomi degli impiegati in una lista

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[2]:
```
def estrai_impiegati(lista):

    ret = []
    for diz in lista:
        ret.append(diz["nome"])
    return ret


assert estrai_impiegati([]) == []

# se non trova db_impiegati, ricordati di eseguire la cella sopra che lo definisce
assert estrai_impiegati(db_impiegati) == ['Mario', 'Pippo', 'Paolo']
```

[2]:
```
def estrai_impiegati(lista):
    raise Exception('TODO IMPLEMENT ME !')

assert estrai_impiegati([]) == []

# se non trova db_impiegati, ricordati di eseguire la cella sopra che lo definisce
assert estrai_impiegati(db_impiegati) == ['Mario', 'Pippo', 'Paolo']
```

estrai_aziende

⊛⊛ RITORNA i nomi di azienda in una lista (i duplicati sono ammessi)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[3]:
```
def estrai_aziende(lista):
    """RITORNA i nomi di azienda in una lista (i duplicati sono ammessi)
    """

    ret = []
    for diz in lista:
        ret.append(diz["azienda"]["nome"])

    return ret



assert estrai_aziende([]) == []
# se non trova db_impiegati, ricordati di eseguire la cella sopra che lo definisce
assert estrai_aziende(db_impiegati) == ["Aringhe Candite Spa","Batworks","Aringhe␣
↪Candite Spa"]
```

[3]:
```
def estrai_aziende(lista):
    """RITORNA i nomi di azienda in una lista (i duplicati sono ammessi)
    """
    raise Exception('TODO IMPLEMENT ME !')


assert estrai_aziende([]) == []
# se non trova db_impiegati, ricordati di eseguire la cella sopra che lo definisce
assert estrai_aziende(db_impiegati) == ["Aringhe Candite Spa","Batworks","Aringhe␣
↪Candite Spa"]
```

età_media

⊛⊛ RITORNA l'età media degli impiegati di azienda

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[4]:
```

def età_media(lista):

    somma = 0
    for diz in lista:
        somma += diz["età"]

    return somma / len(lista)
```

(continues on next page)

(continued from previous page)

```
# visto che la funzione ritorna float non si può comparare per numeri esatti ma per␣
↪numeri vicini
# con funzione math.isclose
import math
assert math.isclose(età_media(db_impiegati), (34 + 20 + 25) / 3)
```

[4]:
```
def età_media(lista):
    raise Exception('TODO IMPLEMENT ME !')


# visto che la funzione ritorna float non si può comparare per numeri esatti ma per␣
↪numeri vicini
# con funzione math.isclose
import math
assert math.isclose(età_media(db_impiegati), (34 + 20 + 25) / 3)
```

tipi_aziende

⊛⊛ RITORNA i tipi (settori) di azienda in una lista, SENZA duplicati !!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[5]:
```
def tipi_aziende(lista):

    ret = []
    for diz in lista:
        settore = diz["azienda"]["settore"]
        if settore not in ret:
            ret.append(settore)

    return ret


assert tipi_aziende([]) == []
assert tipi_aziende(db_impiegati) == ["Alimentari", "Abbigliamento"]
```

[5]:
```
def tipi_aziende(lista):
    raise Exception('TODO IMPLEMENT ME !')

assert tipi_aziende([]) == []
assert tipi_aziende(db_impiegati) == ["Alimentari", "Abbigliamento"]
```

4.22.3 Altri esercizi

medie

⊛⊛ Dato un dizionario strutturato ad albero riguardante i voti di uno studente in classe V e VI, RESTITUIRE un array contente la media di ogni materia

Esempio:

```
medie([
  {'id' : 1, 'subject' : 'math', 'V' : 70, 'VI' : 82},
  {'id' : 1, 'subject' : 'italian', 'V' : 73, 'VI' : 74},
  {'id' : 1, 'subject' : 'german', 'V' : 75, 'VI' : 86}
])
```

ritorna

```
[ (70+82)/2 , (73+74)/2, (75+86)/2 ]
```

ovvero

```
[ 76.0 , 73.5, 80.5 ]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[10]:
```
def medie(lista):

    ret = [0.0, 0.0, 0.0]

    for i in range(len(lista)):
        ret[i] = (lista[i]['V'] + lista[i]['VI']) / 2

    return ret


# INIZIO TEST - NON TOCCARE !
import math


'''
Verifica che i numeri float in lista1 siano simili a quelli di lista2
'''
def is_list_close(lista1, lista2):
    if len(lista1) != len(lista2):
        return False

    for i in range(len(lista1)):
        if not math.isclose(lista1[i], lista2[i]):
            return False

    return True

assert is_list_close(medie([
                            {'id' : 1, 'subject' : 'math', 'V' : 70, 'VI' : 82},
                            {'id' : 1, 'subject' : 'italian', 'V' : 73, 'VI' : 74},
```

(continues on next page)

(continued from previous page)

```
                            {'id' : 1, 'subject' : 'german', 'V' : 75, 'VI' : 86}
                        ]),
                    [ 76.0 , 73.5, 80.5 ])
# FINE TEST
```

[10]:
```
def medie(lista):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
import math


'''
Verifica che i numeri float in lista1 siano simili a quelli di lista2
'''
def is_list_close(lista1, lista2):
    if len(lista1) != len(lista2):
        return False

    for i in range(len(lista1)):
        if not math.isclose(lista1[i], lista2[i]):
            return False

    return True

assert is_list_close(medie([
                            {'id' : 1, 'subject' : 'math', 'V' : 70, 'VI' : 82},
                            {'id' : 1, 'subject' : 'italian', 'V' : 73, 'VI' : 74},
                            {'id' : 1, 'subject' : 'german', 'V' : 75, 'VI' : 86}
                        ]),
                    [ 76.0 , 73.5, 80.5 ])
# FINE TEST
```

ha_pref

⊛⊛ Uno grande magazzino ha un database dei clienti modellato come un dizionario che associa ai nomi dei clienti le loro preferenze riguardo le categorie di articoli che comprano di solito:

```
{
    'aldo':['cinema', 'musica', 'sport'],
    'giovanni':['musica'],
    'giacomo':['cinema', 'videogiochi']
}
```

Dato il dizionario, il nome di un cliente e una categoria, scrivere una funzione `ha_pref` che RITORNA `True` se quel cliente ha la preferenza indicata, `False` altrimenti.

Esempio:

```
ha_pref({
            'aldo':['cinema', 'musica', 'sport'],
            'giovanni':['musica'],
            'giacomo':['cinema', 'videogiochi']
```

(continues on next page)

(continued from previous page)

```
        }, 'aldo', 'musica')
```

deve ritornare `True` perchè ad `aldo` piace la musica, invece

```
ha_pref({'aldo':['cinema', 'musica', 'sport'],
         'giovanni':['musica'],
         'giacomo':['cinema', 'videogiochi']

        }, 'giacomo', 'sport')
```

Deve ritornare `False` perchè a giacomo non piace lo sport

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[7]:
```
def ha_pref(diz, nome, pref):

    if nome in diz:
        return pref in diz[nome]
    else:
        return False


assert ha_pref({}, 'a', 'x') == False
assert ha_pref({'a':[]}, 'a',  'x') == False
assert ha_pref({'a':['x']}, 'a',  'x') == True
assert ha_pref({'a':['x']}, 'b',  'x') == False
assert ha_pref({'a':['x','y']}, 'a',  'y') == True
assert ha_pref({'a':['x','y'],
                'b':['y','x','z']}, 'b',  'y') == True
assert ha_pref({'aldo':['cinema', 'musica', 'sport'],
               'giovanni':['musica'],
               'giacomo':['cinema', 'videogiochi']
              }, 'aldo', 'musica') == True
assert ha_pref({'aldo':['cinema', 'musica', 'sport'],
               'giovanni':['musica'],
               'giacomo':['cinema', 'videogiochi']
              }, 'giacomo', 'sport') == False
```

[7]:
```
def ha_pref(diz, nome, pref):
    raise Exception('TODO IMPLEMENT ME !')

assert ha_pref({}, 'a', 'x') == False
assert ha_pref({'a':[]}, 'a',  'x') == False
assert ha_pref({'a':['x']}, 'a',  'x') == True
assert ha_pref({'a':['x']}, 'b',  'x') == False
assert ha_pref({'a':['x','y']}, 'a',  'y') == True
assert ha_pref({'a':['x','y'],
                'b':['y','x','z']}, 'b',  'y') == True
assert ha_pref({'aldo':['cinema', 'musica', 'sport'],
```

(continues on next page)

(continued from previous page)

```
                'giovanni':['musica'],
                'giacomo':['cinema', 'videogiochi']
               }, 'aldo', 'musica') == True
assert ha_pref({'aldo':['cinema', 'musica', 'sport'],
                'giovanni':['musica'],
                'giacomo':['cinema', 'videogiochi']
               }, 'giacomo', 'sport') == False
```

onomat

⊛⊛ Proviamo ad aggiungere delle espressioni onomatopeiche a delle frasi

INPUT: - frase da arricchire - Il sentimento da usare, che è codificato come un valore numerico. - un dizionario di sentimenti, in cui si associa al codice numerico di ogni sentimento un dizionario contenente un espressione onomatopeica tipica per quel sentimento, e la posizione in cui deve figurare all'interno di una frase. Le posizioni sono indicate come 'i' per inizio e 'f' per fine.

OUTPUT

- La frase arricchita con l'espressione onomatopeica scelta in base al sentimento. L'espressione va aggiunta sempre prima o dopo la frase, e sempre separata da uno spazio.

Per esempio

```
sentimenti = {
                1:  {
                        "espressione": "Gulp!",
                        "posizione": "i"
                    }
                2:  {
                        "espressione": "Sgaragulp !",
                        "posizione": "i"
                    }
                3:  {
                        "espressione": "Uff..",
                        "posizione": "f"
                    }
}


onomat("Ma quelli sono i bassotti!", 1, sentimenti)
```

Deve tornare

```
"Gulp! Ma quelli sono i bassotti!"
```

Mentre

```
onomat("Non voglio alzarmi dall'amaca.", 3, sentimenti)
```

Deve tornare

```
"Non voglio alzarmi dall'amaca. Uff.."
```

NOTA: Ricordarsi lo spazio tra espressione e frase!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[8]:
```
def onomat(frase, sentimento, sentimenti):

    sent = sentimenti[sentimento]
    if sent["posizione"] == "i":
        return sent["espressione"] + " " + frase
    else:
        return frase + " " + sent["espressione"]



# INIZIO TEST - NON TOCCARE !!!


sentimenti = {
                1:  {
                        "espressione": "Gulp!",
                        "posizione": "i"
                    },
                2:  {
                        "espressione": "Sgaragulp!",
                        "posizione": "i"
                    },
                3:  {
                        "espressione": "Uff..",
                        "posizione": "f"
                    },
                4:  {
                        "espressione": "Yuk yuk!",
                        "posizione": "f"
                    },
                5:  {
                        "espressione": "Sgrunt!",
                        "posizione": "i"
                    },
                6:  {
                        "espressione": "Gasp!",
                        "posizione" : "i"
                    }
            }


assert onomat("Mi chiamo Pippo.", 4, sentimenti) == "Mi chiamo Pippo. Yuk yuk!"
assert onomat("Quel topastro mi ha rovinato un'altra rapina!", 5, sentimenti) ==
↪"Sgrunt! Quel topastro mi ha rovinato un'altra rapina!"
assert onomat("Non voglio alzarmi dall'amaca.", 3, sentimenti) == "Non voglio alzarmi␣
↪dall'amaca. Uff.."

# FINE TEST
```

[8]:
```
def onomat(frase, sentimento, sentimenti):
    raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
# INIZIO TEST - NON TOCCARE !!!


sentimenti = {
                1:  {
                        "espressione": "Gulp!",
                        "posizione": "i"
                    },
                2:  {
                        "espressione": "Sgaragulp!",
                        "posizione": "i"
                    },
                3:  {
                        "espressione": "Uff..",
                        "posizione": "f"
                    },
                4:  {
                        "espressione": "Yuk yuk!",
                        "posizione": "f"
                    },
                5:  {
                        "espressione": "Sgrunt!",
                        "posizione": "i"
                    },
                6:  {
                        "espressione": "Gasp!",
                        "posizione" : "i"
                    }
            }


assert onomat("Mi chiamo Pippo.", 4, sentimenti) == "Mi chiamo Pippo. Yuk yuk!"
assert onomat("Quel topastro mi ha rovinato un'altra rapina!", 5, sentimenti) ==
↪"Sgrunt! Quel topastro mi ha rovinato un'altra rapina!"
assert onomat("Non voglio alzarmi dall'amaca.", 3, sentimenti) == "Non voglio alzarmi␣
↪dall'amaca. Uff.."

# FINE TEST
```

```
[ ]:
```

4.23 Condizionali - if else

4.23.1 Scarica zip esercizi

Naviga file online[210]

Possiamo usare il comando condizionale `if` ogni volta che il computer deve prendere una decisione in base al valore di qualche condizione. Se la condizione viene valutata come vera (cioè come il booleano `True`) allora verrà eseguito un blocco di codice, altrimenti ne verrà eseguito un altro.

Riferimenti:

[210] https://github.com/DavidLeoni/softpython-it/tree/master/control-flow

- SoftPython - Basi - booleani[211]

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
control-flow
    flow1-if.ipynb
    flow1-if-sol.ipynb
    flow2-for.ipynb
    flow2-for-sol.ipynb
    flow3-while.ipynb
    flow3-while-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `flow1-if.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.23.2 Il comando base `if else`

Vediamo un programmino che prende decisioni diverse in base al valore di una variabile `caramelle`:

[2]:
```
caramelle = 20

if caramelle > 10 :
    print('Abbiamo trovato...')
    print('Tante caramelle!')
else:
    print("Purtroppo ci sono.. ")
    print('Poche caramelle!')

print()
print('Cerchiamo altri dolci!')
```

```
Abbiamo trovato…
Tante caramelle!

Cerchiamo altri dolci!
```

[211] https://it.softpython.org/basics/basics-sol.html#Booleani

La condizione qui è `caramelle > 10`

[3]:
```
caramelle > 10
```

[3]:
```
True
```

ATTENZIONE: Subito dopo la condizione vanno messi due punti :

```
if caramelle > 10:
```

Dato che nell'esempio sopra `caramelle` vale `20`, la condizione è valutata a `True` e viene quindi eseguito il blocco di codice subito dopo la riga con l'`if`.

Proviamo invece a mettere un numero piccolo, come `caramelle = 5`:

[4]:
```
caramelle = 5

if caramelle > 10 :
    print('Abbiamo trovato...')
    print('Tante caramelle!')
else:
    print("Purtroppo ci sono.. ")
    print('Poche caramelle!')

print()
print('Cerchiamo altri dolci!')
```

```
Purtroppo ci sono..
Poche caramelle!

Cerchiamo altri dolci!
```

In questo caso è stato eseguito il blocco di codice dopo la riga con `else`:

ATTENZIONE all'indentazione nei blocchi !

Come tutti i blocchi di codice in Python, sono preceduti da spazi. Di solito si usano 4 spazi (in qualche progetto Python puoi trovarne solo 2, ma è sconsigliato dalle linee guida di Python)

4.23.3 `else` è opzionale

Non è obbligatorio indicare l'`else`. Se lo omettiamo e la condizione risulta `False`, il controllo passa direttamente ai comandi allo stesso livello di indentazione dell'`if` (senza errori):

[5]:
```
caramelle = 5

if caramelle > 10 :
    print('Abbiamo trovato...')
    print('Tante caramelle!')

print()
print('Cerchiamo altri dolci!')
```

```

Cerchiamo altri dolci!
```

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
x = 3
if x > 2  and if x < 4:
    print('ABBA')
```

2.
```
x = 3
if x > 2 and x < 4
    print('ABBA')
```

3.
```
x = 3
if x > 2 and x < 4:
    print('ABBA')
```

4.
```
x = 2
if x > 1:
    print(x+1, x):
```

5.
```
x = 3
if x > 5 or x:
    print('ACDC')
```

6.
```
x = 7
if x == 7:
print('GLAM')
```

7.
```
x = 7
if x < 1:
    print('BIM')
else:
    print('BUM')
print('BAM')
```

8.
```
x = 30
if x > 8:
    print('DOH')
if x > 10:
    print('DUFF')
if x > 20:
    print('BURP')
```

9.
```
if not True:
    print('sottosopra')
else:
    print('soprasotto')
```

10.
```
if False:
else:
    print('ZORB')
```

11.
```
if False:
    pass
else:
    print('ZORB')
```

12.
```
if 0:
    print('Brandy')
else:
    print('Rum')
```

13.
```
if False:
    print('illustrissimo')
else:
    print('esimio')
else:
    print('dottore')
```

14.
```
if 2 != 2:
    'ATTENTO'
else:
    'STOLTO'
```

15.
```
if 2 != 2:
    print('ATTENTO')
else:
    print('STOLTO')
```

16.
```
x = [1,2,3]
if 4 in x:
    x.append(4)
else:
    x.remove(3)
print(x)
```

17.
```
if 'False':
    print('OCCHIO ALLA STRINGA')
else:
    print('CRUDELE')
```

4.23.4 Esercizio - senza benza

Si vuole fare un viaggio in auto e servono almeno `30` litri. Scrivi del codice che:

- se la variabile `benzina` è inferiore a `30`, stampa `'Senza benza, faccio il pieno'` e incrementa `benzina` di `20` litri
- Altrimenti, stampa `Ho abbastanza carburante!`
- In ogni caso, alla fine stampa `'Si parte con '` seguito dall dalla quantità di benzina finale.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[6]:
```
benzina = 5
#benzina = 30

# scrivi qui

if benzina < 30:
```

(continues on next page)

(continued from previous page)

```
    print('Senza benza, faccio il pieno')
    benzina += 20
else:
    print('Ho abbastanza carburante')

print('Si parte con', benzina, 'litri')
```

```
Senza benza, faccio il pieno
Si parte con 25 litri
```

[6]:
```
benzina = 5
#benzina = 30

# scrivi qui
```

```
Senza benza, faccio il pieno
Si parte con 25 litri
```

4.23.5 Il comando *if - elif - else*

Esaminando il programmino delle caramelle di prima ti sarai forse chiesto cosa stampare quando non ci sono proprio caramelle. Per farlo, potremmo usare più *condizioni in cascata* con il comando `elif` (abbreviazione di *else if*):

[7]:
```
caramelle = 0  # METTIAMO ZERO

if caramelle > 10:
    print('Abbiamo trovato...')
    print('Tante caramelle!')
elif caramelle > 0:
    print("Purtroppo ci sono.. ")
    print('Poche caramelle!')
else:
    print("Che sfortuna! ")
    print('Non ci sono caramelle!')

print()
print('Cerchiamo altri dolci!')
```

```
Che sfortuna!
Non ci sono caramelle!

Cerchiamo altri dolci!
```

ESERCIZIO: Prova a cambiare i valori di `caramelle` nella cella qua sopra e guarda cosa succede

Il programmino si comporta esattamente come i precedenti e quando nessuna condizione è soddisfatta viene eseguito l'ultimo blocco di codice dopo l'`else:`

Possiamo aggiungere quanti `elif` vogliamo, quindi potremmo essere ancora più specifici e mettere un `elif x == 0:` e nell'`else` gestire tutti gli altri casi, anche quelli imprevisti o assurdi come per esempio un numero di caramelle negativo. Perchè mai dovremmo farlo? Gli imprevisti possono sempre succedere, ne avrai trovati in giro di programmi *buggati* … (vedremo meglio come gestire queste situazioni nel tutorial Gestione errori e testing[212])

[212] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html

[8]:
```
caramelle = -2   # PROVIAMO UN NUMERO NEGATIVO

if caramelle > 10:
    print('Abbiamo trovato...')
    print('Tante caramelle!')
elif caramelle > 0:
    print("Purtroppo ci sono.. ")
    print('Poche caramelle!')
elif caramelle == 0:
    print("Che sfortuna! ")
    print('Non ci sono caramelle!')
else:
    print('Qualcosa è andato MOLTO storto! Abbiamo trovato', caramelle, 'caramelle')

print()
print('Cerchiamo altri dolci!')
```

```
Qualcosa è andato MOLTO storto! Abbiamo trovato -2 caramelle

Cerchiamo altri dolci!
```

ESERCIZIO: Prova a cambiare i valori di `caramelle` nella cella qua sopra e guarda cosa succede

Domande

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
y = 2
if y < 3:
    print('bingo')
elif y <= 2:
    print('bango')
```

2.
```
z = 'q'
if not 'quando'.startswith(z):
    print('BAR')
elif not 'spqr'[2] == z:
    print('WAR')
else:
    print('ZAR')
```

3.
```
x = 1
if x < 5:
    print('BARCHE')
    elif x < 3:
        print('ZATTERE')
        else:
            print('SCIALUPPE')
```

4.
```
x = 5
if x < 3:
    print('ORO')
else if x >= 3:
    print('ARGENTO')
```

5.
```
if 0:
    print(0)
elif 1:
    print(1)
```

4.23.6 Domande - Sono equivalenti?

Guarda i seguenti frammenti di codice: in ciascuno, vi sono due parti, A e B. In ciascun frammento, cerca di indovinare se la parte A stamperà esattamente quello che stampa il codice nella parte B, per ogni valore delle variabili da cui dipendono.

- **PRIMA** pensa alla risposta
- **POI** prova ad eseguire con ciascuno dei valori delle variabili suggerite

Sono equivalenti? - fragole

Prova a cambiare il valore di `fragole` togliendo i commenti

[9]:
```
fragole = 5
#fragole = 2
#fragole = 10

print('fragole =', fragole)
print('A:')
if fragole > 5:
    print("Le fragole sono > 5")
elif fragole > 5:
    print("Ho detto che le fragole sono > 5!")
else:
    print("Le fragole sono <= 5")

print('B:')
if fragole > 5:
    print("Le fragole sono > 5")
if fragole > 5:
    print("Ho detto che le fragole sono > 5!")
if fragole <= 5:
    print("Le fragole sono <= 5")
```

```
fragole = 5
A:
Le fragole sono <= 5
B:
Le fragole sono <= 5
```

Sono equivalenti? - max

```
x, y = 3, 5
#x, y = 5, 3
#x, y = 3, 3

print('x =', x)
print('y =', y)

print('A:')
if x > y:
    print(x)
else:
    print(y)

print('B:')
print(max(x,y))
```

Sono equivalenti? - min

```
x, y = 3, 5
#x, y = 5, 3
#x, y = 3, 3

print('x =',x)
print('y =', y)

print('A:')
if x < y:
    print(y)
else:
    print(x)

print('B:')
print(min(x,y))
```

Sono equivalenti? - grande piccolo

```
x = 2
#x = 4
#x = 3

print('x =',x)

print('A:')
if x > 3:
    print('grande')
else:
    print('piccolo')

print('B:')
if x < 3:
    print('piccolo')
```

(continues on next page)

(continued from previous page)

```
else:
    print('grande')
```

Sono equivalenti? - Cippirillo

```
x = 3
#x = 10
#x = 11
#x = 15

print('x =', x)

print('A:')
if x % 5 == 0:
    print('cippirillo')
if x % 3 == 0:
    print('cippirillo')

print('B:')
if x % 3 == 0 or x % 5 == 0:
    print('cippirillo')
```

Esercizio - fattoria

Data una stringa `s`, scrivi del codice che stampa `'BAU!'` se la stringa inizia con `cane` , stampa `'CHICCHIRICHI!'` se le stringa inizia con `'gallo'`, e stampa `'???'` in tutti gli altri casi

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[10]:
```
s = 'cane dalmata'
#s = 'gallo cedrone'
#s = 'sghirimpo'

print(s)

# scrivi qui

if s.startswith('cane'):
    print('BAU')
elif s.startswith('gallo'):
    print('CHICCHIRICHI!')
else:
    print('???')
```

```
cane dalmata
BAU
```

[10]:
```
s = 'cane dalmata'
#s = 'gallo cedrone'
```

(continues on next page)

(continued from previous page)

```
#s = 'sghirimpo'

print(s)

# scrivi qui
```

```
cane dalmata
BAU
```

Esercizio - accenti

Scrivi del codice che stampa se una `parola` termina o meno con una lettera accentata.

- Per determinare se una lettera è accentata, usa le stringhe di accenti `acuti` e `gravi`
- il tuo codice deve funzionare con qualunque `parola`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[11]:
```
acuti = "áéíóú"
gravi = "àèìòù"

parola = 'urrà'          # termina con una accentata
#parola = 'martello'     # non termina con una accentata
#parola = 'ahó'          # termina con una accentata
#parola = 'però'         # termina con una accentata
#parola = 'capitaneria' # non termina con una accentata
#parola = 'viceré'       # termina con una accentata
#parola = 'cioè'         # termina con una accentata
#parola = 'chéto'        # non termina con una accentata
#parola = 'Chi dice che la verità è una sòla?'  # non termina con una accentata

# scrivi qui

if parola[-1] in acuti or parola[-1] in gravi:
    print(parola,'termina con una accentata!')
else:
    print(parola, 'non termina con una accentata')
```

```
urrà termina con una accentata!
```

[11]:
```
acuti = "áéíóú"
gravi = "àèìòù"

parola = 'urrà'          # termina con una accentata
#parola = 'martello'     # non termina con una accentata
#parola = 'ahó'          # termina con una accentata
#parola = 'però'         # termina con una accentata
#parola = 'capitaneria' # non termina con una accentata
#parola = 'viceré'       # termina con una accentata
#parola = 'cioè'         # termina con una accentata
```

(continues on next page)

(continued from previous page)

```
#parola = 'chéto'       # non termina con una accentata
#parola = 'Chi dice che la verità è una sòla?'  # non termina con una accentata

# scrivi qui
```

```
urrà termina con una accentata!
```

Esercizio - arcani

Dato un arcano `x` espresso come una stringa e delle liste di arcani `maggiori` e `minori`, stampare a quale categoria appartiene `x`. Se `x` non appartiene a nessuna categoria, stampare `è un mistero`.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
x = 'La Ruota'     # maggiore
#x = "L'Appeso"       # maggiore
#x = 'Asso di Spade'  # minore
#x = 'Due di Denari'  # minore
#x = 'Il Coding'      # mistero

maggiori = ['La Ruota','Il Carro', "L'Appeso"]
minori = ['Asso di Spade', 'Due di Denari', 'Regina di Coppe']

# scrivi qui
if x in maggiori:
    print(x, 'è un Arcano maggiore')
elif x in minori:
    print(x, 'è un Arcano minore')
else:
    print(x, 'è un mistero')
```

```
La Ruota è un Arcano maggiore
```

[12]:
```
x = 'La Ruota'     # maggiore
#x = "L'Appeso"       # maggiore
#x = 'Asso di Spade'  # minore
#x = 'Due di Denari'  # minore
#x = 'Il Coding'      # mistero

maggiori = ['La Ruota','Il Carro', "L'Appeso"]
minori = ['Asso di Spade', 'Due di Denari', 'Regina di Coppe']

# scrivi qui
```

```
La Ruota è un Arcano maggiore
```

4.23.7 `if` annidati

I comandi `if` sono *blocchi* perciò possono essere annidati come qualunque altro blocco.

Facciamo un esempio. Se hai un punto con coordinate `x` e `y` e vuoi sapere in quale quadrante risiede:

Potresti scrivere qualcosa del genere:

[13]:
```
x,y = 5,9
#x,y = -5,9
#x,y = -5,-9
#x,y = 5,-9

print('x =',x,'y =', y)

if x >= 0:
    if y >= 0:
        print('primo quadrante')
    else:
        print('quarto quadrante')
else:
    if y >= 0:
        print('secondo quadrante')
    else:
        print('terzo quadrante')
```

```
x = 5 y = 9
primo quadrante
```

ESERCIZIO: prova le varie coppie di punti suggeriti togliendo i commenti e convinciti che il codice funziona come atteso.

NOTA: A volte gli `if` annidati possono essere evitati scrivendo sequenze di `elif` con espressioni booleane che verificano due condizioni alla volta:

[14]:
```
x,y = 5,9
#x,y = -5,9
#x,y = -5,-9
#x,y = 5,-9
```

(continues on next page)

(continued from previous page)

```
print('x =',x,'y =', y)

if x >= 0 and y >= 0:
    print('primo quadrante')
elif x >= 0 and y < 0:
    print('quarto quadrante')
elif x < 0 and y >= 0:
    print('secondo quadrante')
elif x < 0 and y < 0:
    print('terzo quadrante')
```

```
x = 5 y = 9
primo quadrante
```

Esercizio - ascisse e ordinate 1

Il codice che abbiamo visto non è molto preciso, perchè non considera il caso di punti che giacciono sugli assi. In questi casi invece del numero del quadrante dovrebbe stampare:

- 'origine' quando `x` e `y` sono uguali a `0`
- 'ascissa' quando `y` è `0`
- 'ordinata' quando `x` è `0`

Scrivi qui sotto una versione modificata **del codice con if annidati** che consideri anche questi casi, poi testalo togliendo i commenti alle varie coppie di coordinate suggerite.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[15]:
```
x,y = 0,0      # origine
#x,y = 0,5     # ordinata
#x,y = 5,0     # ascissa
#x,y = 5,9     # primo
#x,y = -5,9    # secondo
#x,y = -5,-9   # terzo
#x,y = 5,-9    # quarto


print('x =',x,'y =', y)

# scrivi qui

if x == 0 and y == 0:
    print('origine')
elif x == 0:
    print('ordinata')
elif x > 0:
    if y == 0:
        print('ascissa')
    elif y > 0:
        print('primo quadrante')
    else:
        print('quarto quadrante')
else:
```

(continues on next page)

(continued from previous page)

```
    if y == 0:
        print('ascissa')
    elif y > 0:
        print('secondo quadrante')
    else:
        print('terzo quadrante')
```

```
x = 0 y = 0
origine
```

[15]:
```
x,y = 0,0      # origine
#x,y = 0,5     # ordinata
#x,y = 5,0     # ascissa
#x,y = 5,9     # primo
#x,y = -5,9    # secondo
#x,y = -5,-9   # terzo
#x,y = 5,-9    # quarto


print('x =',x,'y =', y)

# scrivi qui
```

```
x = 0 y = 0
origine
```

Esercizio - ascisse e ordinate 2

Volendo essere ancora più specifici, invece che stampare genericamente 'ascissa' o 'ordinata', potremmo stampare:

- 'ascissa tra il primo e quarto quadrante'
- 'ascissa tra il secondo e terzo quadrante'
- 'ordinata tra il primo e il secondo quadrante'
- 'ordinata tra il terzo e il quarto quadrante'

Copia qui sotto il codice dell'esercizio precedente modificandolo per considerare anche questi casi.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[16]:
```
x,y = 0,0      # origine
#x,y = 0,5     # ordinata tra il primo e il secondo quadrante
#x,y = 0,-5    # ordinata tra il terzo e il quarto quadrante
#x,y = 5,0     # ascissa tra il primo e il quarto quadrante
#x,y = -5,0    # ascissa tra il secondo e il terzo quadrante
#x,y = 5,9     # primo
#x,y = -5,9    # secondo
#x,y = -5,-9   # terzo
#x,y = 5,-9    # quarto
```

(continues on next page)

(continued from previous page)

```
print('x =',x,'y =', y)

# scrivi qui

if x == 0 and y == 0:
    print('origine')
elif x == 0:
    if y > 0:
        print('ordinata tra il primo e il secondo quadrante')
    else:
        print('ordinata tra il terzo e il quarto quadrante')
elif x > 0:
    if y == 0:
        print('ascissa tra il primo e il quarto quadrante')
    elif y > 0:
        print('primo quadrante')
    else:
        print('quarto quadrante')
else:
    if y == 0:
        print('ascissa tra il secondo e il terzo quadrante')
    elif y > 0:
        print('secondo quadrante')
    else:
        print('terzo quadrante')
```

```
x = 0 y = 0
origine
```

[16]:
```
x,y = 0,0      # origine
#x,y = 0,5     # ordinata tra il primo e il secondo quadrante
#x,y = 0,-5    # ordinata tra il terzo e il quarto quadrante
#x,y = 5,0     # ascissa tra il primo e il quarto quadrante
#x,y = -5,0    # ascissa tra il secondo e il terzo quadrante
#x,y = 5,9     # primo
#x,y = -5,9    # secondo
#x,y = -5,-9   # terzo
#x,y = 5,-9    # quarto

print('x =',x,'y =', y)

# scrivi qui
```

```
x = 0 y = 0
origine
```

Esercizio - autobus

Devi correre a prendere l'autobus, e hai poco tempo. Per fare il viaggio:

- ti serve lo zaino, altrimenti resti a casa
- ti servono anche dei soldi per il biglietto, oppure la tessera trasporti (o tutte due). Altrimenti resti a casa.

Scrivi del codice che date tre variabili booleane `zaino`, `soldi` e `tessera`, stampa quello che vedi nei commenti a seconda dei vari casi. Una volta scritto il codice, togli i commenti alle assegnazioni per provare i risultati

- **SUGGERIMENTO**: per tener traccia degli oggetti posseduti, ti conviene creare una lista di stringhe dove mettere gli oggetti trovati

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[17]:
```
zaino, soldi, tessera = True, False, True
# Non ho i soldi !
# Ho trovato: zaino,tessera
# Posso partire !

#zaino, soldi, tessera = False, False, True
# Non ho lo zaino, non posso partire !

#zaino, soldi, tessera = True, True, False
# Non ho la tessera !
# Ho trovato: zaino,soldi
# Posso partire !

#zaino, soldi, tessera = True, True, True
# Ho trovato: zaino,soldi,tessera
# Posso partire !

zaino, soldi, tessera = True, False, False
#Non ho i soldi !
#Non ho la tessera !
#Non ho la tessera nè i soldi, non posso partire !


# scrivi qui

trovato = []
if zaino:
    trovato.append('zaino')
    if soldi:
        trovato.append('soldi')
    else:
        print('Non ho i soldi !')
    if tessera:
        trovato.append('tessera')
    else:
        print('Non ho la tessera !')
    if soldi or tessera:
        print('Ho trovato:', ','.join(trovato))
        print('Posso partire !')
    else:
```

(continues on next page)

(continued from previous page)

```
        print('Non ho la tessera nè i soldi, non posso partire !')
else:
    print('Non ho lo zaino, non posso partire !')
```

```
Non ho i soldi !
Non ho la tessera !
Non ho la tessera nè i soldi, non posso partire !
```

[17]:

```
zaino, soldi, tessera = True, False, True
# Non ho i soldi !
# Ho trovato: zaino,tessera
# Posso partire !

#zaino, soldi, tessera = False, False, True
# Non ho lo zaino, non posso partire !

#zaino, soldi, tessera = True, True, False
# Non ho la tessera !
# Ho trovato: zaino,soldi
# Posso partire !

#zaino, soldi, tessera = True, True, True
# Ho trovato: zaino,soldi,tessera
# Posso partire !

zaino, soldi, tessera = True, False, False
#Non ho i soldi !
#Non ho la tessera !
#Non ho la tessera nè i soldi, non posso partire !


# scrivi qui
```

```
Non ho i soldi !
Non ho la tessera !
Non ho la tessera nè i soldi, non posso partire !
```

Esercizio - cronometro

Un cronometro segna le ore, minuti e secondi trascorsi dalla mezzanotte di un certo giorno in una stringa `cronometro`, in cui i numeri delle ore, minuti e secondi sono separati da due punti `:`.

Scrivi del codice che in base al numero di ore trascorse, stampa la fase del giorno:

- dalle ore 6 incluse alle ore 12 escluse : stampa `mattino`
- dalle ore 12 incluse alle ore 18 escluse : stampa `pomeriggio`
- dalle ore 18 incluse alle ore 21 escluse : stampa `sera`
- dalle ore 21 incluse alle ore 6 escluse : stampa `notte`

- **USA** `elif` con espressioni booleane multiple
- Il tuo codice **DEVE** funzionare anche se il cronometro supera le 23:59:59, vedi esempi
- **SUGGERIMENTO**: per avere ore che vadano solo dalle 0 alle 23, usa l'operatore modulo `%`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
cronometro = '10:23:43'   # mattina
#cronometro = '12:00:00'   # pomeriggio
#cronometro = '15:56:02'   # pomeriggio
#cronometro = '19:23:27'   # sera
#cronometro = '21:45:15'   # notte
#cronometro = '02:45:15'   # notte
#cronometro = '27:45:30'   # notte
#cronometro = '32:28:30'   # mattina

# scrivi qui

ora = int(cronometro.split(':')[0]) % 24

if ora >= 6 and ora < 12:
    print('mattina')
elif ora >=12 and ora < 18:
    print('pomeriggio')
elif ora >=18 and ora < 21:
    print('sera')
else:
    print('notte')
```

```
mattina
```

[18]:
```
cronometro = '10:23:43'   # mattina
#cronometro = '12:00:00'   # pomeriggio
#cronometro = '15:56:02'   # pomeriggio
#cronometro = '19:23:27'   # sera
#cronometro = '21:45:15'   # notte
#cronometro = '02:45:15'   # notte
#cronometro = '27:45:30'   # notte
#cronometro = '32:28:30'   # mattina

# scrivi qui
```

```
mattina
```

4.23.8 Domande - Sono equivalenti?

Guarda i seguenti frammenti di codice: in ciascuno, vi sono due parti, A e B. In ciascun frammento, cerca di indovinare se la parte A stamperà esattamente quello che stampa il codice nella parte B, per ogni valore di `x`.

- **PRIMA** pensa alla risposta
- **POI** prova ad eseguire con ciascuno dei valori di `x` suggeriti.

Sono equivalenti? - dentro fuori 1

```
x = 3
#x = 4
#x = 5

print('x =',x)

print('A:')
if x > 3:
    if x < 5:
        print('dentro')
    else:
        print('fuori')
else:
    print('fuori')

print('B:')
if x > 3 and x < 5:
    print('dentro')
else:
    print('fuori')
```

Sono equivalenti? - stelle pianeti

```
x = 2
#x = 3
#x = 4

print('x =', x)

print('A:')
if not x > 3:
    print('stelle')
else:
    print('pianeti')

print('B:')
if x > 3:
    print('pianeti')
else:
    print('stelle')
```

Sono equivalenti? - verde rosso

```
x = 10
#x = 5
#x = 0

print('x =',x)

print('A:')
if x >= 5:
    print('verde')
    if x >= 10:
        print('rosso')


print('B:')
if x >= 10:
    if x >= 5:
        print('verde')
    print('rosso')
```

Sono equivalenti? - cerchi quadrati

```
x = 4
#x = 3
#x = 2
#x = 1
#x = 0

print('x =', x)

print('A:')
if x > 3:
    print('cerchi')
else:
    if x > 1:
        print('quadrati')
    else:
        print('triangoli')

print('B:')
if x <= 1:
    print('triangoli')
elif x <= 3:
    print('quadrati')
else:
    print('cerchi')
```

Sono equivalenti? - dentro fuori 2

```
x = 7
#x = 0
#x = 15

print('x =', x)

print('A:')
if x > 5:
    if x < 10:
        print('dentro')
    else:
        print('fuori')
else:
    print('fuori')

print('B:')
if not x > 5 and not x < 10:
    print('fuori')
else:
    print('dentro')
```

Sono equivalenti? - Ciabanga

```
x = 4
#x = 5
#x = 6
#x = 9
#x = 10
#x = 11

print('x =', x)

print('A:')
if x < 6:
    print('Ciabanga!')
else:
    if x >= 10:
        print('Ciabanga!')

print('B:')
if x <= 5 or not x < 10:
    print('Ciabanga!')
```

Esercizio - Il massimo

Scrivi del codice che stampa il massimo fra tre numeri `x`, `y` e `z`

- usa **if annidati**
- **NON** usare la funzione `max`
- **NON** creare variabili che si chiamano `max` (violerebbe il V Comandamento[213]: non ridifinerai mai funzioni di sistema)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
x,y,z = 1,2,3
#x,y,z = 1,3,2
#x,y,z = 2,1,3
#x,y,z = 2,3,1
#x,y,z = 3,1,2
#x,y,z = 3,2,1

# scrivi qui

if x > y:
    if x > z:
        print(x)
    else:
        print(z)
elif y > z:
    print(y)
else:
    print(z)
```

```
3
```

[19]:
```
x,y,z = 1,2,3
#x,y,z = 1,3,2
#x,y,z = 2,1,3
#x,y,z = 2,3,1
#x,y,z = 3,1,2
#x,y,z = 3,2,1

# scrivi qui
```

[213] https://it.softpython.org/commandments.html#V-COMANDAMENTO

```
3
```

4.23.9 Operatore ternario

In qualche caso è conveniente inizializzare una variabile con valori diversi a seconda di una condizione

Esempio:

Lo sconto che si applica ad un'acquisto dipende dalla quantità acquistata. Crea una variabile `sconto` mettendo il suo valore a 0 se la variabile `spesa` è minore di 100€, oppure `10%` se è maggiore.

[20]:
```
spesa = 200
sconto = 0

if spesa > 100:
    sconto = 0.1
else:
    sconto = 0 # non necessario

print("spesa:", spesa, "  sconto:", sconto)
```

```
spesa: 200   sconto: 0.1
```

Il codice precedente può essere scritto più concisamente così:

[21]:
```
spesa = 200
sconto = 0.1 if spesa > 100 else 0
print("spesa:", spesa, "  sconto:", sconto)
```

```
spesa: 200   sconto: 0.1
```

La sintassi dell'operatore ternario è:

```
VARIABILE = VALORE if CONDIZIONE else ALTRO_VALORE
```

col significato che `VARIABILE` è inizializzata a `VALORE` se la `CONDIZIONE` è `True`, altrimenti viene inizializzata ad `ALTRO_VALORE`

Domande if ternari

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
y = 3
x = 8 if y < 2 else 9
print(x)
```

2.
```
y = 1
z = 2 if y < 3
```

3.
```
y = 10
z = 2 if y < 3 elif y > 5 9
```

Esercizio - scarpe

Scrivere del codice che data la variabile numerica `scarpe`, se `scarpe` è minore di 10 viene incrementata di `1`, altrimenti viene decrementata di `1`

- Usa **SOLO** l'**if ternario**
- Il tuo codice deve funzionare per *qualsiasi* valore di `scarpe`

Esempio 1 - dato:

```
scarpe = 2
```

Dopo il tuo codice, deve risultare:

```
>>> print(scarpe)
3
```

Esempio 2 - dato:

```
scarpe = 16
```

Dopo il tuo codice, deve risultare:

```
>>> print(scarpe)
15
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
scarpe = 2
#scarpe = 16

# scrivi qui

scarpe = scarpe + 1 if scarpe < 10 else scarpe - 1
print('scarpe =', scarpe)
```

```
scarpe = 3
```

[22]:
```
scarpe = 2
#scarpe = 16

# scrivi qui


```

```
scarpe = 3
```

Esercizio - trenino

Scrivi del codice che date 3 stringhe `sa`, `sb` e `sc` assegna alla variabile `x` la stringa `'CIUFCIUF'` se è possibile comporre `sa`, `sb` e `sc` per ottenere la scritta `'trenino'`, altrimenti mette la stringa `':-('`

- **USA** un if ternario
- il tuo codice deve funzionare per **qualsiasi** terna di stringhe
- **NOTA**: ci interessa sapere SE è possibile comporre le scritte come `'trenino'`, NON in che ordine vanno composte
- **SUGGERIMENTO**: ti è consentito creare una lista di appoggio

Esempio 1 - date:

```
sa,sb,sc = "ni","no","tre"
```

dopo il tuo codice, deve risultare:

```
>>> print(x)
CIUFCIUF
```

Esempio 2 - date:

```
sa,sb,sc = "quattro","ni","no"
```

dopo il tuo codice, deve risultare:

```
>>> print(x)
:-(
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
sa,sb,sc = "ni","no","tre"        # CIUFCIUF
#sa,sb,sc = "ni","tre","no"       # CIUFCIUF
#sa,sb,sc = "quattro","ni","no"   # :-(
#sa,sb,sc = "na","tre","no"       # :-(


# scrivi qui
tutte = [sa,sb,sc]
x = 'CIUFCIUF' if "tre" in tutte and "ni" in tutte and "no" in tutte else ':-('

#print(x)
```

[23]:
```
sa,sb,sc = "ni","no","tre"        # CIUFCIUF
#sa,sb,sc = "ni","tre","no"       # CIUFCIUF
#sa,sb,sc = "quattro","ni","no"   # :-(
#sa,sb,sc = "na","tre","no"       # :-(


# scrivi qui
```

```
[ ]:
```

4.24 Controllo di flusso - Cicli for

4.24.1 Scarica zip esercizi

Naviga file online[214]

4.24.2 Introduzione

Se vogliamo compiere una o più azioni per ogni elemento di una sequenza avremo bisogno del cosiddetto ciclo `for`, che ci permette di *iterare* su una sequenza.

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
control-flow
    flow1-if.ipynb
    flow1-if-sol.ipynb
    flow2-for.ipynb
    flow2-for-sol.ipynb
    flow3-while.ipynb
    flow3-while-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `flow2-for.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

[214] https://github.com/DavidLeoni/softpython-it/tree/master/control-flow

4.24.3 Iterazione per elemento

Se abbiamo una sequenza come per esempio una lista:

[2]:
```
sport = ['pallavolo', 'tennis', 'calcio', 'nuoto']
```

e vogliamo in qualche modo usare ogni elemento della lista (per esempio stamparli), li possiamo scorrere (o meglio, *iterare*) con un ciclo `for`:

[3]:
```
for elemento in sport:
    print('Trovato un elemento!')
    print(elemento)

print('Finito!')
```

```
Trovato un elemento!
pallavolo
Trovato un elemento!
tennis
Trovato un elemento!
calcio
Trovato un elemento!
nuoto
Finito!
```

Vediamo cosa succede in Python Tutor:

[4]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[5]:
```
sport = ['pallavolo', 'tennis', 'calcio', 'nuoto']
for elemento in sport:
    print('Trovato un elemento!')
    print(elemento)

print('Finito!')

jupman.pytut()
```

```
Trovato un elemento!
pallavolo
Trovato un elemento!
tennis
Trovato un elemento!
calcio
Trovato un elemento!
nuoto
Finito!
```

[5]:
```
<IPython.core.display.HTML object>
```

Nomi variabili nei `for`

Ad ogni iterazione, un elemento della lista viene assegnato alla variabile `elemento`.

Come nome per la variabile possiamo scegliere quello che ci pare, per esempio questo codice è totalmente equivalente al precedente:

[6]:
```
sport = ['pallavolo', 'tennis', 'calcio', 'nuoto']
for nome in sport:
    print('Trovato un elemento!')
    print(nome)

print('Finito!')
```

```
Trovato un elemento!
pallavolo
Trovato un elemento!
tennis
Trovato un elemento!
calcio
Trovato un elemento!
nuoto
Finito!
```

Bisogna però fare attenzione ad una cosa:

II COMANDAMENTO[215]: **Quando inserisci una variabile in un ciclo `for` , questa variabile deve essere nuova**

Se hai definito la variabile prima, non la reintrodurrai in un `for`, perchè ciò portebbe gran confusione.

Per esempio:

[7]:
```
sport = ['pallavolo', 'tennis', 'calcio', 'nuoto']
prova = 'ciao'

for prova in sport:  # perdi la variabile  prova  originale
    print(prova)

print(prova) # stampa 'nuoto' invece di 'ciao'
```

```
pallavolo
tennis
calcio
nuoto
nuoto
```

Iterare in stringhe

Le stringhe sono sequenze di caratteri perciò possiamo iterarle con il `for`:

[8]:
```
for carattere in "ciao":
    print(carattere)
```

```
c
i
a
o
```

[215] https://it.softpython.org/commandments.html#II-COMANDAMENTO

Iterare in tuple

Anche le tuple sono sequenze quindi possiamo iterarle:

[9]:
```
for parola in ('sto', 'visitando', 'una', 'tupla'):
    print(parola)
```

```
sto
visitando
una
tupla
```

Domande

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
for i in [1,2,3]:
    print(i)
```

2.
```
for x in 7:
    print(x)
```

3.
```
for x in [7]:
    print(x)
```

4.
```
for x in ['a','b','c']:
    x
```

5.
```
for i in []:
    print('GURB')
```

6.
```
for i in [1,2,3]:
    print(type(i))
```

7.
```
for i in '123':
    print(type(i))
```

8.
```
for i in 'abc':
    print(i)
```

9.
```
for x in ((4,5,6)):
    print(x)
```

10.
```
for x in [[1],[2,3],[4,5,6]]:
    print(x)
```

11.
```
x = 5
for x in ['a','b','c']:
    print(x)
print(x)
```

12.
```
for x in ['a','b','c']:
    pass
print(x)
```

13.
```
for x in [1,2,3,4,5,6,7,8]:
    if x % 2 == 0:
        print(x)
```

14.
```
la = [4,5,6]
for x in la:
    print(x)
la.reverse()
for x in la[1:]:
    print(x)
```

Esercizio - tappeto magico

⊛ Qualche mese fa hai acquistato un tappeto da un venditore ambulante. Tempo dopo, alla fine di un giorno particolarmente stressante, esclami 'Ah quanto vorrei andare in vacanza in qualche posto esotico, che so, a *Marrakesh*!' Con sorpresa, il tappeto si alza in aria e risponde: 'Ho sentito e obbedisco!'

Scrivi del codice che data le liste di luoghi `vaggio1` e `viaggio2` stampa tutte le tappe visitate.

Esempio - dati:

```
viaggio1 = ['Marrakesh','Fez','Bazaar','Kasbah']
viaggio2 = ['Koutoubia', 'El Badii', 'Chellah']
```

Stampa:

```
Inizia il primo viaggio
        Tu: Andiamo a  Marrakesh !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Fez !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Bazaar !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Kasbah !
   Tappeto: Ho sentito e obbedisco!
Fine del primo viaggio

Inizia il secondo viaggio
        Tu: Andiamo a  Koutoubia !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  El Badii !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Chellah !
   Tappeto: Ho sentito e obbedisco!
Fine del secondo viaggio
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[10]:
```
viaggio1 = ['Marrakesh','Fez','Bazaar','Kasbah']
viaggio2 = ['Koutoubia', 'El Badii', 'Chellah']


# scrivi qui
print('Inizia il primo viaggio')
for luogo in viaggio1:
    print('        Tu: Andiamo a ',luogo,'!')
    print('   Tappeto: Ho sentito e obbedisco!')
print('Fine del primo viaggio')

print()

print('Inizia il secondo viaggio')
for luogo in viaggio2:
    print('        Tu: Andiamo a ',luogo,'!')
    print('   Tappeto: Ho sentito e obbedisco!')
print('Fine del secondo viaggio')
```

```
Inizia il primo viaggio
        Tu: Andiamo a  Marrakesh !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Fez !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Bazaar !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Kasbah !
   Tappeto: Ho sentito e obbedisco!
Fine del primo viaggio

Inizia il secondo viaggio
        Tu: Andiamo a  Koutoubia !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  El Badii !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Chellah !
   Tappeto: Ho sentito e obbedisco!
Fine del secondo viaggio
```

[10]:
```
viaggio1 = ['Marrakesh','Fez','Bazaar','Kasbah']
viaggio2 = ['Koutoubia', 'El Badii', 'Chellah']


# scrivi qui
```

```
Inizia il primo viaggio
        Tu: Andiamo a  Marrakesh !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Fez !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Bazaar !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Kasbah !
   Tappeto: Ho sentito e obbedisco!
```

(continues on next page)

(continued from previous page)

```
Fine del primo viaggio

Inizia il secondo viaggio
        Tu: Andiamo a  Koutoubia !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  El Badii !
   Tappeto: Ho sentito e obbedisco!
        Tu: Andiamo a  Chellah !
   Tappeto: Ho sentito e obbedisco!
Fine del secondo viaggio
```

Esercizio - sommapari

⊛ Data la lista `numeri`, scrivi del codice che calcola e stampa la somma degli **elementi** pari (**non** gli elementi ad indici pari !)

Esempio - data:

```
numeri = [3,4,1,5,12,7,9]
```

trova `4` e `12` quindi deve stampare:

```
16
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[11]:
```
numeri = [3,4,1,5,12,7,9]

# scrivi qui
s = 0
for x in numeri:
    if x % 2 == 0:
        s += x
print(s)
```

```
16
```

[11]:
```
numeri = [3,4,1,5,12,7,9]

# scrivi qui
```

```
16
```

Esercizio - birbantello

⊛ Data una stringa in minuscolo, scrivere del codice che stampa ogni carattere in maiuscolo seguito dal carattere in minuscolo.

- **SUGGERIMENTO**: per ottenere caratteri in maiuscolo usa il metodo `.upper()`

Esempio - data:

```
s = "birbantello"
```

Stampa:

```
B b
I i
R r
B b
A a
N n
T t
E e
L l
L l
O o
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
s = "birbantello"

# scrivi qui
for x in s:
    print(x.upper(), x)
```

```
B b
I i
R r
B b
A a
N n
T t
E e
L l
L l
O o
```

[12]:
```
s = "birbantello"

# scrivi qui
```

```
B b
I i
R r
B b
```

(continues on next page)

(continued from previous page)

```
A a
N n
T t
E e
L l
L l
O o
```

Esercizio - dirigibile

⊛ A Pierino viene fatta imparare una nuova `parola`. Pierino conosce tante lettere dell'alfabeto ma non tutte. Per ricordarsi quelle che conosce, le tratta come personaggi che divide in 3 categorie: `belli`, `brutti` e `cattivi`. Scrivi del codice che data una `parola` stampa tutti i caratteri e per ciascuno indica se è bello, brutto o cattivo. Se un carattere non è conosciuto da Pierino, stampa 'non mi interessa'.

Esempio - dati:

```
parola = 'dirigibile'

belli = 'abcd'
brutti = 'efgh'
cattivi = 'ilm'
```

Stampa:

```
d è bello
i è cattivo
r non mi interessa
i è cattivo
g è brutto
i è cattivo
b è bello
i è cattivo
l è cattivo
e è brutto
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[13]:
```
parola = 'dirigibile'

belli = 'abcd'
brutti = 'efgh'
cattivi = 'ilm'

# scrivi qui

for c in parola:
    if c in belli:
        print(c,'è bello')
    elif c in brutti:
        print(c, 'è brutto')
    elif c in cattivi:
```

(continues on next page)

(continued from previous page)

```
        print(c, 'è cattivo')
    else:
        print(c, 'non mi interessa')
```

```
d è bello
i è cattivo
r non mi interessa
i è cattivo
g è brutto
i è cattivo
b è bello
i è cattivo
l è cattivo
e è brutto
```

[13]:

```
parola = 'dirigibile'

belli = 'abcd'
brutti = 'efgh'
cattivi = 'ilm'

# scrivi qui
```

```
d è bello
i è cattivo
r non mi interessa
i è cattivo
g è brutto
i è cattivo
b è bello
i è cattivo
l è cattivo
e è brutto
```

Esercizio - gala

⊛ Ad un evento di gala vengono invitate figure altolocate della società. All'inizio della serata sono aperte le porte e gli ospiti si mettono in `coda` all'ingresso. Sfortunatamente, a queste occasioni cercano sempre di presentarsi anche personaggi poco raccomandabili, perciò al concierge nell'atrio viene consegnato un insieme di ospiti `sgraditi`. Qualora un ospite sia riconosciuto come non gradito, verrà affidato alle capaci mani del buttafuori Ferruccio. Gli ospiti illustri vengono invece segnati nella lista `ammessi`.

Scrivi del codice che stampa i vari passi del ricevimento.

Esempio - dati:

```
coda = ['Console','Notaio','Scheletro','Rettore','Goblin','Vampiro', 'Gioielliere']
sgraditi = {'Vampiro','Goblin','Scheletro'}
ammessi = []
```

Stampa:

```
Aprite le porte!

Buonasera Signor Console
  Prego Eccellenza, entri pure
  Avanti il prossimo !
Buonasera Signor Notaio
  Prego Eccellenza, entri pure
  Avanti il prossimo !
Buonasera Signor Scheletro
  Ferruccio, vuoi prenderti cura del signor Scheletro ?
  Avanti il prossimo !
Buonasera Signor Rettore
  Prego Eccellenza, entri pure
  Avanti il prossimo !
Buonasera Signor Goblin
  Ferruccio, vuoi prenderti cura del signor Goblin ?
  Avanti il prossimo !
Buonasera Signor Vampiro
  Ferruccio, vuoi prenderti cura del signor Vampiro ?
  Avanti il prossimo !
Buonasera Signor Gioielliere
  Prego Eccellenza, entri pure
  Avanti il prossimo !

Sono stati ammessi i signori Console, Notaio, Rettore, Gioielliere
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[14]:
```
coda = ['Console','Notaio','Scheletro','Rettore','Goblin','Vampiro', 'Gioielliere']
sgraditi = {'Vampiro','Goblin','Scheletro'}
ammessi = []

# scrivi qui
print('Aprite le porte!')
print()
for ospite in coda:

    print('Buonasera Signor', ospite)
    if ospite in sgraditi:
        print("  Ferruccio, vuoi prenderti cura del signor", ospite, '?')
    else:
        print("  Prego Eccellenza, entri pure")
        ammessi.append(ospite)
    print('  Avanti il prossimo !')

print()
print('Sono stati ammessi i signori', ', '.join(ammessi))
```

```
Aprite le porte!

Buonasera Signor Console
  Prego Eccellenza, entri pure
  Avanti il prossimo !
Buonasera Signor Notaio
  Prego Eccellenza, entri pure
  Avanti il prossimo !
```

(continues on next page)

(continued from previous page)

```
Buonasera Signor Scheletro
  Ferruccio, vuoi prenderti cura del signor Scheletro ?
  Avanti il prossimo !
Buonasera Signor Rettore
  Prego Eccellenza, entri pure
  Avanti il prossimo !
Buonasera Signor Goblin
  Ferruccio, vuoi prenderti cura del signor Goblin ?
  Avanti il prossimo !
Buonasera Signor Vampiro
  Ferruccio, vuoi prenderti cura del signor Vampiro ?
  Avanti il prossimo !
Buonasera Signor Gioielliere
  Prego Eccellenza, entri pure
  Avanti il prossimo !

Sono stati ammessi i signori Console, Notaio, Rettore, Gioielliere
```

[14]:
```
coda = ['Console','Notaio','Scheletro','Rettore','Goblin','Vampiro', 'Gioielliere']
sgraditi = {'Vampiro','Goblin','Scheletro'}
ammessi = []

# scrivi qui
```

```
Aprite le porte!

Buonasera Signor Console
  Prego Eccellenza, entri pure
  Avanti il prossimo !
Buonasera Signor Notaio
  Prego Eccellenza, entri pure
  Avanti il prossimo !
Buonasera Signor Scheletro
  Ferruccio, vuoi prenderti cura del signor Scheletro ?
  Avanti il prossimo !
Buonasera Signor Rettore
  Prego Eccellenza, entri pure
  Avanti il prossimo !
Buonasera Signor Goblin
  Ferruccio, vuoi prenderti cura del signor Goblin ?
  Avanti il prossimo !
Buonasera Signor Vampiro
  Ferruccio, vuoi prenderti cura del signor Vampiro ?
  Avanti il prossimo !
Buonasera Signor Gioielliere
  Prego Eccellenza, entri pure
  Avanti il prossimo !

Sono stati ammessi i signori Console, Notaio, Rettore, Gioielliere
```

Esercizio - bilancia

⊛⊛ E' stato fatto un raccolto di sementi, che verranno messe in un certo numero di `sacchi` da `capienza` kg l'uno (es `15`).

Le sementi arrivano in `recipienti` di capacità variabile. Ogni recipiente viene messo su una bilancia e il suo contenuto versato nel sacco corrente. Nel momento in cui si arriva alla quantità `capienza`, la bilancia si svuota, il sacco viene sostituito con uno nuovo e si comincia a riempire quello nuovo partendo dagli eventuali resti in eccesso dal riempimento precedente. Scrivi del codice che stampa il procedimento.

Esempio - dati:

```
recipienti = [5,1,7,4,3,9,5,2,7,3]
capienza = 15
```

Stampa:

```
Raccolto 5 kg
La bilancia segna 5 kg
Raccolto 1 kg
La bilancia segna 6 kg
Raccolto 7 kg
La bilancia segna 13 kg
Raccolto 4 kg
La bilancia segna 17 kg
Abbiamo raggiunto la capienza di 15 kg, avanzano 2 kg

Raccolto 3 kg
La bilancia segna 5 kg
Raccolto 9 kg
La bilancia segna 14 kg
Raccolto 5 kg
La bilancia segna 19 kg
Abbiamo raggiunto la capienza di 15 kg, avanzano 4 kg

Raccolto 2 kg
La bilancia segna 6 kg
Raccolto 7 kg
La bilancia segna 13 kg
Raccolto 3 kg
La bilancia segna 16 kg
Abbiamo raggiunto la capienza di 15 kg, avanzano 1 kg

Abbiamo riempito 3 sacchi
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[15]:
```
recipienti = [5,1,7,4,3,9,5,2,7,3]
capienza = 15

# scrivi qui
sacchi= 0
k = 0
for n in recipienti:
    k += n
    print('Raccolto', n, 'kg')
```

(continues on next page)

(continued from previous page)

```
    print('La bilancia segna', k,'kg')
    if k >= capienza:

        print('Abbiamo raggiunto la capienza di',capienza,'kg, avanzano', k -␣
↪capienza, 'kg')
        print()
        k = k - capienza
        sacchi += 1

print('Abbiamo riempito', sacchi, 'sacchi')
```

```
Raccolto 5 kg
La bilancia segna 5 kg
Raccolto 1 kg
La bilancia segna 6 kg
Raccolto 7 kg
La bilancia segna 13 kg
Raccolto 4 kg
La bilancia segna 17 kg
Abbiamo raggiunto la capienza di 15 kg, avanzano 2 kg

Raccolto 3 kg
La bilancia segna 5 kg
Raccolto 9 kg
La bilancia segna 14 kg
Raccolto 5 kg
La bilancia segna 19 kg
Abbiamo raggiunto la capienza di 15 kg, avanzano 4 kg

Raccolto 2 kg
La bilancia segna 6 kg
Raccolto 7 kg
La bilancia segna 13 kg
Raccolto 3 kg
La bilancia segna 16 kg
Abbiamo raggiunto la capienza di 15 kg, avanzano 1 kg

Abbiamo riempito 3 sacchi
```

[15]:
```
recipienti = [5,1,7,4,3,9,5,2,7,3]
capienza = 15

# scrivi qui
```

```
Raccolto 5 kg
La bilancia segna 5 kg
Raccolto 1 kg
La bilancia segna 6 kg
Raccolto 7 kg
La bilancia segna 13 kg
Raccolto 4 kg
La bilancia segna 17 kg
Abbiamo raggiunto la capienza di 15 kg, avanzano 2 kg
```

(continues on next page)

(continued from previous page)

```
Raccolto 3 kg
La bilancia segna 5 kg
Raccolto 9 kg
La bilancia segna 14 kg
Raccolto 5 kg
La bilancia segna 19 kg
Abbiamo raggiunto la capienza di 15 kg, avanzano 4 kg

Raccolto 2 kg
La bilancia segna 6 kg
Raccolto 7 kg
La bilancia segna 13 kg
Raccolto 3 kg
La bilancia segna 16 kg
Abbiamo raggiunto la capienza di 15 kg, avanzano 1 kg

Abbiamo riempito 3 sacchi
```

4.24.4 Contare con range

Se abbiamo bisogno di contare a quale iterazione siamo, possiamo usare la sequenza iterabile `range`, che produce una serie di numeri interi da `0` INCLUSO fino al numero specificato ESCLUSO:

[16]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Nota come *non* abbia stampato l'estremo `5`.

Quando chiamiamo `range` possiamo anche specificare l'indice di partenza, che viene INCLUSO nella sequenza generata, mentre l'indice di arrivo è sempre ESCLUSO:

[17]:
```
for i in range(3,7):
    print(i)
```

```
3
4
5
6
```

Contare a intervalli: possiamo specificare l'incremento da applicare al contatore ad ogni iterazione passando un terzo parametero, qua per esempio specifichiamo un incremento di `2` (nota come l'indice finale `18` rimanga ESCLUSO dalla sequenza):

[18]:
```
for i in range(4,18,2):
    print(i)
```

```
4
6
8
10
```

(continues on next page)

(continued from previous page)

```
12
14
16
```

Ordine inverso: Si può contare al contrario usando un incremento negativo:

[19]:
```
for i in range(5,0,-1):
    print(i)
```

```
5
4
3
2
1
```

Nota come il limite `0` *non* sia stato raggiunto, per arrivarci occorre scrivere

[20]:
```
for i in range(5,-1,-1):
    print(i)
```

```
5
4
3
2
1
0
```

Domande - range

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. ```
   for x in range(1):
       print(x)
   ```

2. ```
   for i in range(3):
       i
   ```

3. ```
   for i in range(3):
   print(i)
   ```

4. ```
   for x in range(-1):
       print(x)
   ```

5. ```
   for 'm' in range(3):
       print('m')
   ```

6. ```
   for i in range(3):
       i-1
   ```

7. ```
   for x in range(6,4,-1):
       print(x)
   ```
   ```

8.
```
for x in range(1,0,-1):
    print(x)
```

9.
```
for x in range(3,-3,-2):
    print(x)
```

10.
```
for x in 3:
    print(x)
```

11.
```
x = 3
for i in range(x):
    print(i)
for i in range(x,2*x):
    print(i)
```

12.
```
for x in range(range(3)):
    print(x)
```

Esercizio - stampadoppi

⊛ Dato un numero positivo `n` (per es `n=4`) scrivi del codice che stampa:

```
il doppio di 0 è 0
il doppio di 1 è 2
il doppio di 2 è 4
il doppio di 3 è 6
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[21]:
```
n = 4
# scrivi qui
for i in range(n):
    print('il doppio di', i, 'è', i*2)
```

```
il doppio di 0 è 0
il doppio di 1 è 2
il doppio di 2 è 4
il doppio di 3 è 6
```

[21]:
```
n = 4
# scrivi qui
```

```
il doppio di 0 è 0
il doppio di 1 è 2
il doppio di 2 è 4
il doppio di 3 è 6
```

Esercizio - multipli e non

⊛⊛ Scrivi del codice che dati due numeri interi positivi `k` e `b`:

- prima stampa tutti i numeri da `k` INCLUSO a `b` INCLUSO che sono multipli di `k`
- poi stampa tutti i numeri da `k` ESCLUSO a `b` ESCLUSO che NON sono multipli di `k`

Esempio - dati

```
k,b = 3,15
```

deve stampare:

```
Multipli di 3
3
6
9
12
15

Non divisibili per 3:
4
5
7
8
10
11
13
14
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
k,b = 3,15

# scrivi qui
print('Multipli di', k)
for i in range(k,b+1,k):
    print(i)
print()
print('Non divisibili per', k)
for i in range(k+1,b):
    if i % k != 0:
        print(i)
```

```
Multipli di 3
3
6
9
12
15

Non divisibili per 3
4
5
7
8
```

(continues on next page)

(continued from previous page)

```
10
11
13
14
```

[22]:
```
k,b = 3,15

# scrivi qui
```

```
Multipli di 3
3
6
9
12
15

Non divisibili per 3
4
5
7
8
10
11
13
14
```

Esercizio - intervallo ab

⊛⊛ Data due interi `a` e `b` maggiori o uguali a zero , scrivere del codice che stampa tutti i numeri interi tra i due estremi INCLUSI

- **NOTA**: `a` può essere maggiore, uguale o minore di `b`, il tuo codice deve gestire tutti i casi.

Esempio 1 - dati

```
a,b = 5,9
```

deve stampare

```
5
6
7
8
9
```

Esempio 2 - dati

```
a,b = 8,3
```

deve stampare

```
3
4
5
6
7
8
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
a,b = 5,9  # 5 6 7 8 9
#a,b = 8,3  # 3 4 5 6 7 8
#a,b = 6,6   # 6

# scrivi qui

minimo = min(a,b)
massimo = max(a,b)

for x in range(minimo, massimo + 1):
    print(x)
```

```
5
6
7
8
9
```

[23]:
```
a,b = 5,9  # 5 6 7 8 9
#a,b = 8,3  # 3 4 5 6 7 8
#a,b = 6,6   # 6

# scrivi qui
```

```
5
6
7
8
9
```

4.24.5 Iterare per indici

Se abbiamo una sequenza come una lista, a volte è necessario conoscere in quale cella si è durante l'iterazione: per farlo serve tener traccia degli indici. Possiamo generare gli indici da controllare con `range`, e usarli per accedere ad una lista:

[24]:
```
sport = ['pallavolo', 'tennis', 'calcio', 'nuoto']

for i in range(len(sport)):
    print('posizione', i)
    print(sport[i])
```

```
posizione 0
pallavolo
posizione 1
tennis
posizione 2
calcio
posizione 3
nuoto
```

Nota come abbiamo passato a `range` la dimensione della lista con `len`

Esercizio - cucina

⊛ Scrivi del codice che data una lista di stringhe `cucina` in numero pari, stampa le coppie di elementi che si possono trovare in sequenza, una riga alla volta

Esempio - dati:

```
cucina = ['dado', 'minestra', 'uova', 'torta', 'sugo', 'pasta', 'ragù', 'lasagne']
```

Stampa:

```
dado minestra
uova torta
sugo pasta
ragù lasagne
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[25]:
```
cucina = ['dado', 'minestra', 'uova', 'torta', 'sugo', 'pasta', 'ragù', 'lasagne']

# scrivi qui
for i in range(0,len(cucina)-1, 2):
    print(cucina[i], cucina[i+1])
```

```
dado minestra
uova torta
sugo pasta
ragù lasagne
```

[25]:
```
cucina = ['dado', 'minestra', 'uova', 'torta', 'sugo', 'pasta', 'ragù', 'lasagne']

# scrivi qui
```

```
dado minestra
uova torta
sugo pasta
ragù lasagne
```

Esercizio - neon

⊛ Date due liste `la` e `lb` *di uguale lunghezza* n, scrivi del codice che stampa su n righe i loro caratteri separati da uno spazio

Esempio - dati:

```
la = ['n','e','o','n']
lb = ['s','h','o','w']
```

stampa

```
n s
e h
o o
n w
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[26]:
```
la = ['n','e','o','n']
lb = ['s','h','o','w']

# scrivi qui

for i in range(len(la)):
    print(la[i],lb[i])
```

```
n s
e h
o o
n w
```

[26]:
```
la = ['n','e','o','n']
lb = ['s','h','o','w']

# scrivi qui
```

```
n s
e h
o o
n w
```

Esercizio - emozioni

⊛ Date una lista di stringhe `emozioni` e una `grado` contenente numeri `-1` e `1`, scrivi del codice che stampa le emozioni affiancandole a 'positivo' se il corrispondente grado è un numero maggiore di zero e 'negativa'.

Esempio - dati:

```
emozioni = ['Paura', 'Rabbia','Tristezza','Gioia','Disgusto','Estasi']
grado =    [-1,       -1,     -1,          1,     -1,         1]
```

stampa:

```
Paura : negativa
Rabbia : negativa
Tristezza : negativa
Gioia : positiva
Disgusto : negativa
Estasi : positiva
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[27]:
```
emozioni = ['Paura', 'Rabbia','Tristezza','Gioia','Disgusto','Estasi']
grado =     [-1,       -1,     -1,          1,     -1,         1]

# scrivi qui
for i in range(len(emozioni)):
    if grado[i] > 0:
        print(emozioni[i], ': positiva')
    else:
        print(emozioni[i], ': negativa')
```

```
Paura : negativa
Rabbia : negativa
Tristezza : negativa
Gioia : positiva
Disgusto : negativa
Estasi : positiva
```

[27]:
```
emozioni = ['Paura', 'Rabbia','Tristezza','Gioia','Disgusto','Estasi']
grado =     [-1,       -1,     -1,          1,     -1,         1]

# scrivi qui
```

```
Paura : negativa
Rabbia : negativa
Tristezza : negativa
Gioia : positiva
Disgusto : negativa
Estasi : positiva
```

Esercizio - organetto

⊛ Data una stringa `s`, scrivi del codice che stampa tutte le sottostringhe ottenibili partendo dalla posizione della lettera `'n'` e che terminino con l'ultimo carattere di `s`.

Esempio - data:

```
s = 'organetto'
```

Stampa:

```
netto
etto
tto
to
o
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
s = 'organetto'

# scrivi qui
for i in range(s.index('n'), len(s)):
    print(s[i:])
```

```
netto
etto
tto
to
o
```

[28]:
```
s = 'organetto'

# scrivi qui
```

```
netto
etto
tto
to
o
```

Esercizio - sghiribizzo

Scrivi del codice che data la stringa `s` stampa tutte le possibili combinazioni di coppie di righe tali per cui una riga inizia con i primi caratteri di `s` e la successiva continua con i restanti caratteri.

Esempio - data

```
s = 'sghiribizzo'
```

stampa:

```
s
 ghiribizzo
sg
  hiribizzo
sgh
   iribizzo
sghi
    ribizzo
sghir
     ibizzo
sghiri
      bizzo
sghirib
       izzo
sghiribi
        zzo
sghiribiz
         zo
sghiribizz
          o
sghiribizzo
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[29]:
```
s = 'sghiribizzo'
# scrivi qui
for i in range(len(s)):
    print(s[:i+1])
    print(' '*i,s[i+1:])
```

```
s
 ghiribizzo
sg
  hiribizzo
sgh
   iribizzo
sghi
    ribizzo
sghir
     ibizzo
sghiri
      bizzo
sghirib
       izzo
sghiribi
        zzo
sghiribiz
         zo
sghiribizz
          o
sghiribizzo
```

[29]:
```
s = 'sghiribizzo'
# scrivi qui
```

```
s
 ghiribizzo
sg
  hiribizzo
sgh
   iribizzo
sghi
    ribizzo
sghir
     ibizzo
sghiri
      bizzo
sghirib
       izzo
sghiribi
        zzo
sghiribiz
         zo
sghiribizz
          o
sghiribizzo
```

Esercizio - dna

Date due stringhe di DNA `s1` e `s2` di uguale lunghezza, scrivi del codice che stampa tra la prima e la seconda stringa un'altra stringa costituita da spazi ``` e pipe | in corrispondenza dei caratteri uguali.

- **SUGGERIMENTO**: crea una lista contenente i caratteri spazio o il carattere |, e alla fine converti a stringa usando il metodo `join` delle stringhe (fare così è molto più efficiente che continuare a generare stringhe con l'operatore +)

Esempio - date:

```
s1 = "ATACATATAGGGCCAATTATTATAAGTCAC"
s2 = "CGCCACTTAAGCGCCCTGTATTAAAGTCGC"
```

Stampa:

```
ATACATATAGGGCCAATTATTATAAGTCAC
   ||  || |  |  |   |  ||||| |
CGCCACTTAAGCGCCCTGTATTAAAGTCGC
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[30]:
```
s1 = "ATACATATAGGGCCAATTATTATAAGTCAC"
s2 = "CGCCACTTAAGCGCCCTGTATTAAAGTCGC"
```

(continues on next page)

(continued from previous page)

```
# scrivi qui
lista = []
for i in range(len(s1)):
    if(s1[i] == s2[i]):
        lista.append('|')
    else:
        lista.append(' ')

barre = ''.join(lista)

print(s1)
print(barre)
print(s2)
```

```
ATACATATAGGGCCAATTATTATAAGTCAC
   ||  || |  |  |   |  ||||| |
CGCCACTTAAGCGCCCTGTATTAAAGTCGC
```

[30]:

```
s1 = "ATACATATAGGGCCAATTATTATAAGTCAC"
s2 = "CGCCACTTAAGCGCCCTGTATTAAAGTCGC"

# scrivi qui
```

```
ATACATATAGGGCCAATTATTATAAGTCAC
   ||  || |  |  |   |  ||||| |
CGCCACTTAAGCGCCCTGTATTAAAGTCGC
```

Esercizio - sportello

⊛⊛ Data una stringa `s`, stampa la prima metà di caratteri in minuscolo e la seguente metà in maiuscolo.

- se la stringa è di lunghezza dispari, la prima metà deve avere un carattere *in più* rispetto alla seconda

Esempio - data:

```
s = 'sportello'
```

Il tuo codice deve stampare:

```
s
p
o
r
t
E
L
L
O
```

(nota che 'sportello' ha lunghezza dispari e ci sono *cinque* caratteri nella prima metà e *quattro* nella seconda)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[31]:
```
s = 'sportello'  # sportELLO
#s = 'maglia'     # magLIA

# scrivi qui

if len(s) % 2 == 1:
    mezzo = (len(s) // 2) + 1
else:
    mezzo = (len(s) // 2)

for i in range(mezzo):
    print(s[i])

for i in range(mezzo, len(s)):
    print(s[i].upper())
```

```
s
p
o
r
t
E
L
L
O
```

[31]:
```
s = 'sportello'  # sportELLO
#s = 'maglia'     # magLIA

# scrivi qui
```

```
s
p
o
r
t
E
L
L
O
```

Esercizio - fattoria

⊛⊛ Dato un dizionario `versi` che associa nomi di animali ai versi che fanno, e una lista `stanze` di tuple da 2 elementi contenenti nomi di animali, scrivi del codice che per ogni stanza stampa i versi che si sentono passandoci davanti.

- **NOTA**: le stanze da stampare sono numerate **a partire da 1**

Esempio - dati:

```
versi = {'cane':'Bau!',
         'gatto':'Miao!',
         'mucca':'Muu!',
         'pecora':'Bee!'}

stanze = [('cane', 'pecora'),
          ('gatto','mucca'),
          ('mucca', 'cane')]
```

Stampa:

```
Nella stanza 1 si sentono Bau! e Bee!
Nella stanza 2 si sentono Miao! e Muu!
Nella stanza 3 si sentono Muu! e Bau!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[32]:
```
versi = {'cane':'Bau!',
         'gatto':'Miao!',
         'mucca':'Muu!',
         'pecora':'Bee!'}

stanze = [('cane', 'pecora'),
          ('gatto','mucca'),
          ('mucca', 'cane')]

# scrivi qui

for i in range(len(stanze)):
    stanza = stanze[i]
    print('Nella stanza',i+1,'si sentono',versi[stanza[0]], 'e', versi[stanza[1]])
```

```
Nella stanza 1 si sentono Bau! e Bee!
Nella stanza 2 si sentono Miao! e Muu!
Nella stanza 3 si sentono Muu! e Bau!
```

[32]:
```
versi = {'cane':'Bau!',
         'gatto':'Miao!',
         'mucca':'Muu!',
         'pecora':'Bee!'}

stanze = [('cane', 'pecora'),
          ('gatto','mucca'),
          ('mucca', 'cane')]
```

(continues on next page)

(continued from previous page)

```
# scrivi qui
```

```
Nella stanza 1 si sentono Bau! e Bee!
Nella stanza 2 si sentono Miao! e Muu!
Nella stanza 3 si sentono Muu! e Bau!
```

Esercizio - pokemon

⊛⊛⊛ Data una lista `pokemon` e un numero `g` di gruppi, scrivi del codice per stampare `g` righe mostrando i componenti di ogni gruppo. Forma i gruppi prendendo i pokemon nell'ordine in cui li trovi nella lista.

- **SUGGERIMENTO 1**: per ottenere il numero di componenti in un gruppo dovrai usare la divisione intera `//`
- **SUGGERIMENTO 2**: per stampare i componenti di un gruppo, usa il metodo `join` delle stringhe

Esempio 1 - dati

```
#                0          1           2          3             4          5
pokemon = ['Charizard','Gengar','Arcanine','Bulbasaur','Blaziken','Umbreon',
#                6          7           8          9            10          11
           'Lucario','Gardevoir','Eevee','Dragonite', 'Volcarona', 'Sylveon' ]
g = 3
```

stampa

```
gruppo 1 : Charizard e Gengar e Arcanine e Bulbasaur
gruppo 2 : Blaziken e Umbreon e Lucario e Gardevoir
gruppo 3 : Eevee e Dragonite e Volcarona e Sylveon
```

Esempio 2 - dati

```
#                0          1           2          3             4          5
pokemon = ['Charizard','Gengar','Arcanine','Bulbasaur','Blaziken','Umbreon',
#                6          7           8          9            10          11
           'Lucario','Gardevoir','Eevee','Dragonite', 'Volcarona', 'Sylveon' ]

g = 4
```

stampa

```
gruppo 1 : Charizard e Gengar e Arcanine
gruppo 2 : Bulbasaur e Blaziken e Umbreon
gruppo 3 : Lucario e Gardevoir e Eevee
gruppo 4 : Dragonite e Volcarona e Sylveon
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[33]:
```
#                0          1           2          3             4          5          6
↪       7          8
pokemon = ['Charizard','Gengar','Arcanine','Bulbasaur','Blaziken','Umbreon', 'Lucario
↪','Gardevoir','Eevee',
```

(continues on next page)

(continued from previous page)

```
#                 9          10          11
            'Dragonite', 'Volcarona', 'Sylveon' ]
g = 3
#g = 4

# scrivi qui
k = len(pokemon) // g  # pokemon in un gruppo

for i in range(0, g):
    print('gruppo',i+1,':', ' e '.join(pokemon[i*k:(i+1)*k]))
```

```
gruppo 1 : Charizard e Gengar e Arcanine e Bulbasaur
gruppo 2 : Blaziken e Umbreon e Lucario e Gardevoir
gruppo 3 : Eevee e Dragonite e Volcarona e Sylveon
```

[33]:
```
#                0         1          2         3            4         5         6    ␣
↪       7         8
pokemon = ['Charizard','Gengar','Arcanine','Bulbasaur','Blaziken','Umbreon', 'Lucario
↪','Gardevoir','Eevee',
#                 9          10          11
            'Dragonite', 'Volcarona', 'Sylveon' ]
g = 3
#g = 4

# scrivi qui
```

```
gruppo 1 : Charizard e Gengar e Arcanine e Bulbasaur
gruppo 2 : Blaziken e Umbreon e Lucario e Gardevoir
gruppo 3 : Eevee e Dragonite e Volcarona e Sylveon
```

4.24.6 Modificare durante l'iterazione

Supponi di avere una lista `la` contente caratteri, e ti viene chiesto di duplicare tutti gli elementi, per esempio se hai

```
la = ['a','b','c']
```

dopo il tuo codice, deve risultare

```
>>> print(la)
['a','b','c','a','b','c']
```

Forte delle conoscenze acquisite per l'iterazione, potrebbe venirti l'idea di scrivere qualcosa del genere:

```
for elemento in la:
    la.append(elemento)     # ATTENZIONE !
```

DOMANDA: Vedi forse un problema?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: se *mentre* scorriamo la lista, continuiamo al contempo ad aggiungere pezzi, c'è il rischio concreto che non termineremo mai di esaminare la lista ! Leggi bene quanto segue:

X COMANDAMENTO[216]: **Non aggiungerai o toglierai mai elementi da una sequenza su cui stai iterando con un `for`!**

Abbandonarti in simil tentazioni **produrrebbe comportamenti del tutto imprevedibili** (conosci forse l'espressione *tirare il tappeto da sotto i piedi*?)

E rimuovere? Abbiamo visto che aggiungere è pericoloso, ma lo è anche togliere. Supponi di dover eliminare tutti gli elementi di una lista, potresti essere tentato di scrivere qualcosa del genere:

[34]:
```
lista = ['a','b','c','d','e']

for el in lista:
    lista.remove(el)    # PESSIMA IDEA
```

Guarda bene il codice. Credi che abbiamo rimosso tutto, eh?

[35]:
```
lista
```

[35]:
```
['b', 'd']
```

`O_o'` Il risultato assurdo è legato all'implementazione interna di Python, la mia versione di Python dà questo risultato, la tua potrebbe darne uno completamente diverso. **Quindi attenzione** !

Se proprio devi rimuovere elementi dalla sequenza su cui stai iterando, usa un ciclo while[217] o effettua prima una copia della sequenza originale.

Esercizio - duplica

⊛ Prova a scrivere del codice che MODIFICA una lista `la` duplicandone gli elementi.

- usa un ciclo `for`
- **NON** usare la moltiplicazione di liste

Esempio

```
la = ['a','b','c']
```

dopo il tuo codice, deve risultare

```
>>> la
['a','b','c','a','b','c']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[216] https://it.softpython.org/commandments.html#X-COMANDAMENTO
[217] https://it.softpython.org/control-flow/flow3-while-sol.html

[36]:
```
la = ['a','b','c']

# scrivi qui

for elemento in list(la):  # con list creiamo una *copia* della lista originale, che␣
↪rimane stabile
    la.append(elemento)
print(la)
```

```
['a', 'b', 'c', 'a', 'b', 'c']
```

[36]:
```
la = ['a','b','c']

# scrivi qui
```

```
['a', 'b', 'c', 'a', 'b', 'c']
```

Esercizio - martello

⊛ Data una lista di caratteri `la`, MODIFICA la lista cambiando tutti i caratteri ad indici pari con il carattere `'z'`

Esempio - data:

```
la = ['m', 'a', 'r', 't', 'e', 'l', 'l', 'o']
```

Dopo il tuo codice, deve risultare:

```
>>> print(la)
['z', 'a', 'z', 't', 'z', 'l', 'z', 'o']
```

- **NOTA**: qui *non* stiamo aggiungendo o togliendo celle dalla lista

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[37]:
```
la = ['m', 'a', 'r', 't', 'e', 'l', 'l', 'o']

# scrivi qui
for i in range(len(la)):
    if i % 2 == 0:
        la[i] = 'z'
print(la)
```

```
['z', 'a', 'z', 't', 'z', 'l', 'z', 'o']
```

[37]:
```
la = ['m', 'a', 'r', 't', 'e', 'l', 'l', 'o']

# scrivi qui
```

(continues on next page)

(continued from previous page)

```
['z', 'a', 'z', 't', 'z', 'l', 'z', 'o']
```

Esercizio - Orangutang

⊛⊛ Date due stringhe `sa` e `sb`, scrivi del codice che mette nella stringa `sc` una stringa composta da tutte le lettere in `sa` e `sb` alternate

- se una stringa è più corta dell'altra, in fondo a `sc` metti tutte le lettere restanti dell'altra stringa
- **SUGGERIMENTO**: sebbene sia possibile crescere una scringa un carattere alla volta ad ogni iterazione, ogni volta che lo fai viene creata una nuova stringa (perchè le stringhe sono immutabili). E' quindi più efficiente crescere una lista e convertirla in stringa alla fine.

Esempio - dati:

```
sa,sb = 'gibbone', 'ORANGUTANG'
```

dopo il tuo codice deve risultare:

```
>>> print(sc)
gOiRbAbNoGnUeTANG
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[38]:
```
sa,sb = 'gibbone', 'ORANGUTANG'  # gOiRbAbNoGnUeTANG
#sa,sb = 'crociera', 'BARCA'     # cBrAoRcCiAera
#sa,sb = '', ''                  #

# scrivi qui
temp = []

for i in range(len(sa)):
    temp.append(sa[i])
    if i < len(sb):
        temp.append(sb[i])

if i < len(sb):
    temp.extend(sb[i+1:])

sc = ''.join(temp)
#print(sc)
```

[38]:
```
sa,sb = 'gibbone', 'ORANGUTANG'  # gOiRbAbNoGnUeTANG
#sa,sb = 'crociera', 'BARCA'     # cBrAoRcCiAera
#sa,sb = '', ''                  #

# scrivi qui
```

Esercizio - cesta

⊛⊛⊛ C'è una `cesta` piena di frutta, che rappresentiamo come lista di stringhe. Vogliamo prendere dei frutti e metterli in un piatto, nello stesso ordine in cui li troviamo nella cesta. Bisogna prendere solo i frutti contenuti nell'insieme `preferenze`.

- La `cesta` può contenere duplicati, se sono nelle preferenze devi prenderli tutti
- i frutti vanno presi **nello stesso ordine** in cui sono trovati

Esempio - dati:

```
cesta = ['fragola', 'melone', 'ciliegia', 'anguria', 'mela', 'melone','anguria', 'mela
↪', ]
preferita = {'ciliegia', 'mela', 'fragola'}
piatto = []
```

dopo il tuo codice, deve risultare:

```
>>> print(cesta)
['melone', 'anguria', 'melone', 'anguria']
>>> print(piatto)
['fragola', 'ciliegia', 'mela', 'mela']
```

Si può risolvere il problema in due modi:

- Modo 1 (semplice e consigliabile): crea una lista `nuova_cesta` e infine la assegna alla variabile `cesta`
- Modo 2 (difficile, lento, sconsigliabile ma istruttivo): MODIFICA la lista `cesta` originale, usando il metodo pop[218] e senza mai riassegnare `cesta`, quindi niente righe che iniziano con `cesta =`

Prova a risolvere l'esercizio in entrambi i modi.

ATTENZIONE: Per entrambi i modi, ricordati il sacro X COMANDAMENTO[219]

Non aggiungerai o toglierai mai elementi da una sequenza che stai iterando con un `for`!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[39]:
```
# MODO 1

cesta = ['fragola', 'melone', 'ciliegia', 'anguria', 'mela', 'melone','anguria', 'mela
↪', ]
preferita = {'ciliegia', 'mela', 'fragola'}
piatto = []

# scrivi qui
nuova_cesta = []
for frutto in cesta:
    if frutto in preferita:
        piatto.append(frutto)
    else:
        nuova_cesta.append(frutto)
```

(continues on next page)

[218] https://it.softpython.org/lists/lists3-sol.html#Metodo-pop
[219] https://it.softpython.org/commandments.html#X-COMANDAMENTO

(continued from previous page)

```
cesta = nuova_cesta   # sostituiamo lista originale
print('cesta:',cesta)
print('piatto:',piatto)
```

```
cesta: ['melone', 'anguria', 'melone', 'anguria']
piatto: ['fragola', 'ciliegia', 'mela', 'mela']
```

[39]:
```
# MODO 1

cesta = ['fragola', 'melone', 'ciliegia', 'anguria', 'mela', 'melone','anguria', 'mela
↪', ]
preferita = {'ciliegia', 'mela', 'fragola'}
piatto = []

# scrivi qui
```

```
cesta: ['melone', 'anguria', 'melone', 'anguria']
piatto: ['fragola', 'ciliegia', 'mela', 'mela']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[40]:
```
# MODO 2

cesta = ['fragola', 'melone', 'ciliegia', 'anguria', 'mela', 'melone','anguria', 'mela
↪', ]
preferita = {'ciliegia', 'mela', 'fragola'}
piatto = []

# scrivi qui
copia = list(cesta)
j = 0

# così siamo sicuri di iterare su una sequenza diversa da quella che modifichiamo
for i in range(len(copia)):
    frutto = copia[i]
    if frutto in preferita:
        piatto.append(frutto)
        cesta.pop(j)
    else:
        j += 1

print('cesta:',cesta)
print('piatto:',piatto)
```

```
cesta: ['melone', 'anguria', 'melone', 'anguria']
piatto: ['fragola', 'ciliegia', 'mela', 'mela']
```

[40]:
```
# MODO 2

cesta = ['fragola', 'melone', 'ciliegia', 'anguria', 'mela', 'melone','anguria', 'mela
↪', ]
preferita = {'ciliegia', 'mela', 'fragola'}
piatto = []

# scrivi qui
```

```
cesta: ['melone', 'anguria', 'melone', 'anguria']
piatto: ['fragola', 'ciliegia', 'mela', 'mela']
```

4.24.7 Iterare un insieme

Dato un insieme, possiamo esaminare la sequenza di elementi con un ciclo `for`.

> **ATTENZIONE**: l'ordine di iterazione negli insiemi **non** è prevedibile !
>
> Per capire meglio il perchè, puoi rivedere la guida sugli insiemi[220]

[41]:
```
for elemento in {'questo', 'è', 'un', 'insieme'}:
    print(elemento)
```

```
un
questo
è
insieme
```

Domande - insiemi

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. ```
   s = set()
   s.add('pan')
   s.add('de')
   s.add('mo')
   s.add('nio')
   print(s)
   ```
2. ```
   for x in {'a',12,'34',56,34}[2:4]:
       print(x)
   ```
3. ```
   for x in set(['a']) | set(['b']):
       print(x)
   ```
4. ```
   for x in set(['a']) & set(['b']):
       print(x)
   ```

[220] https://it.softpython.org/sets/sets-sol.html#Creare-un-insieme

4.24.8 Iterare un dizionario

Dato un dizionario, possiamo esaminare la sequenza di chiavi, di valori o entrambi con un ciclo `for`.

> **ATTENZIONE**: l'ordine di iterazione delle chiavi **non** è prevedibile !

Proviamo a scorrere le chiavi:

[42]:
```
diz = {
    'bignè':5,
    'brioche':8,
    'krapfen':2
}
```

[43]:
```
for chiave in diz:
    print('Trovata chiave', chiave)
    print('    con valore', diz[chiave])
```

```
Trovata chiave bignè
    con valore 5
Trovata chiave krapfen
    con valore 2
Trovata chiave brioche
    con valore 8
```

Ad ogni iterazione, alla variabile `chiave` che abbiamo dichiarato viene assegnata una chiave presa dal dizionario, in un ordine che possiamo considerare non prevedibile.

Possiamo anche ottenere direttamente nel `for` sia la chiave che il valore corrispondente con questa notazione

[44]:
```
for chiave, valore in diz.items():
    print('Trovata chiave', chiave)
    print('    con valore', diz[chiave])
```

```
Trovata chiave bignè
    con valore 5
Trovata chiave krapfen
    con valore 2
Trovata chiave brioche
    con valore 8
```

`.items()` ritorna una lista di coppie chiave/valore, e ad ogni iterazione una coppia viene assegnata alle variabili `chiave` e `valore`.

Domande iterazione dizionari

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

> **ATTENZIONE**: Ricordati che l'ordine è IMPOSSIBILE da prevedere, quello che è importante è indovinare tutto quello che verrà stampato

1.
```
for x in {'a':1,'b':2,'c':3}:
    print(x)
```

2.
```
for x in {1:'a',2:'b',3:'c'}:
    print(x)
```

3.
```
diz = {'a':1,'b':2,'c':3}
for x in diz:
    print(x[diz])
```

4.
```
diz = {'a':1,'b':2,'c':3}
for x in diz:
    print(diz[x])
```

5.
```
diz = {'a':1,'b':2,'c':3}
for x in diz:
    if x == 'b':
        print(diz[x])
```

6.
```
for k,v in {1:'a',2:'b',3:'c'}:
    print(k,v)
```

7.
```
for x in {1:'a',2:'b',3:'c'}.values():
    print(x)
```

8.
```
for x in {1:'a',2:'b',3:'c'}.keys():
    print(x)
```

9.
```
for x in {1:'a',2:'b',3:'c'}.items():
    print(x)
```

10.
```
for x,y in {1:'a',2:'b',3:'c'}.items():
    print(x,y)
```

Domande - Sono equivalenti?

Guarda i seguenti frammenti di codice: in ciascuno, vi sono due parti, A e B. In ciascun frammento, cerca di indovinare se la parte A stamperà esattamente quello che stampa il codice nella parte B

- **PRIMA** pensa alla risposta
- **POI** prova ad eseguire

Sono equivalenti ? postin

```
diz = {
    'p':'t',
    'o':'i',
    's':'n',
}

print('A:')
for x in diz.keys():
    print(x)
```

(continues on next page)

(continued from previous page)

```
print('\nB:')
for y in diz:
    print(y)
```

Sono equivalenti ? cortel

```
diz = {
    'c':'t',
    'o':'e',
    'r':'l',
}

print('A:')
for p,q in diz.items():
    print(q)

print('\nB:')
for x in diz.values():
    print(x)
```

Sono equivalenti ? - gel

```
diz = {
    'g':'l',
    'e':'e',
    'l':'g',
}

print('A:')
for x in diz.values():
    print(x)

print('\nB:')
for z in diz.items():
    print(z[0])
```

Sono equivalenti ? - giri

```
diz = {
    'p':'g',
    'e':'i',
    'r':'r',
    'i':'i',
}

print('A:')
for p,q in diz.items():
    if p == q:
        print(p)
```

(continues on next page)

(continued from previous page)

```
print('\nB:')
for x in diz:
    if x == diz[x]:
        print(x)
```

Sono equivalenti? - Trovato

Prima pensa se sono equivalenti, poi verifica con tutti i valori di `k` proposti.

Fai bene attenzione a questo esercizio !

Capire questo vuol dire aver *veramente* capito i dizionari ;-)

```
k = 'p'
#k = 'e'
#k = 'r'
#k = 'z'

diz = {
    'p':'c',
    'e':'h',
    'r':'è',
}

print('A:')
for x in diz:
    if x == k:
        print('Trovato', diz[x])

print('\nB:')
if k in diz:
    print('Trovato', diz[k])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: I due codici sopra riportati sono equivalenti, con una importante differenza: il codice A verrà eseguito in un tempo proporziale alla dimensione di `diz` (perchè deve scorrere tutto il dizionario), invece il codice B verrà sempre eseguito in un tempo costante breve che *non* dipende dalla dimensione di `diz`. Sia il comando `if k in diz` che l'espressione `diz[k]` che recupera il valore corrispondente alla chiave `k` sono estremamente veloci.

ATTENZIONE: assicurati di capire bene questo punto!

In tanti scrivono codice come nella parte A perdendo di fatto la principale utilità dei dizionari che è l'accesso veloce. Finchè i dati sono pochi è facile non accorgersene, ma quando abbiamo megabyte di coppie chiave / valore il tempo perso in cicli inutili si fa sentire! Per ulteriori chiarimenti puoi leggere (o rivedere) la sezione Disordine veloce[221] nel foglio sui dizionari.

[221] https://it.softpython.org/dictionaries/dictionaries2-sol.html#Disordine-veloce

Esercizio - colore di cuori

⊛ Scrivi del codice che dato un dizionario `semi`, per ogni seme stampa il suo colore.

Esempio - dato:

```
semi = {
    'cuori':'rosso',
    'picche':'nero',
    'quadri':'rosso',
    'fiori':'nero'
}
```

stampa:

> **ATTENZIONE**: non ti preoccupare dell'ordine in cui vengono stampati i valori!
>
> Sul tuo computer potresti vedere dei risultati diversi, l'importante è che vengano stampate tutte le righe.

```
Il colore di fiori è nero
Il colore di cuori è rosso
Il colore di picche è nero
Il colore di quadri è rosso
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[45]:
```
semi = {
    'cuori':'rosso',
    'picche':'nero',
    'quadri':'rosso',
    'fiori':'nero'
}

# scrivi qui

for k in semi.keys():
    print('Il colore di', k, 'è', semi[k])
```

```
Il colore di cuori è rosso
Il colore di picche è nero
Il colore di quadri è rosso
Il colore di fiori è nero
```

[45]:
```
semi = {
    'cuori':'rosso',
    'picche':'nero',
    'quadri':'rosso',
    'fiori':'nero'
}

# scrivi qui
```

```
Il colore di cuori è rosso
Il colore di picche è nero
Il colore di quadri è rosso
Il colore di fiori è nero
```

Esercizio - preziosi

⊛ Nel dizionario `preziosi` alcune chiavi sono uguali ai rispettivi valori. Scrivi del codice che trova tali chiavi e le stampa.

Esempio - dato:

```
preziosi = {
    'rubini': 'giada',
    'opali':'topazi',
    'gemme':'gemme',
    'diamanti': 'gemme',
    'rubini':'rubini'
}
```

stampa

```
coppia di elementi uguali: rubini e rubini
coppia di elementi uguali: gemme e gemme
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[46]:
```
preziosi = {
    'rubini': 'giada',
    'opali':'topazi',
    'gemme':'gemme',
    'diamanti': 'gemme',
    'rubini':'rubini'
}

# scrivi qui
for k,v in preziosi.items():
    if k == v:
        print('coppia di elementi uguali:',k, 'e', v)
```

```
coppia di elementi uguali: rubini e rubini
coppia di elementi uguali: gemme e gemme
```

[46]:
```
preziosi = {
    'rubini': 'giada',
    'opali':'topazi',
    'gemme':'gemme',
    'diamanti': 'gemme',
    'rubini':'rubini'
}
```

(continues on next page)

(continued from previous page)

```
# scrivi qui
```

```
coppia di elementi uguali: rubini e rubini
coppia di elementi uguali: gemme e gemme
```

Esercizio - potenze

⊛ Dato un numero `n`, scrvi del codice che crea un NUOVO dizionario `diz` contenente come chiavi i numeri da 1 a `n` INCLUSI, associando ad ogni chiave il suo quadrato.

Esempio - dato:

```
n = 5
```

dopo il tuo codice, deve risultare:

```
>>> print(diz)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[47]:
```
n = 5

# scrivi qui
diz = {}
for i in range(1,n+1):
    diz[i] = i*i

print(diz)
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

[47]:
```
n = 5

# scrivi qui
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Esercizio - fiori

⊛ Data una lista `fiori`, scrivi del codice che crea un NUOVO dizionario `diz` che associa ad ogni fiore `True` se il nome del fiore è scritto tutto in maiuscolo, e `False` altrimenti

- **SUGGERIMENTO**: per verificare se una stringa è tutta in maiuscolo, usa il metodo `.isupper()`

```
fiori = ['girasole','GAROFANO', 'tulipano', 'VIOLA', 'ROSA', 'violetta']
```

stampa (sono in ordine alfabetico perchè stampiamo con `pprint`):

```
>>> from pprint import pprint
>>> pprint(diz)
{'GAROFANO': True,
 'ROSA': True,
 'VIOLA': True,
 'girasole': False,
 'tulipano': False,
 'violetta': False}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[48]:
```
fiori = ['girasole','GAROFANO', 'tulipano', 'VIOLA', 'ROSA', 'violetta']

# scrivi qui

diz = {}
for el in fiori:
    diz[el] = el.isupper()

from pprint import pprint
pprint(diz)
```

```
{'GAROFANO': True,
 'ROSA': True,
 'VIOLA': True,
 'girasole': False,
 'tulipano': False,
 'violetta': False}
```

[48]:
```
fiori = ['girasole','GAROFANO', 'tulipano', 'VIOLA', 'ROSA', 'violetta']

# scrivi qui
```

```
{'GAROFANO': True,
 'ROSA': True,
 'VIOLA': True,
 'girasole': False,
 'tulipano': False,
 'violetta': False}
```

Esercizio - arte

⊛ Un artista ha dipinto una serie di opere con diverse tecniche. Nel dizionario `prezzi` per ciascuna tecnica segna il prezzo. L'artista intende fare una serie di esposizioni, e in ognuna presenterà una particolare tecnica. Supponendo che per ciascuna tecnica abbia prodotto `q` quadri, mostrare quanto guadagnerà in ciascuna esposizione (supponendo venda tutto).

Esempio - dati:

```
q = 20

esposizioni = ['acquerello', 'olio', 'murale', 'tempera', 'carboncino','inchiostro']

prezzi = {'acquerello': 3000,
          'olio':6000,
          'murale': 2000,
          'tempera':4000,
          'carboncino':7000,
          'inchiostro':1000
}
```

Stampa - **questa volta l'ordine conta!!**

```
Guadagni previsti:
  esposizione acquerello : 60000 €
  esposizione olio : 120000 €
  esposizione murale : 40000 €
  esposizione tempera : 80000 €
  esposizione carboncino : 140000 €
  esposizione inchiostro : 20000 €
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[49]:
```
q = 20

esposizioni = ['acquerello', 'olio', 'murale', 'tempera', 'carboncino','inchiostro']

prezzi = {'acquerello': 3000,
          'olio':6000,
          'murale': 2000,
          'tempera':4000,
          'carboncino':7000,
          'inchiostro':1000
}

# scrivi qui

print('Guadagni previsti:')
for i in range(len(esposizioni)):
    tecnica = esposizioni[i]
    print('  esposizione', tecnica, ":", prezzi[tecnica]*q,  '€')
```

```
Guadagni previsti:
  esposizione acquerello : 60000 €
  esposizione olio : 120000 €
  esposizione murale : 40000 €
```

(continues on next page)

(continued from previous page)

```
  esposizione tempera : 80000 €
  esposizione carboncino : 140000 €
  esposizione inchiostro : 20000 €
```

[49]:
```
q = 20

esposizioni = ['acquerello', 'olio', 'murale', 'tempera', 'carboncino','inchiostro']

prezzi = {'acquerello': 3000,
          'olio':6000,
          'murale': 2000,
          'tempera':4000,
          'carboncino':7000,
          'inchiostro':1000
}

# scrivi qui
```

```
Guadagni previsti:
  esposizione acquerello : 60000 €
  esposizione olio : 120000 €
  esposizione murale : 40000 €
  esposizione tempera : 80000 €
  esposizione carboncino : 140000 €
  esposizione inchiostro : 20000 €
```

Esercizio - cartolerie

⊛ Un proprietario di due negozi di cartoleria per riorganizzare il magazzino vuole sapere i materiali in comune disponibili nei negozi che possiede. Dati quindi due dizionari `cartoleria1` e `cartoleria2` che associano oggetti alla loro quantità, scrivi del codice che trova tutte le chiavi in comune e per ciascuna stampa la somma delle quantità trovate.

Esempio - dati:

```
cartoleria1 = {'penne':10,
               'cartelle':20,
               'carta':30,
               'forbici':40}

cartoleria2 = {'penne':80,
               'cartelle':90,
               'goniometri':130,
               'forbici':110,
               'righelli':120,
               }
```

stampa (l'ordine **non** importa):

```
materiali in comune:
   penne : 90
   cartelle : 110
   forbici : 150
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[50]:
```
cartoleria1 = {'penne':10,
        'cartelle':20,
        'carta':30,
        'forbici':40}

cartoleria2 = {'penne':80,
        'cartelle':90,
        'goniometri':130,
        'forbici':110,
        'righelli':120,
        }

# scrivi qui

print('materiali in comune:')
for k in cartoleria1:
    if k in cartoleria2:
        print('  ',k, ':', cartoleria1[k] + cartoleria2[k])
```

```
materiali in comune:
   forbici : 150
   cartelle : 110
   penne : 90
```

[50]:
```
cartoleria1 = {'penne':10,
        'cartelle':20,
        'carta':30,
        'forbici':40}

cartoleria2 = {'penne':80,
        'cartelle':90,
        'goniometri':130,
        'forbici':110,
        'righelli':120,
        }

# scrivi qui
```

```
materiali in comune:
   forbici : 150
   cartelle : 110
   penne : 90
```

Esercizio - legumi

⊛ Un magazzino ha scaffali numerati, ciascuno contenente una quantità di legumi espressa in chili. Rappresentiamo `magazzino` come una lista. E' disponibile anche un `registro` come dizionario che associa ai nomi dei legumi il numero dello scaffale in cui sono contenuti nel magazzino.

Scrivi del codice che data una lista di nomi di legumi, mostra la somma dei chili presenti in magazzino per quei legumi.

Esempio - dati

```
legumi = ['ceci', 'soia']

#            0  1  2  3  4  5
magazzino = [50,90,70,10,20,50]

registro = {'piselli':3,
            'soia':1,
            'lenticchie':5,
            'ceci':4,
            'fave':2,
            'fagioli':0,
}
```

dopo il tuo codice, deve stampare (l'ordine **non** importa):

```
Cerco ceci e soia ...
Trovati 20 kg di ceci
Trovati 90 kg di soia
Totale: 110 kg
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[51]:
```
legumi = ['ceci', 'soia']      # 110
#legumi = ['fagioli', 'fave', 'lenticchie']   # 170

#            0  1  2  3  4  5
magazzino = [50,90,70,10,20,50]

registro = {'piselli':3,
            'soia':1,
            'lenticchie':5,
            'ceci':4,
            'fave':2,
            'fagioli':0,
}

# scrivi qui
print('Cerco', ' e '.join(legumi), '...')
somma = 0
for leg in legumi:
    print('Trovati', magazzino[registro[leg]], 'kg di', leg)
    somma += magazzino[registro[leg]]

print('Totale:',somma, 'kg')
```

```
Cerco ceci e soia …
Trovati 20 kg di ceci
```

(continues on next page)

(continued from previous page)

```
Trovati 90 kg di soia
Totale: 110 kg
```

[51]:
```
legumi = ['ceci', 'soia']      # 110
#legumi = ['fagioli', 'fave', 'lenticchie']    # 170

#             0  1  2  3  4  5
magazzino = [50,90,70,10,20,50]

registro = {'piselli':3,
            'soia':1,
            'lenticchie':5,
            'ceci':4,
            'fave':2,
            'fagioli':0,
}

# scrivi qui
```

```
Cerco ceci e soia …
Trovati 20 kg di ceci
Trovati 90 kg di soia
Totale: 110 kg
```

Esercizio - smog

⊛ Scrivi del codice che dati due dizionari `smog` e `preposizioni` che associano a luoghi rispettivamente valori di smog e preposizioni, stampa tutti i luoghi indicando:

- 'PREPOSIZIONE LUOGO l'inquinamento è eccessivo' se il valore di smog è superiore a 30
- 'PREPOSIZIONE LUOGO l'inquinamento è tollerabile' altrimenti
- **NOTA**: nella stampa la prima lettera delle preposizioni deve essere maiuscola: per trasformare la stringa puoi usare il metodo `.capitalize()`

Esempio - dati:

```
smog = {'strada':40,
        'ciclabile': 20,
        'autostrada': 90,
        'parco': 15,
        'lago':5
       }

preposizioni = {
    'autostrada':'in',
    'ciclabile':'alla',
    'lago':'al',
    'parco':'al',
    'strada':'in',
}
```

stampa (l'ordine **non** importa):

```
In strada l'inquinamento è eccessivo
Al parco l'inquinamento è tollerabile
In autostrada l'inquinamento è eccessivo
Al lago l'inquinamento è tollerabile
Alla ciclabile l'inquinamento è tollerabile
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[52]:
```
smog = {'strada':40,
        'ciclabile': 20,
        'autostrada': 90,
        'parco': 15,
        'lago':5
       }

preposizioni = {
    'autostrada':'in',
    'ciclabile':'alla',
    'lago':'al',
    'parco':'al',
    'strada':'in',
}

# scrivi qui
for x in smog:
    if smog[x] > 30:
        print(preposizioni[x].capitalize(),x, "l'inquinamento è eccessivo")
    else:
        print(preposizioni[x].capitalize(),x,"l'inquinamento è tollerabile")
```

```
In autostrada l'inquinamento è eccessivo
Al parco l'inquinamento è tollerabile
Al lago l'inquinamento è tollerabile
In strada l'inquinamento è eccessivo
Alla ciclabile l'inquinamento è tollerabile
```

[52]:
```
smog = {'strada':40,
        'ciclabile': 20,
        'autostrada': 90,
        'parco': 15,
        'lago':5
       }

preposizioni = {
    'autostrada':'in',
    'ciclabile':'alla',
    'lago':'al',
    'parco':'al',
    'strada':'in',
}

# scrivi qui
```

(continues on next page)

(continued from previous page)

```
In autostrada l'inquinamento è eccessivo
Al parco l'inquinamento è tollerabile
Al lago l'inquinamento è tollerabile
In strada l'inquinamento è eccessivo
Alla ciclabile l'inquinamento è tollerabile
```

Esercizio - sport

⊛⊛ Scrivi del codice che dato un dizionario `sport` in cui persone sono associate allo sport che preferiscono, crea un NUOVO dizionario `conteggi` che associa ad ogni sport il numero di persone che lo preferiscono.

Esempio - dato:

```
diz = {'Gianni':'calcio',
       'Paolo':'tennis',
       'Sara':'pallavolo',
       'Elena':'tennis',
       'Roberto':'calcio',
       'Carla':'calcio',
}
```

Dopo il tuo codice, deve risultare:

```
>>> print(conteggi)
{'calcio': 3, 'pallavolo': 1, 'tennis': 2}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[53]:
```
sport = {'Gianni':'calcio',
         'Paolo':'tennis',
         'Sara':'pallavolo',
         'Elena':'tennis',
         'Roberto':'calcio',
         'Carla':'calcio',
}

# scrivi qui

conteggi = {}

for k,v in diz.items():
    if v in conteggi:
        conteggi[v] += 1
    else:
        conteggi[v] = 1

print(conteggi)
```

```
{False: 3, True: 3}
```

[53]:
```
sport = {'Gianni':'calcio',
         'Paolo':'tennis',
         'Sara':'pallavolo',
         'Elena':'tennis',
         'Roberto':'calcio',
         'Carla':'calcio',
}

# scrivi qui
```

```
{False: 3, True: 3}
```

Modificare un dizionario durante l'iterazione

Supponi di avere un dizionario delle province

```
province = {'tn': 'Trento',
 'mi':'Milano',
 'na':'Napoli',
}
```

e di volerlo MODIFICARE in modo che dopo il tuo codice risultino aggiunte le sigle in maiuscolo:

```
>>> print(province)
{'tn': 'Trento',
 'mi':'Milano',
 'na':'Napoli',
 'TN': 'Trento',
 'MI':'Milano',
 'NA':'Napoli',
}
```

Potrebbe venirti l'idea di scrivere qualcosa del genere:

```
for chiave in province =:
    province[chiave.upper()] = province[chiave]     # ATTENZIONE !
```

DOMANDA: Vedi forse un problema?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: se *mentre* scorriamo il dizionario, continuiamo al contempo ad aggiungere pezzi, c'è il rischio concreto che non termineremo mai di esaminare le chiavi !

Perciò leggi bene quanto segue:

X COMANDAMENTO[222]: **Non aggiungerai o toglierai mai elementi da una sequenza su cui stai iterando con un `for`!**

---

[222] https://it.softpython.org/commandments.html#X-COMANDAMENTO

In questo caso, se proviamo ad eseguire il codice, otterremo un errore esplicito:

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-26-9b20900057e8> in <module>()
----> 1 for chiave in diz:
      2     diz['chiave'.upper()] = diz[chiave]    # ATTENZIONE !

RuntimeError: dictionary changed size during iteration
```

ma in altri casi (come per esempio con le liste) effettuare modifiche **può produrre comportamenti del tutto imprevedibili** (conosci forse l'espressione *tirare il tappeto da sotto i piedi*?)

E rimuovere? Abbiamo visto che aggiungere è pericoloso, ma lo è anche togliere.

Supponiamo di voler togliere qualunque coppia che abbia come valore `'Trento'`

```
province = {
    'tn': 'Trento',
    'mi':'Milano',
    'na':'Napoli',
}
```

affinchè risulti

```
>>> print(province)
{'mi':'Milano',
 'na':'Napoli'}
```

Se proviamo ad eseguire qualcosa del genere per fortuna Python si accorge e ci lancia un'eccezione:

```
province = {
    'tn': 'Trento',
    'mi':'Milano',
    'na':'Napoli',
}

for chiave in province:
    if province[chiave] == 'Trento':
        del province[chiave]    # PESSIMA IDEA
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-23-5df0fd659120> in <module>()
      5     'na':'Napoli'
      6 }
----> 7 for chiave in province:
      8     if province[chiave] == 'Trento':
      9         del province[chiave]    # PESSIMA IDEA

RuntimeError: dictionary changed size during iteration
```

Se proprio devi rimuovere elementi dalla sequenza su cui stai iterando, usa un ciclo while[223] o effettua prima una copia della sequenza originale.

[223] https://it.softpython.org/control-flow/flow3-while-sol.html

Esercizio - zazb

⊛⊛ Scrivere del codice che dato un dizionario `diz` con caratteri come chiavi, MODIFICA il dizionario affinchè gli vengano aggiunte chiavi uguali a quelle esistenti ma precedute dal carattere `'z'` - come valore associato alle nuove chiavi, poni l'intero `10`

Esempio - dato:

```
diz = {
    'a':3,
    'b':8,
    'c':4
}
```

dopo il tuo codice, `diz` dovrebbe risultare MODIFICATO così:

```
>>> diz
{   'a':3,
    'b':8,
    'c':4,
    'za':10,
    'zb':10,
    'zc':10
}
```

DOMANDA: E' forse il caso di scrivere una soluzione come quella qua sotto? Leggi bene questo:

```
diz = {
    'a':3,
    'b':8,
    'c':4
}

for chiave in diz:
    diz['z'+chiave] = 10     # ATTENZIONE !! GUAI IN VISTA !!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Assolutamente no - in questo caso siamo fortunati e otterremo un errore esplicito, in altri potremmo ottenere cicli infiniti o risultati incomprensibili:

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-36-550c4c302120> in <module>()
      5 }
      6
----> 7 for chiave in diz:
      8     diz['z'+chiave] = 10

RuntimeError: dictionary changed size during iteration
```

Fai di meglio: ora prova a scrivere una versione del programma che non abbia questo problema

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a><div class="jupman-sol jupman-sol-code"

style="display:none">

[54]:
```
diz = {
    'a':3,
    'b':8,
    'c':4
}

# scrivi qui

for el in list(diz.keys()):  # list 'fotografa' lo stato delle chiavi
    diz['z'+el] = 10

#diz
```

[54]:
```
diz = {
    'a':3,
    'b':8,
    'c':4
}

# scrivi qui
```

4.24.9 `for` annidati

Quanto già detto in precedenza sul nome delle variabile vale ancor di più per i `for` annidati:

II COMANDAMENTO[224] **Quando inserisci una variabile in un ciclo `for` , questa variabile deve essere nuova**

Se hai definito una variabile in un `for` esterno, non la reintrodurrai in un `for` interno, perchè ciò portebbe gran confusione. Per esempio qua `s` è introdotta sia in quello esterno che in quello interno:

[55]:
```
for s in ['pallavolo', 'tennis', 'calcio', 'nuoto']:

    for s in range(3):  # inferno da debuggare, perdi la s del ciclo for esterno
        print(s)

    print(s)  # stampa 2 invece che uno sport !
```

```
0
1
2
2
0
1
2
2
```

(continues on next page)

[224] https://it.softpython.org/commandments.html#II-COMANDAMENTO

(continued from previous page)

```
0
1
2
2
0
1
2
2
```

Domande - for annidati

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
for y in for x in range(3):
    print(x,y)
```

2.
```
for y in for x in range(2) in range(3):
    print(x,y)
```

3.
```
for y in range(3):
    for x in range(2):
        print(x,y)
```

4.
```
for x in range(2):
    for x in range(3):
        print(x)
    print(x)
```

5.
```
for x in range(2):
    for y in range(3):
        print(x,y)
    print(x,y)
```

6.
```
for x in range(1):
    for y in range(1):
        print(x,y)
```

7.
```
for x in range(2):
    for y in range(3):
        print(x,y)
```

8.
```
la = 'abc'
for x in la:
    for y in la:
        print(x)
```

9.
```
for x in 'ab':
    for y in 'cd':
        print(x,y)
    for y in 'ef':
        print(x,y)
```

10.
```
for x in 'abc':
    for y in 'abc':
        if x == y:
            print(x)
```

11.
```
for x in 'abc':
    for y in 'abc':
        if x != y:
            print(x,y)
```

12.
```
lista = []
for x in 'a':
    for y in 'bc':
        lista.append(x)
        lista.append(y)
print(lista)
```

13.
```
lista = []
for x in 'abc':
    for y in 'de':
        lista.append('z')
print(len(lista))
```

14.
```
c = 1
for x in range(1,4):
    s = ''
    for y in range(1,4):
        s = s + str(c)
        c += 1
    print(s)
```

Esercizio - casting

⊛ Si vuole lanciare una nuova produzione videoculturale italo-nipponica e perciò vengono convocati gli attori per i provini. Il regista vuole provare una scena con tutte le possibili coppie che si possono formare tra attrici e attori. Scrivi del codice che stampa tutte le coppie, mettendo anche messaggi di introduzione.

- **NOTA**: il numero di attori e attrici può essere diverso

Esempio - dati:

```
attrici = ['Licia','Mila']
attori = ['Capitan Harlock', 'Lupin', 'Kenshiro']
```

stampa:

```
Entri in scena Licia !
   Entri in scena Capitan Harlock !
      Licia e Capitan Harlock si preparino... CIAK!
   Grazie Capitan Harlock - il prossimo !
   Entri in scena Lupin !
      Licia e Lupin si preparino... CIAK!
   Grazie Lupin - il prossimo !
   Entri in scena Kenshiro !
      Licia e Kenshiro si preparino... CIAK!
```

(continues on next page)

(continued from previous page)

```
   Grazie Kenshiro - il prossimo !
Grazie Licia - la prossima !
Entri in scena Mila !
   Entri in scena Capitan Harlock !
      Mila e Capitan Harlock si preparino... CIAK!
   Grazie Capitan Harlock - il prossimo !
   Entri in scena Lupin !
      Mila e Lupin si preparino... CIAK!
   Grazie Lupin - il prossimo !
   Entri in scena Kenshiro !
      Mila e Kenshiro si preparino... CIAK!
   Grazie Kenshiro - il prossimo !
Grazie Mila - la prossima !

Fine delle audizioni!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[56]:
```
attrici = ['Licia','Mila']
attori = ['Capitan Harlock', 'Lupin', 'Kenshiro']


# scrivi qui
for attrice in attrici:
    print('Entri in scena', attrice, '!')
    for attore in attori:
        print('   Entri in scena', attore, '!')

        print('     ',attrice, 'e', attore, 'si preparino... CIAK!')
        print('   Grazie', attore, '- il prossimo !')
    print('Grazie', attrice, '- la prossima !')
print()
print('Fine delle audizioni!')
```

```
Entri in scena Licia !
   Entri in scena Capitan Harlock !
      Licia e Capitan Harlock si preparino… CIAK!
   Grazie Capitan Harlock - il prossimo !
   Entri in scena Lupin !
      Licia e Lupin si preparino… CIAK!
   Grazie Lupin - il prossimo !
   Entri in scena Kenshiro !
      Licia e Kenshiro si preparino… CIAK!
   Grazie Kenshiro - il prossimo !
Grazie Licia - la prossima !
Entri in scena Mila !
   Entri in scena Capitan Harlock !
      Mila e Capitan Harlock si preparino… CIAK!
   Grazie Capitan Harlock - il prossimo !
   Entri in scena Lupin !
      Mila e Lupin si preparino… CIAK!
   Grazie Lupin - il prossimo !
   Entri in scena Kenshiro !
      Mila e Kenshiro si preparino… CIAK!
   Grazie Kenshiro - il prossimo !
```

(continues on next page)

(continued from previous page)

```
Grazie Mila - la prossima !

Fine delle audizioni!
```

[56]:
```
attrici = ['Licia','Mila']
attori = ['Capitan Harlock', 'Lupin', 'Kenshiro']


# scrivi qui
```

```
Entri in scena Licia !
   Entri in scena Capitan Harlock !
      Licia e Capitan Harlock si preparino… CIAK!
   Grazie Capitan Harlock - il prossimo !
   Entri in scena Lupin !
      Licia e Lupin si preparino… CIAK!
   Grazie Lupin - il prossimo !
   Entri in scena Kenshiro !
      Licia e Kenshiro si preparino… CIAK!
   Grazie Kenshiro - il prossimo !
Grazie Licia - la prossima !
Entri in scena Mila !
   Entri in scena Capitan Harlock !
      Mila e Capitan Harlock si preparino… CIAK!
   Grazie Capitan Harlock - il prossimo !
   Entri in scena Lupin !
      Mila e Lupin si preparino… CIAK!
   Grazie Lupin - il prossimo !
   Entri in scena Kenshiro !
      Mila e Kenshiro si preparino… CIAK!
   Grazie Kenshiro - il prossimo !
Grazie Mila - la prossima !

Fine delle audizioni!
```

Esercizio - coprire il piano

⊛ Dati gli interi `a` e `b`, scrivi del codice che stampa tutte le coppie possibili di numeri x e y tali che $1 \leq x \leq a$ e $1 \leq y \leq b$

Per esempio, per

```
a,b = 5,3
```

deve stampare

```
1 1
1 2
1 3
2 1
2 2
2 3
```

(continues on next page)

(continued from previous page)

```
3 1
3 2
3 3
4 1
4 2
4 3
5 1
5 2
5 3
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[57]:
```
a,b = 5,3

# scrivi qui
for x in range(1,a+1):
    for y in range(1,b+1):
        print(x,y)
```

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
4 1
4 2
4 3
5 1
5 2
5 3
```

[57]:
```
a,b = 5,3

# scrivi qui
```

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
4 1
4 2
4 3
```

(continues on next page)

(continued from previous page)

```
5 1
5 2
5 3
```

Esercizio - triangolare

⊛ Dato l'intero `a`, scrivi del codice che stampa tutte le coppie possibili di numeri x e y tali che $0 \leq x \leq y < a$

Per esempio, per

```
a = 5
```

deve stampare

```
0 0
0 1
0 2
0 3
0 4
1 1
1 2
1 3
1 4
2 2
2 3
2 4
3 3
3 4
4 4
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[58]:
```
a = 5
# scrivi qui
for x in range(a):
    for y in range(x,a):
        print(x,y)
```

```
0 0
0 1
0 2
0 3
0 4
1 1
1 2
1 3
1 4
2 2
2 3
2 4
3 3
3 4
4 4
```

[58]:
```
a = 5
# scrivi qui
```

```
0 0
0 1
0 2
0 3
0 4
1 1
1 2
1 3
1 4
2 2
2 3
2 4
3 3
3 4
4 4
```

Esercizio - porto

⊛ Scrivi del codice che data una lista `parole` e una lista `lettere`, per ogni parola calcola quante lettere contiene

- vengono contati **SOLO** i caratteri presenti in `lettere`
- stampa il risultato **SOLO** se il numero è maggiore di zero

Esempio - date:

```
parole = ['barca','molo','remo','nassa','vela','strascico']
lettere = ['a','c','s']
```

stampa:

```
barca contiene 2 a
barca contiene 1 c
nassa contiene 2 a
nassa contiene 2 s
vela contiene 1 a
strascico contiene 1 a
strascico contiene 2 c
strascico contiene 2 s
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[59]:
```
parole = ['barca','molo','remo','nassa','vela','strascico']
lettere = ['a','c','s']

# scrivi qui

for x in parole:
    for y in lettere:
```

(continues on next page)

(continued from previous page)

```
        if y in x:
            print(x,'contiene',x.count(y),y)
```

```
barca contiene 2 a
barca contiene 1 c
nassa contiene 2 a
nassa contiene 2 s
vela contiene 1 a
strascico contiene 1 a
strascico contiene 2 c
strascico contiene 2 s
```

[59]:
```
parole = ['barca','molo','remo','nassa','vela','strascico']
lettere = ['a','c','s']

# scrivi qui
```

```
barca contiene 2 a
barca contiene 1 c
nassa contiene 2 a
nassa contiene 2 s
vela contiene 1 a
strascico contiene 1 a
strascico contiene 2 c
strascico contiene 2 s
```

Esercizio - poligoni

⊛⊛ Data una lista `poligoni` con nomi di poligoni ordinati per numero di lati a partire dal triangolo, scrivi del codice che stampa tutte le possibili domande che si possono formare riguardo il numero di lati. Parti da un minimo di 3 lati fino ad un massimo corrispondente al numero di lati dell'ultimo poligono (ricordati che i nomi sono ordinati per numero di lati!)

Esempio - data:

```
#                   0              1              2              3
poligoni = ["Il triangolo","Il quadrato","Il pentagono", "L'esagono"]
```

stampa

```
Il triangolo ha 3 lati? True
Il triangolo ha 4 lati? False
Il triangolo ha 5 lati? False
Il triangolo ha 6 lati? False
Il quadrato ha 3 lati? False
Il quadrato ha 4 lati? True
Il quadrato ha 5 lati? False
Il quadrato ha 6 lati? False
Il pentagono ha 3 lati? False
Il pentagono ha 4 lati? False
Il pentagono ha 5 lati? True
Il pentagono ha 6 lati? False
```

(continues on next page)

(continued from previous page)

```
L'esagono ha 3 lati? False
L'esagono ha 4 lati? False
L'esagono ha 5 lati? False
L'esagono ha 6 lati? True
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[60]:
```
#                 0              1              2              3
poligoni = ["Il triangolo","Il quadrato","Il pentagono", "L'esagono"]

# scrivi qui
for i in range(len(poligoni)):
    for j in range(len(poligoni)):
        print(poligoni[i],'ha', j+3, 'lati?', i+3 == j+3 )
```

```
Il triangolo ha 3 lati? True
Il triangolo ha 4 lati? False
Il triangolo ha 5 lati? False
Il triangolo ha 6 lati? False
Il quadrato ha 3 lati? False
Il quadrato ha 4 lati? True
Il quadrato ha 5 lati? False
Il quadrato ha 6 lati? False
Il pentagono ha 3 lati? False
Il pentagono ha 4 lati? False
Il pentagono ha 5 lati? True
Il pentagono ha 6 lati? False
L'esagono ha 3 lati? False
L'esagono ha 4 lati? False
L'esagono ha 5 lati? False
L'esagono ha 6 lati? True
```

[60]:
```
#                 0              1              2              3
poligoni = ["Il triangolo","Il quadrato","Il pentagono", "L'esagono"]

# scrivi qui
```

```
Il triangolo ha 3 lati? True
Il triangolo ha 4 lati? False
Il triangolo ha 5 lati? False
Il triangolo ha 6 lati? False
Il quadrato ha 3 lati? False
Il quadrato ha 4 lati? True
Il quadrato ha 5 lati? False
Il quadrato ha 6 lati? False
Il pentagono ha 3 lati? False
Il pentagono ha 4 lati? False
Il pentagono ha 5 lati? True
Il pentagono ha 6 lati? False
L'esagono ha 3 lati? False
L'esagono ha 4 lati? False
```

(continues on next page)

(continued from previous page)

```
L'esagono ha 5 lati? False
L'esagono ha 6 lati? True
```

Esercizio - bon jour

⊛⊛⊛ Date due stringhe `sa` e `sb` in minuscolo, scrivi del codice che stampa a turno tutte le lettere di `sa` in maiuscolo seguite da tutte le possibili combinazioni di `sb` che abbiano UNA SOLA lettera maiuscola.

Esempio - date:

```
sa = 'bon'
sb = 'jour'
```

Deve stampare:

```
B Jour
B jOur
B joUr
B jouR
O Jour
O jOur
O joUr
O jouR
N Jour
N jOur
N joUr
N jouR
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[61]:
```
sa = 'bon'
sb = 'jour'

# scrivi qui

for c1 in sa:
    for i in range(len(sb)):
        print(c1.upper() + ' ' + sb[:i] + sb[i].upper() + sb[i+1:])
```

```
B Jour
B jOur
B joUr
B jouR
O Jour
O jOur
O joUr
O jouR
N Jour
N jOur
N joUr
N jouR
```

[61]:
```
sa = 'bon'
sb = 'jour'

# scrivi qui
```

```
B Jour
B jOur
B joUr
B jouR
O Jour
O jOur
O joUr
O jouR
N Jour
N jOur
N joUr
N jouR
```

4.24.10 Comandi `break` e `continue`

Per avere ancora più controllo sull'esecuzione di un ciclo possiamo usare i comandi `break` e `continue` .

NOTA: Cerca di limitarne l'uso!

Quando vi è molto codice nel ciclo è facile 'dimenticarsi' della loro presenza trovandosi con bug difficili da scovare. D'altro canto, in alcuni casi selezionati *possono* rendere il codice più leggibile, quindi come in tutte le cose vanno usati con giudizio.

Terminare con un `break`

Per uscire immediatamente da un ciclo si può usare il comando `break`:

[62]:
```
for x in 'lavato':

    if x == 't':
        print('break, esce dal ciclo!')
        break
        print('Dopo il break')

    print(x)

print('Ciclo finito !')
```

```
l
a
v
a
break, esce dal ciclo!
Ciclo finito !
```

Nota come l'istruzione che stampa `'Dopo il break'` *non* sia stata eseguita.

Proseguire con `continue`

E' possibile portare l'esecuzione immediatamente all'iterazione successiva chiamando `continue`, che salta subito al successivo elemento della sequenza senza eseguire le istruzioni dopo il `continue`.

[63]:
```
i = 1
for x in 'lavato':

    if x == 'a':
        print("continue, salta all'elemento successivo")
        continue
    print(x)
print('Ciclo finito !')
```

```
l
continue, salta all'elemento successivo
v
continue, salta all'elemento successivo
t
o
Ciclo finito !
```

Combinare `break` e `continue`

Proviamo a vedere entrambi in Python Tutor:

[64]:
```
i = 1
for x in 'lavato':
    if x == 'a':
        print("continue, salta all'elemento successivo")
        continue
    if x == 't':
        print('break, esce dal ciclo!')
        break
    print(x)

print('Ciclo finito !')

jupman.pytut()
```

```
l
continue, salta all'elemento successivo
v
continue, salta all'elemento successivo
break, esce dal ciclo!
Ciclo finito !
```

[64]:
```
<IPython.core.display.HTML object>
```

Domande - break e continue

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
for x in ['a','b','c']:
    print(x)
    break
```

2.
```
for x in ['a','b','c']:
    print(x)
    break
    print('GLAM')
```

3.
```
for x in ['a','b','c']:
    print(x)
    break
    break
```

4.
```
for x in ['a','b','c']:
    break
    print(x)
```

5.
```
break
for x in ['a','b','c']:
    print(x)
```

6.
```
for x in ['a','b','c']:
    print(x)
break
```

7.
```
for x in ['a','b','c']:
    continue
    print(x)
```

8.
```
for x in ['a','b','c']:
    print(x)
    continue
```

9.
```
for x in ['a','b','c']:
    print(x)
    continue
    print('BAM')
```

10.
```
continue
for x in ['a','b','c']:
    print(x)
```

11.
```
for x in ['a','b','c']:
    print(x)
continue
```

12.
```
for x in ['a','b','c']:
    break
```

(continues on next page)

(continued from previous page)

```
    1/0
print('BAD KARMA')
```

13.
```
for x in ['a','b','c']:
    1/0
    break
print('BAD KARMA')
```

14.
```
for x in range(8):
    if x < 4:
        continue
    print('ZAM', x)
```

15.
```
for x in range(8):
    if x >= 4:
        break
    print('ZUM', x)
```

16.
```
for x in range(6):
    if x % 2 == 0:
        continue
    print(x)
```

17.
```
for x in ['M','C','M']:
    print(x)
    for y in ['S','P','Q','R']:
        print(y)
        break
```

18.
```
for x in ['M','C','M']:
    print(x)
    break
    for y in ['S','P','Q','R']:
        print(y)
```

19.
```
for x in ['M','C','M']:
    print(x)
    for y in ['S','P','Q','R']:
        print(y)
        continue
```

20.
```
for x in ['M','C','M']:
    print(x)
    continue
    for y in ['S','P','Q','R']:
        print(y)
```

Esercizio - continuamente

⊛ Scrivi del codice che data una stringa `parola`, stampa tutti i caratteri *eccetto* le vocali.

Esempio - data:

```
parola = 'continuamente'
```

stampa

```
c
n
t
n
m
n
t
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[65]:
```
parola = 'continuamente'
#parola='aiuola'

# scrivi qui
for x in parola:
    if x in 'aeiou':
        continue
    else:
        print(x)
```

```
c
n
t
n
m
n
t
```

[65]:
```
parola = 'continuamente'
#parola='aiuola'

# scrivi qui
```

```
c
n
t
n
m
n
t
```

Esercizio - breaking bad

⊛ Scrivi del codice che stampa tutti i caratteri da `stringa` finchè non incontra la stringa `'bad'`.

Esempio - data

```
stringa = 'cascapirillabadgnippobadzarpogno'
```

stampa

```
c
a
s
c
a
p
i
r
i
l
l
a
```

[66]:
```
stringa = 'cascapirillabadgnippobadzarpogno'  # cascapirilla
#stringa = 'sobad'  # 'so'
#stringa = 'bad'  # ''
#stringa = 'badso'  # ''

for i in range(len(stringa)):
    if stringa[i:i+3] == 'bad':
        break
    else:
        print(stringa[i])
```

```
c
a
s
c
a
p
i
r
i
l
l
a
```

Esercizio - punto di rottura

⊛⊛ Data una frase, stampa le parole una per riga *finchè* trova un punto, e in tal caso si ferma.

- **NON** usare `frase.split('.')`, ma split su altri caratteri sono consentite.

Esempio - data:

```
frase = 'Ad un certo punto bisogna fermarsi. Mai oltrepassare il limite.'
```

stampa

```
Ad
un
certo
punto
bisogna
fermarsi
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[67]:
```
frase = 'Ad un certo punto bisogna fermarsi. Mai oltrepassare il limite.'
#frase = "Rispetta l'altolà. Non vorrai farci arrestare?"
#frase = 'Stop.'
#frase = 'No stop'

# scrivi qui

for parola in frase.split():
    if '.' in parola:
        print(parola[:-1])
        break
    else:
        print(parola)
```

```
Ad
un
certo
punto
bisogna
fermarsi
```

[67]:
```
frase = 'Ad un certo punto bisogna fermarsi. Mai oltrepassare il limite.'
#frase = "Rispetta l'altolà. Non vorrai farci arrestare?"
#frase = 'Stop.'
#frase = 'No stop'

# scrivi qui
```

```
Ad
un
certo
punto
bisogna
fermarsi
```

Esercizio - breakdance

⊛⊛ Sei un abile breakdancer e ti viene data della `musica` come lista di suoni. Dovrai fare due balli:

- al primo dovrai ripetere i suoni della musica fino a quando hai incontrato esattmente `3` suoni `'pa'`, al che esclamerai `BREAKDANCE!`.
- al secondo dovrai ripetere i suoni della musica *al contrario* fino a quando hai incontrato esattmente `3` suoni `'pa'`, al che esclamerai `BREAKDANCE!`
- **NON** modificare `musica`, quindi niente `musica.reverse()`

Esempio - data:

```
musica = ['unz','pa','pa','tud','unz','pa','pa','tud','unz','boom','boom','tud']
```

Stampa:

```
unz
pa
pa
tud
unz
pa
BREAKDANCE!

tud
boom
boom
unz
tud
pa
pa
unz
tud
pa
BREAKDANCE!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[68]:
```
musica = ['unz','pa','pa','tud','unz','pa','pa','tud','unz','boom','boom','tud']

# scrivi qui

k = 0
for x in musica:
    print(x)
    if x == 'pa':
        k += 1
    if k == 3:
        print('BREAKDANCE!')
        print()
        break

k = 0
for x in range(len(musica)-1, -1, -1):
    print(musica[x])
```

(continues on next page)

(continued from previous page)

```
    if musica[x] == 'pa':
        k += 1
    if k == 3:
        print('BREAKDANCE!')
        break
```

```
unz
pa
pa
tud
unz
pa
BREAKDANCE!

tud
boom
boom
unz
tud
pa
pa
unz
tud
pa
BREAKDANCE!
```

[68]:
```
musica = ['unz','pa','pa','tud','unz','pa','pa','tud','unz','boom','boom','tud']

# scrivi qui
```

```
unz
pa
pa
tud
unz
pa
BREAKDANCE!

tud
boom
boom
unz
tud
pa
pa
unz
tud
pa
BREAKDANCE!
```

[ ]:

4.25 Controllo di flusso - cicli while

4.25.1 Scarica zip esercizi

Naviga file online[225]

Vediamo come ripetere delle istruzioni eseguendole all'interno dei cicli `while`.

La caratteristica principale del ciclo `while` è che permette di controllare esplicitamente la condizione di fine del ciclo. Tipicamente, si utilizzano questi cicli quando si deve *iterare* su una sequenza la cui dimensione non è nota a priori, varia nel tempo oppure vi sono diverse condizioni che potrebbero determinare la fine del ciclo.

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
control-flow
    flow1-if.ipynb
    flow1-if-sol.ipynb
    flow2-for.ipynb
    flow2-for-sol.ipynb
    flow3-while.ipynb
    flow3-while-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `flow3-while.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

[225] https://github.com/DavidLeoni/softpython-it/tree/master/control-flow

4.25.2 Introduzione

Un ciclo `while` è un blocco di codice che viene eseguito quando si verifica una certa condizione booleana. Il blocco di codice viene ripetutamente eseguito fintantochè la condizione è vera.

Vediamo un esempio:

[2]:
```
i = 1

while i < 4:
    print('Ho contato fino a', i)
    i += 1

print('Ciclo finito !')
```

```
Ho contato fino a 1
Ho contato fino a 2
Ho contato fino a 3
Ciclo finito !
```

Nell'esempio, la condizione booleana è

```
i < 4
```

mentre il blocco di codice da eseguire ripetutamente è

```
print('Ho contato fino a', i)
i += 1
```

Come in tutti blocchi di codice Python, il blocco va indentato con gli spazi (di solito 4).

Guarda meglio l'esecuzione in Python Tutor e leggi il commento che segue.

[3]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[4]:
```
i = 1
while i < 4:
    print('Ho contato fino a', i)
    i += 1

print('Ciclo finito !')

jupman.pytut()
```

```
Ho contato fino a 1
Ho contato fino a 2
Ho contato fino a 3
Ciclo finito !
```

[4]:
```
<IPython.core.display.HTML object>
```

Nell'esempio abbiamo usato una variabile che abbiamo chiamato `i` e l'abbiamo inizializzata a zero.

All'inizio del ciclo `i` vale `1`, perciò l'espressione booleana `i < 4` viene valutata come `True`. Dato che è `True`, l'esecuzione procede all'interno del blocco con la `print` e infine MODIFICA `i` con l'incremento `i += 1`.

A questo punto l'esecuzione riprende alla riga del `while`, e la condizione `i < 4` viene valutata di nuovo. A questa seconda iterazione `i` vale `2` perciò l'espressione booleana `i < 4` viene ancora valutata a `True` e l'esecuzione rimane all'interno del blocco. Di nuovo, viene fatta la stampa con la `print` e incrementata `i`.

Viene fatto ancora un'altro ciclo finchè `i` vale 4. A quel punto `i < 4` produce `False` e in quel momento l'esecuzione *esce* dal blocco `while` e prosegue con i comandi allo stesso livello di indentazione del `while`

`while` che terminano

Quando abbiamo un ciclo `while`, tipicamente vogliamo che prima o poi termini (i programmi che 'si impallano' non sono molto graditi agli utenti ...). Per garantire la terminazione, abbiamo bisogno di:

1. inizializzare una variabile all'esterno del ciclo
2. una condizione dopo la scritta `while` che valuta quella variabile (e opzionalmente altro)
3. almeno una istruzione nel blocco interno che MODIFICA la variabile, portandola prima o poi a soddisfare la condizione 2

Se uno qualsiasi di questi punti viene omesso, avremo problemi. Proviamo di proposito a dimenticarci di rispettarli:

Errore 1: omettere l'inizializzazione. Come in tutti i casi in Python in cui ci si è dimenticati di inizializzare una variabile (proviamo in questo caso `j`), l'esecuzione si interrompe non appena si cerca di usare la variabile:

```
print('Sto per entrare nel ciclo ..')
while j < 4:
    print('Ho contato fino a', j)
    j += 1

print('Ciclo finito !')
```

```
Sto per entrare nel ciclo ..
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-277-3f311955204d> in <module>()
      1 print('Sto per entrare nel ciclo ..')
----> 2 while j < 4:
      3     print('Ho contato fino a', j)
      4     j += 1
      5

NameError: name 'j' is not defined
```

Errore 2: omettere di usare la variabile nella condizione. Se ci si dimentica di valutare la variabile, per esempio usandone per sbaglio un'altra (supponiamo `x`), il ciclo non terminerà mai:

```
i = 1
x = 1
print('Sto per entrare nel ciclo ..')
while x < 4:   # valuta x invece di i
    print('Ho contato fino a', i)
    i += 1

print('Ciclo finito !')
```

```
Sto per entrare nel ciclo ..
Ho contato fino a 1
```

(continues on next page)

(continued from previous page)

```
Ho contato fino a 2
Ho contato fino a 3
Ho contato fino a 4
Ho contato fino a 5
Ho contato fino a 6
.
.
```

Errore 3: Omettere di MODIFICARE la variabile nel blocco interno. Se ci si dimentica di mettere almeno un'istruzione che MODIFICA la variabile usata nella condizione, quando la condizione viene valutata produrrà sempre lo stesso valore booleano `False` impedendo l'uscita dal ciclo:

```
i = 1
print('Sto per entrare nel ciclo ..')
while i < 4:
    print('Ho contato fino a', i)

print('Ciclo finito !')
```

```
Sto per entrare nel ciclo ..
Ho contato fino a 1
Ho contato fino a 1
Ho contato fino a 1
Ho contato fino a 1
Ho contato fino a 1
.
.
```

`while` che non terminano

DOMANDA: Riesci ad immaginare un programma che *non* deve terminare mai?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: se abiti vicino ad una centrale idroelettrica o nucleare, cosa succede se il programma che regola il livello dell'acqua termina ?

Oppure: se sei in aereo e il programma che controlla il flusso di carburante ai motori si blocca improvvisamente è un problema?

Tutti i programmi se scritti bene devono prevedere la terminazione, ma alcuni software vengono eseguiti così a lungo che la terminazione è da considerarsi un evento eccezionale.

Domande

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
i = 0
while i < 3:
print(i)
```

2.
```
k = 0
while k < 5:
    print(k)
    k + 1
```

3.
```
i = 0
while i < 3:
    print(i)
i += 1
```

4.
```
i = 0
while False:
    print(i)
    i += 1
print('Finito !')
```

5.
```
i = 0
while i < 3:
    print(i)
    i += 1
```

6.
```
k = 0
while k < 2
    print(i)
    k += 1
```

7.
```
i = 0
while i < 3:
    print('GAM')
    i = i + 1
```

8.
```
while zanza < 2
    print('ZANZA')
    zanza += 1
```

9.
```
i = 0
while False:
    print(i)
    i = i + 1
print('DARK')
```

10.
```
i = 0
while True:
    print(i)
    i = i + 1
print('LIGHT')
```

11.
```
while 2 + 3:
    print('z')
print('')
```

12.
```
i = 10
while i > 0:
    if i > 5:
        print(i)
        i -= 1
print('WAM')
```

13.
```
i = 10
while i > 0:
    if i > 5:
        print(i)
    i -= 1
print('MAW')
```

14.
```
import random
x = 0
while  x < 7:
    x = random.randint(1,10)
    print(x)

print('LUCK')
```

15.
```
x,y = 0,0
while x + y < 4:
    x += 1
    y += 1
    print(x,y)
```

16.
```
x,y = 0,3
while x < y:
    print(x,y)
    x += 1
    y -= 1
```

Esercizi

Esercizio - stampari

⊛ Scrivi del codice che in un ciclo `while` stampa tutti i numeri dispari da `1` a `k`

- per `k<1` non stampa nulla

Esempio - dati:

```
k = 5
```

dopo il tuo codice deve stampare:

```
1
3
5
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[5]:
```
k = 5   # 1 3 5
#k = 1  # 1
#k = 0  # non stampa

# scrivi qui
i = 1
while i <= k:
    if i % 2 == 1:
        print(i)
    i += 1
```

```
1
3
5
```

[5]:
```
k = 5   # 1 3 5
#k = 1  # 1
#k = 0  # non stampa

# scrivi qui
```

```
1
3
5
```

Esercizio - media

⊛ Scrivi del codice che data una lista `numeri`, calcola la media dei valori della lista usando un `while` e la stampa.

- se la lista è vuota, la media si suppone essere `0.0`
- **NON** usare la funzione `sum`
- NON creare variabili che si chiamano `sum` (violerebbe il V Comandamento[226]: non ridifinerai mai funzioni di sistema)

Esempio - data:

```
numeri = [8,6,5,9]
```

stampa

```
7.0
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[226] https://it.softpython.org/commandments.html#V-COMANDAMENTO

[6]:
```
numeri = [8,6,5,9] # 7.0
#numeri = [3,1,2]  # 2.0
#numeri = []       # 0

# scrivi qui
somma = 0.0
i = 0
while i < len(numeri):
    somma += numeri[i]
    i += 1

if len(numeri) > 0:
    print(somma / len(numeri))
else:
    print(0.0)
```

```
7.0
```

[6]:
```
numeri = [8,6,5,9] # 7.0
#numeri = [3,1,2]  # 2.0
#numeri = []       # 0

# scrivi qui
```

```
7.0
```

4.25.3 Comandi `break` e `continue`

Per avere ancora più controllo sull'esecuzione di un ciclo possiamo usare i comandi `break` e `continue` .

NOTA: Cerca di limitarne l'uso!

Quando vi è molto codice nel ciclo è facile 'dimenticarsi' della loro presenza trovandosi con bug difficili da scovare. D'altro canto, in alcuni casi selezionati *possono* rendere il codice più leggibile, quindi come in tutte le cose vanno usati con giudizio.

Terminare con un `break`

Lo schema visto precedentemente per avere `while` terminanti è quello consigliato, ma se abbiamo una condizione che NON valuta la variabile che incrementiamo (come per esempio l'espressione costante `True`), come alternativa per uscire immediatamente dal ciclo si può usare il comando `break`:

[7]:
```
i = 1
while True:

    print('Ho contato fino a', i)

    if i > 3:
        print('break! Esco dal ciclo!')
```

(continues on next page)

(continued from previous page)

```
        break
        print('Dopo il break')

    i += 1

print('Ciclo finito !')
```

```
Ho contato fino a 1
Ho contato fino a 2
Ho contato fino a 3
Ho contato fino a 4
break! Esco dal ciclo!
Ciclo finito !
```

Nota come `Dopo il break` *non* venga mostrato.

Proseguire con `continue`

E' possibile portare l'esecuzione immediatamente all'iterazione successiva chiamando `continue`, che salta subito alla verifica della condizione senza eseguire le istruzioni dopo il `continue`.

> **ATTENZIONE: i `continue` se usati male possono creare cicli infiniti !**
>
> Quando usi `continue` assicurati che non salti l'istruzione per modificare la varibile usata nella condizione di terminazione (oppure che non salti un `break` necessario per uscire)!

Per evitare problemi qui abbiamo incrementato `i` prima dell'`if` con il `continue`:

[8]:
```
i = 1
while i < 5:
    print('Ho contato fino a', i)

    i += 1

    if i % 2 == 1:
        print('continue, salta alla verifica condizione')
        continue
        print('Dopo il continue')

    print('arrivato in fondo')

print('Ciclo finito !')
```

```
Ho contato fino a 1
arrivato in fondo
Ho contato fino a 2
continue, salta alla verifica condizione
Ho contato fino a 3
arrivato in fondo
Ho contato fino a 4
continue, salta alla verifica condizione
Ciclo finito !
```

Proviamo a combinare `break` e `continue` e vedere il risultato in Python Tutor:

[9]:
```
i = 1
while i < 5:
    print('Ho contato fino a', i)
    if i > 3:
        print('break! Esco dal ciclo!')
        break
        print('Dopo il break')
    i += 1
    if i % 2 == 1:
        print('continue, salta alla verifica condizione')
        continue
        print('Dopo il continue')
    print('arrivato in fondo')

print('Ciclo finito !')

jupman.pytut()
```

```
Ho contato fino a 1
arrivato in fondo
Ho contato fino a 2
continue, salta alla verifica condizione
Ho contato fino a 3
arrivato in fondo
Ho contato fino a 4
break! Esco dal ciclo!
Ciclo finito !
```

[9]:
```
<IPython.core.display.HTML object>
```

Domande su `break` e `continue`

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1.
```
i = 1
while i < 4:
    print('Ho contato fino a', i)
    i += 1
    continue

print('Ciclo finito !')
```

2.
```
i = 1
while i < 4:
    print('Ho contato fino a', i)
    continue
    i += 1

print('Ciclo finito !')
```

3.
```
i = 3
while i > 0:
    print('Ho contato fino a', i)
    if i == 2:
        print('continue, salta alla verifica condizione')
```

(continues on next page)

(continued from previous page)

```
        continue
    i -= 1
    print('arrivato in fondo')

print('Ciclo finito !')
```

4.
```
i = 0
while True:
    i += 1
    print(i)
    if i > 3:
        break

print('BONG')
```

5.
```
i = 0
while True:
    if i < 3:
        continue
    else:
        break
    i += 1

print('ZONG')
```

6.
```
i = 0
while True:
    i += 1
    if i < 3:
        continue
    else:
        break

print('ZANG')
```

Esercizio - cercacar

⊛⊛ Scrivi del codice che usando un `while` che cerca nella lista di caratteri `la` il carattere indicato dalla variabile `car` . Nel momento in cui trova la PRIMA occorrenza del carattere, si ferma e stampa l'indice in cui è stato trovato.

- se non trova il carattere, stampa che la ricerca è andata a vuoto.

Esempio 1 - dati:

```
car = 'z'
la = ['b','a','f','g','z','h','z','r']
```

dopo il tuo codice deve stampare

```
Trovato il primo z all'indice 4
```

Esempio 2 - dati:

```
car = 'z'
la = ['b','a','f','g','h','r']
```

deve stampare

```
Non ho trovato z
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[10]:
```
# scrivi qui

car = 'z'

#      0   1   2   3   4   5   6   7
la = ['b','a','f','g','z','h','z','r']   # Trovato il primo z all'indice 4
#la = ['b','a','f','g','h','r']          # Non ho trovato z


# scrivi qui
i = 0
while i < len(la):
    if la[i] == car:
        print("Trovato il primo", car, "all'indice", i)
        break
    i += 1
if i == len(la):
    print("Non ho trovato", car)
```

```
Trovato il primo z all'indice 4
```

[10]:
```
# scrivi qui
```

```
Trovato il primo z all'indice 4
```

4.25.4 Domande - Sono equivalenti ?

Guarda i seguenti frammenti di codice: in ciascuno, vi sono due parti, A e B. In ciascun frammento, cerca di indovinare se la parte A stamperà esattamente quello che stampa il codice nella parte B.

- **PRIMA** pensa alla risposta
- **POI** prova ad eseguire

Sono equivalenti? - BORG

```
print('A:')
while True:
    print('BORG')
    break

print('\nB:')
while False:
    pass
print('BORG')
```

Sono equivalenti? - al 3

```
print('A:')
x = 0
while x < 3:
    print(x)
    x += 1

print('\nB:')
x = 1
while x <= 3:
    print(x-1)
    x += 1
```

Sono equivalenti? - che caso

Ricordati che `randint(a, b)` restituisce un intero casuale `N` tale che `a <= N <= b`

```
print('A:')
x = 0
while x < 3:
    x += 1
print(x)

print('\nB:')
x = 0
import random
while x != 3:
    x = random.randint(1,5)
print(x)
```

Sono equivalenti? - al sei

```
print('A:')
i = 0
while i < 3:
    print(i)
    i += 1
while i < 6:
    print(i)
    i += 1

print('\nB:')
i = 0
while i < 6:
    print(i)
    i += 1
```

Sono equivalenti? - countdown 1

```
print('A:')
i = 2
print(i)
while i > 0:
    i -= 1
    print(i)

print('\nB:')
i = 2
while i > 0:
    print(i)
    i -= 1
```

Sono equivalenti? - countdown 2

```
print('A:')
i = 2
print(i)
while i > 0:
    i -= 1
    print(i)

print('\nB:')
i = 2
while i > 0:
    print(i)
    i -= 1
print(i)
```

Sono equivalenti? - sortilegio

```
print('A:')
s = 'sortilegio'
i = 0
while s[i] != 'g':
    i += 1
print(s[i:])

print('B:')
s = 'sortilegio'
i = len(s)
while s[i] != 'g':
    i -= 1
print(s[i:])
```

Sono equivalenti? - ping pong

```
print('A:')
ping,pong = 0,3
while ping < 3 or pong > 0:
    print(ping,pong)
    ping += 1
    pong -= 1

print('\nB:')
ping,pong = 0,3
while not(ping >= 3 and pong <= 0):
    print(ping,pong)
    ping += 1
    pong -= 1
```

Sono equivalenti? - zanna

```
print('A:')
n,i,s = 0,0,'zanna'
while i < len(s):
    if s[i] == 'n':
        n += 1
    i += 1
print(n)

print('\nB:')
n,i,s = 0,0,'zanna'
while i < len(s):
    i += 1
    if s[i-1] == 'n':
        n += 1
print(n)
```

Sono equivalenti? - pasticcio

```
print('A:')
c,i,s = 0,0,'pasticcio'
while i < len(s):
    if s[i] == 'c':
        c += 1
    i += 1
print(c)

print('\nB:')
no,k,s = 0,0,'pasticcio'
while k < len(s):
    if s[k] != 'c':
        no += 1
    else:
        k += 1
print(len(s) - no)
```

4.25.5 Esercizi su contatori

Esercizio - don't break 1

⊛ Guarda il codice seguente, e scrivi nella cella seguente del codice che produca lo stesso risultato con un `while` e **senza usare** `break`

[11]:
```
x = 3
while True:
    print(x)
    if x == 0:
        break
    x -= 1
```

```
3
2
1
0
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
x = 3

# scrivi qui

while x >= 0:
    print(x)
    x -= 1
```

```
3
2
1
0
```

[12]:
```
x = 3

# scrivi qui
```

```
3
2
1
0
```

Esercizio - don't break 2

⊛ Guarda il codice seguente, e scrivi nella cella seguente del codice che produca lo stesso risultato con un `while` e **senza usare** `break`

[13]:
```
la = [2,3,7,5,6]
k = 7   # 2 3 7
#k = 5  # 2 3 7 5 6
#k = 13 # 2 3 7 5 6

i = 0
while True:
    print(la[i])
    if i >= len(la)-1 or la[i] == k:
        break
    else:
        i += 1
```

```
2
3
7
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[14]:
```
la = [2,3,7,5,6]
k = 7   # 2 3 7
#k = 6  # 2 3 7 5 6
#k = 13 # 2 3 7 5 6

i = 0

# scrivi qui

while i < len(la) and la[i] != k:
    print(la[i])
    i += 1
if i < len(la) and la[i] == k:
    print(la[i])
```

```
2
3
7
```

[14]:
```
la = [2,3,7,5,6]
k = 7   # 2 3 7
#k = 6  # 2 3 7 5 6
#k = 13 # 2 3 7 5 6

i = 0

# scrivi qui
```

```
2
3
7
```

Esercizio - Dammi un break

⊛ Guarda il codice seguente, e scrivi nella cella seguente del codice che produca lo stesso risultato con un `while` **questa volta usando un** `break`

[15]:
```
x,y = 1,5     # (1,5)   (2,4)
#x,y = 2,8    # (2, 8)  (3, 7)  (4, 6)

while x < y or x == 4:
    print((x,y))
    x += 1
    y -= 1
```

```
(1, 5)
(2, 4)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[16]:
```
x,y = 1,5     # (1,5)   (2,4)
#x,y = 2,8    # (2, 8)  (3, 7)  (4, 6)

# scrivi qui
while True:

    if x >= y or x == 4:
        break
    else:
        print((x,y))
        x += 1
        y -= 1

if x < y or x == 4:
    print((x,y))
```

```
(1, 5)
(2, 4)
```

[16]:
```
x,y = 1,5     # (1,5)   (2,4)
#x,y = 2,8    # (2, 8)  (3, 7)  (4, 6)

# scrivi qui
```

```
(1, 5)
(2, 4)
```

Esercizio - cartone

⊛ Stampa numeri interi da `0` a `k` INCLUSI usando un `while`, e ad ogni numero stampa a fianco una tra le stringhe `'CAR'`, `'TO'` e `'NE'` alternandole

Es - per `k=8` stampa

```
0 CAR
1 TO
2 NE
3 CAR
4 TO
5 NE
6 CAR
7 TO
8 NE
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[17]:
```
k = 8

# scrivi qui
x = 0
while x <= k:
    if x % 3 == 0:
        print(x, 'CAR')
    elif x % 3 == 1:
        print(x, 'TO')
    else:
        print(x, 'NE')
    x += 1
```

```
0 CAR
1 TO
2 NE
3 CAR
4 TO
5 NE
6 CAR
7 TO
8 NE
```

[17]:
```
k = 8

# scrivi qui
```

```
0 CAR
1 TO
2 NE
3 CAR
4 TO
5 NE
6 CAR
```

(continues on next page)

(continued from previous page)

```
7 TO
8 NE
```

Esercizio - al dieci

⊛ Dati due numeri `x` e `y`, scrivi del codice con un `while` che stampa i numeri e li incrementa fermandosi non appena uno dei due raggiunge il dieci.

```
x,y = 5,7
```

dopo il tuo codice deve risultare

```
5 7
6 8
7 9
8 10
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
x,y = 5,7
#x,y = 8,4

# scrivi qui
while x <= 10 and y <= 10:
    print(x,y)
    x += 1
    y += 1
```

```
5 7
6 8
7 9
8 10
```

[18]:
```
x,y = 5,7
#x,y = 8,4

# scrivi qui
```

```
5 7
6 8
7 9
8 10
```

Esercizio - cccc

⊛ Scrivi del codice usando un `while` che dato un numero `y`, stampa `y` righe contenti la lettera `c` tante volte quante il numero di riga.

Esempio - dato

```
y = 4
```

Stampa:

```
c
cc
ccc
cccc
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
y = 4

# scrivi qui
x = 0
while x <= y:
    print('c'*x)
    x += 1
```

```
c
cc
ccc
cccc
```

[19]:
```
y = 4

# scrivi qui
```

```
c
cc
ccc
cccc
```

Esercizio - convergi

⊛ Dati due numeri `x` e `k`, usando un `while` modifica di `1` e stampa `x` finchè `x` non raggiunge `k` incluso.

- **NOTA**: `k` può essere sia maggiore che minore di `x`, devi gestire entrambi i casi

Esempio 1 - dato:

```
x,k = 3,5
```

stampa:

```
3
4
5
```

Esempio 2 - dato:

```
x,k = 6,2
```

stampa:

```
6
5
4
3
2
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[20]:
```
x,k = 3,5     # 3 4 5
#x,k = 6,2     # 6 5 4 3 2
#x,k = 4,4    # 4

# scrivi qui

while x != k:
    print(x)
    if x < k:
        x += 1
    else:
        x -= 1
print(x)
```

```
3
4
5
```

[20]:
```
x,k = 3,5     # 3 4 5
#x,k = 6,2     # 6 5 4 3 2
#x,k = 4,4    # 4
```

(continues on next page)

(continued from previous page)

```
# scrivi qui
```

```
3
4
5
```

Esercizio - wow

⊛ Data una lista `la` di stringhe, scrivere del codice che cerca con un `while` all'interno della lista la prima occorrenza di una stringa che inizi con la lettera `w` (per esempio `'wow'`. Appena la trova, il programma si ferma e stampa `Trovata wow`. Altrimenti, stampa `Niente !`.

Esempio 1 - data:

```
la = ['a','d','g','wow','f','wonder','r']
```

Stampa:

```
esaminato a
esaminato d
esaminato g
Trovato wow
```

Esempio 2 - data:

```
la = ['d','v','q','c','e']
```

stampa

```
esaminato d
esaminato v
esaminato q
esaminato c
esaminato e
Niente !
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[21]:
```
la = ['a','d','g','wow','f','wonder','r']     # a d g Trovato wow
#la = ['a','d','g','f','wonder','r', 'woman']  # a d g f Trovato wonder
#la = ['d','v','q','c','e']                    # d v q c e Niente !

# scrivi qui

i = 0
while i < len(la) and la[i][0] != 'w':
    print('esaminato',la[i])
    i += 1

if i < len(la) and la[i][0] == 'w':
    print('Trovato', la[i])
```

(continues on next page)

(continued from previous page)

```
else:
    print('Niente !')
```

```
esaminato a
esaminato d
esaminato g
Trovato wow
```

[21]:
```
la = ['a','d','g','wow','f','wonder','r']     # a d g Trovato wow
#la = ['a','d','g','f','wonder','r', 'woman']  # a d g f Trovato wonder
#la = ['d','v','q','c','e']                    # d v q c e Niente !

# scrivi qui
```

```
esaminato a
esaminato d
esaminato g
Trovato wow
```

Esercizio - Wild West

⊛⊛ I due banditi Carson e Butch hanno sepolto di comune accordo un tesoro nella ridente cittadina di Tombstone, ma adesso ciascuno dei due vuole riprenderselo senza condividere nulla con il compare.

- Per arrivare al tesoro c'è una `strada` da `Santa Fe` fino a `Tombstone` che rappresentiamo come lista di stringhe
- per rappresentare dove sono i banditi nella strada, usiamo due indici `butch` e `carson`
- ciascun bandito parte da una città diversa
- ad ogni turno Carson si sposta di **una** città
- ad ogni turno Butch si sposta di **due** città, perchè dispone di un veloce cavallo Mustang

Scrivi del codice che stampa la corsa e termina non appena uno dei due arriva nell'ultima città, indicando chi ha preso il tesoro.

- Nel caso entrambi i banditi arrivino contemporaneamente nell'ultima città, stampa 'Duello finale a Tombstone!'
- il tuo codice deve funzionare per *qualsiasi* `strada` e posizioni iniziale `carson` e `butch`

Esempio 1 - dati

```
#              0          1           2               3             4                5
strada = ['Santa Fe','Denver','Dodge City', 'Silverton', 'Agua Caliente', 'Tombstone']
carson,butch  = 3, 0
```

deve stampare:

```
Carson parte da Silverton
Butch parte da Santa Fe
Carson raggiunge Agua Caliente
Butch raggiunge Dodge City
Carson raggiunge Tombstone
Butch raggiunge Agua Caliente
```

(continues on next page)

(continued from previous page)

```
Carson ha trovato il tesoro a Tombstone !
```

Esempio 2 - dati

```
strada = ['Santa Fe','Denver','Dodge City', 'Silverton', 'Agua Caliente', 'Tombstone']
carson,butch  = 3, 2
```

deve stampare:

```
Carson parte da Silverton
Butch parte da Dodge City
Carson raggiunge Agua Caliente
Butch raggiunge Agua Caliente
Carson raggiunge Tombstone
Butch raggiunge Tombstone

Duello finale a  Tombstone !
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
#           0          1         2              3              4              5
strada = ['Santa Fe','Denver','Dodge City', 'Silverton', 'Agua Caliente', 'Tombstone']

carson,butch  = 3, 0    #  Carson ha trovato il tesoro a Tombstone !
#carson,butch  = 0, 0   #  Butch ha trovato il tesoro a Tombstone !
#carson,butch  = 3, 2   #  Duello finale a Tombstone !

# scrivi qui

print('Carson parte da', strada[carson])
print('Butch parte da', strada[butch])

while carson < len(strada)-1 and butch < len(strada)-1:
    carson = min(len(strada)-1, carson + 1)
    butch = min(len(strada)-1, butch + 2)
    print('Carson raggiunge', strada[carson])
    print('Butch raggiunge', strada[butch])


print()
if carson == len(strada)-1 and butch == len(strada)-1:
    print('Duello finale a ', strada[-1], '!')
elif carson == len(strada)-1:
    print('Carson ha trovato il tesoro a', strada[-1], '!')
else:
    print('Butch ha trovato il tesoro a', strada[-1], '!')
```

```
Carson parte da Silverton
Butch parte da Santa Fe
Carson raggiunge Agua Caliente
Butch raggiunge Dodge City
Carson raggiunge Tombstone
Butch raggiunge Agua Caliente
```

(continues on next page)

(continued from previous page)

```
Carson ha trovato il tesoro a Tombstone !
```

[22]:
```
#           0          1          2              3              4              5
strada = ['Santa Fe','Denver','Dodge City', 'Silverton', 'Agua Caliente', 'Tombstone']

carson,butch  = 3, 0    #  Carson ha trovato il tesoro a Tombstone !
#carson,butch  = 0, 0   #  Butch ha trovato il tesoro a Tombstone !
#carson,butch  = 3, 2   #  Duello finale a Tombstone !

# scrivi qui
```

```
Carson parte da Silverton
Butch parte da Santa Fe
Carson raggiunge Agua Caliente
Butch raggiunge Dodge City
Carson raggiunge Tombstone
Butch raggiunge Agua Caliente

Carson ha trovato il tesoro a Tombstone !
```

4.25.6 Modificare sequenze

Nel foglio sui cicli `for` abbiamo visto un importante avvertimento, che ripetiamo qua:

X COMANDAMENTO[227]: **Non aggiungerai o toglierai mai elementi da una sequenza che stai iterando con un `for`!**

Abbandonarti in simil tentazioni **produrrebbe comportamenti del tutto imprevedibili** (conosci forse l'espressione volgare *tirare il tappeto da sotto i piedi*?)

Se proprio devi rimuovere elementi dalla sequenza su cui stai iterando, usa un ciclo `while` o effettua prima una copia della sequenza originale.

Nota che l'avviso è solo per i cicli `for`. In caso di necessità in fondo ci suggerisce di adottare come alternativa i `while`. Vediamo quindi quando e come usarli.

Stack - Pescare da mazzo di carte

Supponiamo di avere un mazzo di carte che rappresentiamo come lista di stringhe e vogliamo pescare tutte le carte, leggendole una per una

Possiamo scrivere un `while` che fintanto che il mazzo contiene carte, continua a togliere la carte in cima con il metodo pop[228] e ne stampa il nome. Ricordati che `pop` MODIFICA la lista rimuovendo l'ultimo elemento E restituisce l'elemento come risultato della chiamata, che possiamo quindi salvare in una variabile che in questo caso chiameremo `carta`:

[227] https://it.softpython.org/commandments.html#X-COMANDAMENTO
[228] https://it.softpython.org/lists/lists3-sol.html#Metodo-pop

[23]:
```
mazzo = ['3 cuori',    # <---- in fondo
         '2 picche',
         '9 cuori',
         '5 quadri',
         '8 fiori']    # <---- in cima

while len(mazzo) > 0:
    carta = mazzo.pop()
    print('pescato', carta)

print('Finite le carte !')

jupman.pytut()
```

```
pescato 8 fiori
pescato 5 quadri
pescato 9 cuori
pescato 2 picche
pescato 3 cuori
Finite le carte !
```

[23]:
```
<IPython.core.display.HTML object>
```

Guardando il codice, possiamo notare che:

1. la variabile `mazzo` viene inizializzata
2. si verifica che la dimensione di `mazzo` sia maggiore di zero
3. ad ogni passo la lista `mazzo` viene MODIFICATA riducendone la dimensione
4. ritorna al punto 2

I primi tre punti sono le condizioni che ci garantiscono che il `while` prima o poi terminerà,

Stack - Pescare fino a condizione

Supponiamo adesso di continuare a pescare carte finchè non ne troviamo una di cuori. La situazione è più complicata, perchè adesso il ciclo può terminare in due modi:

1. troviamo cuori, e interrompiamo la ricerca
2. non ci sono carte di cuori, e il mazzo si esaurisce

In ogni caso, alla fine dobbiamo riportare all'utente un risultato. A tal fine, ci risulta comodo inizializzare all'inizio la variabile `carta` come stringa vuota per gestire il caso non vengano trovate carte di cuori (o il mazzo sia vuoto).

Proviamo una prima implementazione che usa un `if` interno che verifica se abbiamo trovato cuori e in tal caso esce con il comando `break`.

- Prova ad eseguire il codice togliendo il commento al secondo mazzo che non ha carte di cuori e guarda la differenza nell'esecuzione.

[24]:
```
mazzo = ['3 cuori','2 picche','9 cuori','5 quadri','8 fiori']
#mazzo = ['8 picche','2 picche','5 quadri','4 fiori']    # niente cuori !
carta = ''
while len(mazzo) > 0:
    carta = mazzo.pop()
    print('pescato', carta)
    if 'cuori' in carta:
```

(continues on next page)

(continued from previous page)

```
        break

if 'cuori' in carta:
    print('Ho trovato cuori!')
else:
    print('Non ho trovato carte di cuori !')

jupman.pytut()
```

```
pescato 8 fiori
pescato 5 quadri
pescato 9 cuori
Ho trovato cuori!
```

[24]:
```
<IPython.core.display.HTML object>
```

Esercizio - Don't break my heart

⊛ Prova a scrivere del codice che risolve lo stesso problema precedente:

- questa volta **NON** usare il `break`
- assicurati che il codice funzioni con un mazzo senza cuori e anche con un mazzo vuoto
- **SUGGERIMENTO**: metti una condizione multipla nel `while`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[25]:
```
mazzo = ['3 cuori','2 picche','9 cuori','5 quadri','8 fiori']
#mazzo = ['8 picche','2 picche','5 quadri','4 fiori']   # niente cuori !
#mazzo = []  # niente cuori !

carta = ''

# scrivi qui

while len(mazzo) > 0 and 'cuori' not in carta:
    carta = mazzo.pop()
    print('pescato', carta)

if 'cuori' in carta:
    print('Ho trovato cuori!')
else:
    print('Non ho trovato carte di cuori !')
```

```
pescato 8 fiori
pescato 5 quadri
pescato 9 cuori
Ho trovato cuori!
```

[25]:
```
mazzo = ['3 cuori','2 picche','9 cuori','5 quadri','8 fiori']
#mazzo = ['8 picche','2 picche','5 quadri','4 fiori']   # niente cuori !
#mazzo = []  # niente cuori !
```

(continues on next page)

(continued from previous page)

```
carta = ''

# scrivi qui
```

```
pescato 8 fiori
pescato 5 quadri
pescato 9 cuori
Ho trovato cuori!
```

Domande - cosa fanno?

DOMANDA: Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. ```
   while []:
       print('z')
   print('BIG')
   ```

2. ```
   while ['a']:
       print('z')
   print('BUG')
   ```

3. ```
   la = []
   while len(la) < 3:
       la.append('x')
   print(la)
   ```

4. ```
   la = ['x','y','z']
   while len(la) > 0:
       print(la.pop())
   ```

5. ```
   la = ['x','y','z']
   while la:
       print(la.pop(0))
   ```

6. ```
   la = [4,5,8,10]
   while la.pop() % 2 == 0:
       print(la)
   ```

Domande - sono equivalenti?

Guarda i seguenti frammenti di codice: in ciascuno, vi sono due parti, A e B. In ciascun frammento, cerca di indovinare se la parte A stamperà esattamente quello che stampa il codice nella parte B.

- **PRIMA** pensa alla risposta
- **POI** prova ad eseguire

Sono equivalenti? - treno

```
print('A:')
la = ['t','r','e','n','o']
while len(la) > 0:
    print(la.pop())

print('\nB:')
la = ['t','r','e','n','o']
la.reverse()
while len(la) > 0:
    print(la.pop(0))
```

Sono equivalenti? - append nx

```
print('A:')
x,n,la = 2,0,[]
while x not in la:
    la.append(n)
    n += 1
print(la)

print('\nB:')
x,la = 2,[]
while len(la) < 3:
    la.append(x)
    x += 1
print(la)
```

4.25.7 Esercizi su stack

Esercizio - break somma

⊛ Guarda il codice seguente, e riscrivilo nella cella seguente come `while` questa volta **usando il comando** `break`

[26]:
```
lista = []
i = 0
k = 10
while sum(lista) < k:
    lista.append(i)
    i += 1
    print(lista)
```

```
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[27]:
```
lista = []
i = 0

# scrivi qui

while True:
    if sum(lista) >= k:
        break
    else:
        lista.append(i)
        i += 1
        print(lista)
```

```
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
```

[27]:
```
lista = []
i = 0

# scrivi qui
```

```
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
```

Esercizio - libri di viaggi

⊛⊛ Supponiamo di aver visitato la soffitta e raccolto una pila di libri, che rappresentiamo come lista di stringhe. Ogni stringa è preceduta da un etichetta di una lettera che indica la categoria (G per Giallo, V per Viaggi, S per Storia)

```
pila = ['S-Medioevo',    # <---- in fondo
        'V-Australia',
        'V-Scozia',
        'G-Sospetti',
        'V-Caraibi']     # <---- in cima
```

Essendo appassionati di libri di viaggi, vogliamo esaminare da `pila` un libro alla volta a partire da quello più in alto, trasferendo in un'altra catasta inizialmente vuota che chiameremo `viaggi` solo i libri che iniziano con l'etichetta `V` come `('V-Australia')`

```
viaggi = []
```

Scrivi del codice che produce la seguente stampa:

```
All'inizio:
    pila:   ['S-Medioevo', 'V-Australia', 'V-Scozia', 'G-Sospetti', 'V-Caraibi']
    viaggi: []
```

(continues on next page)

(continued from previous page)

```
Preso V-Caraibi
    pila:   ['S-Medioevo', 'V-Australia', 'V-Scozia', 'G-Sospetti']
    viaggi: ['V-Caraibi']
Scartato G-Sospetti
    pila:   ['S-Medioevo', 'V-Australia', 'V-Scozia']
    viaggi: ['V-Caraibi']
Preso V-Scozia
    pila:   ['S-Medioevo', 'V-Australia']
    viaggi: ['V-Caraibi', 'V-Scozia']
Preso V-Australia
    pila:   ['S-Medioevo']
    viaggi: ['V-Caraibi', 'V-Scozia', 'V-Australia']
Scartato S-Medioevo
    pila:   []
    viaggi: ['V-Caraibi', 'V-Scozia', 'V-Australia']
```

- I libri non di viaggi non ci interessano e andranno scartati.
- Il tuo codice deve funzionare per *qualunque* lista `viaggi`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
pila = ['S-Medioevo', 'V-Australia', 'V-Scozia', 'G-Sospetti', 'V-Caraibi']

viaggi = []

# scrivi qui
print("All'inizio:")
print('    pila:  ', pila)
print('    viaggi:', viaggi)

while len(pila) > 0:
    libro = pila.pop()
    if libro.startswith('V'):
        print('Preso',libro)
        viaggi.append(libro)
    else:
        print('Scartato', libro)
    print('    pila:  ', pila)
    print('    viaggi:', viaggi)
```

```
All'inizio:
    pila:   ['S-Medioevo', 'V-Australia', 'V-Scozia', 'G-Sospetti', 'V-Caraibi']
    viaggi: []
Preso V-Caraibi
    pila:   ['S-Medioevo', 'V-Australia', 'V-Scozia', 'G-Sospetti']
    viaggi: ['V-Caraibi']
Scartato G-Sospetti
    pila:   ['S-Medioevo', 'V-Australia', 'V-Scozia']
    viaggi: ['V-Caraibi']
Preso V-Scozia
    pila:   ['S-Medioevo', 'V-Australia']
    viaggi: ['V-Caraibi', 'V-Scozia']
Preso V-Australia
    pila:   ['S-Medioevo']
    viaggi: ['V-Caraibi', 'V-Scozia', 'V-Australia']
```

(continues on next page)

(continued from previous page)

```
Scartato S-Medioevo
    pila:   []
    viaggi: ['V-Caraibi', 'V-Scozia', 'V-Australia']
```

[28]:
```
pila = ['S-Medioevo', 'V-Australia', 'V-Scozia', 'G-Sospetti', 'V-Caraibi']

viaggi = []

# scrivi qui
```

```
All'inizio:
    pila:   ['S-Medioevo', 'V-Australia', 'V-Scozia', 'G-Sospetti', 'V-Caraibi']
    viaggi: []
Preso V-Caraibi
    pila:   ['S-Medioevo', 'V-Australia', 'V-Scozia', 'G-Sospetti']
    viaggi: ['V-Caraibi']
Scartato G-Sospetti
    pila:   ['S-Medioevo', 'V-Australia', 'V-Scozia']
    viaggi: ['V-Caraibi']
Preso V-Scozia
    pila:   ['S-Medioevo', 'V-Australia']
    viaggi: ['V-Caraibi', 'V-Scozia']
Preso V-Australia
    pila:   ['S-Medioevo']
    viaggi: ['V-Caraibi', 'V-Scozia', 'V-Australia']
Scartato S-Medioevo
    pila:   []
    viaggi: ['V-Caraibi', 'V-Scozia', 'V-Australia']
```

Esercizio - BANG !

⊛⊛ Ci sono due pile di oggetti `pila_dx` e `pila_sx` che rappresentiamo come liste di stringhe. Un cowboy per passare il tempo decide di sparare agli oggetti in cima alle pile, alternando ad ogni sparo la pila. Il cowboy è abile e fa sempre centro, quindi ad ogni sparo la pila bersaglio decresce.

- Supponi che gli oggetti in alto siano quelli in fondo alla lista
- Per tenere conto di quale pila colpire, usiamo una variabile `sparo` che tiene all'interno il valore `'dx'` oppure `'sx'`
- Dopo ogni sparo il cowboy se possibile cambierà pila, altrimenti continuerà a sparare alla stessa finchè non rimangono più oggetti
- il tuo codice deve funzionare per *qualsiasi* pila e sparo iniziale

Esempio - dati:

```
pila_sx = ['cassa','stivale','ferro di cavallo','secchio']
pila_dx = ['bidone','sella','latta']
sparo = 'dx'
```

dopo il tuo codice, deve stampare

```
BANG! a destra:    latta
   pila_sx: ['cassa', 'stivale', 'ferro di cavallo', 'secchio']
   pila_dx: ['bidone', 'sella']
BANG! a sinistra:  secchio
   pila_sx: ['cassa', 'stivale', 'ferro di cavallo']
   pila_dx: ['bidone', 'sella']
BANG! a destra:    sella
   pila_sx: ['cassa', 'stivale', 'ferro di cavallo']
   pila_dx: ['bidone']
BANG! a sinistra:  ferro di cavallo
   pila_sx: ['cassa', 'stivale']
   pila_dx: ['bidone']
BANG! a destra:    bidone
   pila_sx: ['cassa', 'stivale']
   pila_dx: []
BANG! a sinistra:  stivale
   pila_sx: ['cassa']
   pila_dx: []
   pila_sx: ['cassa']
   pila_dx: []
BANG! a sinistra:  cassa
   pila_sx: []
   pila_dx: []
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[29]:
```
pila_sx = ['cassa','stivale','ferro di cavallo','secchio']
pila_dx = ['bidone','sella','latta']
sparo = 'dx'

# scrivi qui
while len(pila_dx) > 0 or len(pila_sx) > 0:
    if sparo == 'dx':
        if len(pila_dx) > 0:
            print('BANG! a destra:   ', pila_dx.pop())
        sparo = 'sx'
    else:
        if len(pila_sx) > 0:
            print('BANG! a sinistra: ', pila_sx.pop())
        sparo = 'dx'

    print('   pila_sx:', pila_sx)
    print('   pila_dx:', pila_dx)
```

```
BANG! a destra:    latta
   pila_sx: ['cassa', 'stivale', 'ferro di cavallo', 'secchio']
   pila_dx: ['bidone', 'sella']
BANG! a sinistra:  secchio
   pila_sx: ['cassa', 'stivale', 'ferro di cavallo']
   pila_dx: ['bidone', 'sella']
BANG! a destra:    sella
   pila_sx: ['cassa', 'stivale', 'ferro di cavallo']
   pila_dx: ['bidone']
BANG! a sinistra:  ferro di cavallo
   pila_sx: ['cassa', 'stivale']
   pila_dx: ['bidone']
```

(continues on next page)

(continued from previous page)

```
BANG! a destra:    bidone
   pila_sx: ['cassa', 'stivale']
   pila_dx: []
BANG! a sinistra:  stivale
   pila_sx: ['cassa']
   pila_dx: []
   pila_sx: ['cassa']
   pila_dx: []
BANG! a sinistra:  cassa
   pila_sx: []
   pila_dx: []
```

[29]:
```
pila_sx = ['cassa','stivale','ferro di cavallo','secchio']
pila_dx = ['bidone','sella','latta']
sparo = 'dx'

# scrivi qui
```

```
BANG! a destra:    latta
   pila_sx: ['cassa', 'stivale', 'ferro di cavallo', 'secchio']
   pila_dx: ['bidone', 'sella']
BANG! a sinistra:  secchio
   pila_sx: ['cassa', 'stivale', 'ferro di cavallo']
   pila_dx: ['bidone', 'sella']
BANG! a destra:    sella
   pila_sx: ['cassa', 'stivale', 'ferro di cavallo']
   pila_dx: ['bidone']
BANG! a sinistra:  ferro di cavallo
   pila_sx: ['cassa', 'stivale']
   pila_dx: ['bidone']
BANG! a destra:    bidone
   pila_sx: ['cassa', 'stivale']
   pila_dx: []
BANG! a sinistra:  stivale
   pila_sx: ['cassa']
   pila_dx: []
   pila_sx: ['cassa']
   pila_dx: []
BANG! a sinistra:  cassa
   pila_sx: []
   pila_dx: []
```

Esercizio - Crescere o decrescere ?

⊛⊛ Scrivi del codice che data una lista `la`, MODIFICA continuamente la lista secondo questa procedura:

- se l'ultimo elemento è dispari (es `7`), attacca alla fine della lista un nuovo numero ottenuto moltiplicando per due l'ultimo elemento (es. attacca `14`)
- se l'ultimo elemento è pari, toglie gli ultimi due elementi
- **NOTA 1**: vogliamo proprio MODIFICARE la lista originale, NON vogliamo creare una nuova lista (quindi non vi saranno righe che iniziano con `la =`
- **NOTA 2**: quando vogliamo far sia crescere che decrescere la sequenza che stiamo considerando in un ciclo, dobbiamo convincerci per bene che prima o poi la condizione di terminazione si verifichi, è facile sbagliarsi e finire con un ciclo infinito !
- **SUGGERIMENTO**: per far decrescere la lista, puoi usare il metodo pop[229]

Esempio - data:

```
la = [3,5,6,7]
```

eseguendo il tuo codice, deve stampare

```
Dispari: attacco 14
         la diventa [3, 5, 6, 7, 14]
   Pari: tolgo 14
         tolgo 7
         la diventa [3, 5, 6]
   Pari: tolgo 6
         tolgo 5
         la diventa [3]
Dispari: attacco 6
         la diventa [3, 6]
   Pari: tolgo 6
         tolgo 3
         la diventa []
Finito! la è []
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[30]:
```
la = [3,5,6,7]

# scrivi qui

i = 0
while len(la) > 0:
    if la[-1] % 2 == 1:
        nuovo = la[-1]*2
        la.append(nuovo)
        print('Dispari: attacco', nuovo)
    else:
        print('   Pari: tolgo',la.pop())
        print('         tolgo',la.pop())
    print('         la diventa', la)
    i += 1
```

(continues on next page)

[229] https://it.softpython.org/lists/lists3-sol.html#Metodo-pop

(continued from previous page)

```
print('Finito! la è', la)
```

```
Dispari: attacco 14
         la diventa [3, 5, 6, 7, 14]
   Pari: tolgo 14
         tolgo 7
         la diventa [3, 5, 6]
   Pari: tolgo 6
         tolgo 5
         la diventa [3]
Dispari: attacco 6
         la diventa [3, 6]
   Pari: tolgo 6
         tolgo 3
         la diventa []
Finito! la è []
```

[30]:
```
la = [3,5,6,7]

# scrivi qui
```

```
Dispari: attacco 14
         la diventa [3, 5, 6, 7, 14]
   Pari: tolgo 14
         tolgo 7
         la diventa [3, 5, 6]
   Pari: tolgo 6
         tolgo 5
         la diventa [3]
Dispari: attacco 6
         la diventa [3, 6]
   Pari: tolgo 6
         tolgo 3
         la diventa []
Finito! la è []
```

4.26 Sequenze e comprehensions

4.26.1 Scarica zip esercizi

Naviga file online[230]

In Python è possibile scrivere codice elegante e compatto con le sequenze. Prima vedremo come scorrere le sequenze con gli iteratori e poi come costruirle con le comprehensions di liste, insiemi e dizionari.

[230] https://github.com/DavidLeoni/softpython-it/tree/master/sequences

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
sequences
    sequences.ipynb
    sequences-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `sequences.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

4.26.2 Iterabili - liste

Guardando i cicli abbiamo spesso parlato di *iteratazione* su sequenze, ma nel dettaglio, cosa vuol dire esattamente che una sequenza è *iterabile* ? In concreto, vuol dire che è possibile chiamare la funzione `iter` sulla sequenza.

Proviamo per esempio sulle familiari liste:

[2]:
```
iter(['a','b','c','d'])
```

[2]:
```
<list_iterator at 0x7efd24680b38>
```

Notiamo che Python ha creato un oggetto di tipo `list_iterator`.

NOTA: la lista non viene mostrata!

Puoi immaginare un iteratore come una specie di macchinetta ferma, che ogni volta che viene azionata produce un elemento di una sequenza, uno alla volta.

Tipicamente, un iteratore sa solo in quale *posizione* sta all'interno di una sequenza, e può fornirci gli elementi della sequenza uno per uno se li chiediamo ripetutamente con la funzione `next`:

[3]:
```
iteratore = iter(['a','b','c','d'])
```

[4]:
```
next(iteratore)
```

[4]:
```
'a'
```

```
[5]: next(iteratore)
[5]: 'b'
```

```
[6]: next(iteratore)
[6]: 'c'
```

```
[7]: next(iteratore)
[7]: 'd'
```

Nota come l'iteratore abbia *uno stato* per tenere traccia di dove siamo nella sequenza (in inglese diremmo che è *stateful*). Lo stato viene cambiato ad ogni chiamata della funzione `next`.

Se proviamo a chiedere più elementi di quanti ve ne sono disponibili, Python solleva un'eccezione `StopIteration`:

```
next(iteratore)

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-65-4518bd5da67f> in <module>()
----> 1 next(iteratore)

StopIteration:
```

4.26.3 iterabili - range

Abbiamo provato ad iterare su una lista, che è una sequenza completamente materializzata in memoria su cui scorriamo con l'oggetto iteratore. Esistono anche sequenze particolari che *non* sono materializzate in memoria, come per esempio `range`.

Precedentemente abbiamo usato `range` nei cicli for[231] for ottenere una sequenza di numeri, ma esattamente, che cosa fa la funzione `range`? Proviamo a chiamarla da sola:

```
[8]: range(4)
[8]: range(0, 4)
```

Forse ci attendevamo una sequenza di numeri, invece Python ci mostra un oggetto di tipo `range` (indicando anche l'estremo inferiore del range).

> **NOTA**: Al momento in memoria non è presente nessuna sequenza di numeri
>
> Abbiamo solo un oggetto *iterabile* 'fermo' che se vogliamo può fornirci i numeri

Come possiamo chiederglieli?

Abbiamo già visto che si può usare un `for`:

```
[9]: for x in range(4):
         print(x)
```

[231] https://it.softpython.org/control-flow/flow2-for-sol.html#Contare-con-range

```
0
1
2
3
```

In alternativa, possiamo passare `range` alla funzione `iter` che produce un *iteratore*.

> **ATTENZIONE:** `range` **è iterabile ma NON è un iteratore !!**
>
> Per ottenere l'iteratore dobbiamo chiamare la funzione `iter` sull'oggetto `range`

[10]:
```
iteratore = iter(range(4))
```

Anche `iter` produce un oggetto 'fermo', che non ha ancora materializzato in memoria dei numeri:

[11]:
```
iteratore
```

[11]:
```
<range_iterator at 0x7efd245e7db0>
```

Per chiederglieli dobbiamo usare la funzione `next`:

[12]:
```
next(iteratore)
```

[12]:
```
0
```

[13]:
```
next(iteratore)
```

[13]:
```
1
```

[14]:
```
next(iteratore)
```

[14]:
```
2
```

[15]:
```
next(iteratore)
```

[15]:
```
3
```

Nota come l'iteratore abbia *uno stato*, che viene cambiato ad ogni chiamata di `next` per tenere traccia di dove siamo nella sequenza.

Se proviamo a chiedere più elementi di quanti ve ne sono disponibili, Python solleva un'eccezione `StopIteration`:

```
next(iteratore)

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-65-4518bd5da67f> in <module>()
----> 1 next(iteratore)

StopIteration:
```

4.26.4 Materializzare una sequenza

Abbiamo detto che un oggetto `range` non materializza fisicamente in memoria tutti i numeri contemporaneamente. Solo ottenendo l'iteratore, li materializziamo uno per uno. E volessimo una lista con tutti i numeri ? Nel foglio sulle liste[232] abbiamo visto che passando alla funzione `list` una sequenza, viene creata una nuova lista che contiene tutti gli elementi della sequenza. Abbiamo parlato genericamente di *sequenza*, ma il termine più corretto sarebbe stato *iterabile*.

Se passiamo a `list` un qualunque oggetto *iterabile* allora verrà costruita la lista - come abbiamo visto `range` è iterabile quindi proviamo:

[16]:
```
list(range(4))
```

[16]:
```
[0, 1, 2, 3]
```

Voilà ! Adesso la sequenza è tutta fisicamente presente in memoria.

ATTENZIONE: `list` **consuma l'iteratore!**

Se provi a chiamare due volte `list` sullo stesso iteratore, otterrai una lista vuota:

[17]:
```
sequenza = range(4)
iteratore = iter(sequenza)
```

[18]:
```
nuova1 = list(iteratore)
```

[19]:
```
nuova1
```

[19]:
```
[0, 1, 2, 3]
```

[20]:
```
nuova2 = list(iteratore)
```

[21]:
```
nuova2
```

[21]:
```
[]
```

E se volessimo accedere direttamente ad una posizione specifica della sequenza generata dall'iteratore? Proviamo ad estrarre direttamente la lettera ad indice 2 qui:

[22]:
```
sequenza = range(4)
iteratore = iter(sequenza)
```

```
iteratore[2]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-129-3c080cc9e700> in <module>()
      1 sequenza = range(4)
      2 iteratore = iter(sequenza)
----> 3 iteratore[3]

TypeError: 'range_iterator' object is not subscriptable
```

… purtroppo ci becchiamo un errore.

Non ci restano che due alternative:

[232] https://it.softpython.org/lists/lists1-sol.html#Convertire-sequenze-in-liste

a) prima convertiamo a lista e poi usiamo le quadre

b) chiamiamo 4 volte `next` (ricordati che gli indici partono da zero)

L'opzione a) spesso sembra comoda, ma fai attenzione: **convertire a lista un'iteratore crea una NUOVA lista in memoria**, se la lista di partenza è molto grande e/o si fa questa operazione molte volte si rischia di occupare memoria per niente.

Rivediamo l'esempio in Python Tutor:

[23]:
```
# AFFINCHE' PYTHON TUTOR FUNZIONI, RICORDATI DI ESEGUIRE QUESTA CELLA con Shift+Invio
#   (basta eseguirla una volta sola, la trovi anche all'inizio di ogni foglio)

import jupman
```

[24]:
```
sequenza = range(4)
iteratore = iter(sequenza)
nuova1 = list(iteratore)
nuova2 = list(iteratore)

jupman.pytut()
```

[24]:
```
<IPython.core.display.HTML object>
```

DOMANDA: Occupa più memoria l'oggetto `a` o l'oggetto `b` ?

```
a = range(10)
b = range(10000000)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: entrambi occupano la stessa quantità di memoria

DOMANDA: Occupa più memoria l'oggetto `a` o l'oggetto `b` ?

```
a = list(range(10))
b = list(range(10000000))
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: occupa di più `b` (la lista viene materializzata)

Domande `range`

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. `range(3)`
2. `range()`
3. `list(range(-3))`
4. `range(3,6)`
5. `list(range(5,4))`
6. `list(range(3,3))`
7. `range(3) + range(6)`
8. `list(range(3)) + list(range(6))`
9. `list(range(0,6,2))`
10. `list(range(9,6,-1))`

4.26.5 reversed

`reversed` è una *funzione* che prende come parametro una sequenza e PRODUCE un NUOVO *iteratore* che consente di scorrere la sequenza in ordine inverso.

> **ATTENZIONE**: chiamando `reversed` si ottiene direttamente un *iteratore* !
>
> Quindi *non* serve fare ulteriori chiamate a `iter` come fatto per `range`!

Vediamo meglio cosa vuol dire con un esempio:

```
[25]: la = ['p','r','o','v','a']
```

```
[26]: reversed(la)
[26]: <list_reverseiterator at 0x7efd244a7a58>
```

Vediamo che `reversed` ha prodotto un risultato che è un *iteratore* (non una lista rovesciata).

INFO: gli iteratori occupano poca memoria

Creare un iteratore da una sequenza crea solo una specie di puntatore, *non* crea nuove regioni di memoria.

Inoltre, vediamo che la lista originale associata a `la` *non* è cambiata:

[27]:
```
print(la)
```

```
['p', 'r', 'o', 'v', 'a']
```

ATTENZIONE: la funzione `reversed` è diversa dal metodo reverse[233]

Nota la **d** finale! Se provassiamo a chiamarla come un metodo otterremmo un errore:

```
>>> la.reversed()

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-182-c8d1eec57fdd> in <module>
----> 1 la.reversed()

AttributeError: 'list' object has no attribute 'reversed'
```

iterare con `next`

Come facciamo ad ottenere in memoria una lista rovesciata? In altre parole, come facciamo ad azionare la macchinetta iteratore?

Possiamo chiedere all'iteratore un elemento alla volta con la funzione `next`:

[28]:
```
la = ['a','b','c']
```

[29]:
```
iteratore = reversed(la)
```

[30]:
```
next(iteratore)
```

[30]:
```
'c'
```

[31]:
```
next(iteratore)
```

[31]:
```
'b'
```

[32]:
```
next(iteratore)
```

[32]:
```
'a'
```

Una volta esaurito l'iteratore, chiamando ancora `next` otterremo un errore:

```
next(iteratore)

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-248-4518bd5da67f> in <module>
----> 1 next(iteratore)

StopIteration:
```

Proviamo a crearci manualmente una lista di destinazione `lb` e aggiungere gli elementi che otteniamo mano a mano:

[233] https://it.softpython.org/lists/lists3-sol.html#Metodo-reverse

[33]:
```
la = ['a','b','c']
iteratore = reversed(la)
lb = []
lb.append(next(iteratore))
lb.append(next(iteratore))
lb.append(next(iteratore))
print(lb)

jupman.pytut()
```

```
['c', 'b', 'a']
```

[33]:
```
<IPython.core.display.HTML object>
```

Esercizio - sconcerto

Scrivi del codice che data una lista di caratteri `la`, mette in una lista `lb` tutti i caratteri in posizione dispari presi della lista `la` rovesciata

- usa `reversed` e `next`
- **NON** modificare `la`
- **NON** usare indici negativi
- **NON** usare `list`

Esempio - data

```
#       8    7    6    5    4    3    2    1    0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []
```

Dopo il tuo codice deve mostrare

```
>>> print(lb)
['t', 'e', 'n', 'c']
>>> print(la)
['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
```

Ti invitiamo a risolvere il problema in diversi modi:

MODO 1 - senza ciclo: Supponi che la lunghezza della lista **sia fissa**, e chiama ripetutamente `next` *senza usare un ciclo*

MODO 2 - while: Supponi di avere una lista di lunghezza arbitraria, e prova a generalizzare il precedente codice *usando un ciclo* `while` al cui interno chiamerai `next`

- **SUGGERIMENTO 1**: tieni traccia della posizione a cui sei con un contatore `i`
- **SUGGERIMENTO 2**: non si può chiamare `len` su un iteratore, quindi nella condizione del while dovrai usare la lunghezza della lista originale

MODO 3 - for: questo è il modo più elegante. Supponi di avere una lista di lunghezza arbitraria e *usa un ciclo* `for x in reversed(la)`

- **SUGGERIMENTO**: dovrai comunque tener traccia della posizione a cui sei con un contatore `i`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[34]:
```
# MODO 1 MANUALE

#      8    7    6    5    4    3    2    1    0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

# scrivi qui

iteratore = reversed(la)

next(iteratore)
lb.append(next(iteratore))
next(iteratore)
lb.append(next(iteratore))
next(iteratore)
lb.append(next(iteratore))
next(iteratore)
lb.append(next(iteratore))
print(lb)

#jupman.pytut()
```

```
['t', 'e', 'n', 'c']
```

[34]:
```
# MODO 1 MANUALE

#      8    7    6    5    4    3    2    1    0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

# scrivi qui



```

```
['t', 'e', 'n', 'c']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[35]:
```
# MODO 2 CON IL WHILE

#      8    7    6    5    4    3    2    1    0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

# scrivi qui

iteratore = reversed(la)

i = 1
while i < len(la):
    if i % 2 == 1:
        next(iteratore)
        lb.append(next(iteratore))
    i += 2
```

(continues on next page)

(continued from previous page)

```
print(lb)
```

```
['t', 'e', 'n', 'c']
```

[35]:
```
# MODO 2 CON IL WHILE

#       8    7    6    5    4    3    2    1    0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

# scrivi qui
```

```
['t', 'e', 'n', 'c']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[36]:
```
# MODO 3: for

#       8    7    6    5    4    3    2    1    0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

# scrivi qui
i = 0

for x in reversed(la):
    if i % 2 == 1:
        lb.append(x)
    i += 1
print(lb)
```

```
['t', 'e', 'n', 'c']
```

[36]:
```
# MODO 3: for

#       8    7    6    5    4    3    2    1    0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

# scrivi qui
```

```
['t', 'e', 'n', 'c']
```

Materializzare un iteratore

Per fortuna per ottenere subito una lista dall'iteratore c'è un modo meno laborioso.

Abbiamo visto che quando vogliamo creare una nuova lista a partire da una sequenza, possiamo usare `list` come se fosse una funzione. Possiamo farlo anche in questo caso, interpretando l'iteratore come se fosse una sequenza:

[37]:
```
la = ['p', 'r', 'o', 'v', 'a']
list( reversed(la) )
```

[37]:
```
['a', 'v', 'o', 'r', 'p']
```

Nota bene abbiamo generato una NUOVA lista, quella originale associata ad `la` è sempre la stessa:

[38]:
```
la
```

[38]:
```
['p', 'r', 'o', 'v', 'a']
```

Vediamo meglio cosa è successo usando Python Tutor (per evidenziare i passaggi abbiamo creato delle variabili extra):

[39]:
```
la = ['p','r','o','v','a']
iteratore = reversed(la)
nuova = list(iteratore)
print("la è",la)
print("nuova è",nuova)

jupman.pytut()
```

```
la è ['p', 'r', 'o', 'v', 'a']
nuova è ['a', 'v', 'o', 'r', 'p']
```

[39]:
```
<IPython.core.display.HTML object>
```

DOMANDA Il seguente codice, che effetto produce?

```
la = ['p','o','n','t','e']
lb = list(reversed(reversed(la)))
```

4.26.6 sorted

La **funzione** `sorted` prende una sequenza come parametro ritorna una NUOVA lista ordinata

> **ATTENZIONE**: `sorted` ritorna una LISTA, non un iteratore !

[40]:
```
sorted(['g','a','e','d','b'])
```

[40]:
```
['a', 'b', 'd', 'e', 'g']
```

> **ATTENZIONE**: `sorted` è una **funzione** diversa dal metodo sort[234] !
>
> Nota la **ed** finale! Se provassiamo a chiamarla come un metodo otterremmo un errore:

[234] https://it.softpython.org/lists/lists3-sol.html#Metodo-sort

```
>>> la.sorted()

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-182-c8d1eec57fdd> in <module>
----> 1 la.reversed()

AttributeError: 'list' object has no attribute 'sorted'
```

Esercizio - reversort

⊛ Data una una lista di nomi, produrre una lista ordinata in ordine inverso

Ci sono almeno due modi per farlo in una sola linea di codice, trovarli entrambi

- INPUT: `['Maria','Paolo','Giovanni','Alessia','Greta']`
- OUTPUT: `['Paolo', 'Maria', 'Greta', 'Giovanni', 'Alessia']`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[41]:
```
# scrivi qui

list(sorted(['Maria','Paolo','Giovanni','Alessia','Greta'], reverse=True))

# oppure
#list(reversed(sorted(['Maria','Paolo','Giovanni','Alessia','Greta'])))
```

[41]:
```
['Paolo', 'Maria', 'Greta', 'Giovanni', 'Alessia']
```

[41]:
```
# scrivi qui
```

[41]:
```
['Paolo', 'Maria', 'Greta', 'Giovanni', 'Alessia']
```

4.26.7 zip

Supponiamo di avere due liste `quadri` e `anni`, con rispettivamente nomi di quadri famosi e le date in cui sono stati dipinti:

[42]:
```
quadri = ["La Gioconda", "La Nascita di Venere", "I Girasoli"]
anni = [1503, 1482, 1888]
```

Vogliamo produrre una lista nuova che contenga delle tuple che associano a ciascun quadro l'anno in cui è stato dipinto:

```
[('La Gioconda', 1503),
 ('La Nascita di Venere', 1482),
 ('I Girasoli', 1888)]
```

Per farlo vi sono vari modi ma senz'altro il più elegante è con la **funzione** `zip` che produce un **iteratore**:

[43]:
```
zip(quadri, anni)
```

[43]:
```
<zip at 0x7efd244ac808>
```

Per quanto non si veda scritto 'iteratore' nel nome dell'oggetto, possiamo comunque usarlo come tale con `next`:

[44]:
```
iteratore = zip(quadri, anni)
next(iteratore)
```

[44]:
```
('La Gioconda', 1503)
```

[45]:
```
next(iteratore)
```

[45]:
```
('La Nascita di Venere', 1482)
```

[46]:
```
next(iteratore)
```

[46]:
```
('I Girasoli', 1888)
```

E come fatto in precedenza, possiamo convertire tutto a lista con `list`:

[47]:
```
quadri = ["La Gioconda", "La Nascita di Venere", "I Girasoli"]
anni = [1503, 1482, 1888]

list(zip(quadri,anni))
```

[47]:
```
[('La Gioconda', 1503), ('La Nascita di Venere', 1482), ('I Girasoli', 1888)]
```

Se le liste sono di lunghezza differente, la sequenza prodotta da `zip` sarà lunga come la sequenza di input più corta:

[48]:
```
list(zip([1,2,3], ['a','b','c','d','e']))
```

[48]:
```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

Volendo, si può passare un numero arbitrario di sequenze - per esempio passandone tre otteremo delle triple di valori:

[49]:
```
canzoni = ['Imagine', 'Hey Jude', 'Satisfaction', 'Yesterday' ]
autori = ['John Lennon','The Beatles', 'The Rolling Stones', 'The Beatles']
anni = [1971, 1968, 1965, 1965]
list(zip(canzoni, autori, anni))
```

[49]:
```
[('Imagine', 'John Lennon', 1971),
 ('Hey Jude', 'The Beatles', 1968),
 ('Satisfaction', 'The Rolling Stones', 1965),
 ('Yesterday', 'The Beatles', 1965)]
```

Esercizio - scala

Data una numero `n`, creare una lista di tuple che per ogni numero intero x tale che $0 \leq x \leq n$ associa il numero $n - x$

- INPUT: `n=5`
- OUTPUT: `[(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[50]:
```
n = 5
# scrivi qui
list(zip(range(n), reversed(range(n))))
```

[50]:
```
[(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]
```

[50]:
```
n = 5
# scrivi qui



```

[50]:
```
[(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]
```

4.26.8 List comprehensions

Le list comprehensions servono per generare una NUOVA lista eseguendo la stessa operazione su tutti gli elementi di una sequenza di partenza. Come sintassi imitano le liste, infatti iniziano e finiscono con le parentesi quadre `[ ]`, ma dentro contengono un `for` speciale per ciclare dentro un sequenza :

[51]:
```
numeri = [2,5,3,4]

raddoppiati = [x*2 for x in numeri]

raddoppiati
```

[51]:
```
[4, 10, 6, 8]
```

Nota che la variabile `numeri` è ancora associata alla lista originale:

[52]:
```
numeri
```

[52]:
```
[2, 5, 3, 4]
```

Cosa è successo ? Abbiamo indicato a Python un nome di variabile `x` che ci siamo inventati noi, e gli abbiamo detto di scorrere la lista `numeri`: ad ogni iterazione, la variabile `x` viene associata ad un diverso valore della lista `numeri`. Tale valore lo possiamo riusare nell'espressione che indichiamo a sinistra del `for`, che in questo caso è `x*2`.

Come nome per la variabile abbiamo usato `x`, ma potremmo usare un qualunque altro nome, per esempio questo codice è equivalente al precedente:

[53]:
```
numeri = [2,5,3,4]

raddoppiati = [numero * 2 for numero in numeri]

raddoppiati
```

[53]:
```
[4, 10, 6, 8]
```

Nella parte a sinistra del `for` possiamo scrivere qualunque espressione che produca un valore, per esempio qua scriviamo `x + 1` per incrementare tutti i numeri della lista originale.

[54]:
```
numeri = [2,5,3,4]

aumentati = [x + 1 for x in numeri]

aumentati
```

[54]:
```
[3, 6, 4, 5]
```

DOMANDA: Questo codice cosa produrrà? Se lo visualizziamo in Python Tutor, `la` ed `lb` punteranno ad oggetti diversi?

```
la = [7,5,6,9]
lb = [x for x in la]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Quando viene eseguita `[x for x in la]` alla prima iterazione `x` vale `7`, alla seconda `5`, alla terza `6` e via dicendo. Nella espressione a sinistra del `for` abbiamo messo solo `x` quindi come risultato dell'espressione otterremo lo stesso identico numero preso dalla stringa originale.

Il codice produrrà una NUOVA lista `[7,5,6,9]` e la associerà alla variabile `lb`.

[55]:
```
la = [7,5,6,9]
lb = [x for x in la]

jupman.pytut()
```

[55]:
```
<IPython.core.display.HTML object>
```

list comprehension su stringhe

DOMANDA: Questo codice cosa produrrà?

```
[x for x in 'domanda']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Produrrà `['d', 'o', 'm', 'a', 'n', 'd', 'a']`

Visto che `'domanda'` è una stringa, se la interpretiamo come sequenza ogni suo elemento è un carattere, quindi alla prima iterazione `x` vale `'d'`, alla seconda `'o'`, alla terza `'m'` e via dicendo. Nella espressione a sinistra del `for` abbiamo messo solo `x` quindi come risultato dell'espressione otterremo lo stesso identico carattere preso dalla stringa originale.

Adesso supponiamo di avere una lista di `animali` e vogliamo produrne un'altra con gli stessi nomi ma in maiuscolo. Possiamo farlo in modo compatto con una list comprehension così:

[56]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

nuova_lista = [animale.upper() for animale in animali]
```

[57]:
```
nuova_lista
```

[57]:
```
['CANI', 'GATTI', 'SCOIATTOLI', 'ALCI']
```

Nella parte a sinistra riservata all'espressione abbiamo usato sulla variabile stringa `animale` il metodo `.upper()`. Sappiamo che le stringhe sono immutabili, quindi siamo sicuri che la chiamata al metodo produce una NUOVA stringa. Vediamo cosa è successo in Python Tutor:

[58]:
```
animali = ['cani', 'gatti', 'scoiattoli', 'alci']

nuova_lista = [animale.upper() for animale in animali]

jupman.pytut()
```

[58]:
```
<IPython.core.display.HTML object>
```

⊛ **ESERCIZIO**: Prova qua sotto ad usare una list comprehension per mettere tutti i caratteri in minuscolo (metodo `.lower()`)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[59]:
```
animali = ['caNI', 'gaTTi', 'SCOIATtoli', 'aLLci']

# scrivi qui

[animale.lower() for animale in animali]
```

[59]:
```
['cani', 'gatti', 'scoiattoli', 'allci']
```

[59]:
```
animali = ['caNI', 'gaTTi', 'SCOIATtoli', 'aLLci']

# scrivi qui
```

[59]:
```
['cani', 'gatti', 'scoiattoli', 'allci']
```

4.26.9 Domande list comprehension

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. `[x for [4,2,5]]`
2. `x for x in range(3)`
3. `[x for y in 'cartoccio']`
4. `[for x in 'zappa']`
5. `[for [3,4,5]]`
6. `[k + 1 for k in 'bozza']`
7. `[k + 1 for k in range(5)]`

8.
```
[k > 3 for k in range(7)]
```

9.
```
[s + s for s in ['lam','pa','da']]
```

10.
```
la = ['x','z','z']
[x for x in la] + [y for y in la]
```

11.
```
[x.split('-') for x in ['a-b', 'c-d', 'e-f']]
```

12.
```
['@'.join(x) for x in [['a','b.com'],['c','d.org'],['e','f.net'] ]]
```

13.
```
['z' for y in 'borgo'].count('z') == len('borgo')
```

14.
```
m = [['a','b'],['c','d'],['e','f'] ]
la = [x.pop() for x in m]    # sconsigliabile - perchè ?
print(' m:', m)
print('la:',la)
```

4.26.10 Esercizio - list comprehension

Esercizio - potenza

⊛ Data una lista di numeri, produrre una lista con la potenza quadrata dei numeri di input

- INPUT: `[4,5,9]`
- OUTPUT: `[16, 25, 81]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[60]:
```
import math

# scrivi qui
[x*x for x in [4,5,9]]
```

[60]:
```
[16, 25, 81]
```

[60]:
```
import math

# scrivi qui
```

[60]:
```
[16, 25, 81]
```

Esercizio - radice

⊛ Data una lista di numeri, produrre una lista con la radice quadrata dei numeri di input

- INPUT: `[16,25,81]`
- OUTPUT: `[4.0, 5.0, 9.0]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[61]:
```
import math

# scrivi qui
[math.sqrt(x) for x in [16,25,81]]
```

[61]:
```
[4.0, 5.0, 9.0]
```

[61]:
```
import math

# scrivi qui
```

[61]:
```
[4.0, 5.0, 9.0]
```

Esercizio - primi car

⊛ Data una lista di stringhe, produrre una lista con i primi caratteri di ogni stringa

- INPUT: `['Quando', 'Fuori', 'Piove', 'Programmiamo']`
- OUTPUT: `['Q','F','P','P']`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[62]:
```
# scrivi qui
[x[0] for x in ['Quando', 'Fuori', 'Piove', 'Programmiamo']]
```

[62]:
```
['Q', 'F', 'P', 'P']
```

[62]:
```
# scrivi qui
```

[62]:
```
['Q', 'F', 'P', 'P']
```

Esercizio - porta pazienza

⊛ Da una lista di stringhe, produrre una lista con le lunghezze di tutte le liste

- INPUT: `['non', 'ti', 'preoccupare', 'e', 'porta', 'pazienza']`
- OUTPUT: `[3,2,11,1,5,8]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[63]:
```
# scrivi qui
[len(x) for x in ['non', 'ti', 'preoccupare', 'e', 'porta', 'pazienza']]
```

[63]:
```
[3, 2, 11, 1, 5, 8]
```

[63]:
```
# scrivi qui
```

[63]:
```
[3, 2, 11, 1, 5, 8]
```

Esercizio - maggiore 3

⊛ Data una lista di numeri, produrre una lista con `True` se il corrispondente elemento è maggiore di `3`, `False` altrimenti

- INPUT: `[4,1,0,5,0,9,1]`
- OUTPUT: `[True, False, False, True, False, True, False]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[64]:
```
# scrivi qui
[x > 3 for x in [4,1,0,5,0,9,1]]
```

[64]:
```
[True, False, False, True, False, True, False]
```

[64]:
```
# scrivi qui
```

[64]:
```
[True, False, False, True, False, True, False]
```

Esercizio - pari

⊛ Data una lista di numeri, produrre una lista con `True` se il corrispondente elemento è pari

- INPUT: `[3,2,4,1,5,3,2,9]`
- OUTPUT: `[False, True, True, False, False, False, True, False]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[65]:
```
# scrivi qui
[x % 2 == 0 for x in [3,2,4,1,5,3,2,9]]
```

[65]:
```
[False, True, True, False, False, False, True, False]
```

[65]:
```
# scrivi qui
```

[65]:
```
[False, True, True, False, False, False, True, False]
```

Esercizio - capi

⊛ Data una lista di stringhe di almeno due caratteri, produrre una lista di stringhe con i primi e ultimi caratteri

- INPUT: `['parto', 'per', 'il', 'fronte']`
- OUTPUT: `['po', 'pr', 'il', 'fe']`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[66]:
```
# scrivi qui
[x[0] + x[-1] for x in ['parto', 'per', 'il', 'fronte']]
```

[66]:
```
['po', 'pr', 'il', 'fe']
```

[66]:
```
# scrivi qui
```

[66]:
```
['po', 'pr', 'il', 'fe']
```

Esercizio - trattini

⊛ Data una lista di liste di caratteri, produrre una lista di stringhe con caratteri separati da trattini

- INPUT: `[['a','b'],['c','d','e'], ['f','g']]`
- OUTPUT: `['a-b', 'c-d-e', 'f-g']`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[67]:
```
# scrivi qui
['-'.join(x) for x in [['a','b'],['c','d','e'], ['f','g']]]
```

[67]:
```
['a-b', 'c-d-e', 'f-g']
```

[67]:
```
# scrivi qui
```

[67]:
```
['a-b', 'c-d-e', 'f-g']
```

Esercizio - lollosa

⊛ Data una stringa `s`, produrre una lista di tuple avente per ogni carattere il numero di occorrenze del carattere nella stringa.

- INPUT: `s = 'lollosa'`
- OUTPUT: `[('l', 3), ('o', 2), ('l', 3), ('l', 3), ('o', 2), ('s', 1), ('a', 1)]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[68]:
```
s = 'lollosa'
# scrivi qui
[(car, s.count(car)) for car in s]
```

[68]:
```
[('l', 3), ('o', 2), ('l', 3), ('l', 3), ('o', 2), ('s', 1), ('a', 1)]
```

[68]:
```
s = 'lollosa'
# scrivi qui
```

[68]:
```
[('l', 3), ('o', 2), ('l', 3), ('l', 3), ('o', 2), ('s', 1), ('a', 1)]
```

Esercizio - cane gatto

⊛ Data una lista di stringhe, produrre una lista con le stringhe senza caratteri iniziale e finale

- INPUT: `['bue','gatto','cane', 'mucca' ]`
- OUTPUT: `['u', 'att', 'an', 'ucc']`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[69]:
```
# scrivi qui
[x[1:-1] for x in ['bue','gatto','cane', 'mucca' ]]
```

[69]:
```
['u', 'att', 'an', 'ucc']
```

[69]:
```
# scrivi qui
```

[69]:
```
['u', 'att', 'an', 'ucc']
```

Esercizio - puffi

⊛ Dati dei nomi produrre una lista con i nomi in ordine alfabetico tutti in maiuscolo

- INPUT: `['Quattrocchi', 'Forzuto', 'Puffetta', 'Tontolone']`
- OUTPUT: `['FORZUTO', 'PUFFETTA', 'QUATTROCCHI', 'TONTOLONE']`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[70]:
```
# scrivi qui
[x.upper() for x in sorted(['Quattrocchi', 'Forzuto', 'Puffetta', 'Tontolone'])]
```

[70]:
```
['FORZUTO', 'PUFFETTA', 'QUATTROCCHI', 'TONTOLONE']
```

[70]:
```
# scrivi qui
```

[70]:
```
['FORZUTO', 'PUFFETTA', 'QUATTROCCHI', 'TONTOLONE']
```

Esercizio - metalli preziosi

⊛ Date due liste `valori` e `metalli` produrre una lista contenente tutte le coppie valore metallo come tuple

INPUT:

```
valori = [10,25,50]
metalli = ['argento','oro','platino']
```

OUTPUT: `[(10, 'argento'), (25, 'oro'), (50, 'platino')]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[71]:
```
valori = [10,25,50]
metalli = ['argento','oro','platino']

# scrivi qui
list(zip(valori, metalli))
```

[71]:
```
[(10, 'argento'), (25, 'oro'), (50, 'platino')]
```

[71]:
```
valori = [10,25,50]
metalli = ['argento','oro','platino']

# scrivi qui
```

[71]:
```
[(10, 'argento'), (25, 'oro'), (50, 'platino')]
```

4.26.11 List comprehension filtrate

Durante la costruzione di una list comprehension è possibile filtrare gli elementi presi dalla sequenza che si sta esaminando utilizzando un `if`. Per esempio, l'espressione seguente considera dalla sequenza di numeri solo i numeri maggiori di cinque:

[72]:
```
[x for x in [7,4,8,2,9] if x > 5]
```

[72]:
```
[7, 8, 9]
```

Dopo l'`if` possiamo mettere qualunque espressione che riusi la variabile su cui stiamo iterando, per esempio se stiamo iterando su una stringa possiamo trattenere solo i caratteri maiuscoli:

[73]:
```
[x for x in 'Il Mondo Gira' if x.isupper()]
```

[73]:
```
['I', 'M', 'G']
```

ATTENZIONE: gli `else` non sono supportati

Per esempio, scrivere questo genera un errore:

```
[x for x in [7,4,8,2,9] if x > 5 else x + 1]    # SBAGLIATO!

File "<ipython-input-74-9ba5c135c58c>", line 1
    [x for x in [7,4,8,2,9] if x > 5 else x + 1]
                                         ^
SyntaxError: invalid syntax
```

4.26.12 Domande list comprehension filtrate

Guarda i seguenti frammenti di codice, e per ciascuno cerca di indovinare quale risultato produce (o se da errore):

1. `[x for x in range(100) if False]`
2. `[x for x in range(3) if True]`
3. `[x for x in range(6) if x > 3 else 55]`
4. `[x for x in range(6) if x % 2 == 0]`
5. `[x for x in {'a','b','c'}]  # occhio all'ordine`
6. `[x for x in [[5], [2,3], [4,2,3], [4]] if len(x) > 2]`
7. `[(x,x) for x in 'xyxyxxy' if x != 'x' ]`
8. `[x for x in ['abCdEFg'] if x.upper() == x]`
9.
```
la = [1,2,3,4,5]
[x for x in la if x > la[len(la)//2]]
```

4.26.13 Esercizio - list comprehension filtrate

Esercizio - savana

Data una lista di stringhe, produrre una lista con solo le stringhe di lunghezza maggiore di 6

- INPUT: `['zebra', 'leopardo', 'giraffa', 'gnu', 'rinoceronte', 'leone']`
- OUTPUT: `['leopardo', 'giraffa', 'rinoceronte']`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[74]:
```
# scrivi qui
[x for x in ['zebra', 'leopardo', 'giraffa', 'gnu', 'rinoceronte', 'leone'] if len(x)␣
↪> 6]
```

[74]:
```
['leopardo', 'giraffa', 'rinoceronte']
```

[74]:
```
# scrivi qui
```

[74]:
```
['leopardo', 'giraffa', 'rinoceronte']
```

Esercizio - barZa

Data una lista di stringhe, produrre una lista con solo le stringhe che contengono almeno una `'z'`. Le stringhe selezionate vanno trasformate in modo da mettere la `Z` in maiuscolo.

- INPUT: `['barza', 'palco','porzione', 'corsa', 'maschera', 'zona']`
- OUTPUT: `['barZa', 'porZione', 'Zona']`

[75]:
```
[x.replace('z','Z') for x in ['barza', 'palco','porzione', 'corsa', 'maschera', 'zona
↪'] if 'z' in  x]
```

[75]:
```
['barZa', 'porZione', 'Zona']
```

[ ]:
```

```

4.27 Funzioni - soluzioni

4.27.1 Scarica zip esercizi

Naviga file online[235]

4.27.2 Introduzione

Una funzione prende dei parametri e li usa per produrre o riportare qualche risultato.

> **ATTENZIONE**
>
> **Questi esercizi sulle funzioni sono integrativi a quelli già indicati nella** pagina riferimenti[236], in particolare vedere quelli qui sotto.

Riferimenti

- Pensare in Python, Capitolo 3, Funzioni[237]
- Pensare in Python, Capitolo 6, Funzioni produttive[238] puoi fare tutto saltando la parte 6.5 sulla ricorsione. **NOTA**: nel libro viene usato il termine strano 'funzioni produttive' per quelle funzioni che ritornano un valore, ed il termine ancora più strano 'funzioni vuote' per funzioni che non ritornano nulla ma fanno qualche effetto tipo stampa a video: ignora e dimentica questi termini !
- Nicola Cassetta, Lezione 4, Funzioni[239]

[235] https://github.com/DavidLeoni/softpython-it/tree/master/functions
[236] https://it.softpython.org/references.html
[237] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2004.html
[238] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2007.html
[239] http://ncassetta.altervista.org/Tutorial_Python/Lezione_04.html

4.27.3 Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
functions
    functions.ipynb
    functions-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `functions.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

DOMANDE: Per ciascuna delle espressioni seguenti, prova a indovinare che risultato produce (o se da errore)

1. ```
   def f():
   print('car')
   f()
   ```

2. ```
   def f():
       print('car')
   f()
   ```

3. ```
   def f():
   return 3
   f()
   ```

4. ```
   def f():
       return 3
   f()
   ```

5. ```
   def f()
       return 3
   f()
   ```

6. ```
   def f():
       return 3
   f()f()
   ```

7.
```
def f():
    return 3
f()*f()
```

8.
```
def f():
    pass
```

9.
```
def f(x):
    return x
f()
```

10.
```
def f(x):
    return x
f(5)
```

11.
```
def f():
    print('fire')
x = f()
print(x)
```

12.
```
def f():
    return(print('fire'))
print(f())
```

13.
```
def f(x):
    return 'x'
print(f(5))
```

14.
```
def f(x):
    return 'x'
print(f(5))
```

15.
```
def etc():
    print('etc...')
    return etc()
etc()
```

16.
```
def per(la):
    la.append('è')
per(['c','h'])
print(la)
```

17.
```
def zeb(la):
    la.append('d')
lista = ['a','b','c']
zeb(lista)
print(lista)
```

18.
```
def gu():
    print('GU')
    ru()
def ru():
    print('RU')
```

(continues on next page)

(continued from previous page)

```
    gu()
gu()
```

somma

⊛ Scrivere una funzione `somma` che dati due numeri x e y RITORNA la loro somma

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[1]:
```
# scrivi qui

def somma(x,y):
    return x + y
```

[1]:
```
# scrivi qui
```

[2]:
```
s = somma(3,6)
print(s)
```

```
9
```

[3]:
```
s = somma(-1,3)
print(s)
```

```
2
```

comparap

⊛ Scrivere una funzione `comparap` che dati due numeri `x` e `y`, STAMPA `x è maggiore di y`, `x è minore di y` o `x è uguale a y`

NOTA: nella stampa, mettere i numeri veri. Per es `comparap(10,5)` dovrebbe stampare:

```
10 è maggiore di 5
```

SUGGERIMENTO: per stampare numeri e testo, usare le virgole nella `print`:

```
print(x, " è maggiore di ")
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[4]:
```
# scrivi qui
def comparap(x,y):
    if x > y:
```

(continues on next page)

(continued from previous page)

```
        print(x, " è maggiore di ", y)
    elif x < y:
        print(x, " è minore di ", y)
    else:
        print(x, " è uguale a ", y)
```

[4]:
```
# scrivi qui
```

[5]:
```
comparap(10,5)
```

```
10  è maggiore di  5
```

[6]:
```
comparap(3,8)
```

```
3  è minore di  8
```

[7]:
```
comparap(3,3)
```

```
3  è uguale a  3
```

comparar

⊛ Scrivere una funzione `comparar` che dati due numeri `x` e `y`, RITORNA la STRINGA `'>'` se `x` è maggiore di `y`, la STRINGA `'<'` se `x` è minore di `y` oppure la STRINGA `'=='` se `x` è uguale a `y`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[8]:
```
# scrivi qui
def comparar(x,y):
    if x > y:
        return '>'
    elif x < y:
        return '<'
    else:
        return '=='
```

[8]:
```
# scrivi qui
```

[9]:
```
c = comparar(10,5)
print(c)
```

```
>
```

[10]:
```
c = comparar(3,7)
print(c)
```

```
<
```

[11]:
```
c = comparar(3,3)
print(c)
```

```
==
```

pari

⊛ Scrivere una funzione `pari` che dato un numero `x` RITORNA `True` se il numero `x` in ingresso è pari, altrimenti ritorna `False`

SUGGERIMENTO: un numero è pari quando il resto della divisione per due è zero. Per ottenere il resto della divisione, scrivere `x % 2`.

[12]:
```
# Esempio:
2 % 2
```

[12]:
```
0
```

[13]:
```
3 % 2
```

[13]:
```
1
```

[14]:
```
4 % 2
```

[14]:
```
0
```

[15]:
```
5 % 2
```

[15]:
```
1
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[16]:
```
# scrivi qui
def pari(x):
    return x % 2 == 0
```

[16]:
```
# scrivi qui
```

[17]:
```
p = pari(2)
print(p)
```

```
True
```

[18]:
```
p = pari(3)
print(p)
```

```
False
```

[19]:
```
p = pari(4)
print(p)
```

```
True
```

[20]:
```
p = pari(5)
print(p)
```

```
False
```

[21]:
```
p = pari(0)
print(p)
```

```
True
```

mag

⊛ Scrivere una funzione che dati due numeri `x` e `y` RITORNA il numero maggiore.

Se sono uguali, RITORNA un numero qualsiasi.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
# scrivi qui

def mag(x,y):
    if x > y:
        return x
    else:
        return y
```

[22]:
```
# scrivi qui
```

[23]:
```
m = mag(3,5)
print(m)
```

```
5
```

[24]:
```
m = mag(6,2)
print(m)
```

```
6
```

[25]:
```
m = mag(4,4)
print(m)
```

```
4
```

[26]:
```
m = mag(-5,2)
print(m)
```

```
2
```

[27]:
```
m = mag(-5, -3)
print(m)
```

```
-3
```

is_vocale

⊛ Scrivi una funzione `is_vocale` a cui viene passato un carattere `car` come parametro, e STAMPA `'sì'` se il carattere è una vocale, altrimenti STAMPA `'no'` (usando le `print`).

```
>>> is_vocale("a")
'si'

>>> is_vocale("c")
'no'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
# scrivi qui

def is_vocale(car):
    if car == 'a' or car == 'e' or car == 'i' or car == 'o' or car == 'u':
        print('sì')
    else:
        print('no')
```

[28]:
```
# scrivi qui
```

volume_sfera

⊛ Il volume di una sfera di raggio `r` è $4/3r^3$

Scrivere una funzione `volume_sfera(raggio) che dato un`raggio` di una sfera, STAMPA il volume

NB: assumi pi_greco = 3.14

```
>>> volume_sfera(4)
267.94666666666666
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[1]:
```
# scrivi qui

def volume_sfera(raggio):
```

(continues on next page)

(continued from previous page)

```
    print((4/3)*3.14*(raggio**3))
```

[1]:
```
# scrivi qui
```

ciri

⊛ Scrivi una funzione `ciri(nome)` che prende come parametro la stringa `nome` e RITORNA `True` se è uguale al nome 'Cirillo'

```
>>> r = ciri("Cirillo")
>>> r
True

>>> r = ciri("Cirillo")
>>> r
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[30]:
```
# scrivi qui

def ciri(nome):
    if nome == "Cirillo":
        return True
    else:
        return False
```

[30]:
```
# scrivi qui
```

age

⊛ Scrivi una funzione `age` che prende come parametro `anno` di nascita e RITORNA l'eta' della persona

**Supponi che l'anno corrente sia noto, quindi per rappresentarlo nel corpo della funzione usa una costante come ** `2019`

```
>>> a = age(2003)
>>> print(a)
16
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[31]:
```
# scrivi qui

def age(anno):
    return 2019 - anno
```

[31]:
```
# scrivi qui
```

4.27.4 Verifica comprensione

ATTENZIONE

Gli esercizi che seguono contengono dei test con gli *assert*. Per capire come svolgerli, leggi prima Gestione errori e testing[240]

mag_tre

⊛⊛ Scrivi una funzione `mag_tre(a,b,c)` che prende stavolta tre numeri come parametro e RESTITUISCE il più grande tra loro

Esempi:

```
>>> mag_tre(1,2,4)
4

>>> mag_tre(5,7,3)
7

>>> mag_tre(4,4,4)
4
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[32]:
```
# scrivi qui

def mag_tre(a,b,c):
    if a > b:
        if a>c:
            return a
        else:
            return c
    else:
        if b > c:
```

(continues on next page)

[240] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
            return b
        else:
            return c

assert mag_tre(1,2,4) == 4
assert mag_tre(5,7,3) == 7
assert mag_tre(4,4,4) == 4
```

[32]:
```
# scrivi qui
```

prezzo_finale

⊛⊛ Il prezzo di copertina di un libro è € 24,95, ma una libreria ottiene il 40% di sconto. I costi di spedizione sono € 3 per la prima copia e 75 centesimi per ogni copia aggiuntiva. Quanto costano `n` copie? Scrivi una funzione `prezzo_finale(n)` che RITORNA il prezzo.

ATTENZIONE 1: Python per i numeri vuole il punto, NON la virgola !

ATTENZIONE 2: Se ordinassi zero libri, quanto dovrei pagare ?

SUGGERIMENTO: il 40% di 24,95 si può calcolare moltiplicando il prezzo per 0.40

```
>>> p = prezzo_finale(10)
>>> print(p)

159.45

>>> p = prezzo_finale(0)
>>> print(p)

0
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[33]:
```
def prezzo_finale(n):

    if n == 0:
        return 0
    else:
        return n* 24.95*0.6 + 3 +(n-1)*0.75


assert prezzo_finale(10) == 159.45
assert prezzo_finale(0) == 0
```

[33]:
```
def prezzo_finale(n):
    raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
assert prezzo_finale(10) == 159.45
assert prezzo_finale(0) == 0
```

ora_arrivo

⊛⊛⊛ Correndo a ritmo blando ci mettete 8 minuti e 15 secondi al miglio, e correndo a ritmo moderato ci impiegate 7 minuti e 12 secondi al miglio.

Scrivi una funzione `ora_arrivo(n,m)` che supponendo una partenza alle ore 6 e 52 minuti, date `n` miglia percorse con ritmo blando e `m` con ritmo moderato, STAMPA l'ora di arrivo.

- **SUGGERIMENTO 1**: per calcare una divisione intera, usate `//`
- **SUGGERIMENTO 2**: per calcolare il resto di una divisione intera, usate l'operatore modulo `%`

```
>>> ora_arrivo(2,2)
7:22
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[34]:
```
def ora_arrivo(n,m):

    ore_partenza = 6
    minuti_partenza = 52

    # secondi trascorsi
    secondi = n * 495 + m * 432

    # tempo trascorso
    secondi_due = secondi % 60
    minuti= secondi // 60
    ore= minuti // 60


    ore_arrivo= ore + ore_partenza
    minuti_arrivo= minuti + minuti_partenza

    minuti_finale = minuti_arrivo % 60
    ore_finale = minuti_arrivo // 60 + ore_arrivo

    return str(ore_finale) + ":" + str(minuti_finale)


assert ora_arrivo(0,0) == '6:52'
assert ora_arrivo(2,2) == '7:22'
assert ora_arrivo(2,5) == '7:44'
assert ora_arrivo(8,5) == '9:34'
```

[34]:
```
def ora_arrivo(n,m):
    raise Exception('TODO IMPLEMENT ME !')

assert ora_arrivo(0,0) == '6:52'
```

(continues on next page)

(continued from previous page)

```
assert ora_arrivo(2,2) == '7:22'
assert ora_arrivo(2,5) == '7:44'
assert ora_arrivo(8,5) == '9:34'
```

```
[ ]:
```

4.28 Gestione degli errori e testing

4.28.1 Scarica zip esercizi

Naviga file online[241]

4.28.2 Introduzione

In questa guida tratteremo come deve fare il nostro programma quando incontra situazioni impreviste e vedremo come testare il codice che scriviamo. In particolare, descriveremo il formato degli esercizi proposti nella parte sui Fondamenti Python.

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
errors-and-testing
    errors-and-testing.ipynb
    errors-and-testing-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `errors-and-testing.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

[241] https://github.com/DavidLeoni/softpython-it/tree/master/errors-and-testing/

4.28.3 Situazioni inaspettate

E' sera, c'è da festeggiare un compleanno e perciò ti è stato chiesto di fare una torta. Sai che servono i seguenti passi:

1. prendi il latte
2. prendi lo zucchero
3. prendi la farina
4. impasta
5. scalda nel forno

Prendi il latte, lo zucchero, ma scopri di non avere la farina. E' sera, e non ci sono negozi aperti. Evidentemente, non ha senso proseguire al punto 4 con l'impasto e devi rinunciare a fare la torta, segnalando al festeggiato/a il problema. Puoi solo sperate che il festeggiato/a decida per qualche alternativa di suo gradimento.

Traducendo il tutto in termini di Python, possiamo chiederci se durante l'esecuzione di una funzione, di fronte ad una situazione imprevista è possibile :

1. **interrompere** il flusso di esecuzione del programma
2. **segnalare** a chi ha chiamato la funzione che c'è stato un problema
3. **permettere di gestire** a chi ha chiamato la funzione il problema

La risposta è sì e si fa col meccanismo delle **eccezioni** (`Exception`)

fai_torta_problematica

Vediamo intanto come possiamo rappresentare in Python il problema di sopra. Una versione di base potrebbe essere essere la seguente:

DOMANDA: questa versione ha un problema serio. Riesci a vederlo ??

[1]:
```
def fai_torta_problematica(latte, zucchero, farina):
    """ - supponiamo servano 1.3 kg per il latte, 0.2kg per lo zucchero e 1.0kg per␣
↪la farina

        - prende come parametri le quantità che abbiamo in dispensa
    """

    if latte > 1.3:
        print("prendo il latte")
    else:
        print("Non ho abbastanza latte !")

    if zucchero > 0.2:
        print("prendo lo zucchero")
    else:
        print("Non ho abbastanza zucchero !")

    if farina > 1.0:
        print("prendo la farina")
    else:
        print("Non ho abbastanza farina !")

    print("Impasto")
    print("Scaldo")
    print("Ho fatto la torta !")
```

(continues on next page)

(continued from previous page)

```
fai_torta_problematica(5,1,0.3)  # poca farina ...

print("Festeggia")
```

```
prendo il latte
prendo lo zucchero
Non ho abbastanza farina !
Impasto
Scaldo
Ho fatto la torta !
Festeggia
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: il programma qua sopra festeggia anche quando non abbiamo ingredienti sufficienti !

4.28.4 Controllare con i return

ESERCIZIO: Potremmo correggere i problemi della torta di prima aggiungendo dei comandi `return`. Implementa la seguente funzione.

NOTA: NON spostare il `print("Festeggia")` dentro la funzione. Lo scopo dell'esercizio è proprio tenerlo fuori così da usare il valore ritornato da `fai_torta` per decidere se festeggiare o meno.

Se hai dubbi sulle funzioni con valori di ritorno, puoi consultare il Capitolo 6 di Pensare in Python[242]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

```
[2]: def fai_torta(latte, zucchero, farina):
         """  - supponiamo servano 1.3 kg per il latte, 0.2kg per lo zucchero e 1.0kg per␣
     ↪la farina

              - prende come parametri le quantità che abbiamo in dispensa
              MIGLIORARE CON LE return: RITORNA True se la torta è fattibile,
                                                False altrimenti
              *FUORI* USA IL VALORE RITORNATO PER FESTEGGIARE O MENO

         """
         # implementa qui la funzione

         if latte > 1.3:
             print("prendo il latte")
             # return True  # NO, finisce subito
         else:
             print("Non ho abbastanza latte !")
             return False
```

(continues on next page)

[242] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2007.html

(continued from previous page)

```
    if zucchero > 0.2:
        print("prendo lo zucchero")
    else:
        print("Non ho abbastanza zucchero !")
        return False

    if farina > 1.0:
        print("prendo la farina")
    else:
        print("Non ho abbastanza farina !")
        return False

    print("Impasto")
    print("Scaldo")
    print("Ho fatto la torta !")
    return True



# e scrivi qui la chiamata a funzione, fai_torta(5,1,0.3)
# usando il risultato per dichiarare se si può o meno fare il party :-(


fatta_torta = fai_torta(5,1,0.3)

if fatta_torta == True:
    print("Festeggia")
else:
    print("No party !")
```

```
prendo il latte
prendo lo zucchero
Non ho abbastanza farina !
No party !
```

[2]:
```
def fai_torta(latte, zucchero, farina):
    """  - supponiamo servano 1.3 kg per il latte, 0.2kg per lo zucchero e 1.0kg per␣
↪la farina

         - prende come parametri le quantità che abbiamo in dispensa
         MIGLIORARE CON LE return: RITORNA True se la torta è fattibile,
                                          False altrimenti
         *FUORI* USA IL VALORE RITORNATO PER FESTEGGIARE O MENO

    """
    # implementa qui la funzione



# e scrivi qui la chiamata a funzione, fai_torta(5,1,0.3)
# usando il risultato per dichiarare se si può o meno fare il party :-(
```

```
prendo il latte
prendo lo zucchero
Non ho abbastanza farina !
No party !
```

4.28.5 Le eccezioni

Riferimenti: Nicola Cassetta - 19: La gestione delle eccezioni[243]

Usando i `return` abbiamo migliorato la funzione precedente, ma rimane un problema: la responsabilità di capire se la torta è riuscita rimane al chiamante della funzione, che deve prendere il valore ritornato e decidere in base a quello se festeggiare o meno. Un programmatore sbadato potrebbe dimenticarsi di effettuare il controllo e feteggiare anche con una torta mal riuscita.

Quindi ci chiediamo: è possibile fermare l'esecuzione non solo della funzione ma di tutto il programma non appena riscontriamo una situazione imprevista?

Per migliorare quanto sopra, si possono usare le *eccezioni*. Per dire a Python di **interrompere** l'esecuzione del programma in un punto, si può inserire l'istruzione `raise` così:

```
raise Exception()
```

Volendo, si può anche scrivere un messaggio che aiuti i programmatori (potreste essere voi stessi ...) a capire l'origine del problema. Nel nostro caso potrebbe essere un messaggio di questo tipo:

```
raise Exception("Non c'è abbastanza farina !")
```

Nota: nei programmi professionali, i messaggi delle exception sono intesi per programmatori, sono verbosi, e tipicamente finiscono nascosti nei log di sistema. Agli utenti finali si dovrebbero mostrare messaggi più brevi e comprensibili da un pubblico non tecnico. Al più, si può includere un codice di errore che l'utente può copiare e dare ai tecnici per aiutare a trovare il problema.

ESERCIZIO: Prova a riscrivere la funzione qua sopra sostituendo le righe con `return` con dei `raise Exception()` :

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[3]:
```
def fai_torta_eccezionale(latte, zucchero, farina):
    """ - supponiamo servano 1.3 kg per il latte, 0.2kg per lo zucchero e 1.0kg per␣
↪la farina

        - prende come parametri le quantità che abbiamo in dispensa
        - se mancano ingredienti, lancia Exception

    """
    # implementa la funzione


    if latte > 1.3:
        print("prendo il latte")
    else:
        raise Exception("Non ho abbastanza latte !")
```

(continues on next page)

[243] http://ncassetta.altervista.org/Tutorial_Python/Lezione_19.html

(continued from previous page)

```
    if zucchero > 0.2:
        print("prendo lo zucchero")
    else:
        raise Exception("Non ho abbastanza zucchero!")
    if farina > 1.0:
        print("prendo la farina")
    else:
        raise Exception("Non ho abbastanza farina !")
    print("Impasto")
    print("Scaldo")
    print("Ho fatto la torta !")
```

[3]:
```
def fai_torta_eccezionale(latte, zucchero, farina):
    """ - supponiamo servano 1.3 kg per il latte, 0.2kg per lo zucchero e 1.0kg per␣
↪la farina

        - prende come parametri le quantità che abbiamo in dispensa
        - se mancano ingredienti, lancia Exception

    """
    # implementa la funzione
```

Una volta implmenetata, scrivendo

```
fai_torta_eccezionale(5,1,0.3)
print("Festeggia")
```

Dovresti vedere (nota come "Festeggia" non venga mai stampato):

```
prendo il latte
prendo lo zucchero

---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-10-02c123f44f31> in <module>()
----> 1 fai_torta_eccezionale(5,1,0.3)
      2
      3 print("Festeggia")

<ipython-input-9-030239f08ca5> in fai_torta_eccezionale(latte, zucchero, farina)
     18         print("prendo la farina")
     19     else:
---> 20         raise Exception("Non ho abbastanza farina !")
     21     print("Impasto")
     22     print("Scaldo")

Exception: Non ho abbastanza farina !
```

Vediamo che il programma si è interrotto prima di passare all'impasto (dentro la funzione), e non è neppure stato fatto il festaggiamento (che è fuori dalla funzione). Proviamo adesso a chiamare la funzione con abbastanza elementi in dispensa:

[4]:
```
fai_torta_eccezionale(5,1,20)
print("Festeggia")
```

```
prendo il latte
prendo lo zucchero
prendo la farina
Impasto
Scaldo
Ho fatto la torta !
Festeggia
```

Gestire le eccezioni

E se invece di interrompere brutalmente il programma in caso di problemi, volessimo provare qualche alternativa (come andare a comprare del gelato ...) ? Potremmo usare i blocchi `try except` così:

[5]:
```
try:
    fai_torta_eccezionale(5,1,0.3)
    print("Festeggia")
except:
    print("Non riesco a fare la torta, che ne dite se usciamo a prendere del gelato?")
```

```
prendo il latte
prendo lo zucchero
Non riesco a fare la torta, che ne dite se usciamo a prendere del gelato?
```

Se noti, l'esecuzione ha saltato il `print("Festeggia")` ma non è stata stampata nessuna eccezione, e l'esecuzione è passata alla riga subito dopo l'`except`

Eccezioni particolari

Finora abbiamo usato `Exception` che è un'eccezione generica, ma volendo si può usare eccezioni più specifiche per segnalare meglio il tipo di errore che è accaduto. Per esempio, quando si implementa una funzione, dato che il controllo del valore corretto dei parametri come in `fai_torta_eccezionale` è qualcosa che si fa molto spesso, Python mette a disposizione una eccezione di nome `ValueError`. Usandola invece di `Exception`, si consente a chi chiama la funzione di intercettare solo quella tipologia di errore.

Se invece nella funzione viene lanciato un errore che nel `catch` non viene intercettato, il programma si interromperà.

[6]:
```
def fai_torta_eccezionale_2(latte, zucchero, farina):
    """ - supponiamo servano 1.3 kg per il latte, 0.2kg per lo zucchero e 1.0kg per␣
↪la farina

        - prende come parametri le quantità che abbiamo in dispensa
        - se mancano ingredienti, lancia ValueError
    """

    if latte > 1.3:
        print("prendo il latte")
    else:
        raise ValueError("Non ho abbastanza latte !")
    if zucchero > 0.2:
        print("prendo lo zucchero")
    else:
        raise ValueError("Non ho abbastanza zucchero!")
    if farina > 1.0:
        print("prendo la farina")
```

(continues on next page)

(continued from previous page)

```
    else:
        raise ValueError("Non ho abbastanza farina !")
    print("Impasto")
    print("Scaldo")
    print("Ho fatto la torta !")

try:
    fai_torta_eccezionale_2(5,1,0.3)
    print("Festeggia")
except ValueError:
    print()
    print("Ci deve essere un problema con gli ingredienti!")
    print("Proviamo a chiedere ai vicini !")
    print("Che fortuna, ci hanno dato della farina, riproviamo !")
    print("")
    fai_torta_eccezionale_2(5,1,4)
    print("Festeggia")
except:  # gestisce tutte le altre eccezioni
    print("Ragazzi, è successo un problema grave, no so come fare. Meglio uscire a␣
↪prendere il gelato!")
```

```
prendo il latte
prendo lo zucchero

Ci deve essere un problema con gli ingredienti!
Proviamo a chiedere ai vicini !
Che fortuna, ci hanno dato della farina, riproviamo !

prendo il latte
prendo lo zucchero
prendo la farina
Impasto
Scaldo
Ho fatto la torta !
Festeggia
```

Per ulteriori spiegazioni su i `try catch` rimandiamo alla lezioni di Nicola Cassetta - 19: La gestione delle eccezioni[244]

4.28.6 Gli assert

Ti è stato chiesto di scrivere un programma per controllare un reattore nucleare. Il reattore produce tanta energia, ma ha anche bisogno di almeno 20 metri d'acqua per raffreddarsi, e il tuo programma deve regolare il livello dell'acqua. Senza acqua sufficiente, rischi una fusione. Non ti senti esattamente all'altezza del compito, e cominci a sudare..

Con un certo nervosismo scrivi il codice. Controlli e ricontrolli il programma, credi che sia tutto giusto.

Il giorno dell'inaugurazione, il reattore viene fatto partire. Inaspettatamente, il livello dell'acqua scende a 5 metri, e inizia la reazione. Seguono spettacoli pirotecnici al plutonio.

Avremmo potuto evitare tutto ciò? A volte crediamo sia tutto giusto ma per qualche motivo ci ritroviamo variabili con valori inattesi. Il programma sbagliato descritto qua sopra, in Python avrebbe potuto essere così:

```
[7]: # ci serve acqua per raffreddare il nostro reattore nucleare
livello_acqua = 40 #  sembra sufficiente
```

(continues on next page)

[244] http://ncassetta.altervista.org/Tutorial_Python/Lezione_19.html

(continued from previous page)

```
print("livello_acqua: ", livello_acqua)

# tanto codice

# tanto codice

# tanto codice

# tanto codice

livello_acqua = 5  # abbiamo dimenticato questa riga nefasta !
print("ATTENZIONE: livello acqua basso! ", livello_acqua)

# tanto codice

# tanto codice

# tanto codice

# tanto codice

# dopo tanto codice potremmo non sapere se ci sono le condizioni necessarie
# affinchè tutto funzioni correttamente

print("fai partire reattore nucleare")
```

```
livello_acqua:  40
ATTENZIONE: livello acqua basso!  5
fai partire reattore nucleare
```

Come potremmo migliorarlo? Guardiamo come funziona il comando `assert`, che va scritto facendolo seguire da una condizione booleana.

`assert True` non fa assolutamente niente:

[8]:
```
print("prima")
assert True
print("dopo")
```

```
prima
dopo
```

Invece, `assert False` blocca l'esecuzione del programma, lanciando un'eccezione di tipo `AssertionError` (nota come `"dopo"` non venga stampato):

```
print("prima")
assert False
print("dopo")
```

```
prima
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-7-a871fdc9ebee> in <module>()
----> 1 assert False
```

(continues on next page)

(continued from previous page)

```
AssertionError:
```

Per migliorare il programma precedente, potremmo usare `assert` così:

```
# ci serve acqua per raffreddare il nostro reattore nucleare
livello_acqua = 40 #  livello sufficiente

print("livello_acqua: ", livello_acqua)

# tanto codice

# tanto codice

# tanto codice

# tanto codice

livello_acqua = 5  # abbiamo dimenticato questa riga nefasta !
print("ATTENZIONE: livello acqua basso! ", livello_acqua)

# tanto codice

# tanto codice

# tanto codice

# tanto codice


# dopo tanto codice potremmo non sapere se ci sono le condizioni necessarie
# affinchè tutto funzioni correttamente
# quindi prima di fare cose pericolose, è sempre meglio fare un controllo !
# se l'assert fallisce (cioè se l'espressione booleana è False)
# l'esecuzione si blocca subito
assert livello_acqua >= 20

print("fai partire reattore nucleare")
```

```
livello_acqua:  40
ATTENZIONE: livello acqua basso!  5

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-14-9019745468f3> in <module>
     29 # se l'assert fallisce (cioè se l'espressione booleana è False)
     30 # l'esecuzione si blocca subito
---> 31 assert livello_acqua >= 20
     32
     33 print("fai partire reattore nucleare")

AssertionError:
```

Quando usare gli assert?

Il caso qua sopra è volutamente esagerato, ma mostra come un controllo in più a volte impedisca disastri.

NOTA: Gli assert sono un modo molto spiccio di fare controlli, tanto che Python permette anche anche di ignorarli durante l'esecuzione per migliorare le performance, chiamando python con il parametro `-O` come in:

```
python -O mio_file.py
```

Ma se le performance non sono un problema (nel caso del reattore qui sopra), conviene riscrivere il programma usando un `if` e lanciando esplicitamente una `Exception`:

```
# ci serve acqua per raffreddare il nostro reattore nucleare
livello_acqua = 40 #  seems ok

print("livello_acqua: ", livello_acqua)

# tanto codice

# tanto codice

# tanto codice

# tanto codice

livello_acqua = 5  # abbiamo dimenticato questa riga nefasta !
print("ATTENZIONE: livello acqua basso! ", livello_acqua)

# tanto codice

# tanto codice

# tanto codice

# tanto codice


# dopo tanto codice potremmo non sapere se ci sono le condizioni necessarie
# affinchè tutto funzioni correttamente
# quindi prima di fare cose pericolose, è sempre meglio fare un controllo !
if livello_acqua < 20:
    raise Exception("Livello acqua troppo basso !")  # l'esecuzione si blocca subito

print("fai partire reattore nucleare")
```

```
livello_acqua:  40
ATTENZIONE: livello acqua basso!  5

---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-16-02382ff90f5a> in <module>
     28 # quindi prima di fare cose pericolose, è sempre meglio fare un controllo !
     29 if livello_acqua < 20:
---> 30     raise Exception("Livello acqua troppo basso !")  # l'esecuzione si blocca␣
↪subito
     31
     32 print("fai partire reattore nucleare")
```

(continues on next page)

(continued from previous page)

```
Exception: Livello acqua troppo basso !
```

Usare gli assert per testare

Nella parte sui Fondamenti, usiamo spesso gli `assert` per fare dei test, cioè per verificare che una funzione si comporti come ci si attende.

Per esempio, per questa funzione:

[9]:
```
def somma(x, y):
    s = x + y
    return s
```

Ci si attende che `somma(2,3)` dia `5`. Possiamo scrivere in Python questa attesa usando un `assert`:

[10]:
```
assert somma(2,3) == 5
```

Se `somma` è implementata correttamente:

1. `somma(2,3)` ci dara `5`
2. l'espression booleana `somma(2,3) == 5` darà `True`
3. `assert True` verrà eseguita senza produrre alcun risultato, lasciando proseguire l'esecuzione del programma

Viceversa, se `somma` NON è implementata correttamente come in questo caso:

```
def somma(x,y):
    return 666
```

1. `somma(2,3)` produrrà il numero `666`
2. l'espression booleana `somma(2,3) == 5` darà quindi `False`
3. `assert False` interromperà l'esecuzione del programma, lanciando un eccezione di tipo `AssertionError`

4.28.7 Struttura esercizi parte Fondamenti

Ricapitolando, gli esercizi della parte Fondamenti Python[245] nelle sezioni *Verifica Comprensione* sono spesso strutturati nel seguente formato:

```
def somma(x,y):
    """ RITORNA la somma dei numeri x e y
    """
    raise Exception("IMPLEMENTAMI")


assert somma(2,3) == 5
assert somma(3,1) == 4
assert somma(-2,5) == 3
```

Se tenti di eseguire la cella, vedrai questo errore:

[245] https://it.softpython.org/index.html#A---Fondamenti

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-16-5f5c8512d42a> in <module>()
      6
      7
----> 8 assert somma(2,3) == 5
      9 assert somma(3,1) == 4
     10 assert somma(-2,5) == 3

<ipython-input-16-5f5c8512d42a> in somma(x, y)
      3     """ RITORNA la somma dei numeri x e y
      4     """
----> 5     raise Exception("IMPLEMENTAMI")
      6
      7

Exception: IMPLEMENTAMI
```

Per risolverli, dovrai:

1. sostituire la riga `raise Exception("IMPLEMENTAMI")` con il corpo della funzione
2. eseguire la cella

Se l'esecuzione della cella non risulta in eccezioni lanciate, perfetto ! Vuol dire che la funzione fa quello che ci si attende (gli assert quando vanno a buon fine non producono output)

Se invece vedi qualche AssertionError, probabilmente hai sbagliato qualcosa.

NOTA: Il `raise Exception("IMPLEMENTAMI")` è messo per ricordarti che la funzione ha un problema grosso, e cioè che non ha il codice !!
In programma lunghi, può capitare di sapere che c'è bisogno di una funzione, ma di non sapere al momento quale codice mettere nel corpo. Invece di scrivere nel corpo dei comandi che non fanno niente tipo `print()` o `return None`, è MOLTO meglio lanciare delle eccezioni così che se per caso si esegue il programma per sbaglio, appena Python raggiunge la funzione non implementata, l'esecuzione viene subito fermata segnalando all'utente la natura e posizione del problema. Infatti, diversi editor per programmatori quando generano automaticamente codice mettono negli scheletri di funzione da implementare delle Exception di questo tipo.

Proviamo a scrivere volutamente un corpo di funzione sbagliato, che ritorna sempre `5` indipendentemente dall'`x` e `y` di ingresso:

```
def somma(x,y):
    """ RITORNA la somma dei numeri x e y
    """
    return 5

assert somma(2,3) == 5
assert somma(3,1) == 4
assert somma(-2,5) == 3
```

In questo caso la prima asserzione ha successo e quindi semplicemente passa l'esecuzione passa alla riga successiva che contiene un'altro `assert`. Ci si attende che `somma(3, 1)` dia `4`, ma la nostra funzione implementata male ritorna `5` e quindi questo assert fallisce. Notare come l'esecuzione si interrompa al *secondo* `assert`:

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-19-e5091c194d3c> in <module>()
      6
      7 assert somma(2,3) == 5
----> 8 assert somma(3,1) == 4
      9 assert somma(-2,5) == 3

AssertionError:
```

Implementando la funzione bene ed eseguendo la cella non vedremo nessun output: significa che la funzione ha passato i test e possiamo concludere che è *corretta rispetto ai test*.

ATTENZIONE: ricordati sempre che questo tipo di test *non* sono quasi mai esaustivi! Il fatto che i test passino è solo un'indicazione che la funzione *potrebbe* essere corretta, ma non una certezza !

[11]:
```
def somma(x,y):
    """ RITORNA la somma dei numeri x e y
    """
    return x + y

# Adesso la funzione è corretta, eseguendo questa cella non dovresti vedere alcun␣
↪output

assert somma(2,3) == 5
assert somma(3,1) == 4
assert somma(-2,5) == 3
```

ESERCIZIO: Prova a scrivere il corpo della funzione `moltiplica`:

- sostituisci `raise Exception("IMPLEMENTAMI")` con `return x * y` ed eseguire la cella. Se hai scritto giusto, non dovrebbe succedere niente. In tal caso complimenti, il codice che hai scritto è corretto *rispetto ai test*!
- Prova poi a sostituire invece con `return 10` e guarda che succede.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
def moltiplica(x,y):
    """ RITORNA la somma dei numeri x e y
    """

    return x * y


assert moltiplica(2,5) == 10
assert moltiplica(0,2) == 0
assert moltiplica(3,2) == 6
```

[12]:
```
def moltiplica(x,y):
    """ RITORNA la somma dei numeri x e y
    """
```

(continues on next page)

(continued from previous page)

```
    raise Exception('TODO IMPLEMENT ME !')

assert moltiplica(2,5) == 10
assert moltiplica(0,2) == 0
assert moltiplica(3,2) == 6
```

4.29 Matrici - Liste di liste

4.29.1 Scarica zip esercizi

Naviga file online[246]

ATTENZIONE

Gli esercizi che seguono contengono dei test con gli *assert*. Per capire come svolgerli, leggi prima Gestione errori e testing[247]

4.29.2 Introduzione

Ci sono sostanzialmente due modi in Python di rappresentare matrici: come liste di liste, oppure con la libreria esterna numpy[248]. La più usata è sicuramente numpy ma noi le tratteremo comunque entrambi i modi. Vediamo il motivo e le principali differenze:

Liste di liste - esercizi in questo foglio:

1. native in Python
2. non efficienti
3. le liste sono pervasive in Python, probabilmente incontrerai matrici espresse come liste di liste in ogni caso
4. forniscono un'idea di come costruire una struttura dati annidata
5. possono servire per comprendere concetti importanti come puntatori alla memoria e copie

Numpy - vedere foglio separato Matrici Numpy[249]:

1. non nativamente disponibile in Python
2. efficiente
3. alla base di parecchie librerie di calcolo scientifico (scipy, pandas)
4. la sintassi per accedere agli elementi è lievemente diversa da quella delle liste di liste
5. in alcuni rari casi potrebbe portare problemi di installazione e/o conflitti (l'implementazione non è puro Python)

[246] https://github.com/DavidLeoni/softpython-it/tree/master/matrices-lists
[247] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html
[248] https://www.numpy.org
[249] https://it.softpython.org/matrices-numpy/matrices-numpy-sol.html

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
matrices-lists
    matrices-lists.ipynb
    matrices-lists-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `matrices-lists.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

ATTENZIONE: Ricordati di eseguire sempre la prima cella dentro il notebook. Contiene delle istruzioni come `import jupman` che dicono a Python quali moduli servono e dove trovarli. Per eseguirla, vedi le seguenti scorciatoie

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

Vediamo queste liste di liste. Per esempio, possiamo considerare la seguente matrice con 3 righe e 2 colonne, in breve una matrice 3x2:

[2]:
```
m = [
        ['a','b'],
        ['c','d'],
        ['a','e']
    ]
```

Per convenienza, assumiamo come input per le nostre funzioni non ci saranno matrici senza righe, o righe senza colonne.

Tornando all'esempio, in pratica abbiamo una grande matrice esterna:

```
m = [


]
```

e ciascuno dei suoi elementi è un'altra lista che rappresenta una riga:

```
m = [
        ['a','b'],
        ['c','d'],
        ['a','e']
    ]
```

Quindi, per accedere la prima riga `['a','b']`, semplicemente accediamo all'elemento all'indice 0 della lista esterna `m`:

```
[3]: m[0]
[3]: ['a', 'b']
```

Per accedere alla seconda riga intera `['c','d']`, accediamo all'elemento avete indice 1 della lista esterna `m`:

```
[4]: m[1]
[4]: ['c', 'd']
```

Per accedere alla terza riga intera `['c', 'd']`, accediamo all'elemento ad indice 2 della lista esterna `m`:

```
[5]: m[2]
[5]: ['a', 'e']
```

Per accedere al primo elemento `'a'` della prima riga `['a','b']` aggiungiamo un altro cosiddetto "subscript operator" con indice `0`:

```
[6]: m[0][0]
[6]: 'a'
```

Per accedere il secondo elemento `'b'` della prima riga `['a','b']` usiamo invece indice `1` :

```
[7]: m[0][1]
[7]: 'b'
```

ATTENZIONE: Quando una matrice è una lista di liste, puoi solo accedere valori con notazione `m[i][j]`, **NON** con `m[i,j]` !!

```
[8]: # scrivi qui la notazione sbagliata m[0,0] e guarda che errore ottieni:
```

Adesso implementa le funzioni seguenti.

RICORDA: se la cella è eseguita e non succede niente, è perchè tutti i test degli assert sono passati ! In questo caso il tuo codice è probabilmente corretto ma attenzione, questo tipo di test non sono mai esaustivi perciò potrebbero comunque esserci errori.

III COMANDAMENTO[250]: **Non riassegnerai mai parametri di funzione**

VI COMANDAMENTO[251]: **Userai il comando return solo se vedi scritto "return" nella descrizione della funzione!**

---

[250] https://it.softpython.org/commandments.html#III-COMANDAMENTO
[251] https://it.softpython.org/commandments.html#VI-COMANDAMENTO

4.29.3 Dimensioni della matrice

⊛ **ESERCIZIO**: Per prendere le dimensioni della matrice, possiamo usare normali operazioni su lista. Quali? Puoi assumere che la matrice sia ben formata (tutte le right hanno lunghezza uguale) e almeno una riga e almeno una colonna.

[9]:
```
m = [
        ['a','b'],
        ['c','d'],
        ['a','e']
    ]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[10]:
```
# scrivi qui il codice per stampare righe e colonne

# la lista esterna è una lista di righe, perciò per contarle usiamo
# semplicemente len(m)

print("righe")
print(len(m))

# Se assumiamo che la matrice sia ben formata e ha almeno una riga e una colonna,
# possiamo controllare direttamente la lunghezza della prima riga

print("colonne")
print(len(m[0]))
```

```
righe
3
colonne
2
```

[10]:
```
# scrivi qui il codice per stampare righe e colonne
```

```
righe
3
colonne
2
```

4.29.4 Estrarre righe e colonne

Come estrarre una riga

Una delle prime cose che potresti voler fare è estrarre la riga i-esima. Se stai implementando una funzione che fa questo, hai in sostanza due scelte:

1. ritornare un *puntatore* alla riga *originale*
2. ritornare una *copia* della riga

Dato che copiare consuma memoria, perchè vorresti mai ritornare una copia ? A volte dovresti perchè non sai quale uso verrà fatto della struttura dati. Per esempio, supponi di avere un libro di esercizi che ha spazi vuoti dove scrivere gli

esercizi. E' un libro eccellente, e tutti in classe lo vogliono leggere - ma tu sei preoccupato perchè se il libro comincia a cambiare mani qualche studente poco scrupoloso potrebbe scriverci sopra. Per evitare problemi, fai una copia del libro e la distribuisci (tralasciamo considerazioni sulla violazione del copyright :-)

Estrarre puntatori

Prima vediamo cosa succede quando ritorni semplicemente un *puntatore* alla riga *originale*.

NOTA: Per convenienza, alla fine della cella mettiamo una chiamata magica a `jupman.pytut()` che mostra l'esecuzione di codice come in Python tutor (per info addizionali su `jupman.pytut()`, vedere qua[252]). Se esegui tutto il codice in Python Tutor, vedrai che alla fine hai due puntatori freccia alla riga `['a','b']`, uno che parte dalla lista `m` e uno dalla variabile `riga`.

[12]:
```
def esrigap(mat, i):
    """ RITORNA la riga i-esima da mat
    """
    return mat[i]

m = [
      ['a','b'],
      ['c','d'],
      ['a','e']
    ]
riga = esrigap(m, 0)


jupman.pytut()
```

[12]:
```
<IPython.core.display.HTML object>
```

Estrai riga con for

Cercheremo di implementare una versione che ritorna una **copia** della riga.

DOMANDA: Per farlo, potresti essere tentato di scrivere qualcosa del genere - ma non funzionerebbe. Perchè?

[13]:
```
# ATTENZIONE: CODICE SBAGLIATO!!!!

def esriga_sbagliata(mat, i):
    """ RITORNA la i-esima riga da mat. NOTA: la riga DEVE essere in una nuova lista !
↪ """

    riga = []
    riga.append(mat[i])
    return riga

m = [ ['a','b'],
      ['c','d'],
      ['a','e'] ]
riga = esriga_sbagliata(m,0)

jupman.pytut()
```

[13]:
```
<IPython.core.display.HTML object>
```

[252] https://it.softpython.org/intro/intro-sol.html#Visualizzare-l'esecuzione-con-Python-Tutor

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Il codice sopre aggiunge una LISTA come elemento ad un'altra lista vuota. In altre parole, sta includendo la riga (che è già una lista) in un'altra lista. Se verifichi il problema in Python Tutor, vedrai una freccia che va dalla riga fino alla lista di un elemento che conterrà esattamente una freccia alla riga originale.

Puoi costruire una copia in diversi modi, con un `for`, una slice o una list comprehension. Prova ad implementare tutte le versioni, cominciando con il `for` qui. Assicurati di controllare il risultato con Python tutor - per visualizzare Python tutor nell'output di una cella puoi usare il comando speciale `jupman.pytut()` alla fine della cella come abbiamo fatto prima. In Python tutor, dovresti vedere solo *una* freccia che va dalla riga originale `['a','b']` in `m`, e ci dovrebbe essere *un'altra* copia `['a','b']` da qualche parte, con la variabile with `riga` che ci punta.

⊛ **ESERCIZIO**: Implementa la funzione `esrigaf`, che RITORNA la `i`-esima riga da `mat`

- **NOTA**: la riga DEVE essere una nuova lista! Per creare una nuova lista usa un ciclo for che reitera sugli elementi, *non* gli indici (quindi non usare range) !

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[14]:
```
def esrigaf(mat, i):

    riga = []
    for x in mat[i]:
        riga.append(x)
    return riga


m = [ ['a','b'],
      ['c','d'],
      ['a','e'] ]

assert esrigaf(m, 0) == ['a','b']
assert esrigaf(m, 1) == ['c','d']
assert esrigaf(m, 2) == ['a','e']

# controlla che non abbia cambiato la matrice originale!
r = esrigaf(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

[14]:
```
def esrigaf(mat, i):
    raise Exception('TODO IMPLEMENT ME !')

m = [ ['a','b'],
      ['c','d'],
      ['a','e'] ]

assert esrigaf(m, 0) == ['a','b']
```

(continues on next page)

(continued from previous page)

```
assert esrigaf(m, 1) == ['c','d']
assert esrigaf(m, 2) == ['a','e']

# controlla che non abbia cambiato la matrice originale!
r = esrigaf(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

Estrai riga con range

Vediamo velocemente `range(n)`. Forse pensi che debba ritornare una sequenza di interi, da zero a `n - 1`. E' davvero così?

[15]:
```
range(5)
```

[15]:
```
range(0, 5)
```

Forse ti aspettavi qualcosa come una lista `[0,1,2,3,4]`, invece abbiamo scoperto che Python è piuttosto pigro qua: `range(n)` di fatto ritorna un oggetto *iterabile*, non una sequenza reale materializzata in memoria.

Per prendere una vera lista di interi, dobbiamo chiedere esplicitamente questo oggetto iterabile che ci da gli oggetti uno per uno.

Quando scrivi `for i in range(5)` il ciclo `for` sta facendo esattamente questo, ad ogni round chiede all'oggetto range di generare un numero nella sequenza. Se vogliamo l'intera sequenza materializzata in memoria, possiamo generarla convertendo il range in un oggetto lista:

[16]:
```
list(range(5))
```

[16]:
```
[0, 1, 2, 3, 4]
```

Sii prudente, comunque. A seconda della dimensione della sequenza, questo potrebbe essere pericoloso. Una lista di un miliardo di elementi potrebbe saturare la RAM del tuo computer (i portatili nel 2018 hanno spesso 4 gigabyte di memoria RAM, cioè 4 miliardi di byte).

⊛ Adesso implementa la funzione `esrigar`, che RITORNA la `i`-esima riga da `mat`, iterando su un range di indici di colonna

- **NOTA 1**: la riga DEVE essere una nuova lista! Per creare una nuova lista usa un ciclo `for`
- **NOTA 2**: ricordati di usare un nuovo nome per l'indice di colonna!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[17]:
```
def esrigar(mat, i):

    riga = []
    for j in range(len(mat[0])):
        riga.append(mat[i][j])
    return riga
```

(continues on next page)

(continued from previous page)

```
m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

assert esrigar(m, 0) == ['a','b']
assert esrigar(m, 1) == ['c','d']
assert esrigar(m, 2) == ['a','e']

# controlla che non abbia cambiato la matrice originale!
r = esrigar(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

[17]:
```
def esrigar(mat, i):
    raise Exception('TODO IMPLEMENT ME !')

m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

assert esrigar(m, 0) == ['a','b']
assert esrigar(m, 1) == ['c','d']
assert esrigar(m, 2) == ['a','e']

# controlla che non abbia cambiato la matrice originale!
r = esrigar(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

Estrai riga con slice

⊛ Ricordi che le slice ritornano una *copia* di una lista? Adesso prova ad usarle.

Implementa `esrigas`, che RITORNA la `i`-esima riga da `mat`

- **NOTA**: la riga DEVE essere una nuova lista! Per crearla, usa le slice.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
def esrigas(mat, i):

    return mat[i][:]  # se ometti gli indici di inizio e fine, hai una copia di tutta␣
↪la lista
```

(continues on next page)

(continued from previous page)

```
m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]


assert esrigas(m, 0) == ['a','b']
assert esrigas(m, 1) == ['c','d']
assert esrigas(m, 2) == ['a','e']

# Controlla che non abbia cambiato la matrice originale !
r = esrigas(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

[18]:
```
def esrigas(mat, i):
    raise Exception('TODO IMPLEMENT ME !')

m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]


assert esrigas(m, 0) == ['a','b']
assert esrigas(m, 1) == ['c','d']
assert esrigas(m, 2) == ['a','e']

# Controlla che non abbia cambiato la matrice originale !
r = esrigas(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

Estrai riga con list comprehension

⊛ Implementa `esrigac`, che RITORNA la `i`-esima riga da `mat`. Per creare una nuova lista usa le *list comprehension*

- **NOTA**: la riga DEVE essere in una nuova lista!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
def esrigac(mat, i):

    return [x for x in mat[i]]


m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

assert esrigac(m, 0) == ['a','b']
assert esrigac(m, 1) == ['c','d']
assert esrigac(m, 2) == ['a','e']

# Controlla che non abbia cambiato la matrice originale !
r = esrigac(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

[19]:
```
def esrigac(mat, i):
    raise Exception('TODO IMPLEMENT ME !')

m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

assert esrigac(m, 0) == ['a','b']
assert esrigac(m, 1) == ['c','d']
assert esrigac(m, 2) == ['a','e']

# Controlla che non abbia cambiato la matrice originale !
r = esrigac(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

Estrai colonna con for

⊛ Prova ad estrarre una colonna alla posizione `j`-esima, in questo caso non abbiamo bisogno di pensare se ritornare un puntatore o una copia.

Implementa `escolf`, che RITORNA la `j`-esima colonna da `mat`: per crearla, usa un ciclo for

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[20]:
```
def escolf(mat, j):

    ret = []
    for riga in mat:
        ret.append(riga[j])
    return ret


m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

assert escolf(m, 0) == ['a','c','a']
assert escolf(m, 1) == ['b','d','e']

# Controlla che la colonna ritornata non modifichi m
c = escolf(m,0)
c[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

[20]:
```
def escolf(mat, j):
    raise Exception('TODO IMPLEMENT ME !')

m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

assert escolf(m, 0) == ['a','c','a']
assert escolf(m, 1) == ['b','d','e']

# Controlla che la colonna ritornata non modifichi m
c = escolf(m,0)
c[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

Estrai colonna con list comprehension

⊛ Implementa `escolc`, che RITORNA la `j`-esima colonna da `mat`: per crearla, usa una list comprehension.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[21]:
```
def escolc(mat, j):

    return [riga[j] for riga in mat]


m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

assert escolc(m, 0) == ['a','c','a']
assert escolc(m, 1) == ['b','d','e']

# Controlla che la colonna ritornata non modifichi m
c = escolc(m,0)
c[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

[21]:
```
def escolc(mat, j):
    raise Exception('TODO IMPLEMENT ME !')

m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

assert escolc(m, 0) == ['a','c','a']
assert escolc(m, 1) == ['b','d','e']

# Controlla che la colonna ritornata non modifichi m
c = escolc(m,0)
c[0] = 'z'
assert m[0][0] == 'a'

# togli il commento se vuoi visualizzare l'esecuzione qui
#jupman.pytut()
```

4.29.5 Creare nuove matrici

matrice_vuota

⊛⊛ Ci sono diversi modi di creare una nuova matrice 3x5 vuota come lista di liste che contengono degli zero.

Implementa `matrice_vuota`, che RITORNA una NUOVA matrice nxn come lista di liste riempite con zero

- usa due cicli `for` annidati

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
def matrice_vuota(n, m):

    ret = []
    for i in range(n):
        riga = []
        ret.append(riga)
        for j in range(m):
            riga.append(0)
    return ret


assert matrice_vuota(1,1) == [ [0] ]

assert matrice_vuota(1,2) == [ [0,0] ]

assert matrice_vuota(2,1) == [ [0],
                               [0] ]

assert matrice_vuota(2,2) == [ [0,0],
                               [0,0] ]

assert matrice_vuota(3,3) == [ [0,0,0],
                               [0,0,0],
                               [0,0,0] ]
```

[22]:
```
def matrice_vuota(n, m):
    raise Exception('TODO IMPLEMENT ME !')

assert matrice_vuota(1,1) == [ [0] ]

assert matrice_vuota(1,2) == [ [0,0] ]

assert matrice_vuota(2,1) == [ [0],
                               [0] ]

assert matrice_vuota(2,2) == [ [0,0],
                               [0,0] ]

assert matrice_vuota(3,3) == [ [0,0,0],
                               [0,0,0],
                               [0,0,0] ]
```

matrice_vuota nel modo elegante

Per creare una lista di 3 elementi riempita di zeri, puoi scrivere così:

[23]:
```
[0]*3
```

[23]:
```
[0, 0, 0]
```

Il * in un certo senso sta moltiplicando gli elementi in una lista

Dato quanto sopra, per creare una matrice 5x3 riempita di zeri, che è una lista di liste apparentemente uguali, potresti essere tentato di scrivere così:

[24]:
```
# SBAGLIATO !
[[0]*3]*5
```

[24]:
```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Come mai questo è (probabilmente) sbagliato ? Prova a ispezionarlo in Python Tutor:

[25]:
```
bad = [[0]*3]*5
jupman.pytut()
```

[25]:
```
<IPython.core.display.HTML object>
```

Se guardi da vicino, noterai parecchie frecce che puntano alla stessa lista di 3 zeri. Questo significa che se cambiamo un numero, apparentemente cambieremo 5 di loro nell'intera colonna !

Il modo giusto di creare una matrice come lista di liste con zeri è il seguente:

[26]:
```
# CORRETTO
[[0]*3 for i in range(5)]
```

[26]:
```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

ESERCIZIO: Prova a creare una matrice con 7 righe e 4 colonne e riempila di 5.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[27]:
```
# scrivi qui

[[5]*4 for i in range(7)]
```

[27]:
```
[[5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5]]
```

[27]:
```
# scrivi qui
```

[27]:
```
[[5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5],
 [5, 5, 5, 5]]
```

Copia in profondità

Vediamo come si può produrre un clone *completo* di una matrice, anche chiamato *deep clone*, creando una copia sia della lista esterna e *anche* delle liste interne che rappresentano le righe.

DOMANDA: Per farlo, potresti essere tentato di scrivere codice del genere, ma non funzionerà. Perchè?

[28]:
```
# ATTENZIONE: CODICE SBAGLIATO:
def deep_clone_sbagliato(mat):
    """ RITORNA una NUOVA lista di liste che un DEEP CLONE  di mat (che è una lista␣
↪di liste)
    """
    return mat[:]

m = [ ['a','b'],
      ['b','d'] ]

res = deep_clone_sbagliato(m)

jupman.pytut()
```

[28]:
```
<IPython.core.display.HTML object>
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: `return mat[:]` Non è sufficiente - perchè è un clone SUPERFICIALE (SHALLOW), che copia solo la lista *esterna* ma non quelle interne! Nota che avrai righe nella lista `res` che vanno alla matrice *originale*. Non vogliamo questo !

Nel codice sopra, avrai bisogno di iterare attraverso le righe e *per ciascuna* riga creare una copia di quella riga.

⊛⊛ **ESERCIZIO**: Implementa `deep_clone`, che RITORNA una NUOVA lista come un DEEP CLONE completo di `mat` (che è una lista di liste)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[29]:
```
def deep_clone(mat):


    ret = []
    for riga in mat:
        ret.append(riga[:])
    return ret
```

(continues on next page)

(continued from previous page)

```
m = [ ['a','b'],
      ['b','d'] ]

res = [ ['a','b'],
        ['b','d'] ]

# verifica la copia
c = deep_clone(m)
assert c == res

# verifica che una copia in profondità (cioè, ha anche creato cloni delle righe !)
c[0][0] = 'z'
assert m[0][0] == 'a'
```

[29]:
```
def deep_clone(mat):
    raise Exception('TODO IMPLEMENT ME !')

m = [ ['a','b'],
      ['b','d'] ]

res = [ ['a','b'],
        ['b','d'] ]

# verifica la copia
c = deep_clone(m)
assert c == res

# verifica che una copia in profondità (cioè, ha anche creato cloni delle righe !)
c[0][0] = 'z'
assert m[0][0] == 'a'
```

4.29.6 Modificare matrici

riempic

⊛⊛ Implementa la funzione `riempic`, che prende la matrice in input `mat` (una lista di liste di dimensione `nrighe` x `ncol`) e la MODIFICA mettendo il carattere `c` dentro tutte le celle della matrice.

- per scorrere la matrice usa dei cicli for in range

Ingredienti:

- trovare dimensioni matrice
- due for annidati
- usare range

NOTA : Questa funzione non ritorna nulla!

Se nel testo della funzione non viene menzionato di tornare dei valori, NON bisogna mettere il `return`. Se per caso lo si mette lo stesso non casca il mondo, ma per evitare confusione è molto meglio avere un comportamento consistente col

testo.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[30]:
```
def riempic(mat, c):

    nrighe = len(mat)
    ncol = len(mat[0])

    for i in range(nrighe):
        for j in range(ncol):
            mat[i][j] = c


m1 = [ ['a'] ]
m2 = [ ['z'] ]
riempic(m1,'z')
assert m1 == m2

m3 = [ ['a'] ]
m4 = [ ['y'] ]
riempic(m3,'y')
assert m3 == m4

m5 = [ ['a','b'] ]
m6 = [ ['z','z'] ]
riempic(m5,'z')
assert m5 == m6

m7 = [ ['a','b','c'],
       ['d','e','f'],
       ['g','h','i'] ]

m8 = [ ['y','y','y'],
       ['y','y','y'],
       ['y','y','y'] ]
riempic(m7,'y')
assert m7 == m8

#     j  0   1
m9 = [ ['a','b'],     # 0
       ['c','d'],     # 1
       ['e','f'] ]    # 2

m10 = [ ['x','x'],     # 0
        ['x','x'],     # 1
        ['x','x'] ]    # 2
riempic(m9, 'x')
assert m9 == m10
```

[30]:
```
def riempic(mat, c):
    raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
m1 = [ ['a'] ]
m2 = [ ['z'] ]
riempic(m1,'z')
assert m1 == m2

m3 = [ ['a'] ]
m4 = [ ['y'] ]
riempic(m3,'y')
assert m3 == m4

m5 = [ ['a','b'] ]
m6 = [ ['z','z'] ]
riempic(m5,'z')
assert m5 == m6

m7 = [ ['a','b','c'],
       ['d','e','f'],
       ['g','h','i'] ]

m8 = [ ['y','y','y'],
       ['y','y','y'],
       ['y','y','y'] ]
riempic(m7,'y')
assert m7 == m8

#     j  0   1
m9 = [ ['a','b'],     # 0
       ['c','d'],     # 1
       ['e','f'] ]    # 2

m10 = [ ['x','x'],     # 0
        ['x','x'],     # 1
        ['x','x'] ]    # 2
riempic(m9, 'x')
assert m9 == m10
```

riempix

⊛⊛ Prende una matrice `mat` come lista di liste e un indice di colonna `j`, e MODIFICA `mat` mettendo il carattere `'x'` in tutte le celle della colonna `j`-esima.

Esempio:

```
m = [
      ['a','b','c','d'],
      ['e','f','g','h'],
      ['i','l','m','n']
    ]
```

Dopo la chiamata a

```
riempix(m,2)
```

la matrice `m` sarà cambiata così:

```
print(m)
```

```
[
    ['a','b','x','d'],
    ['e','f','x','h'],
    ['i','l','x','n']
]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[31]:
```
def riempix(mat, j):

    for row in mat:
        row[j] = 'x'


# INIZIO TEST: NON TOCCARE !
m1 = [ ['a'] ]
riempix(m1,0)
assert m1 == [ ['x'] ]

m2 = [ ['a','b'],
       ['c','d'],
       ['e','f'] ]
riempix(m2,0)
assert m2 == [ ['x','b'],
               ['x','d'],
               ['x','f'] ]

m3 = [ ['a','b'],
       ['c','d'],
       ['e','f'] ]
riempix(m3,1)
assert m3 == [ ['a','x'],
               ['c','x'],
               ['e','x'] ]

m4 = [ ['a','b','c','d'],
       ['e','f','g','h'],
       ['i','l','m','n'] ]
riempix(m4,2)
assert m4 == [ ['a','b','x','d'],
               ['e','f','x','h'],
               ['i','l','x','n'] ]
# FINE TEST
```

[31]:
```
def riempix(mat, j):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST: NON TOCCARE !
m1 = [ ['a'] ]
riempix(m1,0)
```

(continues on next page)

(continued from previous page)

```
assert m1 == [ ['x'] ]

m2 = [ ['a','b'],
       ['c','d'],
       ['e','f'] ]
riempix(m2,0)
assert m2 == [ ['x','b'],
               ['x','d'],
               ['x','f'] ]

m3 = [ ['a','b'],
       ['c','d'],
       ['e','f'] ]
riempix(m3,1)
assert m3 == [ ['a','x'],
               ['c','x'],
               ['e','x'] ]

m4 = [ ['a','b','c','d'],
       ['e','f','g','h'],
       ['i','l','m','n'] ]
riempix(m4,2)
assert m4 == [ ['a','b','x','d'],
               ['e','f','x','h'],
               ['i','l','x','n'] ]
# FINE TEST
```

riempiz

⊛⊛ Prende una matrice `mat` come lista di liste e un indice di riga `i`, e MODIFICA `mat` mettendo il carattere `'z'` in tutte le celle della riga `i`-esima.

Esempio:

```
m = [
    ['a','b'],
    ['c','d'],
    ['e','f'],
    ['g','h']
]
```

Dopo la chiamata a

```
riempiz(m,2)
```

la matrice `m` sarà cambiata così:

```
print(m)
```

```
[
    ['a','b'],
    ['c','d'],
    ['z','z'],
```

(continues on next page)

(continued from previous page)

```
    ['g','h']
]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[32]:
```
def riempiz(mat, i):

    ncol=len(mat[0])
    for j in range(ncol):
        mat[i][j] = 'z'


# INIZIO TEST: NON TOCCARE !
m1 = [ ['a'] ]
riempiz(m1,0)
assert m1 == [ ['z'] ]

m2 = [ ['a','b'],
       ['c','d'],
       ['e','f'] ]
riempiz(m2,0)
assert m2 == [ ['z','z'],
               ['c','d'],
               ['e','f'] ]

m3 = [ ['a','b'],
       ['c','d'],
       ['e','f'] ]
riempiz(m3,1)
assert m3 == [ ['a','b'],
               ['z','z'],
               ['e','f'] ]

m4 = [ ['a','b'],
       ['c','d'],
       ['e','f'] ]
riempiz(m4,2)
assert m4 == [ ['a','b'],
               ['c','d'],
               ['z','z'] ]
# FINE TEST
```

[32]:
```
def riempiz(mat, i):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST: NON TOCCARE !
m1 = [ ['a'] ]
riempiz(m1,0)
assert m1 == [ ['z'] ]

m2 = [ ['a','b'],
       ['c','d'],
```

(continues on next page)

(continued from previous page)

```
       ['e','f'] ]
riempiz(m2,0)
assert m2 == [ ['z','z'],
               ['c','d'],
               ['e','f'] ]

m3 = [ ['a','b'],
       ['c','d'],
       ['e','f'] ]
riempiz(m3,1)
assert m3 == [ ['a','b'],
               ['z','z'],
               ['e','f'] ]

m4 = [ ['a','b'],
       ['c','d'],
       ['e','f'] ]
riempiz(m4,2)
assert m4 == [ ['a','b'],
               ['c','d'],
               ['z','z'] ]
# FINE TEST
```

attacca_sotto

⊛⊛ Date le matrici `mat1` e `mat2` come lista di liste, con `mat1` di dimensione `u` x `n` e `mat2` di dimensione `d` x `n`, RITORNA una NUOVA matrice di dimensione `(u+d)` x `n` come lista di liste, attaccando la seconda matrice in fondo a `mat1`

- **NOT**A: per NUOVA matrice intendiamo una matrice con nessun puntatore alle righe originali (vedi il precedente esercizio deep_clone)
- Per esempi, vedere gli assert

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[33]:
```
def attacca_sotto(mat1, mat2):

    res = []
    for riga in mat1:
        res.append(riga[:])
    for riga in mat2:
        res.append(riga[:])
    return res


m11 = [ ['a'] ]
m12 = [ ['b'] ]
assert attacca_sotto(m11, m12) == [ ['a'],
                                    ['b'] ]

# controlla che non stiamo dando indietro un deep clone
r = attacca_sotto(m11, m12)
r[0][0] = 'z'
```

(continues on next page)

(continued from previous page)

```
assert m11[0][0] == 'a'

m21 = [ ['a','b','c'],
        ['d','b','a'] ]
m22 = [ ['f','b', 'h'],
        ['g','h', 'w'] ]
assert attacca_sotto(m21, m22) == [ ['a','b','c'],
                                    ['d','b','a'],
                                    ['f','b','h'],
                                    ['g','h','w'] ]
```

[33]:
```
def attacca_sotto(mat1, mat2):
    raise Exception('TODO IMPLEMENT ME !')

m11 = [ ['a'] ]
m12 = [ ['b'] ]
assert attacca_sotto(m11, m12) == [ ['a'],
                                    ['b'] ]

# controlla che non stiamo dando indietro un deep clone
r = attacca_sotto(m11, m12)
r[0][0] = 'z'
assert m11[0][0] == 'a'

m21 = [ ['a','b','c'],
        ['d','b','a'] ]
m22 = [ ['f','b', 'h'],
        ['g','h', 'w'] ]
assert attacca_sotto(m21, m22) == [ ['a','b','c'],
                                    ['d','b','a'],
                                    ['f','b','h'],
                                    ['g','h','w'] ]
```

attacca_sopra

⊛⊛ Date le matrici `mat1` e `mat2` come lista di liste, con `mat1` di dimensione `u` x `n` e `mat2` di dimensione `d` x `n`, RITORNA una NUOVA matrice di dimensione (`u`+`d`) x `n` come lista di liste, attaccando la prima `mat` alla fine di `mat2`

- **NOTA**: per NUOVA matrice intendiamo una matrice con nessun puntatore alle righe originali (vedi il precedente esercizio `deep_clone`)
- Per implementare questa funzione, usa una chiamata al metodo attacca_sotto che hai implementato prima
- Per esempi, vedere gli assert

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[34]:
```
def attacca_sopra(mat1, mat2):

    return attacca_sotto(mat2, mat1)
```

(continues on next page)

(continued from previous page)

```
m1 = [
        ['a']
     ]
m2 = [
        ['b']
     ]
assert attacca_sopra(m1, m2) == [
                                  ['b'],
                                  ['a']
                                ]

# controlla che stiamo ritornando un deep clone
s = attacca_sopra(m1, m2)
s[0][0] = 'z'
assert m1[0][0] == 'a'

m1 = [
        ['a','b','c'],
        ['d','b','a']
     ]
m2 = [
        ['f','b', 'h'],
        ['g','h', 'w']
     ]

res = [
        ['f','b','h'],
        ['g','h','w'],
        ['a','b','c'],
        ['d','b','a']
     ]

assert attacca_sopra(m1, m2) == res
```

[34]:
```
def attacca_sopra(mat1, mat2):
    raise Exception('TODO IMPLEMENT ME !')

m1 = [
        ['a']
     ]
m2 = [
        ['b']
     ]
assert attacca_sopra(m1, m2) == [
                                  ['b'],
                                  ['a']
                                ]

# controlla che stiamo ritornando un deep clone
s = attacca_sopra(m1, m2)
s[0][0] = 'z'
assert m1[0][0] == 'a'

m1 = [
        ['a','b','c'],
```

(continues on next page)

(continued from previous page)

```
        ['d','b','a']
     ]
m2 = [
        ['f','b', 'h'],
        ['g','h', 'w']
     ]

res = [
        ['f','b','h'],
        ['g','h','w'],
        ['a','b','c'],
        ['d','b','a']
     ]

assert attacca_sopra(m1, m2) == res
```

attacca_dx

⊛⊛⊛ Date le matrici `mat1` e `mat2` come lista di liste, con `mat1` di dimensione `n` x `l` e `mat2` di dimensione `n` x `r`, RITORNA una NUOVA matrice di dimensione `n` x (`l` + `r`) come lista di liste, attaccando `mat2` alla destra di `mat1`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[35]:
```
def attacca_dx(mat1,mat2):

    ret = []
    for i in range(len(mat1)):
        riga_da_aggiungere =  mat1[i][:]
        riga_da_aggiungere.extend(mat2[i])
        ret.append(riga_da_aggiungere)
    return ret



m1 = [
        ['a','b','c'],
        ['d','b','a']
     ]
m2 = [
        ['f','b'],
        ['g','h']
     ]

res = [
        ['a','b','c','f','b'],
        ['d','b','a','g','h']
      ]

assert attacca_dx(m1, m2) == res
```

[35]:
```
def attacca_dx(mat1,mat2):
    raise Exception('TODO IMPLEMENT ME !')


m1 = [
        ['a','b','c'],
        ['d','b','a']
     ]
m2 = [
        ['f','b'],
        ['g','h']
     ]

res = [
        ['a','b','c','f','b'],
        ['d','b','a','g','h']
      ]

assert attacca_dx(m1, m2) == res
```

soglia

⊛⊛ Prende una matrice `mat` come lista di liste (ogni lista ha la stessa dimensione) e un numero `t`, e RITORNA una NUOVA matrice come lista di liste dove c'è `True` se l'elemento di input corrispondente è maggiore di `t`, altrimenti ritorna `False`.

Ingredienti:

- una variabile per la matrice da ritornare
- per ogni riga originale, dobbiamo creare una nuova lista

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[36]:
```
def soglia(mat, t):

    ret = []
    for riga in mat:
        nuova_riga = []
        ret.append(nuova_riga)
        for el in riga:
            nuova_riga.append(el > t)

    return ret


morig = [ [1,4,2],
          [7,9,3] ]


m = [ [1,4,2],
      [7,9,3] ]

s = [ [False,False,False],
      [True, True, False] ]
```

(continues on next page)

(continued from previous page)

```
assert soglia(m,4) == s
assert m == morig   # verifica che original non sia cambiata


m = [ [5,2],
      [3,7] ]

s = [
    [True,False],
    [False,True]
]
assert soglia(m,4) == s
```

[36]:
```
def soglia(mat, t):
    raise Exception('TODO IMPLEMENT ME !')

morig = [ [1,4,2],
          [7,9,3] ]


m = [ [1,4,2],
      [7,9,3] ]

s = [ [False,False,False],
      [True, True, False] ]
assert soglia(m,4) == s
assert m == morig   # verifica che original non sia cambiata


m = [ [5,2],
      [3,7] ]

s = [
    [True,False],
    [False,True]
]
assert soglia(m,4) == s
```

scambia_righe

Proveremo a scambiare due righe di una matrice

Ci sono diversi modi di procedere. Prima di continuare, assicurati di sapere come scambiare solo due valori risolvendo questo semplice esercizio - controlla inoltre il risultato in Python Tutor.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[37]:
```
x = 3
y = 7

# scrivi qui il codice per scambiare x e y (non usare direttamente le costanti 3 e 7!)
```

(continues on next page)

(continued from previous page)

```
k = x
x = y
y = k

#jupman.pytut()
```

[37]:
```
x = 3
y = 7

# scrivi qui il codice per scambiare x e y (non usare direttamente le costanti 3 e 7!)
```

⊛⊛ Prende una matrice `mat` come lista di liste, e RITORNA una NUOVA matrice dove le righe agli indici `i1` e `i2` sono scambiate

Ingredienti:

- prima clona in profondità
- poi scambia le righe

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[38]:
```
def scambia_righe(mat, i1, i2):


    # prima clona in profondita
    ret = []
    for riga in mat:
        ret.append(riga[:])
    # poi scambia
    s = ret[i1]
    ret[i1] = ret[i2]
    ret[i2] = s
    return ret


m = [ ['a','d'],
      ['b','e'],
      ['c','f'] ]

res = scambia_righe(m, 0, 2)

assert res == [ ['c','f'],
                ['b','e'],
                ['a','d'] ]

res[0][0] = 'z'
assert m[0][0] == 'a'


m = [ ['a','d'],
```

(continues on next page)

(continued from previous page)

```
      ['b','e'],
      ['c','f'] ]


# scambia con se stesso dovrebbe nei fatti generare un deep clone
res = scambia_righe(m, 0, 0)

assert res == [ ['a','d'],
                ['b','e'],
                ['c','f'] ]

res[0][0] = 'z'
assert m[0][0] == 'a'
```

[38]:
```
def scambia_righe(mat, i1, i2):
    raise Exception('TODO IMPLEMENT ME !')

m = [ ['a','d'],
      ['b','e'],
      ['c','f'] ]

res = scambia_righe(m, 0, 2)

assert res == [ ['c','f'],
                ['b','e'],
                ['a','d'] ]

res[0][0] = 'z'
assert m[0][0] == 'a'


m = [ ['a','d'],
      ['b','e'],
      ['c','f'] ]


# scambia con se stesso dovrebbe nei fatti generare un deep clone
res = scambia_righe(m, 0, 0)

assert res == [ ['a','d'],
                ['b','e'],
                ['c','f'] ]

res[0][0] = 'z'
assert m[0][0] == 'a'
```

scambia_colonne

⊛⊛ Prende una matrice `mat` e due indici di colonna `j1` e `j2` e RITORNA una NUOVA matrice dove le colonne `j1` e `j2` sono scambiate

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[39]:
```
def scambia_colonne(mat, j1, j2):

    ret = []
    for riga in mat:
        nuova_riga = riga[:]
        nuova_riga[j1] = riga[j2]
        nuova_riga[j2] = riga[j1]
        ret.append(nuova_riga)
    return ret


m = [ ['a','b','c'],
      ['d','e','f'] ]

res = scambia_colonne(m, 0,2)

assert res == [ ['c','b','a'],
                ['f','e','d'] ]

res[0][0] = 'z'
assert m[0][0] == 'a'
```

[39]:
```
def scambia_colonne(mat, j1, j2):
    raise Exception('TODO IMPLEMENT ME !')

m = [ ['a','b','c'],
      ['d','e','f'] ]

res = scambia_colonne(m, 0,2)

assert res == [ ['c','b','a'],
                ['f','e','d'] ]

res[0][0] = 'z'
assert m[0][0] == 'a'
```

4.29.7 Altri esercizi

diag

`diag` estrae la diagonale di una matrice. Per farlo, `diag` richiede una matrice nxn come input. Per essere sicuri che ci prendiamo effettivamente una matrice nxn, questa volta dovrai validare l'input, cioè controllare che il numero di righe sia uguale al numero di colonne (come al solito assumi che la matrice abbia almeno una riga e almeno una colonna). Se la matrice non è nxn, la funzione dovrebbe fermarsi e lanciare una eccezione. In particolare, dovrebbe lanciare un ValueError[253], che è il modo standard in Python di lanciare un'eccezione quando l'input atteso non è corretto e non puoi trovare errori più specifici.

Per scopi illustrativi, mostriamo qui i numeri indice `i` e `j` e evitiamo di mettere gli apici intorno alle stringhe:

```
\ j  0,1,2,3
i
   [
0   [a,b,c,d],
1   [e,f,g,h],
2   [p,q,r,s],
3   [t,u,v,z]
   ]
```

Vediamo una esecuzione passo-passo:

```
                                \ j  0,1,2,3
                                i
                                   [
estrai dalla riga a  i=0 -->    0   [a,b,c,d],        'a' è estratto da mat[0][0]
                                1   [e,f,g,h],
                                2   [p,q,r,s],
                                3   [t,u,v,z]
                                   ]
```

```
                                \ j  0,1,2,3
                                i
                                   [
                                0   [a,b,c,d],
estrai dalla riga a i=1  -->    1   [e,f,g,h],        'f' è estratto da mat[1][1]
                                2   [p,q,r,s],
                                3   [t,u,v,z]
                                   ]
```

```
                                \ j  0,1,2,3
                                i
                                   [
                                0   [a,b,c,d],
                                1   [e,f,g,h],
estrai dalla riga a i=2  -->    2   [p,q,r,s],        'r' è estratto da mat[2][2]
                                3   [t,u,v,z]
                                   ]
```

```
                                 \ j  0,1,2,3
                                 i
                                    [
```

(continues on next page)

[253] https://docs.python.org/3/library/exceptions.html#ValueError

(continued from previous page)

```
                                0   [a,b,c,d],
                                1   [e,f,g,h],
                                2   [p,q,r,s],
 estrai dalla riga a i=3  -->   3   [t,u,v,z]         'z' è estratto da mat[3][3]
                                  ]

Da quanto sopra, notiamo che abbiamo bisogno di elementi da questi indici:

 i, j
 1, 1
 2, 2
 3, 3
```

Ci sono due modi di risolvere questo esercizio, uno è usare un doppio for (un for annidato, per essere precisi), mentre l'altro metodo usa solo un for. Prova a risolverlo in entrambi i modi. Di quanti passi hai bisogno con un doppio for? e con uno solo?

⊛⊛ **ESERCIZIO**: Implementa la funzione `diag`, che data una matrice `nxn` come lista di liste, RITORNA una lista che contiene gli elementi della diagonale (da sinistra in alto fino all'angolo basso destro)

- se `mat` non è `nxn` solleva l'eccezione `ValueError`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[40]:
```
def diag(mat):

    if len(mat) != len(mat[0]):
        raise ValueError("mat dovrebbe essere nxn, trovato invece %s x %s" %␣
↪(len(mat), len(mat[0])))
    ret = []
    for i in range(len(mat)):
        ret.append(mat[i][i])
    return ret


m = [ ['a','b','c'],
      ['d','e','f'],
      ['g','h','i'] ]

assert diag(m) == ['a','e','i']

try:
    diag([              # 1x2 dimension, non quadrata
           ['a','b']
         ])
    raise Exception("Dovrei aver fallito !") # se diag solleva un'eccezione che è␣
↪ValueError come ci
                                             # aspettiamo che faccia, il codice non␣
↪dovrebbe mai arrivare qui
except ValueError: # questo cattura solo ValueError. Altri tipi di errori non sono␣
↪catturati
    pass  # In una calusola except devi sempre mettere del codice
                  # Qui mettiamo il comando pass che non fa niente
```

[40]:
```
def diag(mat):
    raise Exception('TODO IMPLEMENT ME !')

m = [ ['a','b','c'],
      ['d','e','f'],
      ['g','h','i'] ]

assert diag(m) == ['a','e','i']

try:
    diag([              # 1x2 dimension, non quadrata
           ['a','b']
         ])
    raise Exception("Dovrei aver fallito !") # se diag solleva un'eccezione che è␣
↪ValueError come ci
                                             # aspettiamo che faccia, il codice non␣
↪dovrebbe mai arrivare qui
except ValueError: # questo cattura solo ValueError. Altri tipi di errori non sono␣
↪catturati
    pass  # In una calusola except devi sempre mettere del codice
                  # Qui mettiamo il comando pass che non fa niente
```

anti_diag

⊛⊛ Data una matrice `nxn` come lista di liste, RITORNA una lista che contiene gli elementi della antidiagonale (dall'angolo destro fino all'angolo in basso a sinistra).

- Se `mat` non è `nxn` solleva `ValueError`.

Prima di implementarla, ricordati di scrivere gli indici richiesti come abbiamo fatto per l'esempio della funzione *diag*

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[41]:
```
def anti_diag(mat):

    n = len(mat)
    ret = []
    for i in range(n):
        ret.append(mat[i][n-i-1])
    return ret


m = [ ['a','b','c'],
      ['d','e','f'],
      ['g','h','i'] ]

assert anti_diag(m) == ['c','e','g']

# Se hai dubbi riguardo gli indici ricordati di provare il codice in Python Tutor !
# jupman.pytut()
```

[41]:
```
def anti_diag(mat):
    raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
m = [ ['a','b','c'],
      ['d','e','f'],
      ['g','h','i'] ]

assert anti_diag(m) == ['c','e','g']

# Se hai dubbi riguardo gli indici ricordati di provare il codice in Python Tutor !
# jupman.pytut()
```

is_utriang

Chiediamoci cosa vuol dire iterare solo la parte triangolare inferiore di una matrice. Vediamo un esempio:

[42]:
```
m = [
        [3,2,5,8],
        [0,6,2,3],
        [0,0,4,9],
        [0,0,0,5]
    ]
```

Solo per propositi illustrativi, mostriamo qui i numeri di indice `i` e `j`:

```
\ j  0,1,2,3
i
   [
0   [3,2,5,8],
1   [0,6,2,3],
2   [0,0,4,9],
3   [0,7,0,5]
   ]
```

Vediamo una esecuzione passo passo su una matrice triangolare non-superiore:

```
                                \ j  0,1,2,3
                                i
                                   [
                                0   [3,2,5,8],
inizia da riga a indice i=1 --> 1   [0,6,2,3],      Controlla fino a colonna limite␣
↪j=0 inclusa
                                2   [0,0,4,9],
                                3   [0,7,0,5]
                                   ]
```

Viene trovato uno zero, è ora di controllare la riga successiva.

```
                                \ j  0,1,2,3
                                i
                                   [
                                0   [3,2,5,8],
                                1   [0,6,2,3],
controlla riga a indice i=2 --> 2   [0,0,4,9],      Controlla fino a colonna limite ␣
↪j=1 inclusa
                                3   [0,7,0,5]
                                   ]
```

Due zeri sono trovati, è ora di controllare la riga successiva.

```
                                \ j  0,1,2,3
                                i
                                   [
                                0   [3,2,5,8],
                                1   [0,6,2,3],
                                2   [0,0,4,9],
controlla riga a indice i=3 --> 3   [0,7,0,5]       Controlla fino a colonna limite ␣
↪j=2 inclusa
                                   ]                MA può fermarsi prima a j=1␣
↪perchè il numero a j=1
                                                    è differente da zero. Appena 7 è␣
↪trovato, può ritornare False
                                                    In questo caso la matrice non è␣
↪triangolare superiore
```

VII COMANDAMENTO[254]: **Scriverai anche su carta!**

Quando sviluppi questi algoritmi, è fondamentale scrivere un esempio passo passo come quello sopra per avere un'idea chiara di cosa sta succedendo. Inoltre, se scrivi giù gli indici correttamente, sarai facilmente in grado di derivare una generalizzazione. Per trovarla, prova a scrivere ulteriormente gli indici trovati in una tabella.

Per esempio, da quanto sopra per ciascuna riga indice `i` possiamo facilmente trovare di quale indice limite `j` abbiamo bisogno per raggiungere nella nostra caccia agli zero:

```
| i | limit j (included) |            Notes                    |
|---|--------------------|-------------------------------------|
| 1 |          0         |  cominciamo dalla riga a indice i=1 |
| 2 |          1         |                                     |
| 3 |          2         |                                     |
```

Dalla tabella, possiamo vedere che il limite per `j` può essere calcolato in termini dell'indice riga corrente `i` con una semplice formula `i - 1`

Il fatto che tu debba muoverti attraverso righe e colonne suggerisce che hai bisogno di due `for`, uno per le righe e uno per le colonne - cioè un *for annidato*

per svolgere l'esercizio:

- usa range di indici (quindi niente `for riga in mat` ..)
- usa i caratteri `i` come indice per le righe, `j` come indice per le colonne e in caso tu ne abbia bisogno la lettera `n` come dimensione della matrice

SUGGERIMENTO 1: ricorda che puoi consentire a range di partire da un indice specifico, come `range(3,7)` che partirà da 3 e finira a 6 *incluso* (l'ultimo 7 è *escluso*).

SUGGERIMENTO 2: Per implementare questo, è meglio guardare a numeri *diversi* da zero. Appena ne trovi uno, puoi fermare la funzione e ritornare False. Solo dopo che *tutto* il controllo dei numeri è fatto puoi ritornare `True`

Infine, ricordati di questo:

II COMANDAMENTO[255]: **Quando inserisci una variabile in un ciclo for, questa variabile deve essere nuova**

[254] https://it.softpython.org/commandments.html#VII-COMANDAMENTO
[255] https://it.softpython.org/commandments.html#II-COMANDAMENTO

⊛⊛⊛ **ESERCIZIO**: Se hai letto *tutto* quanto sopra, comincia ad implementare la funzione `is_utriang`, che RITORNA `True` se la matrice `nxn` fornita è triangolare superiore, cioè, ha tutte le celle sotto la diagonale a zero. Altrimenti, ritorna `False`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[43]:
```
def is_utriang(mat):

    n = len(mat)
    m = len(mat[0])

    for i in range(1,n):
        for j in range(i): # nota che arriva fino alla i *esclusa*, cioè, arriva a i -
↪ 1 *inclusa*
            if mat[i][j] != 0:
                return False
    return True


assert is_utriang([ [1] ]) == True
assert is_utriang([ [3,2,5],
                    [0,6,2],
                    [0,0,4] ]) == True

assert is_utriang([ [3,2,5],
                    [0,6,2],
                    [1,0,4] ]) == False

assert is_utriang([ [3,2,5],
                    [0,6,2],
                    [1,1,4] ]) == False

assert is_utriang([ [3,2,5],
                    [0,6,2],
                    [0,1,4] ]) == False


assert is_utriang([ [3,2,5],
                    [1,6,2],
                    [1,0,4] ]) == False
```

[43]:
```
def is_utriang(mat):
    raise Exception('TODO IMPLEMENT ME !')

assert is_utriang([ [1] ]) == True
assert is_utriang([ [3,2,5],
                    [0,6,2],
                    [0,0,4] ]) == True

assert is_utriang([ [3,2,5],
                    [0,6,2],
                    [1,0,4] ]) == False
```

(continues on next page)

(continued from previous page)

```
assert is_utriang([ [3,2,5],
                    [0,6,2],
                    [1,1,4] ]) == False

assert is_utriang([ [3,2,5],
                    [0,6,2],
                    [0,1,4] ]) == False


assert is_utriang([ [3,2,5],
                    [1,6,2],
                    [1,0,4] ]) == False
```

attacca_sx_mod

Questa volta proviamo a *modificare* `mat1` sul posto (*in place*), attaccando `mat2` *alla sinistra* di `mat1`.

Perciò questa volta **non** mettere una istruzione `return`.

Avrai bisogno di eseguire una inserzione di lista, che può essere problematica. Ci sono molti modi di farlo in Python, uno potrebbe essere usare l'inserzione cosiddetta di 'splice assignment' (che può apparire un po' strana):

```
mia_lista[0:0] = lista_da_inserire
```

Guarda qui per altre info (in inglese): https://stackoverflow.com/a/10623383

⊛⊛⊛ **ESERCIZIO**: implementa `attacca_sx_mod`, che date le matrici `mat1` e `mat2` come lista di liste, con `mat1` di dimensioni `n` x `l` e `mat2` di dimensioni `n` x `r`, MODIFICA `mat1` così che diventi di dimensioni `n` x (`l` + `r`), attaccando la `mat2` alla sinistra di `mat1`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[44]:
```
def attacca_sx_mod(mat1,mat2):


    for i in range(len(mat1)):
        mat1[i][0:0] = mat2[i]



m1 = [ ['a','b','c'],
       ['d','b','a'] ]
m2 = [ ['f','b'],
       ['g','h'] ]

res = [ ['f','b','a','b','c'],
        ['g','h','d','b','a'] ]

attacca_sx_mod(m1, m2)
assert m1 == res
```

[44]:
```
def attacca_sx_mod(mat1,mat2):
    raise Exception('TODO IMPLEMENT ME !')


m1 = [ ['a','b','c'],
       ['d','b','a'] ]
m2 = [ ['f','b'],
       ['g','h'] ]

res = [ ['f','b','a','b','c'],
        ['g','h','d','b','a'] ]

attacca_sx_mod(m1, m2)
assert m1 == res
```

trasposta_1

Vediamo come trasporre una una matrice *sul posto* (*in-place*). La trasposta M^T di una matrice M è definita come

$M^T[i][j] = M[j][i]$

La definizione è semplice eppure l'implementazione può essere insidiosa. Se non stai attento, potresti facilmente finire a scambiare valori due volte e riottenere la stessa matrice originale. Per evitare ciò, itera solo la parte triangolare superiore e ricordati che la funzione `range` può avere un indice di partenza:

[45]:
```
list(range(3,7))
```

[45]:
```
[3, 4, 5, 6]
```

Inoltre, assicurati di sapere come scambiare solo due valori risolvendo questo semplice esercizio - controlla inoltre il risultato in Python Tutor.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[46]:
```
x = 3
y = 7

# scrivi qui il codice per scambiare x e y (non usare direttamente le costanti 3 e 7!)

k = x
x = y
y = k

jupman.pytut()
```

[46]:
```
<IPython.core.display.HTML object>
```

[46]:
```
x = 3
y = 7

# scrivi qui il codice per scambiare x e y (non usare direttamente le costanti 3 e 7!)
```

[46]:
```
<IPython.core.display.HTML object>
```

Tornando alla trasposta, per adesso consideriamo solo una matrice nxn. Per assicurarci che ci prendiamo in effetti una matrice nxn, valideremo l'input come fatto in precedenza

IV COMANDAMENTO[256] **(adattato per matrici): Non riassegnerai mai parametri di funzione**

```
def myfun(M):

    # M è un parametro, perciò *non* commetterai mai malvagità come:

    M = [
            [6661,6662],
            [6663,6664 ]
        ]


    # Per il solo caso di parametri composti come liste (o liste di liste...),
    # puoi scrivere cose come questa SE E SOLO SE le specifiche della funzione
    # ti richiedono di modificare gli elementi interni del parametro (es. trasopsta _
↪in-place_)

    M[0][1] =  6663
```

⊛⊛⊛ **ESERCIZIO**: Se hai letto *tutto* quanto sopra, adesso puoi procedere implementando la funzione `trasposta_1`, che MODIFICA la matrice `nxn` data `mat`, facendo la trasposta *in-place*

- Se la matrice non è `nxn`, lancia l'eccezione `ValueError`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[47]:
```
def trasposta_1(mat):

    if len(mat) != len(mat[0]):
        raise ValueError("La matrice dovrebbe essere nxn, trovato invece %s x %s" %␣
↪(len(mat), len(mat[0])))
    for i in range(len(mat)):
        for j in range(i+1,len(mat[i])):
            el = m[i][j]
            mat[i][j] = m[j][i]
            mat[j][i] = el


# Controlliamo le dimensioni sbagliate della matrice:

try:
    trasposta_1([ [3,5] ])
    raise Exception("AVREI DOVUTO FALLIRE !")
except ValueError:
    pass

m = [ ['a'] ]
```

(continues on next page)

[256] https://it.softpython.org/commandments.html#IV-COMANDAMENTO

(continued from previous page)

```
trasposta_1(m)
assert m == [ ['a'] ]

m = [ ['a','b'],
      ['c','d'] ]

trasposta_1(m)
assert m == [ ['a','c'],
              ['b','d'] ]
```

[47]:
```
def trasposta_1(mat):
    raise Exception('TODO IMPLEMENT ME !')

# Controlliamo le dimensioni sbagliate della matrice:

try:
    trasposta_1([ [3,5] ])
    raise Exception("AVREI DOVUTO FALLIRE !")
except ValueError:
    pass

m = [ ['a'] ]

trasposta_1(m)
assert m == [ ['a'] ]

m = [ ['a','b'],
      ['c','d'] ]

trasposta_1(m)
assert m == [ ['a','c'],
              ['b','d'] ]
```

transpose_2

⊛⊛ Adesso proviamo a trasporre una matrice generica nxm. Questa volta per semplicità ritorneremo una intera nuova matrice.

Prende una matrice `mat nxm` come lista di liste e RITORNA una NUOVA matrice `mxn` che è la trasposta della matrice di input:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[48]:
```
def trasposta_2(mat):

    n = len(mat)
    m = len(mat[0])
    ret = [[0]*n for i in range(m)]
    for i in range(n):
        for j in range(m):
```

(continues on next page)

(continued from previous page)

```
            ret[j][i] = mat[i][j]
    return ret


m1 = [ ['a'] ]

r1 = trasposta_2(m1)

assert  r1 == [ ['a'] ]
r1[0][0] = 'z'
assert m1[0][0] == 'a'

m2 = [ ['a','b','c'],
      ['d','e','f'] ]

assert trasposta_2(m2) == [ ['a','d'],
                            ['b','e'],
                            ['c','f'] ]
```

[48]:
```
def trasposta_2(mat):
    raise Exception('TODO IMPLEMENT ME !')

m1 = [ ['a'] ]

r1 = trasposta_2(m1)

assert  r1 == [ ['a'] ]
r1[0][0] = 'z'
assert m1[0][0] == 'a'

m2 = [ ['a','b','c'],
      ['d','e','f'] ]

assert trasposta_2(m2) == [ ['a','d'],
                            ['b','e'],
                            ['c','f'] ]
```

flip

⊛⊛ Prende una matrice come lista di liste in ingresso contenenti zeri e uni, e RITORNA una nuova matrice (sempre come lista di liste), costruita prima invertendo tutte le righe della matrice di input e poi rovesciando tutte le righe

- Invertire una lista vuol dire trasformare gli 0 in 1 e gli 1 in 0. Per esempio,

  [0,1,1] diventa [1,0,0][0,0,1] diventa [1,1,0]

- Rovesciare una lista vuol dire che rovesciare l'ordine degli elementi:

  Per esempio [0,1,1] diventa [1,1,0][0,0,1] diventa [1,0,0]

Combinando inversione e rovesciamento, per esempio se partiamo da

```
[
  [1,1,0,0],
  [0,1,1,0],
```

(continues on next page)

(continued from previous page)

```
  [0,0,1,0]
]
```

Prima invertiamo ciascun elemento:

```
[
  [0,0,1,1],
  [1,0,0,1],
  [1,1,0,1]
]
```

e poi rovesciamo ciascuna riga:

```
[
  [1,1,0,0],
  [1,0,0,1],
  [1,0,1,1]

]
```

Suggerimenti

- per rovesciare una lista usare il metodo `.reverse()` come in `mia_lista.reverse()` NOTA: `mia_lista.reverse()` modifica `mia_lista`, *non* ritorna una nuova lista !!
- ricordarsi ll `return` !!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[49]:
```
def flip(matrice):

    ret = []
    for riga in matrice:
        nuova_riga = []
        for elem in riga:
            nuova_riga.append(1 - elem)

        nuova_riga.reverse()
        ret.append(nuova_riga)
    return ret



# INIZIO TEST - NON TOCCARE !!!

assert flip([[]]) == [[]]
assert flip([[1]]) == [[0]]
assert flip([[1,0]]) == [[1,0]]

m1 = [ [1,0,0],
       [1,0,1] ]
r1 = [ [1,1,0],
       [0,1,0] ]
assert flip(m1) == r1
```

(continues on next page)

(continued from previous page)

```
m2 = [ [1,1,0,0],
       [0,1,1,0],
       [0,0,1,0] ]

r2 = [ [1,1,0,0],
       [1,0,0,1],
       [1,0,1,1] ]

assert flip(m2) == r2

# verifica che l'm originale non sia cambiato !
assert m2 == [ [1,1,0,0],
               [0,1,1,0],
               [0,0,1,0] ]
# FINE TEST
```

[49]:
```
def flip(matrice):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST - NON TOCCARE !!!

assert flip([[]]) == [[]]
assert flip([[1]]) == [[0]]
assert flip([[1,0]]) == [[1,0]]

m1 = [ [1,0,0],
       [1,0,1] ]
r1 = [ [1,1,0],
       [0,1,0] ]
assert flip(m1) == r1


m2 = [ [1,1,0,0],
       [0,1,1,0],
       [0,0,1,0] ]

r2 = [ [1,1,0,0],
       [1,0,0,1],
       [1,0,1,1] ]

assert flip(m2) == r2

# verifica che l'm originale non sia cambiato !
assert m2 == [ [1,1,0,0],
               [0,1,1,0],
               [0,0,1,0] ]
# FINE TEST
```

toepliz

⊛⊛⊛ RESTITUISCE `True` se la matrice come lista di liste in input è Toeplitz, mentre RESTITUISCE `False` se non lo è.

Una matrice è Toeplitz se e solo se tutti gli elementi su ogni diagonale contiene gli stessi elementi.

assumiamo che la matrice contenga sempre almeno una riga di almeno un elemento

SUGGERIMENTO: usare due `for`, nel primo scorrere la matrice per righe, nel secondo per colonne

Chiedersi: - da che riga occorre partire per la scansione? La prima è utile? - da che colonna occorre partire per la scansione? La prima è utile? - se scorriamo le righe dalla prima verso l'ultima e stiamo esaminando un certo numero ad una certa riga, che condizione deve rispettare quel numero affinchè la matrice sia toepliz ?

ESEMPIO:

```
m1 = [
        [1,2,3,4],
        [5,1,2,3],
        [9,5,1,2]
     ]

toepliz(m1)
```

Su ogni diagonale ci sono gli stessi numeri e quindi viene restituito `True`

```
m2 = [
        [1, 2, 3, 4],
        [5, 1, 4, 3],
        [9, 3, 1, 2]
     ]

toepliz(m2)
```

Restituisce `False`. Ci sono due diagonali con numeri diversi: (5,3) e (2,4,2)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[50]:
```
def toepliz(matrix):

    for i in range(1,len(matrix)):
        for j in range(1,len(matrix[0])):
            if matrix[i][j] != matrix[i-1][j-1]:
                return False
    return True


# INIZIO TEST - NON TOCCARE !
assert toepliz([ [1] ]) == True
assert toepliz([ [3,7],
                 [5,3] ]) == True
assert toepliz([ [3,7],
                 [3,5] ]) == False
assert toepliz([ [3,7],
                 [3,5] ]) == False
assert toepliz([ [3,7,9],
```

(continues on next page)

(continued from previous page)

```
                 [5,3,7] ]) == True
assert toepliz([ [3,7,9],
                 [5,3,8] ]) == False

assert toepliz([ [1,2,3,4],
                 [5,1,2,3],
                 [9,5,1,2] ]) == True

assert toepliz([ [1,2,3,4],
                 [5,9,2,3],
                 [9,5,1,2] ]) == False
# FINE TEST
```

[50]:
```
def toepliz(matrix):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST - NON TOCCARE !
assert toepliz([ [1] ]) == True
assert toepliz([ [3,7],
                 [5,3] ]) == True
assert toepliz([ [3,7],
                 [3,5] ]) == False
assert toepliz([ [3,7],
                 [3,5] ]) == False
assert toepliz([ [3,7,9],
                 [5,3,7] ]) == True
assert toepliz([ [3,7,9],
                 [5,3,8] ]) == False

assert toepliz([ [1,2,3,4],
                 [5,1,2,3],
                 [9,5,1,2] ]) == True

assert toepliz([ [1,2,3,4],
                 [5,9,2,3],
                 [9,5,1,2] ]) == False
# FINE TEST
```

Moltiplicazione di matrici

Guarda la definizione di moltiplicazione di matrici[257] su Wikipedia e prova ad implementarla nella funzione seguente.

In sostanza, data una matrice nxm A e una matrice mxp B devi ritornare come output una matrice nxp C calcolando le celle c_{ij} con la formula

$c_{ij} = a_{i1}b_{1j} + \cdots + a_{im}b_{mj} = \sum_{k=1}^{m} a_{ik}b_{kj}$

Devi riempire tutte le nxp celle di C, perciò per assicurarti di riempire un rettangolo hai bisogno di due `for`. Hai forse bisogno di un'altro `for`? Aiutati con il diagramma seguente.

[257] https://en.wikipedia.org/w/index.php?title=Matrix_multiplication&section=2#Definition

⊛⊛⊛ **ESERCIZIO**: Implementa la funzione `mul` che date le matrici `n` x `m` `mat1` e `m` x `p` `mat2`, RITORNA una NUOVA matrice `n` x `p` che è il risultato della moltiplicazione di `mat1` per `mat2`.

- Se `mat1` ha il numero di colonna diverso dal numero di righe di `mat2`, lancia un `ValueError`.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[51]:
```
def mul(mat1, mat2):

    n = len(mat1)
    m = len(mat1[0])
    p = len(mat2[0])
    if m != len(mat2):
        raise ValueError("il numero di colonne di mat1 %s deve essere uguale al␣
↪numero di righe di mat2 %s !" % (m, len(mat2)))
    ret = [[0]*p for i in range(n)]
    for i in range(n):
        for j in range(p):
            ret[i][j] = 0
            for k in range(m):
                ret[i][j] += mat1[i][k] * mat2[k][j]
    return ret
```

(continues on next page)

(continued from previous page)

```
# let's try wrong matrix dimensions:
try:
    mul([[3,5]], [[7]])
    raise Exception("AVREI DOVUTO FALLIRE !")
except ValueError:
    pass

m11 = [ [3] ]
m12 = [ [5] ]
assert mul(m11,m12) == [ [15] ]

m21 = [ [3],
        [5] ]
m22 = [ [2,6] ]

assert mul(m21,m22) == [ [3*2, 3*6],
                         [5*2, 5*6] ]

m31 = [ [3,5] ]
m32 = [ [2],
        [6] ]
assert mul(m31,m32) == [ [3*2 + 5*6] ]

m41 = [ [3,5],
        [7,1],
        [9,4] ]
m42 = [ [4,1,5,7],
        [8,5,2,7] ]
assert mul(m41,m42) == [ [52, 28, 25, 56],
                         [36, 12, 37, 56],
                         [68, 29, 53, 91] ]
```

[51]:
```
def mul(mat1, mat2):
    raise Exception('TODO IMPLEMENT ME !')

# let's try wrong matrix dimensions:
try:
    mul([[3,5]], [[7]])
    raise Exception("AVREI DOVUTO FALLIRE !")
except ValueError:
    pass

m11 = [ [3] ]
m12 = [ [5] ]
assert mul(m11,m12) == [ [15] ]

m21 = [ [3],
        [5] ]
m22 = [ [2,6] ]

assert mul(m21,m22) == [ [3*2, 3*6],
                         [5*2, 5*6] ]

m31 = [ [3,5] ]
m32 = [ [2],
```

(continues on next page)

(continued from previous page)

```
        [6] ]
assert mul(m31,m32) == [ [3*2 + 5*6] ]

m41 = [ [3,5],
        [7,1],
        [9,4] ]
m42 = [ [4,1,5,7],
        [8,5,2,7] ]
assert mul(m41,m42) == [ [52, 28, 25, 56],
                         [36, 12, 37, 56],
                         [68, 29, 53, 91] ]
```

check_nqueen

⊛⊛⊛⊛ Questo è un problema difficile ma non ti preoccupare, il resto del corso è ben più semplice !

Hai una matrice nxn di booleani che rappresenta una scacchiera dove il valore True significa che c'è una regina nella cella, e False significa cella vuota.

Ai fini della visualizzazione, possiamo rappresentare una configurazione usando `.` per significare `False` e lettere come `A` e `B` per indicare che c'è una regina in una cella. Contrariamente a quanto abbiamo fatto sino ad adesso, per convenienza mostriamo la matrice con le `j` che vanno dal basso fino alla sommità.

Vediamo un esempio. In questo caso A e B non possono attaccare ciascun'altro, perciò l'algoritmo ritorna `True`:

```
    7  ......B.
    6  ........
    5  ........
    4  ........
    3  ....A...
    2  ........
    1  ........
    0  ........
    i
     j 01234567


Vediamo perchè evidenziando le linee di attacco di A ..

    7  \...|.B.
    6  .\..|../
    5  ..\.|./.
    4  ...\|/..
    3  ----A---
    2  .../|\..
    1  ../.|.\.
    0  ./..|..\
    i
     j 01234567


... e quelle di B :

    7  ------B-
    6  ...../|\
    5  ..../.|.
```

(continues on next page)

(continued from previous page)

```
    4  .../..|.
    3  ../.A.|.
    2  ./....|.
    1  /.....|.
    0  ......|.
    i
     j 01234567
```

In quest'altro caso l'algoritmo ritornerebbe `False` perchè `A` e `B` possono attaccare ciascun altro:

```
7  \./.|...
6  -B--|--/
5  /|\.|./.
4  .|.\|/..
3  ----A---
2  .|./|\..
1  .|/.|.\.
0  ./..|..\
i
 j 01234567
```

Nel tuo algoritmo, prima devi cercare le regine. Quando ne trovi una (e per ciascuna di esse !), devi controllare se può essere colpita da un'altra regina. Vediamo come:

In questa tabella 7x7 abbiamo solo una regina A, alla posizione `i=1` e `j=4`:

```
6  ....|..
5  \...|..
4  .\..|..
3  ..\.|./
2  ...\|/.
1  ----A--
0  .../|\.
i
 j 0123456
```

Per comprendere completamente il range della regina e come calcolare le diagonali, è conveniente estendere visualmente la tabella così da avere le diagonali che intersichino gli assi. Nota inoltre che abbiamo aggiunto le lettere `y` e `x`

> **NOTA**: nell'algoritmo **non hai** bisogno di estendere la matrice ! in the algorithm you **do not** need to extend the matrix !

```
 y
 6  ....|....
 5  \...|.../
 4  .\..|../.
 3  ..\.|./..
 2  ...\|/...
 1  ----A----
 0  .../|\...
-1  ../.|.\..
-2  ./..|..\.
-3  /...|...\
 i
  j 01234567 x
```

Vediamo che la diagonale da sinistra in alto a in basso a destra interseca l'asse verticale a `y = 5` e che la diagonale da in basso a sinistra a in alto a destra interseca l'asse a `y = -3`. Dovresti usare queste informazioni per calcolare le equazioni di linea.

ESERCIZIO: Adessi dovresti avere tutti i suggerimenti necessari per procedere con l'implementazione. Implementa la funzione `check_nqueen`, che prende una matrice `nxn` di booleani che rappresentano una scacchiera dove `True` sinifica che c'è una regina nella cella, e `False` che non c'è niente. La funzione RITORNA `True` se nessuna regina può attaccare le altre, `False` altrimenti.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[52]:
```
def check_nqueen(mat):


    # equazione linea dal basso sinistra fino ad alto destra
    # y = x - 3
    # -3 = -j + i
    # y = x -j + i

    # equazione linea dall'alto sinistra al basso destra
    # y = x + 5
    # 5 = j + i
    # y = x + j + i

    n = len(mat)
    for i in range(n):
        for j in range(n):
            if mat[i][j]:  # queen is found at i,j
                for y in range(n):            # vertical scan
                    if y != i and mat[y][j]:
                        return False
                for x in range(n):            # horizontal scan
                    if x != j and mat[i][x]:
                        return False
                for x in range(n):
                    y = x + j + i        # top-left to bottom-right
                    if y >= 0 and y < n and y != i and x != j and mat[y][x]:
                        return False
                    y = x - j + i        # bottom-left to top-right
                    if y >= 0 and y < n and y != i and x != j and mat[y][x]:
                        return False

    return True


assert check_nqueen([ [True] ])
assert check_nqueen([ [True, True],
                      [False, False] ]) == False

assert check_nqueen([ [True, False],
                      [False, True] ]) == False

assert check_nqueen([ [True, False],
                      [True, False] ]) == False
```

(continues on next page)

(continued from previous page)

```
assert check_nqueen([ [True,  False, False],
                      [False, False, True],
                      [False, False, False] ]) == True

assert check_nqueen([ [True,  False, False],
                      [False, False, False],
                      [False, False, True] ]) == False


assert check_nqueen([ [False, True,  False],
                      [False, False, False],
                      [False, False, True] ]) == True

assert check_nqueen([ [False, True,  False],
                      [False, True, False],
                      [False, False, True] ]) == False
```

[52]:

```
def check_nqueen(mat):
    raise Exception('TODO IMPLEMENT ME !')

assert check_nqueen([ [True] ])
assert check_nqueen([ [True, True],
                      [False, False] ]) == False

assert check_nqueen([ [True, False],
                      [False, True] ]) == False

assert check_nqueen([ [True, False],
                      [True, False] ]) == False

assert check_nqueen([ [True,  False, False],
                      [False, False, True],
                      [False, False, False] ]) == True

assert check_nqueen([ [True,  False, False],
                      [False, False, False],
                      [False, False, True] ]) == False


assert check_nqueen([ [False, True,  False],
                      [False, False, False],
                      [False, False, True] ]) == True

assert check_nqueen([ [False, True,  False],
                      [False, True, False],
                      [False, False, True] ]) == False
```

[ ]:

4.30 Matrici - Numpy

4.30.1 Scarica zip esercizi

Naviga file online[258]

ATTENZIONE

Gli esercizi che seguono contengono dei test con gli *assert*. Per capire come svolgerli, leggi prima Gestione errori e testing[259]

4.30.2 Introduzione

Ci sono sostanzialmente due modi in Python di rappresentare matrici: come liste di liste, oppure con la libreria esterna numpy[260]. La più usata è sicuramente numpy, vediamo il motivo e le principali differenze:

Liste di liste - vedere foglio separato[261]

1. native in Python
2. non efficienti
3. le liste sono pervasive in Python, probabilmente incontrerai matrici espresse come liste di liste in ogni caso
4. forniscono un'idea di come costruire una struttura dati annidata
5. possono servire per comprendere concetti importanti come puntatori alla memoria e copie

Numpy - questo foglio

1. non nativamente disponibile in Python
2. efficiente
3. alla base di parecchie librerie di calcolo scientifico (scipy, pandas)
4. la sintassi per accedere agli elementi è lievemente diversa da quella delle liste di liste
5. in alcuni rari casi potrebbe portare problemi di installazione e/o conflitti (l'implementazione non è puro Python)

Qui vedremo i tipi di dati e comandi essenziali della libreria numpy[262], ma non ci addentreremo nei dettagli.

L'idea è semplicemente passare ad usare il formato dati `ndarray` senza badare molto alle performance: per esempio, anche se i cicli `for` in Python sono lenti perchè operano cella per cella, li useremo comunque. Qualora tu abbia effettivamente necessità di eseguire calcoli velocemente, vorrete usare operazioni su vettori ma per questo invitiamo alla lettura dei link qua sotto.

ATTENZIONE: se vuoi usare Numpy in Python tutor[263], invece dell'interprete di default `Python 3.6` dovete selezionare `Python 3.6 with Anaconda` (che a Maggio 2019 risulta marcato come sperimentale)

Per riferimenti, vedere:

[258] https://github.com/DavidLeoni/softpython-it/tree/master/matrices-numpy
[259] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html
[260] https://www.numpy.org
[261] https://it.softpython.org/matrices-lists/matrices-lists-sol.html
[262] https://www.numpy.org
[263] http://www.pythontutor.com/visualize.html#mode=edit

- i tutorial Nicola Zoppetti, parte Numpy[264]
- Python Data Science Handbook, parte Numpy (inglese)[265]

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
matrices-numpy
    matrices-numpy.ipynb
    matrices-numpy-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `matrices-numpy.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive. Gli esercizi sono graduati per difficoltà, da una stellina ⊛ a quattro ⊛⊛⊛⊛

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

Creare una matrice

[1]:
```
# Innanzitutto importiamo la libreria, e per comodità la rinominiamo in 'np'

import numpy as np
```

Con le liste di liste abbiamo spesso costruito le matrici una riga alla volta, aggiundo liste all'occorrenza. In numpy invece di solito si crea in un colpo solo tutta la matrice, riempiendola di zeri.

In particolare, questo comando crea un ndarray riempito di zeri:

[2]:
```
mat = np.zeros( (2,3)  )    # 2 righe, 3 colonne
```

[3]:
```
mat
```

[3]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

Nota come all'interno di `array( )` il contenuto sembra che venga rappresentato come una lista di liste, MA in realtà nelle memoria fisica i dati sono strutturati in una sequenza lineare che permette a Python di accedere ai numeri in modo molto più rapido.

Possiamo anche crearci un `ndarray` a partire da una lista di liste:

[264] http://www.ifac.cnr.it/~zoppetti/corsopython/
[265] https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html

[4]:
```
mat = np.array( [ [5.0,8.0,1.0],
                  [4.0,3.0,2.0]])
```

[5]:
```
mat
```

[5]:
```
array([[5., 8., 1.],
       [4., 3., 2.]])
```

[6]:
```
type(mat)
```

[6]:
```
numpy.ndarray
```

Dimensioni di una matrice

Per ottenere le dimensioni, scriviamo così:

> **ATTENZIONE**: dopo `shape` **non** ci sono le parentesi tonde !
>
> `shape` è un attributo, non una funzione da chiamare

[7]:
```
mat = np.array( [ [5.0,8.0,1.0],
                  [4.0,3.0,2.0]])

mat.shape
```

[7]:
```
(2, 3)
```

Se vogliamo memorizzare le dimensioni in variabili separate, possiamo usare questo modo più pythonico: (notare la virgola tra num_righe e num_colonne):

[8]:
```
num_righe, num_colonne = mat.shape
```

[9]:
```
num_righe
```

[9]:
```
2
```

[10]:
```
num_colonne
```

[10]:
```
3
```

Lettura e scrittura

Per accedere ai dati o sovrascriverli si utilizza la notazione con le quadre, con l'importante differenza che in numpy è consentito scrivere *entrambi* gli indici *dentro* le stesse quadre, separati da una virgola:

ATTENZIONE: la notazione `mat[i,j]` è solo per numpy!

Con le liste di liste **non** funziona.

[11]:
```
mat = np.array( [ [5.0,8.0,1.0],
                  [4.0,3.0,2.0]])

# mettiamo il numero 0 nella cella alla riga 0 e colonna 1
```

(continues on next page)

(continued from previous page)

```
mat[0,1] = 9
```

[12]:
```
mat
```

[12]:
```
array([[5., 9., 1.],
       [4., 3., 2.]])
```

[13]:
```
# Accediamo alla cella alla riga 0 e colonna 1

mat[0,1]
```

[13]:
```
9.0
```

[14]:
```
# mettiamo il numero 7 nella cella alla riga 1 e colonna 2

mat[1,2] = 7
```

[15]:
```
mat
```

[15]:
```
array([[5., 9., 1.],
       [4., 3., 7.]])
```

⊛ **ESERCIZIO**: prova a scrivere così: che succede?

```
mat[0,0] = "c"
```

[16]:
```
# scrivi qui
```

[17]:
```
mat[1,1]
```

[17]:
```
3.0
```

⊛ **ESERCIZIO**: Prova a scrivere così e vedere che succede:

```
mat[1,1.0]
```

[18]:
```
# scrivi qui
```

4.30.3 NaN e infinità

I numeri float possono essere numeri, *non numeri* , e anche *infinità* . A volte durante i calcoli accadono condizioni estreme, come per esempio dividere un piccolo numero per un numero enorme. In tali casi, potresti finire con un float che è il temuto *Not a Number* , *NaN* in breve, o potresti ottenere una *infinità* . Questo potrebbe portare a comportamenti terribilmente imprevedibili, perciò devi saper riconoscere situazioni potenzialmente problematiche.

I comportamenti descritti in seguito sono dettati dallo standard IEEE per l'Aritmetica in virgola mobile binaria (IEEE 754) usato da Numpy e che è implementato in tutti i processori (*CPU*), perciò di fatto riguarda *tutti* i linguaggi di programmazione.

NaN

Un NaN dal nome *Non è un Numero*. Che è già un nome poco chiaro, visto che il NaN in realtà è un membro molto speciale dei floats, con questa stupefacente proprietà:

> **ATTENZIONE: NaN NON E' UGUALE A SE' STESSO !!!**
>
> Sì hai letto bene, NaN davvero *non* è uguale a sè stesso.

Persino se la tua mente vuole rifiutare questa nozione, la confermeremo a breve.

Per ottenere un NaN, puoi usare il modulo Python `math` che contiene questo oggetto alieno:

[19]:
```
import math
math.nan    # nota che stampa 'nan' con n minuscolo
```

[19]:
```
nan
```

Come detto, un NaN è considerato un float:

[20]:
```
type(math.nan)
```

[20]:
```
float
```

Eppure, si comporta molto diversamente dai suoi compagni float, o da ogni altro oggetto nelluniverso conosciuto:

[21]:
```
math.nan == math.nan   # Eh ????
```

[21]:
```
False
```

Rilevare i NaN

Dato quanto sopra, se vuoi controllare se una variabile `x` è un NaN, *non* puoi scrivere così:

[22]:
```
x = math.nan
if x == math.nan:  # SBAGLIATO
    print("Sono un NaN ")
else:
    print("x è qualcos'altro ??")
```

```
x è qualcos'altro ??
```

Per gestire correttamente questa situazione, devi usare la funzione `math.isnan`:

[23]:
```
x = math.nan
if math.isnan(x):  # CORRETTO
    print("x è un NaN ")
else:
    print("x è qualcos'altro ??")
```

```
x è un NaN
```

Nota che `math.isnan` funziona anche con NaN *negativi*:

[24]:
```
y = -math.nan
if math.isnan(y):  # CORRETTO
    print("y è un NaN ")
```

(continues on next page)

(continued from previous page)

```
else:
    print("y è quacos'altro ??")
```

```
y è un NaN
```

Sequenze con i NaN

Per fortuna, non tutto è completamente assurdo. Se compari sequenze che contengono NaN ad altre, ottieni risultati ragionevoli:

[25]:
```
[math.nan, math.nan] == [math.nan, math.nan]
```

[25]:
```
True
```

[26]:
```
[math.nan, math.nan] == [math.nan, 5.0]
```

[26]:
```
False
```

Esercizio - NaN due variabili

Date due variabili `x` e `y`, scrivi del codice che stampa `"stessa cosa"` quando sono lo stesso, *anche* quando sono `NaN`. Altrimenti, stampa `"non sono la stessa cosa"`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[27]:
```
import math

# output atteso: stessa cosa
x = math.nan
y = math.nan

# output atteso: non sono la stessa cosa
#x = 3
#y = math.nan

# output atteso: non sono la stessa cosa
#x = math.nan
#y = 5

# output atteso: non sono la stessa cosa
#x = 2
#y = 7

# output atteso: stessa cosa
#x = 4
#y = 4

# scrivi qui
if math.isnan(x) and math.isnan(y):
    print('stessa cosa')
elif x == y:
    print('stessa cosa')
```

(continues on next page)

(continued from previous page)

```
else:
    print('non sono la stessa cosa')
```

```
stessa cosa
```

[27]:
```
import math

# output atteso: stessa cosa
x = math.nan
y = math.nan

# output atteso: non sono la stessa cosa
#x = 3
#y = math.nan

# output atteso: non sono la stessa cosa
#x = math.nan
#y = 5

# output atteso: non sono la stessa cosa
#x = 2
#y = 7

# output atteso: stessa cosa
#x = 4
#y = 4

# scrivi qui
```

```
stessa cosa
```

Operazioni sui NaN

Qualunque operazione sui NaN genera un altro NaN:

[28]:
```
5 * math.nan
```

[28]:
```
nan
```

[29]:
```
math.nan + math.nan
```

[29]:
```
nan
```

[30]:
```
math.nan / math.nan
```

[30]:
```
nan
```

L'unica cosa che non puoi fare è dividere per zero un NaN 'fuori scatola':

```
math.nan / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-94-1da38377fac4> in <module>
----> 1 math.nan / 0

ZeroDivisionError: float division by zero
```

NaN corrisponde al valore logico booleano `True`:

[31]:
```
if math.nan:
    print("That's True")
```

```
That's True
```

I NaN e Numpy

Quando usi Numpy è abbastanza probabile incontrare NaN, al punto che sono ridefiniti dentro Numpy - ma di fatto sono esattamente gli stessi che nel modulo `math`:

[32]:
```
np.nan
```

[32]:
```
nan
```

[33]:
```
math.isnan(np.nan)
```

[33]:
```
True
```

[34]:
```
np.isnan(math.nan)
```

[34]:
```
True
```

In Numpy quando hai numeri sconosciuti potresti essere tentato di mettere un `None`. Puoi anche farlo, ma guarda attentamente il risultato:

[35]:
```
import numpy as np
np.array([4.9,None,3.2,5.1])
```

[35]:
```
array([4.9, None, 3.2, 5.1], dtype=object)
```

L'array risultante *non* è un array di float64 che permette calcoli veloci, invece è un array che contiene generici *object* , perchè Numpy assume che l'array contenga dati eterogenei. Perciò quello che guadagni in generalità lo perdi in performance, che dovrebbe essere il motivo principale di usare Numpy.

Per quanto appaiano strani, i NaN sono considerati come dei `float` e quindi possono essere salvati nell'array:

[36]:
```
np.array([4.9,np.nan,3.2,5.1])    # NOTA: il `dtype=object` è sparito
```

[36]:
```
array([4.9, nan, 3.2, 5.1])
```

Dove sono i NaN ?

Vediamo dove possiamo incontrare dei NaN e altri oggetti strani come le infinità.

Prima, controlliamo cosa succede quando chiamiamo la funzione `log` del modulo standard `math`. Dalle lezioni di matematica, sappiamo che la funzione `log` si comporta così:

- $x < 0$: non definita
- $x = 0$: tende a meno infinito
- $x > 0$: definita

Perciò possiamo chiederci cosa succede se gli passiamo un valore per il quale non è definita. Proviamo prima con `math.log` della libreria standard di Python:

```
>>> math.log(-1)
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-38-d6e02ba32da6> in <module>
----> 1 math.log(-1)

ValueError: math domain error
```

In questo caso viene sollevato `ValueError` **e l'esecuzione viene interrotta**.

Vediamo ora l'equivalente in Numpy:

```
[37]: np.log(-1)
```

```
/home/da/.local/lib/python3.5/site-packages/ipykernel_launcher.py:1: RuntimeWarning:␣
↪invalid value encountered in log
  """Entry point for launching an IPython kernel.
```

```
[37]: nan
```

In questo caso **abbiamo ottenuto come risultato** `np.nan`, quindi l'esecuzione non si è interrotta, Jupyter ci ha solo informato con una stampa addizionale che abbiamo compiuto qualcosa di pericoloso.

Il comportamento di default di Numpy quando incontra calcoli pericolosi è effettuare in ogni caso il calcolo e salvare il risultato come NaN o altri oggetti limite. Questo vale anche per i calcoli sugli array:

[38]:
```
np.log(np.array([3,7,-1,9]))
```

```
/home/da/.local/lib/python3.5/site-packages/ipykernel_launcher.py:1: RuntimeWarning:␣
↪invalid value encountered in log
  """Entry point for launching an IPython kernel.
```

[38]:
```
array([1.09861229, 1.94591015,        nan, 2.19722458])
```

Infinità

Come abbiamo detto in precedenza, Numpy usa lo standard IEEE per l'aritmetica binaria in virgola mobile (IEEE 754). Dato che qualcuno all'IEEE ha deciso di racchiudere i misteri dell'infinito nei numeri float, abbiamo ancora un'altro cittadino da considerare quando facciamo calcoli (per altre informazioni, vedere Numpy documentation on constants[266]):

Infinità positiva `np.inf`

[39]:
```
 np.array( [ 5 ] ) / 0
```

```
/home/da/.local/lib/python3.5/site-packages/ipykernel_launcher.py:1: RuntimeWarning:␣
↪divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[39]:
```
array([inf])
```

[40]:
```
np.array( [ 6,9,5,7 ] ) / np.array( [ 2,0,0,4 ] )
```

```
/home/da/.local/lib/python3.5/site-packages/ipykernel_launcher.py:1: RuntimeWarning:␣
↪divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[40]:
```
array([3.  ,  inf,  inf, 1.75])
```

Attenzione che:

- Not a Number **non** è equivalente all'infinità
- l'infinità positiva **non** è equivalente all'infinità negativa
- l'infinità è equivalente all'infinità positiva

Questa volta, l'infinità è equivalente all'infinità:

[41]:
```
np.inf == np.inf
```

[41]:
```
True
```

perciò possiamo in sicurezza equiparare due infinità con ==:

[42]:
```
x = np.inf

if x == np.inf:
    print("x è infinito")
else:
    print("x è finito")
```

```
x è infinito
```

Alternativamente, possiamo usare la funzione `np.isinf`:

[266] https://numpy.org/devdocs/reference/constants.html

[43]:
```
np.isinf(np.inf)
```
[43]:
```
True
```

Infinità negativa

Possiamo anche avere un'infinità negativa, che è differente dall'inifinità positiva:

[44]:
```
-np.inf == np.inf
```
[44]:
```
False
```

Nota che `isinf` può rilevare sia infinità positive che negative:

[45]:
```
np.isinf(-np.inf)
```
[45]:
```
True
```

Per rilevare specificamente un'infinità negativa dei usare `isneginf`:

[46]:
```
np.isneginf(-np.inf)
```
[46]:
```
True
```

[47]:
```
np.isneginf(np.inf)
```
[47]:
```
False
```

Dove possiamo trovarle? Come esempio, proviamo la funzione `np.log`:

[48]:
```
np.log(0)
```
```
/home/da/.local/lib/python3.5/site-packages/ipykernel_launcher.py:1: RuntimeWarning:␣
↪divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```
[48]:
```
-inf
```

Combinare infinità e NaN

Quando esegui operazioni che riguardano le infinità e i NaN, l'aritmetica IEEE prova a imitare l'analisi classica, a volte includendo NaN come risultato:

[49]:
```
np.inf + np.inf
```
[49]:
```
inf
```

[50]:
```
- np.inf - np.inf
```
[50]:
```
-inf
```

[51]:
```
np.inf * -np.inf
```
[51]:
```
-inf
```

Un risultato che in analisi classica sarebbe non definito, qui diventa NaN:

[52]:
```
np.inf - np.inf
```
[52]:
```
nan
```

[53]:
```
np.inf / np.inf
```
[53]:
```
nan
```

Come al solito, combinare con un NaN risulta in NaN:

[54]:
```
np.inf + np.nan
```
[54]:
```
nan
```

[55]:
```
np.inf / np.nan
```
[55]:
```
nan
```

Zero negativo

Puoi persino avere uno zero *negativo* - chi l'avrebbe pensato?

[56]:
```
np.NZERO
```
[56]:
```
-0.0
```

Lo zero negativo naturalmente fa coppia bene con il più conosciuto e apprezzato zero *positivo*:

[57]:
```
np.PZERO
```
[57]:
```
0.0
```

NOTA: Scrivere `np.NZERO` o `-0.0` è *esattamente* la stessa cosa. Lo stesso vale per lo zero positivo.

A questo punto, potresti cominciare a chiederti con qualche se sono davvero considerati *uguali*. Verifichiamo:

[58]:
```
0.0 == -0.0
```
[58]:
```
True
```

Grandioso! Finalmente qualcosa che ha senso.

Dato quanto sopra, potresti pensare che in una formula puoi sostituire one per l'altro e ottenere gli stessi risultati, in armonia con le regole dell'universo.

Facciamo un tentativo di sostituzione, come esempio prima cercheremo di dividere un numero per uno zero positivo (persino se gli insegnanti di matematica ci dicono che tali divisioni siano vietate) - cosa potremmo mai ottenere?

$\frac{5.0}{0.0} = ???$

In termini di Numpy, potremmo scrivere così per 'inscatolare' tutto in arrays:

[59]:
```
np.array( [ 5.0 ] ) / np.array( [ 0.0 ] )
```
```
/home/da/.local/lib/python3.5/site-packages/ipykernel_launcher.py:1: RuntimeWarning:␣
↪divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```
[59]:
```
array([inf])
```

Mmm, abbiamo ottenuto un array con dentro `np.inf`.

Se `0.0` e `-0.0` sono davvero la stessa cosa, dividendo un numero per `-0.0` dovremmo ottenere lo stesso identico risultato, no?

Proviamo:

[60]:
```
np.array( [ 5.0 ] ) / np.array( [ -0.0 ] )
```

```
/home/da/.local/lib/python3.5/site-packages/ipykernel_launcher.py:1: RuntimeWarning:␣
↪divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[60]:
```
array([-inf])
```

Ecchecaspita. Questa volta ci ritroviamo con una infinità *negativa* `-np.inf`

Se tutto ciò ti pare strano, non dare la colpa a Numpy o Python. Questo è il modo con cui praticamente ogni processore (*CPU*) compie operazioni in virgola mobile, perciò lo troverai in quasi TUTTI i linguaggi di programmazione.

Quello che i linguaggi di programmazione possono fare è aggiungere ulteriori controlli per proteggerti da queste situazioni paradossali, come per esempio lanciare `ZeroDivisionError` quando scrivi direttamente `1.0/0.0` (bloccando quindi l'esecuzione) o stampare un warning nel caso di operazioni su array Numpy.

Esercizio: rilevare numeri propri

Scrivi del codice che STAMPA `numeri uguali` se due numeri `x` e `y` sono uguali e veri numeri, e STAMPA `non uguali` altrimenti.

NOTA: `numeri non uguali` va stampato se uno qualunque dei numeri è infinito o NaN.

Per risolverlo, sentiti libero di chiamare funzioni indicate nella documentazione di Numpy riguardo le costanti[267]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[61]:
```
import numpy as np

# atteso: numeri uguali
x = 5
y = 5

# atteso: numeri non uguali
#x = np.inf
#y = 3

# atteso: numeri non uguali
#x = 3
#y = np.inf

# atteso: numeri non uguali
#x = np.inf
#y = np.nan

# atteso: numeri non uguali
#x = np.nan
#y = np.inf
```

(continues on next page)

[267] https://docs.scipy.org/doc/numpy/reference/constants.html

(continued from previous page)

```
# atteso: numeri non uguali
#x = np.nan
#y = 7

# atteso: numeri non uguali
#x = 9
#y = np.nan

# atteso: numeri non uguali
#x = np.nan
#y = np.nan


# scrivi qui

# SOLUZIONE 1 - quella brutta
if np.isinf(x) or np.isinf(y) or np.isnan(x) or np.isnan(y):
    print('numeri non uguali')
else:
    print('numeri uguali')

# SOLUZIONE 2 - quella bella
if np.isfinite(x) and np.isfinite(y):
    print('numeri uguali')
else:
    print('numeri non uguali')
```

```
numeri uguali
numeri uguali
```

[61]:
```
import numpy as np

# atteso: numeri uguali
x = 5
y = 5

# atteso: numeri non uguali
#x = np.inf
#y = 3

# atteso: numeri non uguali
#x = 3
#y = np.inf

# atteso: numeri non uguali
#x = np.inf
#y = np.nan

# atteso: numeri non uguali
#x = np.nan
#y = np.inf

# atteso: numeri non uguali
#x = np.nan
```

(continues on next page)

(continued from previous page)

```
#y = 7

# atteso: numeri non uguali
#x = 9
#y = np.nan

# atteso: numeri non uguali
#x = np.nan
#y = np.nan


# scrivi qui
```

```
numeri uguali
numeri uguali
```

Domande - NaN

Per ciascuna delle espressioni seguenti, prova ad indovinare il risultato

> **ATTENZIONE: ciò che segue può causare nausea e gravi convulsioni.**
>
> Durante i test clinici, sia pazienti con inclinazioni matematiche che soggetti con repulsione per le scienze esatte hanno lamentato malessere per ragioni differenti che sono ancora oggetto di ricerca.

```
a.  0.0 * -0.0
b.  (-0.0)**3
c.  np.log(-7) == math.log(-7)
d.  np.log(-7) == np.log(-7)
e.  np.isnan( 1 / np.log(1) )
f.  np.sqrt(-1) * np.sqrt(-1)    # sqrt = square root
g.  3 ** np.inf
h   3 ** -np.inf
i.  1/np.sqrt(-3)
j.  1/np.sqrt(-0.0)
m.  np.sqrt(np.inf) - np.sqrt(-np.inf)
n.  np.sqrt(np.inf) + ( 1 / np.sqrt(-0.0) )
o.  np.isneginf(np.log(np.e) / np.sqrt(-0.0))
p.  np.isinf(np.log(np.e) / np.sqrt(-0.0))
q.  [np.nan, np.inf] == [np.nan, np.inf]
r.  [np.nan, -np.inf] == [np.nan, np.inf]
s.  [np.nan, np.inf] == [-np.nan, np.inf]
```

4.30.4 Verifica comprensione

ATTENZIONE

Gli esercizi che seguono contengono dei test con gli *assert*. Per capire come svolgerli, leggi prima Gestione errori e testing[268]

Prova adesso a implementare queste funzioni

quadro

⊛⊛⊛ Restituisce una NUOVA matrice numpy di n righe e n colonne, in cui tutti i valori sono a zero eccetto quelli sui bordi, che devono essere uguali a k

Per esempio, `quadro(4, 7.0)` deve dare:

```
array([[7.0, 7.0, 7.0, 7.0],
       [7.0, 0.0, 0.0, 7.0],
       [7.0, 0.0, 0.0, 7.0],
       [7.0, 7.0, 7.0, 7.0]])
```

Ingredienti:

- crea una matrice pieni di zeri. ATTENZIONE: quali dimensioni ha? Bisogna usare `n` o `k` ? Leggi BENE il testo.
- comincia riempiendo le celle della prima riga con i valori `k`. Per iterare lungo le colonne della prima riga, usa un `for j in range(n)`
- riempi le altre righe e colonne, usando opportuni `for`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[62]:
```
def quadro(n, k):

    mat = np.zeros( (n,n)  )
    for i in range(n):
        mat[0, i] = k
        mat[i, 0] = k
        mat[i, n-1] = k
        mat[n-1, i] = k
    return mat




r1 = np.array( [[7.0, 7.0, 7.0, 7.0],
                [7.0, 0.0, 0.0, 7.0],
                [7.0, 0.0, 0.0, 7.0],
                [7.0, 7.0, 7., 7.0]])

# all_close ritorna True se tutti i valori nella prima matrice sono abbastanza vicini
# (cioè entro una certa tolleranza) ai corrispondenti nella seconda
```

(continues on next page)

[268] https://it.softpython.org/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
assert np.allclose(quadro(4, 7.0), r1)

r2 = np.array( [ [7.0] ])
assert np.allclose(quadro(1, 7.0), r2)

r3 = np.array( [ [7.0, 7.0],
                 [7.0, 7.0]])
assert np.allclose(quadro(2, 7.0), r3)
```

[62]:
```
def quadro(n, k):
    raise Exception('TODO IMPLEMENT ME !')



r1 = np.array( [[7.0, 7.0, 7.0, 7.0],
                [7.0, 0.0, 0.0, 7.0],
                [7.0, 0.0, 0.0, 7.0],
                [7.0, 7.0, 7., 7.0]])

# all_close ritorna True se tutti i valori nella prima matrice sono abbastanza vicini
# (cioè entro una certa tolleranza) ai corrispondenti nella seconda
assert np.allclose(quadro(4, 7.0), r1)

r2 = np.array( [ [7.0] ])
assert np.allclose(quadro(1, 7.0), r2)

r3 = np.array( [ [7.0, 7.0],
                 [7.0, 7.0]])
assert np.allclose(quadro(2, 7.0), r3)
```

media_righe

⊛⊛⊛ Prende una matrice numpy n x m e RITORNA una NUOVA matrice numpy di una sola colonna in cui i valori sono la media dei valori nelle corrispondenti righe della matrice in input

Esempio:

Input: matrice 5x4

```
3 2 1 4
6 2 3 5
4 3 6 2
4 6 5 4
7 2 9 3
```

Output: matrice 5x1

```
(3+2+1+4)/4
(6+2+3+5)/4
(4+3+6+2)/4
(4+6+5+4)/4
(7+2+9+3)/4
```

Ingredienti:

- create una matrice n x 1 da ritornare, riempiendola di zeri
- visitate tutte le celle della matrice originale con due for in range annidati
- durante la visita, accumulate nella matrice da ritornare la somma degli elementi presi da ciascuna riga della matrice originale
- una volta completata la somma di una riga, potete dividerla per la dimensione ncolonne della matrice originale
- ritornate la matrice

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[63]:
```
def media_righe(mat):

    righe, colonne = mat.shape

    ret = np.zeros( (righe,1)  )

    for i in range(righe):

        for j in range(colonne):
            ret[i] += mat[i,j]

        ret[i] = ret[i] / colonne
        # per brevità potremmo anche scrivere
        # ret[i] /= colonne

    return ret

m1 = np.array([  [5.0]  ])
r1 = np.array([  [5.0]  ])
assert np.allclose(media_righe(m1), r1)

m2 = np.array([ [5.0, 3.0] ])
r2 = np.array([ [4.0]  ])
assert np.allclose(media_righe(m2), r2)

m3 = np.array(
    [[3,2,1,4],
     [6,2,3,5],
     [4,3,6,2],
     [4,6,5,4],
     [7,2,9,3]])

r3 = np.array([ [(3+2+1+4)/4],
                [(6+2+3+5)/4],
                [(4+3+6+2)/4],
                [(4+6+5+4)/4],
                [(7+2+9+3)/4] ])

assert np.allclose(media_righe(m3), r3)
```

[63]:
```
def media_righe(mat):
    raise Exception('TODO IMPLEMENT ME !')
    return ret
```

(continues on next page)

(continued from previous page)

```
m1 = np.array([  [5.0]  ])
r1 = np.array([  [5.0]  ])
assert np.allclose(media_righe(m1), r1)

m2 = np.array([ [5.0, 3.0] ])
r2 = np.array([ [4.0]  ])
assert np.allclose(media_righe(m2), r2)

m3 = np.array(
    [[3,2,1,4],
     [6,2,3,5],
     [4,3,6,2],
     [4,6,5,4],
     [7,2,9,3]])

r3 = np.array([ [(3+2+1+4)/4],
                [(6+2+3+5)/4],
                [(4+3+6+2)/4],
                [(4+6+5+4)/4],
                [(7+2+9+3)/4] ])

assert np.allclose(media_righe(m3), r3)
```

matrot

⊛⊛⊛ RITORNA una NUOVA matrice numpy che ha i numeri della matrice numpy di input ruotati di una colonna.

Per ruotati intendiamo che:

- se un numero nella matrice di input si trova alla colonna j, nella matrice di output si troverà alla colonna j+1 nella stessa riga.
- Se un numero si trova nell'ultima colonna, nella matrice di output si troverà nella colonna zeresima.

Esempio:

Se abbiamo come input

```
np.array(   [
                [0,1,0],
                [1,1,0],
                [0,0,0],
                [0,1,1]
            ])
```

Ci aspettiamo come output

```
np.array(   [
                [0,0,1],
                [0,1,1],
                [0,0,0],
                [1,0,1]
            ])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a><div class="jupman-sol jupman-sol-code"

style="display:none">

[64]:
```
import numpy as np

def matrot(matrice):

    ret = np.zeros(matrice.shape)

    for i in range(matrice.shape[0]):
        ret[i,0] = matrice[i,-1]
        for j in range(1, matrice.shape[1]):
            ret[i,j] = matrice[i,j-1]
    return ret


m1 = np.array(  [ [1] ])
r1 = np.array( [  [1]  ])
assert np.allclose(matrot(m1), r1)

m2 = np.array(  [ [0,1] ])
r2 = np.array( [ [1,0] ])
assert np.allclose(matrot(m2), r2)

m3 = np.array([ [0,1,0] ])
r3 = np.array([ [0,0,1] ])
assert np.allclose(matrot(m3), r3)

m4 = np.array(  [[0,1,0],
                 [1,1,0]])
r4 = np.array(  [[0,0,1],
                 [0,1,1]])
assert np.allclose(matrot(m4), r4)


m5 = np.array([ [0,1,0],
                [1,1,0],
                [0,0,0],
                [0,1,1]])
r5 = np.array([ [0,0,1],
                [0,1,1],
                [0,0,0],
                [1,0,1]])
assert np.allclose(matrot(m5), r5)
```

[64]:
```
import numpy as np

def matrot(matrice):
    raise Exception('TODO IMPLEMENT ME !')

m1 = np.array(  [ [1] ])
r1 = np.array( [  [1]  ])
assert np.allclose(matrot(m1), r1)

m2 = np.array(  [ [0,1] ])
r2 = np.array( [ [1,0] ])
assert np.allclose(matrot(m2), r2)
```

(continues on next page)

(continued from previous page)

```
m3 = np.array([ [0,1,0] ])
r3 = np.array([ [0,0,1] ])
assert np.allclose(matrot(m3), r3)

m4 = np.array(  [[0,1,0],
                 [1,1,0]])
r4 = np.array(  [[0,0,1],
                 [0,1,1]])
assert np.allclose(matrot(m4), r4)


m5 = np.array([ [0,1,0],
                [1,1,0],
                [0,0,0],
                [0,1,1]])
r5 = np.array([ [0,0,1],
                [0,1,1],
                [0,0,0],
                [1,0,1]])
assert np.allclose(matrot(m5), r5)
```

disp

⊛⊛⊛ Prende una matrice Numpy `mat` di dimensioni`nrighe` x `ncol` contenente numeri interi in input e RITORNA una NUOVA matrice numpy di dimensioni `nrighe` x `ncol` che è come quella originale, ma nelle celle che contenevano numeri pari adesso ci saranno numero dispari ottenuti sommando `1` al numero pari esistente.

Esempio:

```
disp(np.array( [
                    [2,5,6,3],
                    [8,4,3,5],
                    [6,1,7,9]
               ]))
```

Deve dare in output

```
array([[ 3.,  5.,  7.,  3.],
       [ 9.,  5.,  3.,  5.],
       [ 7.,  1.,  7.,  9.]])
```

Suggerimenti:

- Visto che dovete ritornare una nuova matrice, cominciate con il crearne una vuota
- scorrete con indici i e j tutta la matrice iniziale

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[65]:
```
import numpy as np

def disp(mat):
```

(continues on next page)

(continued from previous page)

```
    nrighe, ncol = mat.shape
    ret = np.zeros( (nrighe, ncol) )

    for i in range(nrighe):
        for j in range(ncol):
            if mat[i,j] % 2 == 0:
                ret[i,j] = mat[i,j] + 1
            else:
                ret[i,j] = mat[i,j]
    return ret


m1 = np.array([ [2] ])
r1 = np.array([ [3] ])
assert np.allclose(disp(m1), r1)
assert m1[0][0] == 2  # controlla non si stia modificando la matrice originale

m2 = np.array( [ [2,5,6,3],
                 [8,4,3,5],
                 [6,1,7,9]
               ])
r2 = np.array( [ [3,5,7,3],
                 [9,5,3,5],
                 [7,1,7,9]])
assert np.allclose(disp(m2), r2)
```

[65]:
```
import numpy as np

def disp(mat):
    raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([ [2] ])
r1 = np.array([ [3] ])
assert np.allclose(disp(m1), r1)
assert m1[0][0] == 2  # controlla non si stia modificando la matrice originale

m2 = np.array( [ [2,5,6,3],
                 [8,4,3,5],
                 [6,1,7,9]
               ])
r2 = np.array( [ [3,5,7,3],
                 [9,5,3,5],
                 [7,1,7,9]])
assert np.allclose(disp(m2), r2)
```

radalt

⊛⊛⊛ Prende una matrice numpy `mat` di dimensioni`nrighe` x `ncol` contenente numeri interi in input e RITORNA una NUOVA matrice numpy di dimensioni `nrighe` x `ncol`, avente alle righe di **indice** pari i numeri della matrice originale moltiplicati per due, e alle righe di **indice** dispari gli stessi numeri della matrice originale

Esempio:

```
m  = np.array( [                        #  indice
                    [ 2, 5, 6, 3],      #    0      pari
                    [ 8, 4, 3, 5],      #    1      dispari
                    [ 7, 1, 6, 9],      #    2      pari
                    [ 5, 2, 4, 1],      #    3      dispari
                    [ 6, 3, 4, 3]       #    4      pari
               ])
```

Una chiamata a

```
radalt(m)
```

ritornerà la matrice numpy

```
array([[ 4, 10, 12,  6],
       [ 8,  4,  3,  5],
       [14,  2, 12, 18],
       [ 5,  2,  4,  1],
       [12,  6,  8,  6]])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[66]:
```
import numpy as np

def radalt(mat):

    nrighe, ncol = mat.shape
    ret = np.zeros( (nrighe, ncol) )

    for i in range(nrighe):
        for j in range(ncol):
            if i % 2 == 0:
                ret[i,j] = mat[i,j] * 2
            else:
                ret[i,j] = mat[i,j]
    return ret


# INIZIO TEST: NON TOCCARE !

m1 = np.array([ [2] ])
r1 = np.array([ [4] ])
assert np.allclose(radalt(m1),r1)
assert m1[0][0] == 2  # controlla non si stia modificando la matrice originale

m2 = np.array( [ [ 2, 5, 6],
                 [ 8, 4, 3]])
r2 = np.array( [ [ 4,10,12],
```

(continues on next page)

(continued from previous page)

```
                 [ 8, 4, 3]])
assert np.allclose(radalt(m2), r2)

m3 = np.array( [ [ 2, 5, 6, 3],
                 [ 8, 4, 3, 5],
                 [ 7, 1, 6, 9],
                 [ 5, 2, 4, 1],
                 [ 6, 3, 4, 3] ])
r3 = np.array( [ [ 4,10,12, 6],
                 [ 8, 4, 3, 5],
                 [14, 2,12,18],
                 [ 5, 2, 4, 1],
                 [12, 6, 8, 6] ])
assert np.allclose(radalt(m3), r3)
# FINE TEST
```

[66]:
```
import numpy as np

def radalt(mat):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST: NON TOCCARE !

m1 = np.array([ [2] ])
r1 = np.array([ [4] ])
assert np.allclose(radalt(m1),r1)
assert m1[0][0] == 2  # controlla non si stia modificando la matrice originale

m2 = np.array( [ [ 2, 5, 6],
                 [ 8, 4, 3]])
r2 = np.array( [ [ 4,10,12],
                 [ 8, 4, 3]])
assert np.allclose(radalt(m2), r2)

m3 = np.array( [ [ 2, 5, 6, 3],
                 [ 8, 4, 3, 5],
                 [ 7, 1, 6, 9],
                 [ 5, 2, 4, 1],
                 [ 6, 3, 4, 3] ])
r3 = np.array( [ [ 4,10,12, 6],
                 [ 8, 4, 3, 5],
                 [14, 2,12,18],
                 [ 5, 2, 4, 1],
                 [12, 6, 8, 6] ])
assert np.allclose(radalt(m3), r3)
# FINE TEST
```

scacchiera

⊛⊛⊛ RITORNA una NUOVA matrice numpy di `n` righe e `n` colonne, in cui le celle si alternano tra zero e uno.

Esempio: `scacchiera(4)` deve dare:

```
array([[1.0, 0.0, 1.0, 0.0],
       [0.0, 1.0, 0.0, 1.0],
       [1.0, 0.0, 1.0, 0.0],
       [0.0, 1.0, 0.0, 1.0]])
```

Ingredienti:

- per alternare, potete usare la range nella forma in cui prende 3 parametri, per esempio `range(0,n,2)` parte da `0`, arriva fino a `n` escluso e salta di due in due, generando 0,2,4,6,8, …
- invece `range(1,n,2)` genererebbe 1,3,5,7, …

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[67]:
```
def scacchiera(n):

    mat = np.zeros( (n,n)  )

    for i in range(0,n, 2):
        for j in range(0,n, 2):
            mat[i, j] = 1

    for i in range(1,n, 2):
        for j in range(1,n, 2):
            mat[i, j] = 1

    return mat


r1 = np.array( [ [1.0, 0.0, 1.0, 0.0],
                 [0.0, 1.0, 0.0, 1.0],
                 [1.0, 0.0, 1.0, 0.0],
                 [0.0, 1.0, 0.0, 1.0] ])

# all_close ritorna True se tutti i valori nella prima matrice sono abbastanza vicini
# (cioè entro una certa tolleranza) ai corrispondenti nella seconda
assert np.allclose(scacchiera(4), r1)

r2 = np.array( [ [1.0] ])
assert np.allclose(scacchiera(1), r2)

r3 = np.array( [ [1.0, 0.0],
                 [0.0, 1.0] ])
assert np.allclose(scacchiera(2), r3)
```

[67]:
```
def scacchiera(n):
    raise Exception('TODO IMPLEMENT ME !')

r1 = np.array( [ [1.0, 0.0, 1.0, 0.0],
```

(continues on next page)

(continued from previous page)

```
                 [0.0, 1.0, 0.0, 1.0],
                 [1.0, 0.0, 1.0, 0.0],
                 [0.0, 1.0, 0.0, 1.0] ])

# all_close ritorna True se tutti i valori nella prima matrice sono abbastanza vicini
# (cioè entro una certa tolleranza) ai corrispondenti nella seconda
assert np.allclose(scacchiera(4), r1)

r2 = np.array( [ [1.0] ])
assert np.allclose(scacchiera(1), r2)

r3 = np.array( [ [1.0, 0.0],
                 [0.0, 1.0] ])
assert np.allclose(scacchiera(2), r3)
```

somma_alterna

⊛⊛⊛ MODIFICA la matrice numpy in input (n x n), sommando a tutte le righe dispari le righe pari. Per esempio:

```
m = [[1.0, 3.0, 2.0, 5.0],
     [2.0, 8.0, 5.0, 9.0],
     [6.0, 9.0, 7.0, 2.0],
     [4.0, 7.0, 2.0, 4.0]]
somma_alterna(m)
```

adesso `m` dovrebbe essere :

```
m = [[1.0, 3.0, 2.0, 5.0],
     [3.0, 11.0,7.0, 14.0],
     [6.0, 9.0, 7.0, 2.0],
     [10.0,16.0,9.0, 6.0]]
```

Ingredienti:

- per alternare, potete usare la range nella forma in cui prende 3 parametri, per esempio range(0,n,2) parte da 0, arriva fino a n escluso e salta di due in due, generando 0,2,4,6,8, …
- invece range(1,n,2) genererebbe 1,3,5,7, …

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[68]:
```
def somma_alterna(mat):
    """ MODIFICA la matrice numpy in input (n x n), sommando a tutte le righe dispari␣
↪le righe pari.
    """

    nrows, ncols = mat.shape
    for i in range(1,nrows, 2):
        for j in range(0,ncols):
            mat[i, j] = mat[i,j] + mat[i-1, j]



m1 = np.array( [ [1.0, 3.0, 2.0, 5.0],
```

(continues on next page)

(continued from previous page)

```
                        [2.0, 8.0, 5.0, 9.0],
                        [6.0, 9.0, 7.0, 2.0],
                        [4.0, 7.0, 2.0, 4.0] ])

r1 = np.array( [ [1.0, 3.0, 2.0, 5.0],
                 [3.0, 11.0,7.0, 14.0],
                 [6.0, 9.0, 7.0, 2.0],
                 [10.0,16.0,9.0, 6.0] ])

somma_alterna(m1)
assert np.allclose(m1, r1)  # controlla che abbiamo MODIFICATO la matrice originale

m2 = np.array( [ [5.0] ])
r2 = np.array( [ [5.0] ])
somma_alterna(m1)
assert np.allclose(m2, r2)

m3 = np.array( [ [6.0, 1.0],
                 [3.0, 2.0] ])
r3 = np.array( [ [6.0, 1.0],
                 [9.0, 3.0] ])
somma_alterna(m3)
assert np.allclose(m3, r3)
```

[68]:
```
def somma_alterna(mat):
    """ MODIFICA la matrice numpy in input (n x n), sommando a tutte le righe dispari
 ↪le righe pari.
    """
    raise Exception('TODO IMPLEMENT ME !')


m1 = np.array( [ [1.0, 3.0, 2.0, 5.0],
                        [2.0, 8.0, 5.0, 9.0],
                        [6.0, 9.0, 7.0, 2.0],
                        [4.0, 7.0, 2.0, 4.0] ])

r1 = np.array( [ [1.0, 3.0, 2.0, 5.0],
                 [3.0, 11.0,7.0, 14.0],
                 [6.0, 9.0, 7.0, 2.0],
                 [10.0,16.0,9.0, 6.0] ])

somma_alterna(m1)
assert np.allclose(m1, r1)  # controlla che abbiamo MODIFICATO la matrice originale

m2 = np.array( [ [5.0] ])
r2 = np.array( [ [5.0] ])
somma_alterna(m1)
assert np.allclose(m2, r2)

m3 = np.array( [ [6.0, 1.0],
                 [3.0, 2.0] ])
r3 = np.array( [ [6.0, 1.0],
                 [9.0, 3.0] ])
somma_alterna(m3)
assert np.allclose(m3, r3)
```

media_meta

⊛⊛⊛ Prende in input una matrice numpy con un numero pari di colonne, e RITORNA in output una matrice numpy 1x2: il primo elemento sarà la media della metà sinistra della matrice, il secondo elemento sarà la media della metà destra.

Ingredienti:

- per ottenere il numero di colonne diviso 2 come numero intero, usare l'operatore `//`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[69]:
```
def media_meta(mat):

    righe, colonne = mat.shape
    meta_colonne = colonne // 2

    media_sx = 0.0
    media_dx = 0.0

    # scrivi qui
    for i in range(righe):
        for j in range(meta_colonne):
            media_sx += mat[i,j]
        for j in range(meta_colonne, colonne):
            media_dx += mat[i,j]

    mezzi_elementi = righe * meta_colonne
    media_sx /=  mezzi_elementi
    media_dx /= mezzi_elementi
    return np.array([media_sx, media_dx])


# INIZIO TEST
m1 = np.array([[3,2,1,4],
               [6,2,3,5],
               [4,3,6,2],
               [4,6,5,4],
               [7,2,9,3]])

r1 = np.array([(3+2+6+2+4+3+4+6+7+2)/10, (1+4+3+5+6+2+5+4+9+3)/10  ])

assert np.allclose( media_meta(m1), r1)
# FINE TEST
```

[69]:
```
def media_meta(mat):
    raise Exception('TODO IMPLEMENT ME !')

# INIZIO TEST
m1 = np.array([[3,2,1,4],
               [6,2,3,5],
               [4,3,6,2],
               [4,6,5,4],
               [7,2,9,3]])

r1 = np.array([(3+2+6+2+4+3+4+6+7+2)/10, (1+4+3+5+6+2+5+4+9+3)/10  ])
```

(continues on next page)

(continued from previous page)

```
assert np.allclose( media_meta(m1), r1)
# FINE TEST
```

matxarr

⊛⊛⊛ Prende una matrice numpy `n` x `m` e un `ndarray` di `m` elementi, e RITORNA una NUOVA matrice numpy in cui i valori di ogni colonna della matrice di input sono moltiplicati per il corrispondente valore dell'array di `n` elementi.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[70]:
```
def matxarr(mat, arr):

    ret = np.zeros( mat.shape )


    for i in range(mat.shape[0]):
        for j in range(mat.shape[1]):
            ret[i,j] = mat[i,j] * arr[j]

    return ret


m1 = np.array( [ [3,2,1],
                 [6,2,3],
                 [4,3,6],
                 [4,6,5] ] )
a1 = [5, 2, 6]
r1 = [ [3*5, 2*2, 1*6],
       [6*5, 2*2, 3*6],
       [4*5, 3*2, 6*6],
       [4*5, 6*2, 5*6] ]
assert np.allclose(matxarr(m1,a1), r1)
```

[70]:
```
def matxarr(mat, arr):
    raise Exception('TODO IMPLEMENT ME !')

m1 = np.array( [ [3,2,1],
                 [6,2,3],
                 [4,3,6],
                 [4,6,5] ] )
a1 = [5, 2, 6]
r1 = [ [3*5, 2*2, 1*6],
       [6*5, 2*2, 3*6],
       [4*5, 3*2, 6*6],
       [4*5, 6*2, 5*6] ]
assert np.allclose(matxarr(m1,a1), r1)
```

quadranti

⊛⊛⊛ Data una matrice `2n * 2n`, dividere la matrice in 4 parti quadrate uguali (vedi esempio per capire meglio) e RESTITUIRE una NUOVA matrice `2 * 2` contente la media di ogni quadrante

Si assume che la matrice sia sempre di dimensioni pari

SUGGERIMENTO: per dividere per 2 e ottenere un numero intero, usare l'operatore //

Esempio:

```
1, 2 , 5 , 7
4, 1 , 8 , 0
2, 0 , 5 , 1
0, 2 , 1 , 1
```

si divide in

```
  1, 2 | 5 , 7
  4, 1 | 8 , 0
-----------------
  2, 0 | 5 , 1
  0, 2 | 1 , 1
```

e si restituisce

```
(1+2+4+1)/ 4   | (5+7+8+0)/4                          2.0 , 5.0
-----------------------------             =>         1.0 , 2.0
(2+0+0+2)/4    | (5+1+1+1)/4
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[71]:
```
import numpy as np

def quadranti(matrice):

    ret = np.zeros( (2,2) )

    dim = matrice.shape[0]
    n = dim // 2
    elementi_per_quadrante = n * n

    for i in range(n):
        for j in range(n):
            ret[0,0] += matrice[i,j]
    ret[0,0] /=   elementi_per_quadrante


    for i in range(n,dim):
        for j in range(n):
            ret[1,0] += matrice[i,j]
    ret[1,0] /= elementi_per_quadrante

    for i in range(n,dim):
        for j in range(n,dim):
            ret[1,1] += matrice[i,j]
```

(continues on next page)

(continued from previous page)

```
    ret[1,1] /= elementi_per_quadrante

    for i in range(n):
        for j in range(n,dim):
            ret[0,1] += matrice[i,j]
    ret[0,1] /= elementi_per_quadrante

    return ret



# INIZIO TEST - NON TOCCARE !
m1 = np.array( [ [3.0, 5.0],
                 [4.0, 9.0] ])
r1 = np.array([  [3.0, 5.0],
                 [4.0, 9.0],
               ])
assert np.allclose(quadranti(m1),r1)

m2 = np.array( [ [1.0, 2.0 , 5.0 , 7.0],
                 [4.0, 1.0 , 8.0 , 0.0],
                 [2.0, 0.0 , 5.0 , 1.0],
                 [0.0, 2.0 , 1.0 , 1.0] ])
r2 = np.array( [ [2.0, 5.0],
                 [1.0, 2.0] ] )
assert np.allclose(quadranti(m2),r2)
# FINE TEST
```

[71]:
```
import numpy as np

def quadranti(matrice):
    raise Exception('TODO IMPLEMENT ME !')


# INIZIO TEST - NON TOCCARE !
m1 = np.array( [ [3.0, 5.0],
                 [4.0, 9.0] ])
r1 = np.array([  [3.0, 5.0],
                 [4.0, 9.0],
               ])
assert np.allclose(quadranti(m1),r1)

m2 = np.array( [ [1.0, 2.0 , 5.0 , 7.0],
                 [4.0, 1.0 , 8.0 , 0.0],
                 [2.0, 0.0 , 5.0 , 1.0],
                 [0.0, 2.0 , 1.0 , 1.0] ])
r2 = np.array( [ [2.0, 5.0],
                 [1.0, 2.0] ] )
assert np.allclose(quadranti(m2),r2)
# FINE TEST
```

Altri esercizi numpy

- Prova a svolgere gli esercizi delle liste di liste[269], ma usando invece Numpy.
- Leggi i tutorial Nicola Zoppetti, parte Numpy[270] e prova a rendere gli esercizi già visti più efficienti sostituendo ai cicli `for` delle funzioni specifiche di numpy che operano su vettori
- (in inglese) machinelearningplus[271] Esercizi su Numpy (Fermarsi a difficoltà L1, L2 e se vuoi prova L3)

[ ]:

```

```

4.31 Comandamenti

Il Comitato Supremo per la Dottrina del Coding ha emanato importanti comandamenti che seguirai scrupolosamente.

Se accetti le loro sagge parole, diventerai un veri Jedi Python.

ATTENZIONE: se non segui i Comandamenti, finirai nel *Debugging Hell* !

4.31.1 I COMANDAMENTO

Scriverai codice Python

Chi non scrive codice Python, non impara Python

4.31.2 II COMANDAMENTO

Quando inserisci una variabile in un ciclo `for`, questa variabile deve essere nuova

Se hai definito la variabile prima, non la reintrodurrai in un `for`, perchè ciò portebbe confusione nelle menti di chi legge.

Perciò evita questi peccati:

[1]:
```
i = 7
for i in range(3):  # peccato, perdi la variabile i
    print(i)

print(i)   # stampa 2 e non 7  !!
```

```
0
1
2
2
```

[269] https://it.softpython.org/matrices-list-of-lists/list-of-lists-sol.html
[270] http://www.ifac.cnr.it/~zoppetti/corsopython/
[271] https://www.machinelearningplus.com/python/101-numpy-exercises-python/

[2]:
```
def f(i):
    for i in range(3): # altro peccato, perdi il parametro i
        print(i)

    print(i)  # stampa 2, e non il 7 che gli abbiamo passato !

f(7)
```

```
0
1
2
2
```

[3]:
```
for i in range(2):

    for i in  range(5):  # inferno da debuggare, perdi l'i del ciclo for esterno
        print(i)

    print(i)  # stampa 4 !!
```

```
0
1
2
3
4
4
0
1
2
3
4
4
```

4.31.3 III COMANDAMENTO

Noi riassegnerai mai parametri di funzione

Non farai mai nessuna di queste assegnazioni, pena la perdita del parametro passato quando viene chiamata la funzione:

[4]:
```
def peccato(intero):
    intero = 666            # hai perso il 5 passato dall'esterno !
    print(intero)           # stampa 666

x = 5
peccato(x)
```

```
666
```

Lo stesso discorso si applica per tutti gli altri tipi:

[5]:
```
def male(stringa):
    stringa = "666"
```

[6]:
```
def disgrazia(lista):
    lista = [666]
```

[7]:
```
def delirio(dizionario):
    dizionario = {"evil":666}
```

Per il solo caso di parametri compositi come liste o dizionari, puoi scrivere come sotto SE E SOLO SE le specifiche della funzione ti richiedono di MODIFICARE gli elementi interni del parametro (come per esempio ordinare una lista o cambiare il campo di un dizionario)

[8]:
```
# MODIFICA lista in qualche modo
def consentito(lista):
    lista[2] = 9

fuori = [8,5,7]
consentito(fuori)
print(fuori)
```

```
[8, 5, 9]
```

[9]:
```
# MODIFICA dizionario in qualche modo
def daccordo(dizionario):
    dizionario["mio campo"] = 5
```

[10]:
```
# MODIFICA istanza in qualche modo
def va_bene(istanza_di_classe):
    istanza_di_classe.mio_campo = 7
```

Se invece il testo di una funzione ti chiede di RITORNARE un NUOVO oggetto, non cadrai nella tentazione di modificare l'input:

[11]:
```

# RITORNA una NUOVA lista ordinata
def dolore(lista):
    lista.sort()          # MALE, stai modificando la lista di input invece di crearne␣
↪una nuova!
    return lista
```

[12]:
```
# RITORNA una NUOVA lista
def crisi(lista):
    lista[0] = 5            # MALE, come sopra
    return lista
```

[13]:
```
# RITORNA un NUOVO dizionario
def tormento(dizionario):
    dizionario['a'] = 6  # MALE, stai modificando il dizionario di input invece di␣
↪crearne uno nuovo!
    return dizionario
```

[14]:
```
# RITORNA una NUOVA istanza di classe
def disperazione(istanza):
    istanza.mio_campo = 6  # MALE, stai modificando l'oggetto di input
                                        # invece di crearne uno nuovo!
    return istanza
```

4.31.4 IV COMANDAMENTO

Non riassegnerai mai valori a chiamate a funzioni o metodi

SBAGLIATISSIMO:

```
mia_funzione() = 666
mia_funzione() = 'evil'
mia_funzione() = [666]
```

CORRETTO:

```
x = 5
y = my_fun()
z = []
z[0] = 7
d = dict()
d["a"] = 6
```

Chiamate a funzione come `mia_funzione()` ritornano risultati di calcoli e li mettono in una scatola che è creata solo per lo scopo della chiamata e Python non ci consentirà di riusarla come una variabile.

Quando vedi `nome()` alla parte sinistra, *non può* essere seguito da un segno di uguaglianza = (ma può essere seguito da due segni di uguaglianza == se stai eseguendo una comparazione).

4.31.5 V COMANDAMENTO

Non ridifinerai mai funzioni di sistema

Python ha diverse funzioni di sistema predefinite. Per esempio `list` è un tipo Python: come tale, puoi usarlo per esempio come funzione per convertire un qualche tipo a lista:

```
[15]: list("ciao")
[15]: ['c', 'i', 'a', 'o']
```

Quando consenti alle Forze del Male di prendere il sopravvento, potresti essere tentato di usare tipi e funzioni di sistema (per es. `list`) come una variabile per i tuoi miserabili propositi personali:

```
list = ['la', 'mia', 'lista', 'raccapricciante']
```

Python ti permette di farlo, ma **noi no**, poichè le conseguenze sono disastrose.

Per esempio, se adesso usi `list` per il proposito per cui è stata creata, cioè conversione a lista, non funzionerà più:

```
list("ciao")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-c63add832213> in <module>()
----> 1 list("ciao")

TypeError: 'list' object is not callable
```

In particolare, raccomandiamo di **non ridefinire** queste preziose funzioni:

- `bool`, `int`,`float`,`tuple`,`str`,`list`,`set`,`dict`
- `max`, `min`, `sum`
- `next`, `iter`
- `id`, `dir`, `vars`,`help`

4.31.6 VI COMANDAMENTO

Userai il comando `return` solo se vedi scritto RITORNA nella descrizione di funzione!

Se non c'è un `return` nella descrizione di funzione, si intende che la funzione ritorni `None`. In questo caso non devi nemmeno scrivere `return None`, perchè Python lo farà implicitamente per te.

4.31.7 VII COMANDAMENTO

Scriverai anche su carta!

Se fissare il monitor non funziona, aiutati e disegna su carta una rappresentazione dello stato del programma. Tabelle, nodi, frecce, tutto può aiutare nel trovare una soluzione al problema.

4.31.8 VIII COMANDAMENTO

Non riassegnerai mai `self` !

Non scriverai mai empietà come questa:

```
[16]: class MiaClasse:
          def mio_metodo(self):
              self = {'mio_campo':666}
```

Dato che `self` è una specie di dizionario, potresti essere tentato di scrivere come sopra, ma al mondo esterno questo non porterà alcun effetto.

Per esempio, supponiamo che qualcuno da fuori faccia una chiamata come questa:

```
[17]: mc = MiaClasse()
      mc.mio_metodo()
```

Dopo la chiamata `mc` non punterà a `{'mio_campo':666}`

```
[18]: mc
[18]: <__main__.MiaClasse at 0x7f5b345a0550>
```

e non avrà `mio_campo`:

```
mc.mio_campo

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-26-5c4e6630908d> in <module>()
----> 1 mc.mio_campo

AttributeError: 'MiaClasse' object has no attribute 'mio_campo'
```

Per lo stesso ragionamento, non devi riassegnare `self` a liste o altro:

[19]:
```
class MiaClasse:
    def mio_metodo(self):
        self = ['evil']
        self = 666
```

4.31.9 IX COMANDAMENTO

Testerai il codice!

Il codice non testato per definizione *non funziona*. Per idee su come testare, guarda *Gestione degli errori e testing*

[ ]:
```
```

4.31.10 X COMANDAMENTO

Non aggiungerai o toglierai mai elementi da una sequenza che stai iterando con un `for`!

Abbandonarti in simil tentazioni **produrrebbe comportamenti del tutto imprevedibili** (conosci forse l'espressione *tirare il tappeto da sotto i piedi?*)

Non aggiungere, poichè rischi di camminare su un tapis roulant che mai si spegne:

```
lista = ['a','b','c','d','e']
for el in lista:
    lista.append(el)  # STAI INTASANDO LA MEMORIA DEL COMPUTER
```

Non togliere, poichè rischi di corrompere l'ordine naturale delle cose:

[20]:
```
lista = ['a','b','c','d','e']

for el in lista:
    lista.remove(el)   # PESSIMA IDEA
```

Guarda bene il codice. Credi che abbiamo rimosso tutto, eh?

[21]:
```
lista
```

[21]:
```
['b', 'd']
```

○_○' Non provar a capacitarti di cotal sortilegio - nessuno capirlo può, poichè esso è legato all'implementazione interna di Python.

La mia versione di Python dà questo risultato assurdo, la vostra potrebbe darne un'altro. Il discorso vale anche per iterazione su insiemi e dizionari. **Siete avvertiti**.

Se proprio devi rimuovere elementi dalla sequenza su cui stai iterando, usa un ciclo while[272] o effettua prima una copia della sequenza originale.

[272] https://it.softpython.org/control-flow/flow3-while-sol.html

CHAPTER

FIVE

B - ANALISI DATI

5.1 Formati dati

5.1.1 Scarica zip esercizi

Naviga file online[273]

5.1.2 Introduzione

In questo tutorial parleremo di formati dei dati:

- file testuali
  - File a linee
  - CSV
  - breve panoramica sui cataloghi open data
  - menzione licenze (Creative Commons CC-Zero)
  - menzione JSON
  - menzione XML
- menzione file binari
  - immagini
  - fogli Excel

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
formats
    formats.ipynb
    formats-sol.ipynb
    jupman.py
```

[273] https://github.com/DavidLeoni/softpython-it/tree/master/formats

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `formats.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

5.1.3 1. File a linee

I file a linee tipicamente sono file di testo che contengono informazioni raggruppate per linee. Un esempio usando personaggi storici potrebbe essere un file così:

```
Leonardo
da Vinci
Sandro
Botticelli
Niccolò
Macchiavelli
```

Si nota subito una regolarità: le prime due linee contengono i dati di Leonardo da Vinci, prima il nome e poi il cognome. Le successive due linee hanno invece i dati di Sandro Botticelli, di nuovo prima il nome e poi il cognome, e così via

Un programma che potremo voler fare potrebbe essere leggere le linee e stampare a video nomi e cognomi così:

```
Leonardo da Vinci
Sandro Botticelli
Niccolò Macchiavelli
```

Per iniziare ad avere un'approssimazione del risultato finale, possiamo aprire il file, leggere solo la prima linea e stamparla:

[1]:
```
with open('people-simple.txt', encoding='utf-8') as f:
    linea=f.readline()
    print(linea)
```

```
Leonardo
```

Che è successo? Esaminiamo le varie linee:

il comando open

Il comando

```
open('people-simple.txt', encoding='utf-8')
```

ci permette di aprire il file di testo dicendo a Python il percorso del file `'people-simple.txt'` e la codifica con cui è stato scritto (`encoding='utf-8'`).

La codifica

La codifica dipende dal sistema operativo e dell'editor con cui è stato scritto il file. Quando apriamo un file, Python non può divinare la codifica, e se non gliela specifichiamo potrebbe aprirlo assumendo una codifica diversa dall'originale - tradotto, affidandoci al caso o sbagliando codifica in seguito potremmo vedere dei caratteri strani (tipo quadratini invece di lettere accentate).

In genere, quando apri un file, prova prima a specificare la codifica `utf-8` che è la più comune scrivendo `encoding='utf-8'`, e se per caso non va bene prova invece `encoding='latin-1'` (solitamente utile se il file è stato scritto su sistemi Windows). Se apri file scritti in posti più esotici, tipo in Cina, potresti dover usare un'altro encoding. Per approfondire queste questioni, quando hai tempo leggi Immersione in Python - Cap 4 - Stringhe[274] e Immersione in Python - Cap 11 - File[275], **entrambe letture caldamente consigliate**.

il with

Il `with` definisce un blocco con all'interno le istruzioni:

```
with open('people-simple.txt', encoding='utf-8') as f:
    linea=f.readline()
    print(linea)
```

Abbiamo usato il `with` per dire a Python che in ogni caso, anche se accadono errori, vogliamo che dopo aver usato il file, e cioè eseguito le istruzioni nel blocco interno (il `linea=f.readline()` e `print(linea)`) Python deve chiudere automaticamente il file. Chiudere propriamente un file evita di sprecare risorse di memoria e creare errori paranormali. Se vuoi evitare di andare a caccia di file zombie mai chiusi, ricordati sempre di aprire i file nei blocchi `with`! Inoltre, alla fine della riga nella parte `as f:` abbiamo assegnato il file ad una variabile chiamata qui `f`, ma potevamo usare un qualunque altro nome.

> **ATTENZIONE**: Per indentare il codice, usa SEMPRE sequenze di 4 spazi bianchi. Sequenze di 2 soli spazi per quanto consentite non sono raccomandate.

> **ATTENZIONE**: A seconda dell'editor che usi, premendo TAB potresti ottenere una sequenza di spazi bianchi come accade in Jupyter (4 spazi che sono raccomandati), oppure un carattere speciale di tabulazione (da evitare)! Per quanto noiosa questa distinzione ti possa apparire, ricordatela perchè potrebbe generare errori molto difficili da scoprire.

> **ATTENZIONE**: Nei comandi che creano blocchi come il `with`, ricordati di mettere sempre il carattere dei doppi punti `:` alla fine della linea !

Il comando

```
linea=f.readline()
```

mette nella variabile `linea` l'intera linea, come una stringa. Attenzione: la stringa conterrà alla fine anche il carattere speciale di ritorno a capo !

Ti chiederai da dove venga fuori quel `readline`. Come quasi tutto in Python, la nostra variabile `f` che rappresenta il file appena aperto è un oggetto, e ogni oggetto, a seconda del suo tipo, ha dei *metodi* particolari che possiamo usare su di esso. In questo caso il metodo è `readline`. Clicca qua[276] per maggiori informazioni sugli oggetti.

Il comando seguente stampa il contenuto della stringa:

[274] http://gpiancastelli.altervista.org/dip3-it/stringhe.html
[275] http://gpiancastelli.altervista.org/dip3-it/stringhe.html
[276] http://it.softpython.org/intro.html#Usare-metodi-degli-oggetti

```
print(linea)
```

⊛ **1.1 ESERCIZIO**: Prova a riscrivere nella cella qua il blocco with appena visto, ed esegui la cella premendo Control-Invio. Riscrivi il codice con le dita, non con il copia e incolla ! Fai attenzione ad indentare correttamente con gli spazi il blocco.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[2]:
```
# scrivi qui

with open('people-simple.txt', encoding='utf-8') as f:
    linea=f.readline()
    print(linea)
```

```
Leonardo
```

[2]:
```
# scrivi qui
```

```
Leonardo
```

⊛ **1.2 ESERCIZIO**: immagino ti starai chiedendo che cosa è esattamente quella `f`, e cosa faccia esattamente il metodo `readlines`. Quando ti trovi in queste situazioni, puoi aiutarti con le funzioni `type` e `help`. Questa volta, copia e incolla direttamente sempre lo stesso codice qua sotto, ma aggiungi a mano dentro il blocco `with` i comandi:

- `print(type(f))`
- `print(help(f))`
- `print(help(f.readline)) # Attenzione: ricordati il 'f.' prima del readline !!`

Ogni volta che aggiungi qualcosa, prova ad eseguire con Control+Invio e vedere cosa succede.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[3]:
```
# scrivi qui il codice (copia e incolla)
with open('people-simple.txt', encoding='utf-8') as f:
    linea=f.readline()
    print(type(f))
    print(help(f.readline))
    print(help(f))
    print(linea)
```

```
<class '_io.TextIOWrapper'>
Help on built-in function readline:

readline(size=-1, /) method of _io.TextIOWrapper instance
    Read until newline or EOF.
```

(continues on next page)

(continued from previous page)

```
    Returns an empty string if EOF is hit immediately.

None
Help on TextIOWrapper object:

class TextIOWrapper(_TextIOBase)
 |  Character and line based layer over a BufferedIOBase object, buffer.
 |
 |  encoding gives the name of the encoding that the stream will be
 |  decoded or encoded with. It defaults to locale.getpreferredencoding(False).
 |
 |  errors determines the strictness of encoding and decoding (see
 |  help(codecs.Codec) or the documentation for codecs.register) and
 |  defaults to "strict".
 |
 |  newline controls how line endings are handled. It can be None, '',
 |  '\n', '\r', and '\r\n'.  It works as follows:
 |
 |  * On input, if newline is None, universal newlines mode is
 |    enabled. Lines in the input can end in '\n', '\r', or '\r\n', and
 |    these are translated into '\n' before being returned to the
 |    caller. If it is '', universal newline mode is enabled, but line
 |    endings are returned to the caller untranslated. If it has any of
 |    the other legal values, input lines are only terminated by the given
 |    string, and the line ending is returned to the caller untranslated.
 |
 |  * On output, if newline is None, any '\n' characters written are
 |    translated to the system default line separator, os.linesep. If
 |    newline is '' or '\n', no translation takes place. If newline is any
 |    of the other legal values, any '\n' characters written are translated
 |    to the given string.
 |
 |  If line_buffering is True, a call to flush is implied when a call to
 |  write contains a newline character.
 |
 |  Method resolution order:
 |      TextIOWrapper
 |      _TextIOBase
 |      _IOBase
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __getstate__(…)
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __repr__(self, /)
 |      Return repr(self).
```

(continues on next page)

(continued from previous page)

```
 |
 |  close(self, /)
 |      Flush and close the IO object.
 |
 |      This method has no effect if the file is already closed.
 |
 |  detach(self, /)
 |      Separate the underlying buffer from the TextIOBase and return it.
 |
 |      After the underlying buffer has been detached, the TextIO is in an
 |      unusable state.
 |
 |  fileno(self, /)
 |      Returns underlying file descriptor if one exists.
 |
 |      OSError is raised if the IO object does not use a file descriptor.
 |
 |  flush(self, /)
 |      Flush write buffers, if applicable.
 |
 |      This is not implemented for read-only and non-blocking streams.
 |
 |  isatty(self, /)
 |      Return whether this is an 'interactive' stream.
 |
 |      Return False if it can't be determined.
 |
 |  read(self, size=-1, /)
 |      Read at most n characters from stream.
 |
 |      Read from underlying buffer until we have n characters or we hit EOF.
 |      If n is negative or omitted, read until EOF.
 |
 |  readable(self, /)
 |      Return whether object was opened for reading.
 |
 |      If False, read() will raise OSError.
 |
 |  readline(self, size=-1, /)
 |      Read until newline or EOF.
 |
 |      Returns an empty string if EOF is hit immediately.
 |
 |  seek(self, cookie, whence=0, /)
 |      Change stream position.
 |
 |      Change the stream position to the given byte offset. The offset is
 |      interpreted relative to the position indicated by whence.  Values
 |      for whence are:
 |
 |      * 0 -- start of stream (the default); offset should be zero or positive
 |      * 1 -- current stream position; offset may be negative
 |      * 2 -- end of stream; offset is usually negative
 |
 |      Return the new absolute position.
 |
 |  seekable(self, /)
```

(continues on next page)

(continued from previous page)

```
 |      Return whether object supports random access.
 |
 |      If False, seek(), tell() and truncate() will raise OSError.
 |      This method may need to do a test seek().
 |
 |  tell(self, /)
 |      Return current stream position.
 |
 |  truncate(self, pos=None, /)
 |      Truncate file to size bytes.
 |
 |      File pointer is left unchanged.  Size defaults to the current IO
 |      position as reported by tell().  Returns the new size.
 |
 |  writable(self, /)
 |      Return whether object was opened for writing.
 |
 |      If False, write() will raise OSError.
 |
 |  write(self, text, /)
 |      Write string to stream.
 |      Returns the number of characters written (which is always equal to
 |      the length of the string).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  buffer
 |
 |  closed
 |
 |  encoding
 |      Encoding of the text stream.
 |
 |      Subclasses should override.
 |
 |  errors
 |      The error setting of the decoder or encoder.
 |
 |      Subclasses should override.
 |
 |  line_buffering
 |
 |  name
 |
 |  newlines
 |      Line endings translated so far.
 |
 |      Only line endings translated during reading are considered.
 |
 |      Subclasses should override.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _IOBase:
 |
 |  __del__(…)
 |
```

(continues on next page)

(continued from previous page)

```
 |  __enter__(…)
 |
 |  __exit__(…)
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  readlines(self, hint=-1, /)
 |      Return a list of lines from the stream.
 |
 |      hint can be specified to control the number of lines read: no more
 |      lines will be read if the total size (in bytes/characters) of all
 |      lines so far exceeds hint.
 |
 |  writelines(self, lines, /)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _IOBase:
 |
 |  __dict__

None
Leonardo
```

[3]:
```
# scrivi qui il codice (copia e incolla)
```

```
<class '_io.TextIOWrapper'>
Help on built-in function readline:

readline(size=-1, /) method of _io.TextIOWrapper instance
    Read until newline or EOF.

    Returns an empty string if EOF is hit immediately.

None
Help on TextIOWrapper object:

class TextIOWrapper(_TextIOBase)
 |  Character and line based layer over a BufferedIOBase object, buffer.
 |
 |  encoding gives the name of the encoding that the stream will be
 |  decoded or encoded with. It defaults to locale.getpreferredencoding(False).
 |
 |  errors determines the strictness of encoding and decoding (see
 |  help(codecs.Codec) or the documentation for codecs.register) and
 |  defaults to "strict".
 |
 |  newline controls how line endings are handled. It can be None, '',
 |  '\n', '\r', and '\r\n'.  It works as follows:
 |
 |  * On input, if newline is None, universal newlines mode is
 |    enabled. Lines in the input can end in '\n', '\r', or '\r\n', and
```

(continues on next page)

(continued from previous page)

```
 |    these are translated into '\n' before being returned to the
 |    caller. If it is '', universal newline mode is enabled, but line
 |    endings are returned to the caller untranslated. If it has any of
 |    the other legal values, input lines are only terminated by the given
 |    string, and the line ending is returned to the caller untranslated.
 |
 |  * On output, if newline is None, any '\n' characters written are
 |    translated to the system default line separator, os.linesep. If
 |    newline is '' or '\n', no translation takes place. If newline is any
 |    of the other legal values, any '\n' characters written are translated
 |    to the given string.
 |
 |  If line_buffering is True, a call to flush is implied when a call to
 |  write contains a newline character.
 |
 |  Method resolution order:
 |      TextIOWrapper
 |      _TextIOBase
 |      _IOBase
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __getstate__(…)
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  close(self, /)
 |      Flush and close the IO object.
 |
 |      This method has no effect if the file is already closed.
 |
 |  detach(self, /)
 |      Separate the underlying buffer from the TextIOBase and return it.
 |
 |      After the underlying buffer has been detached, the TextIO is in an
 |      unusable state.
 |
 |  fileno(self, /)
 |      Returns underlying file descriptor if one exists.
 |
 |      OSError is raised if the IO object does not use a file descriptor.
 |
 |  flush(self, /)
 |      Flush write buffers, if applicable.
 |
 |      This is not implemented for read-only and non-blocking streams.
```

(continues on next page)

(continued from previous page)

```
 |
 |  isatty(self, /)
 |      Return whether this is an 'interactive' stream.
 |
 |      Return False if it can't be determined.
 |
 |  read(self, size=-1, /)
 |      Read at most n characters from stream.
 |
 |      Read from underlying buffer until we have n characters or we hit EOF.
 |      If n is negative or omitted, read until EOF.
 |
 |  readable(self, /)
 |      Return whether object was opened for reading.
 |
 |      If False, read() will raise OSError.
 |
 |  readline(self, size=-1, /)
 |      Read until newline or EOF.
 |
 |      Returns an empty string if EOF is hit immediately.
 |
 |  seek(self, cookie, whence=0, /)
 |      Change stream position.
 |
 |      Change the stream position to the given byte offset. The offset is
 |      interpreted relative to the position indicated by whence.  Values
 |      for whence are:
 |
 |      * 0 -- start of stream (the default); offset should be zero or positive
 |      * 1 -- current stream position; offset may be negative
 |      * 2 -- end of stream; offset is usually negative
 |
 |      Return the new absolute position.
 |
 |  seekable(self, /)
 |      Return whether object supports random access.
 |
 |      If False, seek(), tell() and truncate() will raise OSError.
 |      This method may need to do a test seek().
 |
 |  tell(self, /)
 |      Return current stream position.
 |
 |  truncate(self, pos=None, /)
 |      Truncate file to size bytes.
 |
 |      File pointer is left unchanged.  Size defaults to the current IO
 |      position as reported by tell().  Returns the new size.
 |
 |  writable(self, /)
 |      Return whether object was opened for writing.
 |
 |      If False, write() will raise OSError.
 |
 |  write(self, text, /)
 |      Write string to stream.
```

(continues on next page)

(continued from previous page)

```
 |      Returns the number of characters written (which is always equal to
 |      the length of the string).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  buffer
 |
 |  closed
 |
 |  encoding
 |      Encoding of the text stream.
 |
 |      Subclasses should override.
 |
 |  errors
 |      The error setting of the decoder or encoder.
 |
 |      Subclasses should override.
 |
 |  line_buffering
 |
 |  name
 |
 |  newlines
 |      Line endings translated so far.
 |
 |      Only line endings translated during reading are considered.
 |
 |      Subclasses should override.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _IOBase:
 |
 |  __del__(…)
 |
 |  __enter__(…)
 |
 |  __exit__(…)
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  readlines(self, hint=-1, /)
 |      Return a list of lines from the stream.
 |
 |      hint can be specified to control the number of lines read: no more
 |      lines will be read if the total size (in bytes/characters) of all
 |      lines so far exceeds hint.
 |
 |  writelines(self, lines, /)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _IOBase:
 |
 |  __dict__
```

(continues on next page)

(continued from previous page)

```
None
Leonardo
```

Prima abbiamo messo il contenuto della prima linea nella variabile `line`, ora potremmo metterlo in una variabile dal nome più significativo, come `nome`. Non solo, possiamo anche direttamente leggere la linea successiva nella variabile `cognome` e poi stampare la concatenazione delle due:

[4]:
```
with open('people-simple.txt', encoding='utf-8') as f:
    nome=f.readline()
    cognome=f.readline()
    print(nome + ' ' + cognome)
```

```
Leonardo
 da Vinci
```

PROBLEMA ! La stampa mette comunque uno strano ritorno a capo. Come mai? Se ti ricordi, prima ho detto che `readline` legge il contenuto della linea in una stringa aggiungendo alla fine anche il carattere speciale di ritorno a capo. Per eliminarlo, puoi usare il comando `rstrip()`:

[5]:
```
with open('people-simple.txt', encoding='utf-8') as f:
    nome=f.readline().rstrip()
    cognome=f.readline().rstrip()
    print(nome + ' ' + cognome)
```

```
Leonardo da Vinci
```

⊛ **1.3 ESERCIZIO**: Di nuovo, riscrivi il blocco qua sopra nella cella sotto, ed esegui la cella con Control+Invio. Domanda: che succede se usi `strip()` invece di `rstrip()`? Ed `lstrip()`? Riesci a dedurre il significato di `r` e `l`? Se non ci riesci, prova ad usare il comando python `help` chiamando `help(string.rstrip)`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[6]:
```
# scrivi qui

with open('people-simple.txt', encoding='utf-8') as f:
    nome=f.readline().rstrip()
    cognome=f.readline().rstrip()
    print(nome + ' ' + cognome)
```

```
Leonardo da Vinci
```

[6]:
```
# scrivi qui
```

```
Leonardo da Vinci
```

Benissimo, abbiamo la prima linea ! Adesso possiamo leggere tutte le linee in sequenza. A tal fine possiamo usare un ciclo `while`:

[7]:
```
with open('people-simple.txt', encoding='utf-8') as f:
    linea=f.readline()
    while linea != "":
        nome = linea.rstrip()
        cognome=f.readline().rstrip()
        print(nome + ' ' + cognome)
        linea=f.readline()
```

```
Leonardo da Vinci
Sandro Botticelli
Niccolò Macchiavelli
```

NOTA In Python ci sono metodi più concisi[277] per leggere un file tdi testo linea per linea, ma abbiamo usato questo approccio per esplicitare tutti i passaggi

Cosa abbiamo fatto? Per prima cosa, abbiamo aggiunto un ciclo `while` in un nuovo blocco

Attenzione: nel nuovo blocco while, dato che è già all'interno del blocco esterno `with`, le istruzioni sono indentate di 8 spazi e non più 4! Se per caso sbagli gli spazi, possono succedere brutti guai!

Prima leggiamo una linea, e due casi sono possibili:

a. siamo alla fine del file (o il file è vuoto): in questo caso la chiamata a `readline()` ritorna una stringa vuota
b. non siamo alla fine del file: la prima linea è messa come una stringa dentro la variabile `linea`.
c. we are not at the end of the file: the first line is put as a string inside the variable `line`. Dato che Python internamente usa un puntatore per tenere traccia della posizione a cui si è durante la lettura del file, dopo una lettura questo puntatore è mosso all'inizio della riga successiva. In questo modo una chiamata successiva a `readline()` leggera la linea dalla nuova posizione.

Nel blocco `while` diciamo a Python di continuare il ciclo fintanto che `linea` *non* è vuota. In questo caso, dentro il blocco `while` estraiamo il nome dalla linea e lo mettiamo nella variabile `nome` (rimuovendo il carattere extra di ritorno a capo con la `rstrip()` come fatto in precedenza), e poi procediamo leggendo la nuova riga ed estraendo il risultato dentro la variabile `cognome`. Infine, leggiamo di nuovo la linea dentro la variabile `linea` così sarà pronta per la prossima iterazione di estrazione nome. Se la `linea` è vuota il ciclo terminerà:

```
while linea != "":                    # entra il ciclo se la linea contiene caratteri
    nome = linea.rstrip()             # estrae il nome
    cognome=f.readline().rstrip()     # legge la nuova linea ed estrae il cognome
    print(nome + ' ' + cognome)
    linea=f.readline()                # legge la prossima linea
```

⊛ **1.4 ESERCIZIO**: Di nuovo come prima, riscrivi nella cella qua sotto il codice col while appena spiegato, facendo MOLTA attenzione all'indentazione (per la linea del with esterno fai pure copia e incolla):

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[8]:
```
# scrivi qui il codice col while interno

with open('people-simple.txt', encoding='utf-8') as f:
```

(continues on next page)

[277] https://thispointer.com/5-different-ways-to-read-a-file-line-by-line-in-python/

(continued from previous page)

```
    linea=f.readline()
    while linea != "":
        nome = linea.rstrip()
        cognome=f.readline().rstrip()
        print(nome + ' ' + cognome)
        linea=f.readline()
```

```
Leonardo da Vinci
Sandro Botticelli
Niccolò Macchiavelli
```

[8]:
```
# scrivi qui il codice col while interno
```

```
Leonardo da Vinci
Sandro Botticelli
Niccolò Macchiavelli
```

File a linee people-complex:

Guarda il file `people-complex.txt`, più complesso:

```
nome: Leonardo
cognome: da Vinci
data di nascita: 1452-04-15
nome: Sandro
cognome: Botticelli
data di nascita: 1445-03-01
nome: Niccolò
cognome: Macchiavelli
data di nascita: 1469-05-03
```

Supponendo di leggere il file per voler stampare questo output, come faresti ?

```
Leonardo da Vinci, 1452-04-15
Sandro Botticelli, 1445-03-01
Niccolò Macchiavelli, 1469-05-03
```

Suggerimento 1: Per ottenere dalla stringa `'abcde'` la sottostringa `'cde'`, che inizia all'indice 2, puoi usare l'operatore di parentesi quadre, indicando l'indice di inizio seguito dai doppi punti `:`

[9]:
```
x = 'abcde'
x[2:]
```

[9]:
```
'cde'
```

[10]:
```
x[3:]
```

[10]:
```
'de'
```

Suggerimento 2: Per sapere la lunghezza di una stringa, usa la funzione `len`:

[11]:
```
len('abcde')
```

[11]:
```
5
```

⊛ **1.5 ESERCIZIO**: Scrivi qua sotto la soluzione dell'esercizio 'People complex':

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
# scrivi qui

with open('people-complex.txt', encoding='utf-8') as f:
    linea=f.readline()
    while linea != "":
        nome = linea.rstrip()[len("nome: "):]
        cognome= f.readline().rstrip()[len("cognome: "):]
        nato = f.readline().rstrip()[len("data di nascita: "):]
        print(nome + ' ' + cognome + ', ' + nato)
        linea=f.readline()
```

```
Leonardo da Vinci, 1452-04-15
Sandro Botticelli, 1445-03-01
Niccolò Macchiavelli, 1469-05-03
```

[12]:
```
# scrivi qui
```

```
Leonardo da Vinci, 1452-04-15
Sandro Botticelli, 1445-03-01
Niccolò Macchiavelli, 1469-05-03
```

Esercizio file a linee immersione-in-python-toc

Questo esercizio è più difficile, se sei alle prime armi puoi saltarlo e passare ai CSV.

Il libro Immersione in Python in Italiano è bello e ti fornisce un PDF, che però ha un problema: se provi a stamparlo scoprirai che manca l'indice. Senza perderci d'animo, abbiamo trovato un programmino che ci ha estratto i titoli in un file come segue, che però come scoprirai non è esattamente bello da vedere. Dato che siamo Python ninja, abbiamo deciso di trasformare i titoli grezzi in una tabella dei contenuti vera e propria[278]. Sicuramente ci sono metodi più furbi per compiere questa operazione, come caricare il pdf in Python con una apposita libreria per i pdf, ma pareva un esercizio interessante.

Ti viene consegnato il file immersione-in-python-toc.txt:

```
BookmarkBegin
BookmarkTitle: Il vostro primo programma Python
BookmarkLevel: 1
BookmarkPageNumber: 38
BookmarkBegin
BookmarkTitle: Immersione!
BookmarkLevel: 2
BookmarkPageNumber: 38
BookmarkBegin
BookmarkTitle: Dichiarare funzioni
```

(continues on next page)

[278] http://it.softpython.org/_static/toc-immersione-in-python-3.txt

(continued from previous page)

```
BookmarkLevel: 2
BookmarkPageNumber: 41
BookmarkBeginint
BookmarkTitle: Argomenti opzionali e con nome
BookmarkLevel: 3
BookmarkPageNumber: 42
BookmarkBegin
BookmarkTitle: Scrivere codice leggibile
BookmarkLevel: 2
BookmarkPageNumber: 44
BookmarkBegin
BookmarkTitle: Stringhe di documentazione
BookmarkLevel: 3
BookmarkPageNumber: 44
BookmarkBegin
BookmarkTitle: Il percorso di ricerca di import
BookmarkLevel: 2
BookmarkPageNumber: 46
BookmarkBegin
BookmarkTitle: Ogni cosa &#232; un oggetto
BookmarkLevel: 2
BookmarkPageNumber: 47
```

Scrivi un programma python per stampare e video il seguente output:

```
Il vostro primo programma Python  38
   Immersione!  38
   Dichiarare funzioni  41
      Argomenti opzionali e con nome  42
   Scrivere codice leggibile  44
      Stringhe di documentazione  44
   Il percorso di ricerca di import  46
   Ogni cosa è un oggetto  47
```

Per questo esercizio, dovrai inserire nell'output degli spazi artificiali, in una quantità determinata dalle righe `BookmarkLevel`.

DOMANDA: che cos'è lo strano valore `è` alla fine del file originale ? Dobbiamo riportarlo nell'output ?

SUGGERIMENTO 1: Per convertire una stringa in numero intero, usa la funzione `int`:

[13]:
```
x = '5'
```

[14]:
```
x
```

[14]:
```
'5'
```

[15]:
```
int(x)
```

[15]:
```
5
```

Attenzione: `int(x)` ritorna un valore, e non modifica mai l'argomento `x` !

SUGGERIMENTO 2: Per sostituire un sottostringa in una stringa, si può usare la funzione replace:

[16]:
```
x = 'abcde'
x.replace('cd', 'HELLO' )
```

[16]:
```
'abHELLOe'
```

SUGGERIMENTO 4: Finchè c'è una sola sequenza da sostituire, va bene il replace, ma se avessimo un milione di sequenze orribili come `>`, `>`, `&x3e;`, che faremmo? Da bravi data cleaner, possiamo riconoscere che sono sequenze di escape HTML[279], perciò potremmo usare dei metodi specifici per queste sequenze come html.unescape[280]. Provalo invece del replace e guarda se funziona!

NOTA: Prima di usare il metodo `html.unescape`, importa il modulo `html` con il comando:

```
import html
```

SUGGERIMENTO 3: Per scrivere *n* copie di un carattere, usa `*` come qua:

[17]:
```
"b" * 3
```

[17]:
```
'bbb'
```

[18]:
```
"b" * 7
```

[18]:
```
'bbbbbbb'
```

⊛⊛⊛ **1.6 ESERCIZIO**: Scrivi qua sotto la soluzione per file a linee immersione-in-python-toc, e prova ad eseguirla premendo Control + Invio:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
# scrivi qui

import html

with open("immersione-in-python-toc.txt", encoding='utf-8') as f:

    linea=f.readline()
    while linea != "":
        linea = f.readline().strip()
        titolo = html.unescape(linea[len("BookmarkTitle: "):])
        linea=f.readline().strip()
        livello = int(linea[len("BookmarkLevel: "):])
        linea=f.readline().strip()
        pagina = linea[len("BookmarkPageNumber: "):]
        print(("   " * livello) + titolo + "  " + pagina)
        linea=f.readline()
```

```
   Il vostro primo programma Python  38
      Immersione!  38
      Dichiarare funzioni  41
         Argomenti opzionali e con nome  42
      Scrivere codice leggibile  44
         Stringhe di documentazione  44
      Il percorso di ricerca di import  46
      Ogni cosa è un oggetto  47
```

[279] https://corsidia.com/materia/web-design/caratterispecialihtml
[280] https://docs.python.org/3/library/html.html#html.unescape

[19]:
```
# scrivi qui
```

```
Il vostro primo programma Python  38
   Immersione!  38
   Dichiarare funzioni  41
      Argomenti opzionali e con nome  42
   Scrivere codice leggibile  44
      Stringhe di documentazione  44
   Il percorso di ricerca di import  46
   Ogni cosa è un oggetto  47
```

5.1.4 2. File CSV

Ci possono essere vari formati per i file tabulari, tra cui sicuramente conoscerai gli Excel (`.xls` o `.xslx`). Peccato che se vuoi processare dati programmaticamente, faresti meglio ad evitarli e preferire se possibile i file CSV, letteralmente 'Comma separated Value'. Per capire il perchè, quando hai tempo potresti guardare questo tutorial[281] in cui si spiega a produttori di dati (in questo caso, dipendenti pubblici) come trasformare Excel in CSV, evidenziando i vari grattacapi che un Excel può presentare a chi poi riusa i dati.

Oggi proveremo ad aprire qualche CSV, prendendo in considerazione i possibili problemi che possono insorgere. I CSV non sono la panacea per tutti i mali, ma offrono maggiore controllo sulla lettura e tipicamente se saltano fuori errori di conversione è perchè siamo stati noi a sbagliare, e non perchè la libreria di lettura ha magari deciso da sola di scambiare nelle date i giorni con i mesi.

Perchè scorrere un CSV ?

Per caricare e processare CSV esistono già librerie molto potenti ed intuitive come Pandas in Python o i dataframe di R, che forse avrai già visto. Oggi invece caricheremo i CSV usando il mezzo più semplice possibile, che è la lettura riga per riga, più o meno come fatto nella prima parte del tutorial. Non bisogna pensare che questo metodo sia primitivo o stupido, a seconda della situazione può salvare la giornata. Come mai? Dato che alcuni file potenzialmente potrebbero occupare terabyte, e nei moderni laptop di solito abbiamo mediamente solo 4 Gigabyte di RAM, che è la memoria dove Python mette le variabili, le funzioni di base di Python per leggere file evitano di caricare tutto in RAM. Tipicamente invece un file viene scorso un po' alla volta, mettendo in RAM solo una riga alla volta.

DOMANDA 2.1: se vogliamo sapere se un certo file da 100 terabyte contiene almeno 3 milioni di righe in cui è presente la parola 'ciao', dobbiamo mettere in RAM contemporaneamente tutte le righe?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: no, basta mettere una riga alla volta, e tenere una variabile per il conteggio

DOMANDA 2.2: E se volessimo, partendo da un file da 100 terabyte crearne un'altro con gli stessi contenuti del primo file a cui a tutte le righe è aggiunta la parola 'ciao' alla fine, dovremmo mettere in RAM contemporaneamente tutte le righe del primo file? E quelle del secondo ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

[281] https://docs.google.com/presentation/d/1RPiRFm0g9QtSiC3aO32aFldSYThl11CB9HI_HHG6hvY/

RISPOSTA: No, basta tenere in RAM una riga alla volta, che viene prima letta dal primo file e subito scritta nel secondo

CSV di esempio

Cominciamo con dei CSV artificiali di esempio, sul modello di esercizi già fatti nella introduzione[282]

Per iniziare, vedremo il CSV esempio-1.csv che trovi nella stessa cartella di questo foglio Jupyter. Riportiamo qui il contenuto del file:

```
animale, anni
cane, 12
gatto, 14
pellicano, 30
scoiattolo, 6
aquila, 25
```

Notiamo subito che il CSV è più strutturato dei file visti nella sezione precedente

- la prima linea sono i nomi delle colonne, separati da virgole (*comma* in inglese): `animale, anni`
- I campi nelle righe successive sono pure separati da virgole `,` : `cane, 12`

Proveremo ora ad importare questo file in Python:

[20]:
```
import csv
with open('esempio-1.csv', encoding='utf-8', newline='') as f:

    # creiamo un oggetto 'lettore' che pescherà righe dal file
    lettore = csv.reader(f, delimiter=',')

    # 'lettore' è un oggetto cosiddetto 'iterabile', cioè se usato in un for produce␣
↪una
    #  sequenza di righe dal csv
    #  NOTA: qui ogni riga del file viene convertita in una una lista di stringhe␣
↪Python!
    for riga in lettore:
        print('Abbiamo appena letto una riga!')
        print(riga)  # stamperà la variabile 'riga', che è una lista di stringhe
        print('')    # stampa una stringa vuota, per separare in verticale
```

```
Abbiamo appena letto una riga!
['animale', 'anni']

Abbiamo appena letto una riga!
['cane', '12']

Abbiamo appena letto una riga!
['gatto', '14']

Abbiamo appena letto una riga!
['pellicano', '30']

Abbiamo appena letto una riga!
['scoiattolo', '6']
```

(continues on next page)

[282] https://it.softpython.org/intro/intro-sol.html

(continued from previous page)

```
Abbiamo appena letto una riga!
['aquila', '25']
```

Notiamo subito dall'output della print che viene stampato il file di esempio, ma ci sono delle parentesi quadre ('[]'). Cosa significano? Quelle che abbiamo stampato sono una *liste* di *stringhe*.

Analizziamo meglio quanto fatto:

```
import csv
```

Python è fornito nativamente di un modulo per il trattamento dei csv, col nome intuitivo `csv`. Con questa istruzione, abbiamo appena caricato questo modulo.

Cosa succede dopo ? Come già fatto per i file a linee in precedenza, apriamo il file in un blocco `with`:

```
with open('esempio-1.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=',')
    for riga in lettore:
        print(riga)
```

Per adesso ignora il `newline=''` e nota che come prima abbiamo specificato l'encoding

Una volta aperto il file, nella riga

```
lettore = csv.reader(f, delimiter=',')
```

chiediamo al modulo `csv` di crearci un oggetto lettore chiamato `lettore` per il nostro file, dicendo a Python che il delimitatore per i campi sono le virgole.

NOTA: `lettore` è il nome di una variabile che stiamo creando, potremmo dare un nome qualunque.

Questo oggetto lettore può essere sfruttato come una specie di generatore di righe usando un ciclo `for`.

```
for riga in lettore:
    print(riga)
```

Nel ciclo `for` sfruttiamo l'oggetto `lettore` per iterare nella lettura del file, producendo ad ogni iterazione una riga che chiamiamo `riga` (ma potrebbe essere un qualunque nome a nostro piacimento). Ad ogni iterazione, la variabile `riga` viene stampata.

Se guardi bene le stampe delle prime liste, vedrai che ogni volta a ogni riga riga viene assegnata una sola lista Python. La lista contiene tanti elementi quanti campi ci sono nel CSV.

⊛ **ESERCIZIO 2.3**: Riscrivi nella cella qua sotto le istruzioni per leggere e stampare il csv, facendo come sempre attenzione all'indentazione:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[21]:
```
# scrivi qui

import csv
with open('esempio-1.csv', encoding='utf-8', newline='') as f:

    # creiamo un oggetto 'lettore' che pescherà righe dal file
    lettore = csv.reader(f, delimiter=',')
```

(continues on next page)

(continued from previous page)

```
    # 'lettore' è un oggetto cosiddetto 'iterabile', cioè se usato in un for produce una
    #  sequenza di righe dal csv
    #  NOTA: qui ogni riga del file viene convertita in una una lista di stringhe Python!
    for riga in lettore:
        print('Abbiamo appena letto una riga!')
        print(riga)  # stamperà la variabile 'riga', che è una lista di stringhe
        print('')    # stampa una stringa vuota, per separare in verticale
```

```
Abbiamo appena letto una riga!
['animale', 'anni']

Abbiamo appena letto una riga!
['cane', '12']

Abbiamo appena letto una riga!
['gatto', '14']

Abbiamo appena letto una riga!
['pellicano', '30']

Abbiamo appena letto una riga!
['scoiattolo', '6']

Abbiamo appena letto una riga!
['aquila', '25']
```

[21]:
```
# scrivi qui
```

```
Abbiamo appena letto una riga!
['animale', 'anni']

Abbiamo appena letto una riga!
['cane', '12']

Abbiamo appena letto una riga!
['gatto', '14']

Abbiamo appena letto una riga!
['pellicano', '30']

Abbiamo appena letto una riga!
['scoiattolo', '6']

Abbiamo appena letto una riga!
['aquila', '25']
```

⊛⊛ **ESERCIZIO 2.4**: prova a mettere in una variabile `listona` una lista contenente tutte le righe estratte dal file, che

quindi sarà una lista di liste che dovrebbe apparire così.

```
[['animale', ' anni'],
 ['cane', '12'],
 ['gatto', '14'],
 ['pellicano', '30'],
 ['scoiattolo', '6'],
 ['aquila', '25']]
```

SUGGERIMENTO: Comincia creando una lista vuota e poi aggiungendo elementi con il metodo `.append`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
# scrivi qui

import csv
with open('esempio-1.csv', encoding='utf-8', newline='') as f:

    # creiamo un oggetto 'lettore' che pescherà righe dal file
    lettore = csv.reader(f, delimiter=',')

    # 'lettore' è un oggetto cosiddetto 'iterabile', cioè se usato in un for produce␣
↪una
    #  sequenza di righe dal csv
    #  NOTA: qui ogni riga del file viene convertita in una una lista di stringhe␣
↪Python!
    listona = []
    for riga in lettore:
        listona.append(riga)
    print(listona)
```

```
[['animale', 'anni'], ['cane', '12'], ['gatto', '14'], ['pellicano', '30'], [
↪'scoiattolo', '6'], ['aquila', '25']]
```

[22]:
```
# scrivi qui
```

```
[['animale', 'anni'], ['cane', '12'], ['gatto', '14'], ['pellicano', '30'], [
↪'scoiattolo', '6'], ['aquila', '25']]
```

⊛⊛ **ESERCIZIO 2.5**: Forse avrai notato che i numeri nella liste sono rappresentati come stringhe tipo `'12'` (nota gli apici), invece che come numeri interi Python (rappresentati senza apici), `12`:

```
Abbiamo appena letto una riga!
['cane', '12']
```

Quindi, leggendo il file e usando dei normali cicli `for`, prova a creare nua variabile `listona` formata come questa, che

- ha solo i dati, la riga con le intestazioni non è presente
- i numeri sono rappresentati propriamente come interi

```
[['cane', 12],
 ['gatto', 14],
```

(continues on next page)

(continued from previous page)

```
 ['pellicano', 30],
 ['scoiattolo', 6],
 ['aquila', 25]]
```

SUGGERIMENTO 1: per saltare una riga puoi usare l'istruzione `next(lettore)`

SUGGERIMENTO 2: per convertire una stringa in un intero, si può usare per es. `int('25')`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
# scrivi qui

import csv
with open('esempio-1.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=',')
    listona = []
    next(lettore)
    for riga in lettore:
        listona.append([riga[0], int(riga[1])])
    print(listona)
```

```
[['cane', 12], ['gatto', 14], ['pellicano', 30], ['scoiattolo', 6], ['aquila', 25]]
```

[23]:
```
# scrivi qui
```

```
[['cane', 12], ['gatto', 14], ['pellicano', 30], ['scoiattolo', 6], ['aquila', 25]]
```

Cos'è esattamente `lettore` ?

Abbiamo detto che `lettore` genera una sequenza di righe, ed è *iterabile*. Nel ciclo `for`, ad ogni ciclo gli chiediamo di leggere una nuova riga, che viene messa nella variabile `riga`. Quindi possiamo chiederci, cosa succede se stampiamo direttamente `lettore`, senza usare nessun `for` ? Vedremo una bella lista o qualcos'altro? Proviamo:

[24]:
```
import csv
with open('esempio-1.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=',')
    print(lettore)
```

```
<_csv.reader object at 0x7f985c63d908>
```

Quello che vediamo è piuttosto deludente.

⊛ **ESERCIZIO 2.6**: Se ti ricordi, nell'introduzione[283] a un certo punto ci siamo trovati nella stessa situazione, quando abbiamo provato a stampare una map[284] . Che potremmo fare per risolvere la situazione?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[283] https://it.softpython.org/intro/intro-sol.html
[284] https://it.softpython.org/intro/intro-sol.html#Trasformazioni-con-le-map

[25]:
```
# scrivi qui

import csv
with open('esempio-1.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=',')
    print(list(lettore))
```

```
[['animale', 'anni'], ['cane', '12'], ['gatto', '14'], ['pellicano', '30'], [
↪'scoiattolo', '6'], ['aquila', '25']]
```

[25]:
```
# scrivi qui
```

```
[['animale', 'anni'], ['cane', '12'], ['gatto', '14'], ['pellicano', '30'], [
↪'scoiattolo', '6'], ['aquila', '25']]
```

Consumare un file

Non tutte le sequenze sono uguali. Da quello che hai visto finora, in Python scorrere un file assomiglia molto a scorrere una lista. Che è molto comodo, ma bisogna stare attenti ad alcune cose. Dato che i file potenzialmente potrebbero occupare terabyte, le funzioni di base di Python per leggere file evitano di caricarli tutti in memoria e tipicamente il file viene scorso un po' alla volta. Ma se il file non viene caricato tutto nell'ambiente di Python in un colpo solo, cosa succede se proviamo a scorrerlo due volte all'interno della stessa `with`? E se proviamo ad usarlo fuori dal `with`, che succede? Guarda i prossimi esercizi per scoprirlo.

⊛ **ESERCIZIO 2.7**: Prendendo la soluzione all'esercizio di prima, prova a chiamare `print(list(lettore))` due volte, in sequenza. Ottieni la stessa stampa entrambe le volte?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[26]:
```
# scrivi qui il codice

import csv
with open('esempio-1.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=',')
    print(list(lettore))
    print(list(lettore))
```

```
[['animale', 'anni'], ['cane', '12'], ['gatto', '14'], ['pellicano', '30'], [
↪'scoiattolo', '6'], ['aquila', '25']]
[]
```

[26]:
```
# scrivi qui il codice
```

```
[['animale', 'anni'], ['cane', '12'], ['gatto', '14'], ['pellicano', '30'], [
↪'scoiattolo', '6'], ['aquila', '25']]
[]
```

⊛ **ESERCIZIO 2.8**: Prendendo la soluzione all'esercizio di prima (usando una sola print), prova qua sotto a spostare la print tutta a sinistra (eliminando gli spazi). Funziona ancora?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[27]:
```
# scrivi qui

import csv
with open('esempio-1.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=',')
#print(list(lettore))     # COMMENTATA, PERCHE' LANCEREBBE ERRORE DI FILE CHIUSO
                          # Non possiamo usare comandi che leggono il file fuori dal␣
↪with !
```

[27]:
```
# scrivi qui
```

⊛⊛⊛ **ESERCIZIO 2.9**: Adesso che abbiamo capito un po' che tipo di bestia è `lettore`, proviamo adesso a produrre questo risultato come già fatto in precedenza, ma usando una *list comprehension* invece del del ciclo `for`:

```
[['cane', 12],
 ['gatto', 14],
 ['pellicano', 30],
 ['scoiattolo', 6],
 ['aquila', 25]]
```

- Se riesci, prova anche a scrivere tutta la trasformazione per creare `listona` in una sola riga, usando la funzione itertools.islice[285] per saltare l'intestazione (per es. `itertools.islice(['A', 'B', 'C', 'D', 'E'], 2, None)` salta i primi due elementi e produce la sequenza C D E F G - nel nostro caso gli elementi prodotti da `lettore` sarebbero righe)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
import csv
import itertools
with open('esempio-1.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=',')
    # scrivi qui
    listona = [[riga[0], int(riga[1])] for riga in itertools.islice(lettore, 1, None)]
    print(listona)
```

```
[['cane', 12], ['gatto', 14], ['pellicano', 30], ['scoiattolo', 6], ['aquila', 25]]
```

[28]:
```
import csv
import itertools
with open('esempio-1.csv', encoding='utf-8', newline='') as f:
```

(continues on next page)

[285] https://docs.python.org/3/library/itertools.html#itertools.islice

(continued from previous page)

```
    lettore = csv.reader(f, delimiter=',')
    # scrivi qui
```

```
[['cane', 12], ['gatto', 14], ['pellicano', 30], ['scoiattolo', 6], ['aquila', 25]]
```

⊛ **ESERCIZIO 2.10**: Crea un file `mio-esempio.csv` nella stessa cartella dove c'è questo foglio Jupyter, copiandoci dentro il contenuto del file `esempio-1.csv`. Poi aggiungici una colonna `descrizione`, ricordandoti di separare il nome di colonna dalla precedente con una virgola. Come valori della colonna, metti nelle righe successive stringhe tipo `i cani camminano`, `i pellicani volano` a seconda dell'animale, etc ricordandoti di separarle dagli anni usando una virgola, così:

```
cane,12,i cani camminano
```

Dopo di che, copia e incolla qua sotto il codice Python per caricare il file, mettendo il nome di file `mio-esempio.csv`, e prova a caricare il tutto, tanto per vedere se funziona:

[29]:
```
# scrivi qui
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

```
animale,anni,descrizione
cane,12,i cani camminano
gatto,14,i gatti camminano
pellicano,30,i pellicani volano
scoiattolo,6,gli scoiattoli volano
aquila,25,le aquile volano
```

⊛ **ESERCIZIO 2.11**: Non tutti i CSV sono strutturati in modo uguale, e a volte quando scriviamo i csv o li importiamo alcuni accorgimenti sono necessari. Cominciamo a vedere che problemi potrebbero sorgere:

- Nel file, prova a mettere uno o due spazi prima dei numeri, per es scrivi come sotto e guarda che succede:

```
cane, 12,i cani volano
```

DOMANDA 2.11.1: Lo spazio viene importato o no?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: sì

DOMANDA 2.11.2: se convertiamo ad intero, lo spazio è un problema?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: no

DOMANDA 2.11.3 Modifica solo la descrizione dei cani da `i cani camminano` a `i cani camminano, ma non volano` e prova a rieseguire la cella che legge il file. Che succede?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Python legge un elemento in più nella lista

DOMANDA 2.11.4: Per ovviare al problema precedente, una soluzione che si può adottare nei CSV è circondare stringhe contenti virgole da doppi apici, così: `"i cani camminano, ma non volano"`. Funziona ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: sì

Leggere come dizionari

Invece di leggere un CSV come se fosse una sequenza di liste, potrebbe essere più conveniente dire a Python di interpretare le linee come se fossero dizionari.

Tramite l'oggetto csv.DictReader[286] sarai in grado di recuperare dizionari, in cui le chiavi saranno i nomi dei campi presi dall'intestazione.

NOTA: diverse versioni di Python producono diversi dizionari:

- $<$ 3.6: `dict`
- 3.6, 3.7: `OrderedDict`
- $\geq$ 3.8: `dict`

La 3.8 è ritornata al vecchio `dict` perchè nella sua nuova implementazione dei dizionari l'ordine delle chiavi è garantito, quindi sarà consistente con quello degli header del CSV.

[30]:
```
import csv
with open('esempio-1.csv', encoding='utf-8', newline='') as f:
    lettore = csv.DictReader(f, delimiter=',')    # Notice we now used DictReader
    for diz in lettore:
        print(diz)
```

```
{'animale': 'cane', 'anni': '12'}
{'animale': 'gatto', 'anni': '14'}
{'animale': 'pellicano', 'anni': '30'}
{'animale': 'scoiattolo', 'anni': '6'}
{'animale': 'aquila', 'anni': '25'}
```

[286] https://docs.python.org/3/library/csv.html#csv.DictReader

Scrivere un CSV

Puoi facilmente creare un CSV instanziando un oggetto `writer`:

> **ATTENZIONE: ASSICURATI DI SCRIVERE NEL FILE GIUSTO!**
>
> Se non stai più che attento ai nomi dei file, **rischi di cancellare dati** !!!

[31]:
```
import csv

# Per scrivere, RICORDATI di specificare l'opzione 'w'
# ATTENZIONE: 'w' rimpiazza *completamente* eventuali file  esistenti!
with open('file-scritto.csv', 'w', newline='') as csv_da_scrivere:

    scrittore = csv.writer(csv_da_scrivere, delimiter=',')

    scrittore.writerow(['This', 'is', 'a header'])
    scrittore.writerow(['some', 'example', 'data'])
    scrittore.writerow(['some', 'other', 'example data'])
```

Leggere e scrivere un CSV

Per scrivere un nuovo CSV prendendo dati da un CSV esistente, potresti annidare un `with` per la lettura dentro uno per la scrittura:

> **ATTENZIONE A SCAMBIARE I NOMI DEI FILE!**
>
> Quando leggiamo e scriviamo è facile commettere un errore e sovrascrivere accidentalmente i nostri preziosi dati.

Per evitare problemi:

- usa nomi espliciti sia per i file di output (es: `esempio-1-arricchito.csv'`) che per gli handle (es: `csv_da_scrivere`)
- fai una copia di backup dei dati da leggere
- controlla sempre prima di eseguire il codice !

[32]:
```
import csv

# Per scrivere, RICORDATI di specificare l'opzione 'w'
# ATTENZIONE: 'w' rimpiazza *completamente* eventuali file  esistenti!
# ATTENZIONE: l'handle *esterno* l'abbiamo chiamato  csv_da_scrivere
with open('esempio-1-arricchito.csv', 'w', encoding='utf-8', newline='') as csv_da_
↪scrivere:
    scrittore = csv.writer(csv_da_scrivere, delimiter=',')

    # Nota come questo 'with' sia dentro quello esterno
    # ATTENZIONE: l'handle *interno* l'abbiamo chiamato csv_da_leggere
    with open('esempio-1.csv', encoding='utf-8', newline='') as csv_da_leggere:
        lettore = csv.reader(csv_da_leggere, delimiter=',')

        for riga in lettore:
            riga.append("qualcos'altro")
```

(continues on next page)

(continued from previous page)

```
            scrittore.writerow(riga)
            scrittore.writerow(riga)
            scrittore.writerow(riga)
```

Vediamo se il file è stato effettivamente scritto provando a leggerlo:

```
[33]: with open('esempio-1-arricchito.csv', encoding='utf-8', newline='') as csv_da_leggere:
    lettore = csv.reader(csv_da_leggere, delimiter=',')

    for riga in lettore:
        print(riga)
```

```
['animale', 'anni', "qualcos'altro"]
['animale', 'anni', "qualcos'altro"]
['animale', 'anni', "qualcos'altro"]
['cane', '12', "qualcos'altro"]
['cane', '12', "qualcos'altro"]
['cane', '12', "qualcos'altro"]
['gatto', '14', "qualcos'altro"]
['gatto', '14', "qualcos'altro"]
['gatto', '14', "qualcos'altro"]
['pellicano', '30', "qualcos'altro"]
['pellicano', '30', "qualcos'altro"]
['pellicano', '30', "qualcos'altro"]
['scoiattolo', '6', "qualcos'altro"]
['scoiattolo', '6', "qualcos'altro"]
['scoiattolo', '6', "qualcos'altro"]
['aquila', '25', "qualcos'altro"]
['aquila', '25', "qualcos'altro"]
['aquila', '25', "qualcos'altro"]
```

CSV Impianti funiviari

Di solito sui cataloghi open data come il popolare CKAN (es dati.trentino.it[287], data.gov.uk[288], European data portal[289]) i file sono organizzati in *dataset*, che sono collezioni di *risorse*: ogni risorsa contiene direttamente un file dentro al catalogo (tipicamente CSV, JSON o XML) oppure un link al file vero e proprio su un server di proprietà dell'organizzazione che ha creato i dati.

Il primo dataset che guarderemo sarà 'Impianti funiviari in esercizio pubblico:

http://dati.trentino.it/dataset/impianti-funiviari-in-esercizio-pubblico

Qua troverai alcune informazioni generiche sul dataset, di importante nota la licenza che è Creative Commons Zero v1.0[290], praticamente è la licenza più permissiva che si possa trovare, e garantisce la certezza di poter riusare i dati senza alcun vincolo.

All'interno della pagina del dataset, è presente una risorsa chiamata "Elenco degli impianti bifuni con movimento a va e vieni":

http://dati.trentino.it/dataset/impianti-funiviari-in-esercizio-pubblico/resource/ceb488a9-696c-4b9d-b8c8-6ca727863931

Alla pagina della risorsa, troviamo un link ad un file CSV (ci si arriva anche cliccando sul bottone blu 'Vai alla risorsa'):

[287] http://dati.trentino.it/
[288] https://data.gov.uk/
[289] https://www.europeandataportal.eu/
[290] http://creativecommons.org/publicdomain/zero/1.0/deed.it

http://www.sif.provincia.tn.it/binary/pat_sif/opendata/Elenco_degli_impianti_bifuni_con_movimento_a_va_e_vieni_aggiornato_a_marzo_2015.1435662007.csv

Se apro il CSV in una tab di Firefox, sul mio computer che ha sistema operativo Linux, vedo una cosa del genere:

```
Denominazione impianto,Codice S.I.F.,Comune,Esercente,Data Primo collaudo,Data Ultimo␣
↪Collaudo,Lunghezza Inclinata (m),Dislivello (m),Stazione di Valle (m.s.l.m.),
↪CapacitÃ  Veicolo (pers.),Portata Max (Pers/h),Opere Paravalanghe ,Servizio
Trento - Sardagna,B007e,Trento,Soc. Trentino Trasporti Esercizio,08-apr-64,18-mar-03,
↪1157,400,200,13,270,#,annuale
Mezzocorona - Monte,B008e,Mezzocorona,Soc.Fun. Monte Mezzocorona,22-dic-04,22-dic-04,
↪933,623,267,7,130,si (frane),annuale
Alba - Ciampac,B014m,Canazei,Soc.Fun. Ciampac e Contrin,20-mag-75,27-nov-14,1902,658,
↪1496,75,840,#,bistagionale
Pecol - Col dei Rossi,B017m,Canazei,S.I.T.C.,09-feb-79,05-dic-97,1313,450,1932,76,
↪1100,si,bistagionale
P. S.Pellegrino - Col Margherita,B022m,Moena,Soc.Fun. Col Margherita,11-mar-82,14-dic-
↪01,1395,639,1874,97,1330,#,bistagionale
Vigo di Fassa - Ciampedie,B026m,Vigo di Fassa,Soc.Catinaccio Impianti a Fune,24-lug-
↪85,23-giu-05,1523,570,1431,101,1300,#,bistagionale
Campitello - Col Rodella ,B027m,Canazei,S.I.T.C.,06-dic-86,07-lug-05,2472,984,1411,
↪126,1160,si,bistagionale
Passo Pordoi - Sass Pordoi,B028m,Canazei,S.I.T.C.,23-mag-95,23-mag-95,1487,708,2240,
↪65,820,#,bistagionale
Col Verde - Rosetta,B029b,Tonadico,Imprese e Territorio Soc. Cons. s.r.l.,21-lug-04,
↪21-lug-04,1336,674,1935,41,420,si + M.G.,bistagionale
Tarlenta - Rifugio Mantova,B030g,Peio,Soc.Fun. Peio,30-dic-10,30-dic-10,2856,992,2001,
↪101,860,si + frane,bistagionale
```

Come atteso, vediamo dei campi separati da virgole.

Problema: caratteri sbagliati ??

Si vede subito un problema nella prima riga delle intestazioni, alla colonna `CapacitÃ Veicolo (pers.)`. Pare che il file abbia la 'à' accentata sbagliata. Ma è davvero un problema del file? Vi dico subito di no. Probabilmente, è il server che non sta dicendo a Firefox quale è il giusto encoding per il file. Firefox non ha capacità di divinazione, e fa solo il suo meglio per mostrare il CSV basandosi sulle informazioni che ha, che possono essere limitate e/o addirittura incorrette. Il mondo non è mai come lo vorremmo…

⊛ **2.12 ESERCIZIO**: scarica il CSV, e prova ad aprirlo in Excel, e / o in LibreOffice Calc. Vedi la à accentata corretta? Se no, prova ad impostare l'encoding (per es. in Calc si chiama 'Character set') a 'Unicode (UTF-8)'

ATTENZIONE SE USI Excel!

Facendo direttamente `File->Apri in Excel`, probabilmente Excel cercherà di immaginarsi da solo come intabellare il CSV, e sbaglierà metterà tutto le righe in una colonna. Per ovviare al problema, dobbiamo dire ad Excel di mostrare un pannello per chiederci come vogliamo aprire il CSV, facendo così:

- In Excel vecchi, cerca `File-> Importa`
- In Excel recenti, clicca la scheda `Dati` e poi seleziona `Da testo`. Per ulteriori riferimenti su Excel, vedere guida di Salvatore Aranzulla[291]

- **NOTA**: Casomai il file non fosse disponibile, nella cartella dove c'è questo notebook Jupyter troverai anche lo stesso file rinominato in `impianti-bifuni.csv`

[291] https://www.aranzulla.it/come-aprire-file-csv-672903.html

Poi dovrebbe risultare più o meno una tabella come questa:

Denominazione impianto	Codice S.I.F.	Comune	Esercente	Data Primo collaudo	Data Ultimo Collaudo	Lunghezza Inclinata (m)	Dislivello (m)	Stazione di Valle (m.s.l.m.)	Capacità Veicolo (pers.)	Portata Max (Pers/h)	Opere Paravalanghe	Servizio
Trento - Sardagna	B007e	Trento	Soc. Trentino Trasporti Esercizio	08-apr-64	18-mar-03	1157	400	200	13	270	#	annuale
Mezzocorona - Monte	B008e	Mezzocorona	Soc.Fun. Monte Mezzocorona	22-dic-04	22-dic-04	933	623	267	7	130	si (frane)	annuale
Alba - Ciampac	B014m	Canazei	Soc.Fun. Ciampac e Contrin	20-mag-75	27-nov-14	1902	658	1496	75	840	#	bistagionale
Pecol - Col dei Rossi	B017m	Canazei	S.I.T.C.	09-feb-79	05-dic-97	1313	450	1932	76	1100	si	bistagionale
P. S.Pellegrino - Col Margherita	B022m	Moena	Soc.Fun. Col Margherita	11-mar-82	14-dic-01	1395	639	1874	97	1330	#	bistagionale
Vigo di Fassa - Ciampedie	B026m	Vigo di Fassa	Soc.Catinaccio Impianti a Fune	24-lug-85	23-giu-05	1523	570	1431	101	1300	#	bistagionale
Campitello - Col Rodella	B027m	Canazei	S.I.T.C.	06-dic-86	07-lug-05	2472	984	1411	126	1160	si	bistagionale
Passo Pordoi - Sass Pordoi	B028m	Canazei	S.I.T.C.	23-mag-95	23-mag-95	1487	708	2240	65	820	#	bistagionale
Col Verde - Rosetta	B029b	Tonadico	Imprese e Territorio Soc. Cons. s.r.l.	21-lug-04	21-lug-04	1336	674	1935	41	420	si + M.G.	bistagionale
Tarlenta - Rifugio Mantova	B030g	Peio	Soc.Fun. Peio	30-dic-10	30-dic-10	2856	992	2001	101	860	si + frane	bistagionale

Importare in Python

Adesso che abbiamo capito due cose sull'encoding, proviamo ad importare il file in Python (ho rinominato il file originale nel più corto `impianti-bifuni.csv`):

[34]:
```
import csv
with open('impianti-bifuni.csv',  encoding='utf-8', newline='') as f:

    # creiamo un oggetto 'lettore' che pescherà righe dal file
    lettore = csv.reader(f, delimiter=',')

    # 'lettore' è un oggetto cosiddetto 'iterabile', cioè se usato in un for produce␣
↪una
    #  sequenza di righe dal csv
    #  NOTA: qui ogni riga del file viene convertita in una una lista di stringhe␣
↪Python!
    for riga in lettore:
        print('Abbiamo appena letto una riga!')
        print(riga)  # stamperà la variabile 'riga', che è una lista di stringhe
        print('')    # stampa una stringa vuota, per separare in verticale
```

```
Abbiamo appena letto una riga!
['Denominazione impianto', 'Codice S.I.F.', 'Comune', 'Esercente', 'Data Primo␣
↪collaudo', 'Data Ultimo Collaudo', 'Lunghezza Inclinata (m)', 'Dislivello (m)',
↪'Stazione di Valle (m.s.l.m.)', 'Capacità Veicolo (pers.)', 'Portata Max (Pers/h)',
↪'Opere Paravalanghe ', 'Servizio']

Abbiamo appena letto una riga!
['Trento - Sardagna', 'B007e', 'Trento', 'Soc. Trentino Trasporti Esercizio', '08-apr-
↪64', '18-mar-03', '1157', '400', '200', '13', '270', '#', 'annuale']

Abbiamo appena letto una riga!
['Mezzocorona - Monte', 'B008e', 'Mezzocorona', 'Soc.Fun. Monte Mezzocorona', '22-dic-
↪04', '22-dic-04', '933', '623', '267', '7', '130', 'si (frane)', 'annuale']

Abbiamo appena letto una riga!
['Alba - Ciampac', 'B014m', 'Canazei', 'Soc.Fun. Ciampac e Contrin', '20-mag-75', '27-
↪nov-14', '1902', '658', '1496', '75', '840', '#', 'bistagionale']
```

(continues on next page)

(continued from previous page)

```
Abbiamo appena letto una riga!
['Pecol - Col dei Rossi', 'B017m', 'Canazei', 'S.I.T.C.', '09-feb-79', '05-dic-97',
↪'1313', '450', '1932', '76', '1100', 'si', 'bistagionale']

Abbiamo appena letto una riga!
['P. S.Pellegrino - Col Margherita', 'B022m', 'Moena', 'Soc.Fun. Col Margherita', '11-
↪mar-82', '14-dic-01', '1395', '639', '1874', '97', '1330', '#', 'bistagionale']

Abbiamo appena letto una riga!
['Vigo di Fassa - Ciampedie', 'B026m', 'Vigo di Fassa', 'Soc.Catinaccio Impianti a␣
↪Fune', '24-lug-85', '23-giu-05', '1523', '570', '1431', '101', '1300', '#',
↪'bistagionale']

Abbiamo appena letto una riga!
['Campitello - Col Rodella ', 'B027m', 'Canazei', 'S.I.T.C.', '06-dic-86', '07-lug-05
↪', '2472', '984', '1411', '126', '1160', 'si', 'bistagionale']

Abbiamo appena letto una riga!
['Passo Pordoi - Sass Pordoi', 'B028m', 'Canazei', 'S.I.T.C.', '23-mag-95', '23-mag-95
↪', '1487', '708', '2240', '65', '820', '#', 'bistagionale']

Abbiamo appena letto una riga!
['Col Verde - Rosetta', 'B029b', 'Tonadico', 'Imprese e Territorio Soc. Cons. s.r.l.',
↪ '21-lug-04', '21-lug-04', '1336', '674', '1935', '41', '420', 'si + M.G.',
↪'bistagionale']

Abbiamo appena letto una riga!
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
```

⊛⊛ **ESERCIZIO 2.13** Vediamo che nel dataset alla terza colonna ci sono i comuni dove ci sono gli impianti. Prova a estrarli in una variabile `comuni`.

- Attenzione alla prima riga, non vogliamo che la scritta `Comune` appaia nel risultato
- Se li metti in una lista, avrai un problema, perchè ci sono comuni che appaiono più volte. Ti conviene quindi creare un insieme vuoto con `set([])` e aggiungere elementi con il metodo `add` (gli insiemi non hanno metodo `append`). Stampando `comuni` dovrebbe apparire qualcosa del genere:

```
{'Vigo di Fassa', 'Trento', 'Mezzocorona', 'Moena', 'Peio', 'Canazei',
'Tonadico'}
```

NOTA: gli insiemi non hanno ordine, potresti vedere un insieme ordinato diversamente !

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[35]:
```
import csv
with open('impianti-bifuni.csv',  encoding='utf-8', newline='') as f:

    lettore = csv.reader(f, delimiter=',')

    # scrivi qui
```

(continues on next page)

(continued from previous page)

```
    next(lettore)
    comuni = set([])
    for riga in lettore:
        comuni.add(riga[2])
    print(comuni)
```

```
{'Tonadico', 'Canazei', 'Moena', 'Trento', 'Mezzocorona', 'Vigo di Fassa', 'Peio'}
```

[35]:

```
import csv
with open('impianti-bifuni.csv',  encoding='utf-8', newline='') as f:

    lettore = csv.reader(f, delimiter=',')

    # scrivi qui
```

```
{'Tonadico', 'Canazei', 'Moena', 'Trento', 'Mezzocorona', 'Vigo di Fassa', 'Peio'}
```

⊛⊛ **ESERCIZIO 2.14**: prova a conteggiare in un dizionario quanti tipi di servizio vengono trovati. Dovresti ottenere un risultato così:

```
{
 'bistagionale': 8,
 'annuale': 2
}
```

SUGGERIMENTO: `'Servizio'` è all'ultima posizione, per ottenerla dalla riga puoi usare l'indice `-1`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[36]:

```
import csv
with open('impianti-bifuni.csv',  encoding='utf-8', newline='') as f:

    lettore = csv.reader(f, delimiter=',')

    # scrivi qui

    next(lettore)
    servizi = {}
    for riga in lettore:
        if riga[-1] in servizi:
            servizi[riga[-1]] += 1
        else:
            servizi[riga[-1]] = 1
    print(servizi)
```

```
{'bistagionale': 8, 'annuale': 2}
```

[36]:
```
import csv
with open('impianti-bifuni.csv',  encoding='utf-8', newline='') as f:

    lettore = csv.reader(f, delimiter=',')

    # scrivi qui
```

```
{'bistagionale': 8, 'annuale': 2}
```

Il mistero delle opere paravalanghe

Se guardiamo nella colonna 'Opere paravalanghe', notiamo che i valori possibili sono:

- `#`
- `si`
- `si (frane)`
- `si + frane`
- `si + M.G`

Sono sempre valori testuali, ma pare esserci una qualche regolarità. Visto che il nostro scopo è convertire in oggetti Python i file in ingresso, forse si potrebbero compiere alcune operazioni per semplificare e adattare al meglio questi dati?

⊛⊛ **2.15 ESERCIZIO**: `si + frane` è la stessa cosa di `si (frane)` ? Assumendo cha sia vero, prova a stampare righe modificate in cui per uniformare gli elementi uguali a `si (frane)` settandoli a `si + frane`. Sia che conosci o non conosci Python, avendo guardato la soluzione della domanda precedente dovresti essere in grado di farlo (ricorda che per modificare un elemento di una lista all'indice `N` si scrive `lista[N] = NUOVO_VALORE`)

Scrivi qua sotto la soluzione

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[37]:
```
# scrivi qui

import csv

with open('impianti-bifuni.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=',')
    next(lettore, None) # saltiamo la riga con l'intestazione
    for row in lettore:
        if 'si (frane)' in riga[11]: # ricordati che l'indice delle colonne inizia da␣
↪0
            riga[11] = 'si + frane'
        print(riga)
```

```
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
```

(continues on next page)

(continued from previous page)

```
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
```

[37]:
```
# scrivi qui
```

```
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
```

⊛ **2.16 DOMANDA**: Quel `'M.G'` non chiarissimo, come neanche se ci sia qualche differenza tra `frane` e `valanghe`. Capire il significato di cosa manipoliamo è importante, forse potresti scoprire il significato nella pagina del dataset[292] ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

Nella sezione *Descrizione Campi* nella pagina del dataset[293] vediamo scritto:

```
Opere paravalanghe = Presenza di opere paravalanghe o misure gestionali
```

(continues on next page)

[292] http://dati.trentino.it/dataset/impianti-funiviari-in-esercizio-pubblico
[293] http://dati.trentino.it/dataset/impianti-funiviari-in-esercizio-pubblico

(continued from previous page)

```
    Mis. Ges. / M.G. = Misure gestionali (piano P.I.S.T.E. o piano P.I.D.A.V.)
    CLV = Commissione locale valanghe
    Si = Presenza di opere paravalanghe
```

Scopriamo quindi che M.G. vuol dire 'Misure Gestionali', purtroppo non ci dicono nulla sulle frane. Questo non è un caso eccezionale, spesso accade che le descrizioni dei dati siano incomplete a si è costretti ad operare in regime di incertezza.

⊛ **2.17 ESERCIZIO**: Cosa significa quel cancelletto # ? Conosci un oggetto di Python che potrebbe rappresentarlo ? Riesci a sostituire tutti i cancelletti con quell'oggetto prima di stampare le liste?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[38]:
```
# scrivi qui

import csv

with open('impianti-bifuni.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=',')
    next(lettore, None) # saltiamo la riga con l'intestazione
    for riga in lettore:
        if '#' in riga[11]: # ricordati che l'indice delle colonne inizia da 0
            riga[11] = None
        print(riga)
```

```
['Trento - Sardagna', 'B007e', 'Trento', 'Soc. Trentino Trasporti Esercizio', '08-apr-
↪64', '18-mar-03', '1157', '400', '200', '13', '270', None, 'annuale']
['Mezzocorona - Monte', 'B008e', 'Mezzocorona', 'Soc.Fun. Monte Mezzocorona', '22-dic-
↪04', '22-dic-04', '933', '623', '267', '7', '130', 'si (frane)', 'annuale']
['Alba - Ciampac', 'B014m', 'Canazei', 'Soc.Fun. Ciampac e Contrin', '20-mag-75', '27-
↪nov-14', '1902', '658', '1496', '75', '840', None, 'bistagionale']
['Pecol - Col dei Rossi', 'B017m', 'Canazei', 'S.I.T.C.', '09-feb-79', '05-dic-97',
↪'1313', '450', '1932', '76', '1100', 'si', 'bistagionale']
['P. S.Pellegrino - Col Margherita', 'B022m', 'Moena', 'Soc.Fun. Col Margherita', '11-
↪mar-82', '14-dic-01', '1395', '639', '1874', '97', '1330', None, 'bistagionale']
['Vigo di Fassa - Ciampedie', 'B026m', 'Vigo di Fassa', 'Soc.Catinaccio Impianti a␣
↪Fune', '24-lug-85', '23-giu-05', '1523', '570', '1431', '101', '1300', None,
↪'bistagionale']
['Campitello - Col Rodella ', 'B027m', 'Canazei', 'S.I.T.C.', '06-dic-86', '07-lug-05
↪', '2472', '984', '1411', '126', '1160', 'si', 'bistagionale']
['Passo Pordoi - Sass Pordoi', 'B028m', 'Canazei', 'S.I.T.C.', '23-mag-95', '23-mag-95
↪', '1487', '708', '2240', '65', '820', None, 'bistagionale']
['Col Verde - Rosetta', 'B029b', 'Tonadico', 'Imprese e Territorio Soc. Cons. s.r.l.',
↪ '21-lug-04', '21-lug-04', '1336', '674', '1935', '41', '420', 'si + M.G.',
↪'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
```

[38]:
```
# scrivi qui
```

```
['Trento - Sardagna', 'B007e', 'Trento', 'Soc. Trentino Trasporti Esercizio', '08-apr-
↪64', '18-mar-03', '1157', '400', '200', '13', '270', None, 'annuale']
['Mezzocorona - Monte', 'B008e', 'Mezzocorona', 'Soc.Fun. Monte Mezzocorona', '22-dic-
↪04', '22-dic-04', '933', '623', '267', '7', '130', 'si (frane)', 'annuale']
['Alba - Ciampac', 'B014m', 'Canazei', 'Soc.Fun. Ciampac e Contrin', '20-mag-75', '27-
↪nov-14', '1902', '658', '1496', '75', '840', None, 'bistagionale']
['Pecol - Col dei Rossi', 'B017m', 'Canazei', 'S.I.T.C.', '09-feb-79', '05-dic-97',
↪'1313', '450', '1932', '76', '1100', 'si', 'bistagionale']
['P. S.Pellegrino - Col Margherita', 'B022m', 'Moena', 'Soc.Fun. Col Margherita', '11-
↪mar-82', '14-dic-01', '1395', '639', '1874', '97', '1330', None, 'bistagionale']
['Vigo di Fassa - Ciampedie', 'B026m', 'Vigo di Fassa', 'Soc.Catinaccio Impianti a␣
↪Fune', '24-lug-85', '23-giu-05', '1523', '570', '1431', '101', '1300', None,
↪'bistagionale']
['Campitello - Col Rodella ', 'B027m', 'Canazei', 'S.I.T.C.', '06-dic-86', '07-lug-05
↪', '2472', '984', '1411', '126', '1160', 'si', 'bistagionale']
['Passo Pordoi - Sass Pordoi', 'B028m', 'Canazei', 'S.I.T.C.', '23-mag-95', '23-mag-95
↪', '1487', '708', '2240', '65', '820', None, 'bistagionale']
['Col Verde - Rosetta', 'B029b', 'Tonadico', 'Imprese e Territorio Soc. Cons. s.r.l.',
↪ '21-lug-04', '21-lug-04', '1336', '674', '1935', '41', '420', 'si + M.G.',
↪'bistagionale']
['Tarlenta - Rifugio Mantova', 'B030g', 'Peio', 'Soc.Fun. Peio', '30-dic-10', '30-dic-
↪10', '2856', '992', '2001', '101', '860', 'si + frane', 'bistagionale']
```

⊛⊛ **2.18 ESERCIZIO**: prova a salvare i dati sistemati (quindi con entrambe le traasformazioni `si (frane) -> si + frane` e `# -> None` nel NUOVO file `impianti-bifuni-sistemato.csv`

- se nella cartella trovi già un file chiamato `impianti-bifuni-sistemato.csv`, ricordati di cancellarlo manualmente prima di iniziare l'esercizio
- il valore `None` come verrà scritto ? Controlla nel file generato dal tuo codice.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[39]:
```
# scrivi qui

import csv

# Per scrivere, RICORDATI di specificare l'opzione 'w'
# ATTENZIONE: 'w' rimpiazza *completamente* eventuali file  esistenti!
# ATTENZIONE: l'handle *esterno* l'abbiamo chiamato  csv_da_scrivere
with open('impianti-bifuni-sistemato.csv', 'w', encoding='utf-8', newline='') as csv_
↪da_scrivere:

    scrittore = csv.writer(csv_da_scrivere, delimiter=',')

    # Nota come questo 'with' sia dentro quello esterno
    # ATTENZIONE: l'handle *interno* l'abbiamo chiamato csv_da_leggere
    with open('impianti-bifuni.csv', encoding='utf-8', newline='') as csv_da_leggere:
        lettore = csv.reader(csv_da_leggere, delimiter=',')
        next(lettore, None) # saltiamo la riga con l'intestazione
        for riga in lettore:
            if 'si (frane)' in row[11]: # ricordati che l'indice delle colonne inizia␣
↪da 0
                row[11] = 'si + frane'
            elif '#' in riga[11]: # ricordati che l'indice delle colonne inizia da 0
                riga[11] = None
```

(continues on next page)

(continued from previous page)

```
            scrittore.writerow(riga)
print()
print('Completata scrittura di:  impianti-bifuni-sistemato.csv')
```

```
Completata scrittura di:  impianti-bifuni-sistemato.csv
```

[39]:
```
# scrivi qui
```

```
Completata scrittura di:  impianti-bifuni-sistemato.csv
```

5.1.5 3. File JSON

Il JSON è un formato più elaborato, molto diffuso nel mondo delle applicazioni web.

Un file json è semplicemente un file di testo, strutturato *ad albero*. Vediamone un esempio, tratto dal dataset stazioni di Bike sharing di Lavis trovato su dati.trentino :

- Fonte dati: dati.trentino.it[294], Servizio Trasporti Provincia Autonoma di Trento
- Licenza: CC-BY 4.0[295]

File bike-sharing-lavis.json:

```
[
  {
    "name": "Grazioli",
    "address": "Piazza Grazioli - Lavis",
    "id": "Grazioli - Lavis",
    "bikes": 3,
    "slots": 7,
    "totalSlots": 10,
    "position": [
      46.139732902099794,
      11.111516155225331
    ]
  },
  {
    "name": "Pressano",
    "address": "Piazza della Croce - Pressano",
    "id": "Pressano - Lavis",
    "bikes": 2,
    "slots": 5,
    "totalSlots": 7,
    "position": [
      46.15368174037716,
      11.106601229430453
    ]
  },
```

(continues on next page)

[294] https://dati.trentino.it/dataset/stazioni-bike-sharing-emotion-trentino
[295] http://creativecommons.org/licenses/by/4.0/deed.it

(continued from previous page)

```
  {
    "name": "Stazione RFI",
    "address": "Via Stazione - Lavis",
    "id": "Stazione RFI - Lavis",
    "bikes": 4,
    "slots": 6,
    "totalSlots": 10,
    "position": [
      46.148180371138814,
      11.096753997622727
    ]
  }
]
```

Come si può notare, il formato del json è molto simile a strutture dati che già abbiamo in Python, come stringhe, numeri interi, float, liste e dizionari. L'unica differenza sono i campi `null` in json che diventano `None` in Python. Quindi la conversione a Python è quasi sempre facile e indolore, e per farla basta usare il modulo già pronto `json` con la funzione `json.load`, che interpreta il testo dal file json e lo converte in strutture dati Python:

[40]:
```
import json

with open('bike-sharing-lavis.json',  encoding='utf-8') as f:
    contenuto_python = json.load(f)

print(contenuto_python)
```

```
[{'position': [46.139732902099794, 11.111516155225331], 'bikes': 3, 'name': 'Grazioli
↪', 'slots': 7, 'totalSlots': 10, 'id': 'Grazioli - Lavis', 'address': 'Piazza␣
↪Grazioli - Lavis'}, {'position': [46.15368174037716, 11.106601229430453], 'bikes':␣
↪2, 'name': 'Pressano', 'slots': 5, 'totalSlots': 7, 'id': 'Pressano - Lavis',
↪'address': 'Piazza della Croce - Pressano'}, {'position': [46.148180371138814, 11.
↪096753997622727], 'bikes': 4, 'name': 'Stazione RFI', 'slots': 6, 'totalSlots': 10,
↪'id': 'Stazione RFI - Lavis', 'address': 'Via Stazione - Lavis'}]
```

Notiamo che quanto letto con la funzione `json.load` non è più semplice testo ma oggetti Python. Per questo json, l'oggetto più esterno è una lista (notate le quadre all'inizio e alla fine del file), e usando `type` su `contenuto_python` ne abbiamo la conferma:

[41]:
```
type(contenuto_python)
```

[41]:
```
list
```

Guardando meglio il JSON, vedrete che è una lista di dizionari. Quindi, per accedere al primo dizionario (cioè quello all0indice zeresimo), possiamo scrivere

[42]:
```
contenuto_python[0]
```

[42]:
```
{'address': 'Piazza Grazioli - Lavis',
 'bikes': 3,
 'id': 'Grazioli - Lavis',
 'name': 'Grazioli',
 'position': [46.139732902099794, 11.111516155225331],
 'slots': 7,
 'totalSlots': 10}
```

Vediamo che è la stazione in Piazza Grazioli. Per accedere al nome esatto, accederemo alla chiave `'address'` del primo dizionario :

[43]:
```
contenuto_python[0]['address']
```

[43]:
```
'Piazza Grazioli - Lavis'
```

Per accedere alla posizione, usiamo la chiave corrispondente:

[44]:
```
contenuto_python[0]['position']
```

[44]:
```
[46.139732902099794, 11.111516155225331]
```

Notiamo come essa sia a sua volta una lista. In JSON, possiamo avere alberi ramificati arbitrariamente, senza necessariamente una struttura regolare (per quanto quando generiamo noi un json sia sempre auspicabile mantenere uno schema regolare nei dati).

JSONL

C'è un particolare tipo di file JSON che si chiama JSONL[296] (nota 'L' alla fine), e cioè un file di testo contenente una sequenza di linee, ciascuna rappresentante un valido oggetto json.

Guardiamo per esempio il file impiegati.jsonl:

```
{"nome": "Mario", "cognome":"Rossi"}
{"nome": "Paolo", "cognome":"Bianchi"}
{"nome": "Luca", "cognome":"Verdi"}
```

Per leggerlo, possiamo aprire il file, separarlo nelle linee di testo e poi interpretare ciascuna come singolo oggetto JSON

[45]:
```
import json

with open('./impiegati.jsonl', encoding='utf-8',) as f:
    lista_testi_json = list(f)        # converte le linee del file di testo in una␣
↪lista Python

# in questo caso avremo un contenuto python per ciascuna riga del file originale

i = 0
for testo_json in lista_testi_json:
    contenuto_python = json.loads(testo_json)    # converte testo json in oggetto␣
↪python
    print('Oggetto ', i)
    print(contenuto_python)
    i = i + 1
```

```
Oggetto  0
{'nome': 'Mario', 'cognome': 'Rossi'}
Oggetto  1
{'nome': 'Paolo', 'cognome': 'Bianchi'}
Oggetto  2
{'nome': 'Luca', 'cognome': 'Verdi'}
```

[ ]:
```

```

[296] http://jsonlines.org/

5.2 Visualizzazione dati

5.2.1 Scarica zip esercizi

Naviga file online[297]

5.2.2 Introduzione

Excel ci permette di creare molti tipi di visualizzazione per i nostri dati ma è molto più limitato rispetto a Python ed il risultato solitamente è di qualità inferiore. In questo tutorial in particolare guarderemo:

Grafici Matplotlib

- grafici a punti e a linee
- posizionare grafici
- mettere oggetti e scritte in sovraimpressione
- istogrammi, grafici a torte e barre

Infografiche SVG e interattivi (cenni)

- RawGraphs
- DataWrapper

Incorporare codice HTML in Jupyter

- calendari
- video
- mappe

Jupyter è molto flessibile, e permette di fare grafici interattivi, mettere insieme collezioni di notebook per creare dei veri e propri libri in formato pdf, così come creare siti web. Qua di seguito mettiamo dei cenni - in future versioni del tutorial le tratteremo più in dettaglio.

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
visualization
    visualization.ipynb
    visualization-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `visualization.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

[297] https://github.com/DavidLeoni/softpython-it/tree/master/visualization

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

5.2.3 1. Matplotlib e Numpy

Matplotlib è una libreria molto famosa per disegnare grafici in Python; è alla base di molti framework per la visualizzazione dei dati e quindi è importante capire il suo funzionamento.

Installazione

Per prima cosa bisogna installare la libreria usando da console:

Windows / Anaconda: Con Anaconda, in teoria già hai Matplotlib! Prova ad eseguire il codice che trovate nel *primo esempio* qua sotto e vedi se viene mostrato qualcosa in Jupyter. Dovessero esserci problemi, si può sempre tentare di eseguire questo codice nell'Anaconda Prompt:

```
conda install matplotlib -c conda-forge
```

Ubuntu: eseguire nel terminale

```
sudo apt-get install python3-matplotlib
```

Mac / Linux generico: eseguire nel terminale

```
python3 -m pip install --user matplotlib
```

Nota Se vedi errori riguardo permessi non sufficienti, potrebbe essere necessario lanciare il comando come amministratore. se questo accade, prova ad installare a livello di sistema con il comando:

```
sudo python3 -m pip install matplotlib
```

Primo esempio

A sua volta Matplotlib[298] utilizza una libreria matematica chiamata Numpy[299]: questa libreria viene automaticamente installata quando installiamo Matplotlib e quindi non dobbiamo servirà installarla manualmente. Queste due librerie sono molto potenti e estensive, tanto da poter coprire un corso intero per ognuna di queste: il nostro obbiettivo però è quello di imparare le funzioni più importanti e capire il funzionamento in generale, per la documentazione completa è possibile accedere alla lista delle funzioni disponibili sui rispettivi siti internet.

Vediamo un primo esempio:

> **NOTA**: La prima volta che esegui la cella qua sotto potrebbe sembrare tutto bloccato!
>
> Potresti anche vedere comparire un messaggio come questo: *UserWarning: Matplotlib is building the font cache using fc-list. This may take a moment. Matplotlib is building the font cache using fc-list.*

[298] http://matplotlib.org
[299] http://numpy.org

Niente paura, non è un errore ma solo un avvertimento (la linea inizia dicendo `UserWarning` non `UserError`): il rallentamento è causato soltanto dal fatto che Matplotlib vuole sapere quali font (tipi di carattere) può utilizzare per disegnare i grafici. È sufficiente aspettare qualche minuto e il processo riprenderà in maniera automatica appena la libreria completerà la ricerca.

[1]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 11, 1.)
y = 2*x + 10

plt.plot(x, y, 'bo')
plt.title('Performance Attesa Esame Fisica')
plt.xlabel('Esercizi svolti')
plt.ylabel('Votazione esame')

plt.show()
```

Il codice qui sopra serve per disegnare il risultato atteso dei voti in relazione al numero di esercizi svolti. Vediamo la prima riga:

```
%matplotlib inline
```

Perché inizia con un `%`? La prima riga in realtà non è una istruzione Python ma è una istruzione per l'integrazione tra Jupyter e Matplotlib, e serve per comunicare a queste due librerie in che modo vogliamo visualizzare i grafici generati da Matplotlib. In questo caso `inline` significa che vogliamo vedere i grafici all'interno del notebook appena eseguiamo la cella che li disegna.

Guardiamo le linee successive:

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
```

Queste importano `matplotlib`, `matplotlib.pyplot` e `numpy`; per le ultime due useremo per brevità degli *alias* `plt` e `np`, che sono dei nomi alternativi che assegnamo noi al momento, per riferirci in modo rapido alle librerie nel codice che segue.

Una volta importate le librerie prepariamo i valori che vogliamo visualizzare. Supponiamo di avere una formula lineare che collega il numero di esercizi svolti con il risultato dell'esame, per esempio possiamo usare questa funzione:

$$y = 2x + 10$$

Per tradurla in Python usando la libreria Numpy, possiamo fare così:

```
x = np.arange(0, 11, 1.)
y = 2*x + 10
```

Per visualizzare la retta sul grafico è necessario mettere dei valori per la `x` in un vettore di Numpy. A tal fine, per eseguire questo *campionamento* scegliendo dei valori di x, abbiamo usato la funzione `arange` di numpy: i parametri sono simili alla funzione già fornita da Python `range` (che restituisce una serie di numeri selezionando un intervallo e opzionalmente un incremento) ma in questo caso `arange` restituisce un oggetto di tipo `numpy.ndarray` che permette di essere utilizzato all'interno di espressioni (al contrario di `tuple` o `list` che sono meno flessibili).

Guardiamo meglio la prima riga:

```
x = np.arange(0, 11, 1.)
```

- Il primo parametro `0` rappresenta il limite inferiore (*compreso* nella serie)
- il secondo `11` il limite superiore (*escluso* dalla serie)
- mentre il terzo `1.` rappresenta l'incremento tra un numero e quello successivo nella serie generata.

⊛ **ESERCIZIO 1.1**: Prova a usare il comando `type` per controllare quale è il tipo di valore ritornato dalla chiamata a `np.arange`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[2]:
```
# scrivi qui il comando type

type(np.arange(0, 11, 1.))
```

[2]:
```
numpy.ndarray
```

[2]:
```
# scrivi qui il comando type
```

[2]:
```
numpy.ndarray
```

Dopo aver generato i valori di `x` e `y` in due vettori possiamo disegnare un grafico. Il grafico più semplice che si possa plottare è un grafico con dei punti nel piano e la funzione da chiamare per farlo è `plt.plot()`:

```
plt.plot(x, y, 'bo')
```

Questa funzione può ricevere come parametro due liste di oggetti aventi *la stessa dimensione* rappresentando posizionalmente le coordinate dei punti mentre il terzo parametro (opzionale), serve per indicare lo stile dell'oggetto da disegnare: nel nostro caso `"bo"` significa colore **b**lue e la **o** dice a Python di stampare cerchi (per maggiori informazioni scrivi `help(plt.plot)`).

Ora l'oggetto `plt` contiene le informazioni riguardanti il grafico che vogliamo vedere, ma mancano ancora alcune informazioni come il titolo e le etichette sugli assi. Per settare questi valori utilizziamo i metodi `plt.title()` (per il titolo), `plt.xlabel()` (per l'etichetta dell'asse x) e `plt.ylabel()` (per l'etichetta dell'asse y):

```
plt.title('Performance Attesa Esame Fisica')
plt.xlabel('Esercizi svolti')
plt.ylabel('Votazione esame')
```

L'ultima istruzione:

```
plt.show()
```

è il metodo che veramente genera il grafico e pulisce l'oggetto `plt` per renderlo pronto a disegnare un nuovo grafico. Per il momento, consideriamo che dopo che aver chiamato questo metodo *non sarà più possibile apportare modifiche al grafico* quindi lo chiameremo per ultimo.

⊛ **ESERCIZIO 1.2**: Riscrivi a mano qua sotto il codice visto sopra, e prova ad eseguirlo con `Ctrl+Invio`:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[3]:
```
# scrivi qui il codice

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 11, 1.)
y = 2*x + 10

plt.plot(x, y, 'bo')
plt.title('Performance Attesa Esame Fisica')
plt.xlabel('Esercizi svolti')
plt.ylabel('Votazione esame')

plt.show()
```

[3]:
```
# scrivi qui il codice
```

⊛ **ESERCIZIO 1.3**: Copia e incolla qua sotto l'esempio precendente, questa volta cambiando il colore della linea (usa `r` per il rosso) e lo stile della linea, usando una linea continua con il carattere `-`.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[4]:
```
# scrivi qui il codice

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 11, 1.)
y = 2*x + 10

plt.plot(x, y, 'r-')
plt.title('Performance Attesa Esame Fisica')
plt.xlabel('Esercizi svolti')
plt.ylabel('Votazione esame')

plt.show()
```

[4]:
```
# scrivi qui il codice
```

⊛ **ESERCIZIO 1.4**: Ricopia con il copia e incolla il codice qua sotto, e prova ad aggiungere la griglia con il comando `plt.grid()`, ricordandoti che puoi sempre usare lo help con `help(plt.grid)` (nota: quando chiedi lo help non devi mettere le parentesi tonde `()` dopo il nome del metodo `grid` !)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[5]:
```
# scrivi qui il codice

%matplotlib inline
import matplotlib
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 11, 1.)
y = 2*x + 10

plt.plot(x, y, 'r-')
plt.title('Performance Attesa Esame Fisica')
plt.xlabel('Esercizi svolti')
plt.ylabel('Votazione esame')
plt.grid()

plt.show()
```

[5]:
```
# scrivi qui il codice
```

⊛ **ESERCIZIO 1.5**: Copia e incolla il codice dell'esempio precedente qua sotto, e prova ad aggiungere l'istruzione:

```
plt.annotate(
    "Risultato minimo\nper la sufficienza",
    xy=(4, 18), arrowprops={'arrowstyle':'->'}, xytext=(6, 17.2))
```

Che cosa succede? Che cosa fanno i parametri? Prova a variare i parametri cercando nella guida di matplotlib[300]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[6]:
```
# scrivi qui il codice

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 11, 1.)
y = 2*x + 10

plt.grid()
plt.plot(x, y, 'r-')
plt.axis([0, 10, 0, 30])
plt.annotate(
        "Risultato minimo\nper la sufficienza",
        xy=(4, 18), arrowprops={'arrowstyle':'->'}, xytext=(6, 17.2))
plt.title('Performance attesa Esame Fisica')
plt.xlabel('Esercizi svolti')
plt.ylabel('Votazione esame')
plt.show()
```

[300] https://matplotlib.org/users/annotations_guide.html

[6]:
```
# scrivi qui il codice
```

5.2.4 2. Stile MATLAB vs. Object-Oriented

Finora abbiamo usato un sistema per la creazione di grafici chiamato `pyplot` (spesso chiamato solo `plt` nel codice). Questo sistema è di proposito molto simile a quello presente in altri software molto usati come MATLAB o GNUplot, che però non sono stati originariamente scritti in Python.

Matplotlib permette anche di disegnare grafici utilizzando un paradigma più proprio di Python, e quindi più flessibile e consigliato.

Purtroppo è importante saper leggere entrambe le notazioni in quanto sono entrambe molto utilizzate.

Come prima cosa è importante conoscere i nomi degli elementi all'interno dei grafici per poter capire meglio queste differenze, esistono 4 elementi principali:

- `Figure` sono la figura completa, cioè l'immagine composta da uno (o più grafici); questo è l'unico elemento a poter essere disegnato.
- `Axes` sono i grafici all'interno di una figura, questi contengono la rappresentazione dei grafici che ci interessa,
- `Axis` sono le assi di un sistema cartesiano, ogni oggetto di tipo `Axes` ne contiene 2 o 3 e ne compongono il sistema di riferimento.
- `Artist` tutto quello che viene disegnato nell'immagine (`Figure`, `Axes`, `Axis`).

Nell'esempio qui sotto viene riportata lo stesso grafico dell'esempio 1 usando il metodo *object-oriented*:

[7]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 11, 1.)
y = 2*x + 10

fig = plt.figure(figsize=(10,2))  # larghezza 10 pollici, altezza 2 pollici
ax = fig.add_subplot(111)
ax.plot(x, y, 'o')
ax.set_title('Performance attesa Esame Fisica')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')
plt.tight_layout()
plt.show()
```

Guardiamo la prima (linea 1):

```
fig = plt.figure()
```

utilizzo `plt.figure()` per creare recuperare una `Figure` dal modulo pyplot,

Poi in linea 2:

```
ax = fig.add_subplot(111)
```

genero gli `Axes`, cioè il grafico vero e proprio usando il metodo `fig.add_subplot()`. Questo metodo prende in ingresso un numero di 3 cifre, ognuna di queste ha un significato particolare:

- La prima cifra rappresenta il numero di righe in cui dividere la *figura*
- La seconda cifra rappresenta il numero di colonne in cui dividere la *figura*
- La terza cifra è la cella corrispondente nella griglia generata con le prime due cifre.

Restituisce un `Axes` all'interno della figura, la cui cella è enumerata partendo da 1, da sinistra verso destra, dall'alto verso il basso.

In questo caso `111` significa che l'`Axes` ritornato sarà allineato ad una griglia di **1** riga, **1** colonna, ed occuperà il posto del **#1** grafico.

Il metodo successivo disegna il grafico nell'`Axes` selezionato (linea 3).

I comandi successivi sono analoghi a quelli negli esempi precedenti:

```
ax.plot(x, y, 'o')
ax.set_title('Performance attesa Esame Fisica')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')
```

, ma fai attenzione al nome e all'oggetto sul quale sono chiamati: adesso abbiamo il metodo `ax.set_title()` invece di `plt.title()` per settare il titolo, `ax.set_xlabel()` invece di `plt.xlabel()` per settare l'etichetta dell'asse delle asciesse e `ax.set_ylabel()` invece di `plt.ylabel()` per settare l'etichetta dell'asse delle ordinate.

L'istruzione `plt.tight_layout()`:

```
plt.tight_layout()
plt.show()
```

infine fa spazio tra i grafici per ottimizzarlo in maniera che non si sormontino tra loro: funziona in maniera automatica e fa tutto il possibile perché questo non accada ma non può fare i miracoli: alcuni layout potrebbero comunque soffrire di qualche sovrapposizione se lo spazio disponibile è davvero limitato.

⊛ **ESERCIZIO 2.1**: Come al solito, inizia a copiare manualmente qua sotto il codice dell'esempio precedente, ed eseguilo con Control+Invio:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[8]:
```
# scrivi qui

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 11, 1.)
y = 2*x + 10

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x, y, 'o')
ax.set_title('Performance attesa Esame Fisica')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

plt.tight_layout()
plt.show()
```

[8]:
```
# scrivi qui
```

Proviamo adesso a mettere due grafici, uno di fianco all'altro. in maniera che ci siano due grafici per gli stessi dati, ma su due righe: nel grafico superiore ci sarà una linea rossa e in quello inferiore i punti saranno blu. Per realizzare questo effetto, dovrai aggiungere dei subplot alla figura. Prova a giocare un po' con i codici per i quadranti di subplot per vedere cosa succede.

[9]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 11, 1.)
y = 2*x + 10

fig = plt.figure()
ax = fig.add_subplot(121)  # griglia a 1 riga, 2 colonne, grafico numero 1
ax.plot(x, y, 'bo')
ax.set_title('GRAFICO A SINISTRA')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

ax = fig.add_subplot(122)  # griglia a 1 riga, 2 colonne, grafico numero 2
ax.plot(x, -y, 'bo')  # notate che mettendo meno davanti a y tutti i valori nell
↪'ndarray diventano negativi
ax.set_title('GRAFICO A DESTRA')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

plt.tight_layout()
plt.show()
```

⊛⊛ **ESERCIZIO 2.2**: Adesso prova a copiare il grafico (anche di/con copia incolla) in maniera che ci siano due grafici per gli stessi dati, ma su due righe: nel grafico superiore ci sarà una linea rossa e in quello inferiore i punti saranno blu. Per realizzare questo effetto, dovrai aggiungere dei subplot alla figura. Prova a giocare un po' con i codici per i quadranti di subplot per vedere cosa succede.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[10]:
```
# scrivi qui

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 11, 1.)
y = 2*x + 10


fig = plt.figure()
ax = fig.add_subplot(211)
ax.plot(x, y, 'r-')
ax.set_title('Performance attesa\nFisica')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

ax = fig.add_subplot(212)
ax.plot(x, -y, 'bo')
ax.set_title("Performance attesa\nFisica")
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

plt.tight_layout()
plt.show()
```

[10]:
```
# scrivi qui
```

⊛⊛ **ESERCIZIO 2.3**: Prova a fare 6 grafici su 3 righe e 2 colonne

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[11]:
```
# scrivi qui

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
```

(continues on next page)

(continued from previous page)

```
x = np.arange(0, 11, 1.)
y = 2*x + 10


fig = plt.figure()
ax = fig.add_subplot(321)
ax.plot(x, y, 'bo')
ax.set_title('grafico 1')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

ax = fig.add_subplot(322)
ax.plot(x, y, 'bo')
ax.set_title('grafico 2')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

ax = fig.add_subplot(323)
ax.plot(x, y, 'bo')
ax.set_title('grafico 3')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

ax = fig.add_subplot(324)
ax.plot(x, y, 'bo')
ax.set_title('grafico 4')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

ax = fig.add_subplot(325)
ax.plot(x, y, 'bo')
ax.set_title('grafico 5')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

ax = fig.add_subplot(326)
ax.plot(x, y, 'bo')
ax.set_title('grafico 6')
ax.set_xlabel('Esercizi svolti')
ax.set_ylabel('Votazione esame')

plt.tight_layout()
plt.show()
```

[11]:
```
# scrivi qui
```

5.2.5 3. Altri tipi di grafici

Oltre a questo tipo di grafico Matplotlib permette ulteriori tipi di grafici come grafici a barre, istogrammi, piechart, scatter, polari, etc. Nella documentazione è possibile trovare la spiegazione dettagliata di tutti i tipi di grafico, nei prossimi esempi ne sono riportati alcuni.

Generiamo una distribuzione

Per cominciare, generiamo dei valori secondo una distribuzione gaussiana e mettiamoli nell'ndarray che chiamiamo `segnale`. Questi valori ci serviranno per esprimenti in seguito. Per questi esempi, useremo un nuovo modo per ottenere le variabili `fig` e `ax` con il metodo `subplots`:

[12]:
```
plt.subplots()
```

[12]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fb0c5773be0>)
```

Se vedi `plt.subplots` ci ritorna due valori come una tupla (e ci mostra anche il grafico per ora vuoto in Jupyter). Il primo valore è la `Figure` e il secondo è un `Axes`. Per metterli rapidamente in variabili con nomi che piacciono a noi come `fig` e `ax`, possiamo usare questa notazione:

[13]:
```
fig, ax = plt.subplots()
```

Proviamo adesso a generare un ndarray di numeri casuali, distribuiti secondo una distribuzione gaussiana. Definiamo la media `mu`, lo scarto quadratico medio `sigma`. La funzione `np.random.normal()` richiede come parametro la media, l'SQM e il numero di esempi che devono essere estratti, in questo caso 500:

[14]:
```
mu = 0  # media
sigma = 1  # sqm
num_bins = 50    # numero di colonne per l'istogramma
segnale = np.random.normal(mu, sigma, 500) # generiamo 500 valori distribuiti come␣
↪una gaussiana, e mettiamoli nell'ndarray 'x'

fig, ax = plt.subplots()  # subplots restituisce una tupla con figura e asse
ax.plot(segnale)
plt.show()
```

Come atteso, i numeri sono centrati sulla linea corrispondente a 0.

Istogrammi

Sarebbe ora interessante produrre un istogramma che mostri in percentuale quanti numeri generati nel paragrafo precedente sono stati pari a -2, quanti pari a -1, 0, 1 , 2 , etc..

[15]:
```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()  # creiamo Figure and Axes in un comando solo

# generiamo il segnale secondo distribuzione gaussiana

mu = 0  # media
sigma = 1  # sqm
# generiamo 500 valori distribuiti come una gaussiana, e mettiamoli nell'ndarray 'x'
segnale = np.random.normal(mu, sigma, 500)

# aggiungiamo l'istogramma
num_bins = 50    # numero di colonne per l'istogramma
# in questo caso hist ritorna tre valori che mettiamo in altrettante variabili
n, bins, columns = ax.hist(segnale, num_bins)

ax.set_xlabel('Segnale')
ax.set_ylabel('Numero di elementi')
ax.set_title('Istogramma di una Gaussiana con $\mu=0$, $\sigma=1$')
fig.tight_layout()
plt.show()
```

Nella parte per l'istogramma, chiamiamo il metodo `ax.hist()`: questo prende come parametri l'array contenente i dati che abbiamo generato (`segnale`), e il numero di partizioni dell'istogramma. Oltre a disegnare la funzione di probabilità dentro l'Axes `ax` restituisce anche i valori numerici per ogni colonna in `n`, i valori per usati per partizionare i dati nelle varie colonne in `bins` e le colonne vere e proprie, intesi come i "rettangoli colorati" che compongono il grafico, in `columns`.

I comandi successivi li conosciamo già, ma facciamo attenzione al `set_title` questa volta: come puoi vedere ci sono dei caratteri `$` all interno del titolo: se hai mai usato o conosci *Latex* avrai sicuramente riconosciuto la notazione, infatti Matplotlib permette di inserire testo *Latex* all'interno dei grafici generati; per chi non lo conoscesse Latex è un linguaggio

di markup[301] che viene utilizzato per scrivere documenti di testo, molto utilizzato in ambito scientifico anche grazie alla potenza e semplicità nell'esprimere formule matematiche.

⊛ **ESERCIZIO 3.1**: Copia sotto il codice per plottare l'istogramma di qua sopra, ma invece di generare il `segnale` con distribuzione gaussiana, prova invece a settarlo uguale a liste come queste. Che grafici prevedi ? Dove saranno allineati lungo l'asse y ?

- `[1,1,1,1,1, 2,2, 3,3,3,3,3,3,3,3]`
- `[3,5,3,5]`
- `[-3,-3,-3,7,7,7,7,7]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[16]:
```
import matplotlib.pyplot as plt
import numpy as np

# scrivi qui il primo grafico


fig, ax = plt.subplots()  # creiamo Figure and Axes in un comando solo

# aggiungiamo l'istogramma
num_bins = 50    # numero di colonne per l'istogramma
# in questo caso hist ritorna tre valori che mettiamo in altrettante variabili
n, bins, columns = ax.hist([1,1,1,1,1, 2,2, 3,3,3,3,3,3,3,3], num_bins)

ax.set_xlabel('Segnale')
ax.set_ylabel('Numero di elementi')
ax.set_title('Istogramma di una Gaussiana con $\mu=0$, $\sigma=1$')
fig.tight_layout()
plt.show()
```

[301] http://wwwcdf.pd.infn.it/AppuntiLinux/latex_ambienti_matematici.htm

[16]:
```
import matplotlib.pyplot as plt
import numpy as np

# scrivi qui il primo grafico
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[17]:
```
# scrivi qui il secondo grafico

import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()  # creiamo Figure and Axes in un comando solo

# aggiungiamo l'istogramma
num_bins = 50    # numero di colonne per l'istogramma
# in questo caso hist ritorna tre valori che mettiamo in altrettante variabili
n, bins, columns = ax.hist([3,5,3,5], num_bins)

ax.set_xlabel('Segnale')
ax.set_ylabel('Numero di elementi')
ax.set_title('Istogramma di una Gaussiana con $\mu=0$, $\sigma=1$')
fig.tight_layout()
plt.show()
```

[17]:
```
# scrivi qui il secondo grafico
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[18]:
```
# scrivi qui il terzo grafico

import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()  # creiamo Figure and Axes in un comando solo
```

(continues on next page)

(continued from previous page)

```
# aggiungiamo l'istogramma
num_bins = 50    # numero di colonne per l'istogramma
# in questo caso hist ritorna tre valori che mettiamo in altrettante variabili
n, bins, columns = ax.hist([-3,-3,-3,7,7,7,7,7], num_bins)

ax.set_xlabel('Segnale')
ax.set_ylabel('Numero di elementi')
ax.set_title('Istogramma di una Gaussiana con $\mu=0$, $\sigma=1$')
fig.tight_layout()
plt.show()
```

[18]:
```
# scrivi qui il terzo grafico
```

Aggiungiamo la curva di fitting

⊛⊛⊛ **ESERCIZIO 3.2** Tipicamente, quando otteniamo da esperimenti una distribuzione di valori, ci interessa ricavare un modello matematico dei dati osservati. In questo caso, siamo fortunati e già sappiamo qual'è il modello giusto dei dati in `segnale`, e cioè una distribuzione gaussiana con i parametri `mu` e `sigma`. Se oltre all'istogramma facciamo anche un plot in sovraimpressione di una curva gaussiana con quei `mu` e `sigma`, dovremmo quindi vedere una linea che segue l'istogramma, che per questo la chiameremo curva di *fitting*. Per ottenere la curva, possiamo usare i valori usati per dividere le colonne come punti sull'asse x e calcolare i valori corrispondenti sull'asse y: questo può essere fatto utilizzando la libreria `scipy.norm` e più precisamente il metodo `scipy.norm.pdf()`, che sta per Normal distribution's Probability Density Function. A questo metodo si passano:

- i valori delle x
- i parametri della normale `mu`
- il parametro `sigma`

e lui restituisce i valori sulla curva di densità corrispondente.

Prova ad aggiungere la funzione di fit come descritto sopra, disegnando una linea tratteggiata con il metodo `ax.plot` visto in precedenza, aggiungendo le *due linee* di codice dove segnalato dal commento.

> **NOTA** Come forse hai notato i valori sull'asse y sono cambiati e la funzione `ax.hist()` ha acquisito un nuovo parametro: `density=True`: questo serve per normalizzare i valori dell'istogramma dividendo il numero di elementi in ogni *bin* per il numero totale di elementi e permettendo di comparare l'istogramma con la funzione di probabilità associata.`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[19]:
```
from scipy.stats import norm
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()  # creiamo Figure and Axes in un comando solo

# generiamo il segnale secondo distribuzione gaussiana

mu = 0  # media
sigma = 1  # sqm
segnale = np.random.normal(mu, sigma, 500) # generiamo 500 valori distribuiti come␣
↪una gaussiana, e mettiamoli nell'ndarray 'x'

# aggiungiamo l'istogramma
num_bins = 50    # numero di colonne per l'istogramma

# in questo caso hist ritorna tre valori che mettiamo in altrettante variabili
# notare density=True per avere valori tra 0 e 1
n, bins, columns = ax.hist(segnale, num_bins, density=True)

# ESERCIZIO: Inserisci qui le due linee per disegnare la curva di fitting

# Qua usiamo direttamente mu e sigma che già conosciamo,
# ma con esperimenti nel mondo reale dovremmo ricavarli in qualche modo
# solo a partire dall'istogramma
y = norm.pdf(bins, mu, sigma)
ax.plot(bins, y, '--')
```

(continues on next page)

(continued from previous page)

```
ax.set_xlabel('Segnale')
ax.set_ylabel('Densità di probabilità')
ax.set_title('Istogramma di una Gaussiana con $\mu=0$, $\sigma=1$')
fig.tight_layout()
plt.show()
```

[19]:
```
from scipy.stats import norm
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots()  # creiamo Figure and Axes in un comando solo

# generiamo il segnale secondo distribuzione gaussiana

mu = 0  # media
sigma = 1  # sqm
segnale = np.random.normal(mu, sigma, 500) # generiamo 500 valori distribuiti come␣
↪una gaussiana, e mettiamoli nell'ndarray 'x'

# aggiungiamo l'istogramma
num_bins = 50    # numero di colonne per l'istogramma

# in questo caso hist ritorna tre valori che mettiamo in altrettante variabili
# notare density=True per avere valori tra 0 e 1
n, bins, columns = ax.hist(segnale, num_bins, density=True)

# ESERCIZIO: Inserisci qui le due linee per disegnare la curva di fitting


ax.set_xlabel('Segnale')
ax.set_ylabel('Densità di probabilità')
ax.set_title('Istogramma di una Gaussiana con $\mu=0$, $\sigma=1$')
fig.tight_layout()
plt.show()
```

5.2.6 Grafici a torta e altro ancora...

Nel prossimo esempio abbiamo il celeberrimo grafico a torta (piechart), la cui creazione è semplicissima:

1. assegnamo delle etichette (`labels`) a tutti gli spicchi;
2. decidiamo le quantità (la larghezza degli spicchi) per ogniuno degli spicchi (usando la stessa posizione in cui abbiamo enumerato le etichette in precedenza);
3. selezioniamo di quanto vogliamo separare ogni spicchio dagli altri (`esplodi` conterrà questa informazione);
4. creiamo *Figure* e *Axes*
5. disegnamo la torta usando il metodo `pie`, questo metodo prende in ingresso le quantità ma ha anche una lunga lista di parametri opzionali, nel nostro caso noi abbiamo usato:
   - `labels` cioè le etichette da apporre ad ogni spicchio,
   - `explode` vedi *punto 3*,
   - `autopct` è una stringa che serve per stampare la percentuale su ogni fetta, richiede come parametro una stringa di formattazione[302], in questo caso riserva una cifra intera (%``1``.1f%%) e assegna la precisione ad una cifra decimale (%1.``1``f%%), ed aggiunge il carattere `%` alla fine (%1.1f``%%``).
   - `startangle` è l'angolo di partenza dal quale iniziare a disegnare il grafico, 90 significa la verticale superiore del grafico.

[20]:
```
import matplotlib.pyplot as plt

labels = ['Pippo', 'Pluto', 'Paperino']
y = [3, 4, 1]
esplodi = [0, 0, 0.1]

fig, ax1 = plt.subplots()
ax1.pie(y, labels=labels, explode=esplodi, autopct='%1.1f%%', startangle=90)
ax1.set_title("Spar(t)izione della pizza")
```

(continues on next page)

[302] https://docs.python.org/2/library/stdtypes.html#string-formatting

(continued from previous page)

```
fig.tight_layout()
#fig.show()
```

⊛ **ESERCIZIO 3.3**: copia qua sotto manualmente il codice per disegnare il grafico a torta

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[21]:
```
# scrivi qui

import matplotlib.pyplot as plt

labels = ['Pippo', 'Pluto', 'Paperino']
y = [3, 4, 1]
esplodi = [0, 0, 0.1]

fig, ax1 = plt.subplots()
ax1.pie(y, labels=labels, explode=esplodi, autopct='%1.1f%%', startangle=90)
ax1.set_title("Spar(t)izione della pizza")

fig.tight_layout()
#fig.show()
```

[21]:
```
# scrivi qui
```

⊛⊛⊛ **ESERCIZIO 3.4** Prova a disegnare una figura con due colonne: in quella di sinistra copia un grafico a torta, e in quella di sinistra metti un grafico a barre verticali utilizzando il metodo `ax.bar()` (primo parametro la posizione x delle barre e secondo l'altezza) per disegnare un diagramma a barre equivalente. Quale ti sembra più chiaro? Prova a giocare con i parametri e `explode startangle`, noterai che nel grafico a torta le proporzioni sembrano cambiare, specialmente se la dimensione degli spicchi è simile.

Non diamo qua tutte le istruzioni per visualizzare bene il grafico a barre, prova un po' a cercare nella documentazione di Matplotlib[303]. Prova anche a:

- impostare colori uguali a quelli della torta

[303] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.bar.html

- impostare la larghezza delle colonne, provando diversi valori. Domanda: la larghezza delle barre può influire sulla percezione dei valori in chi osserva?
- mettere le label sotto le barre (più difficile)

Se non ti viene in mente niente puoi sempre guardare la soluzione.

Ricordati sempre che il comando `help()` è molto importante, usalo quando vuoi sapere di più sui parametri o sulle funzioni che stai utilizzando

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
# scrivi qui

import matplotlib.pyplot as plt

labels = ['Pippo', 'Pluto', 'Paperino']
y = [3, 4, 1]
esplodi = [0, 0, 0.1]

fig, (ax1, ax2) = plt.subplots(ncols=2)
ax1.pie(y, labels=labels, explode=esplodi, autopct='%1.1f%%', startangle=90)
ax1.set_title("Spar(t)izione della pizza")

xticks = [1,2,3] # ci serve per posizionare le barre e anche le label

ax2.bar(xticks,y, color=['b','g','r'], width=0.3, align="center")
ax2.set_title("Spar(t)izione della pizza")
ax2.set_xticklabels(labels)  # verranno posizionate dove sono gli xticks
ax2.set_xticks(xticks)

fig.tight_layout()
```

[22]:
```
# scrivi qui
```

(continues on next page)

(continued from previous page)

5.2.7 Conclusione matplotlib

Come anticipato questa lezione non copre il 100% dei grafici e delle funzionalità presenti in Matplotlib, quindi la principale risorsa nella quale cercare esempi e documentazione è sicuramente la pagina ufficiale di matplotlib[304] ma esistono altre risorse utili che raggruppano le funzioni utili e le loro interfacce cheatsheet per matplotlib[305] e cheatsheet per numpy[306].

5.2.8 Grafici SVG

E' possibile creare velocemente grafici grafici accattivanti senza programmare in siti come Rawgraphs[307]. Una volta creati i grafici, si può esportarli in file SVG e caricarli in Jupyter usando semplice codice Markdown[308]. Un'altro sito da guardare è DataWrapper[309]

Nel caso di grafici interattivi, potresti dover incollare in Jupyter del codice HTML che rappresenta il grafico - vediamo come si fa. In ogni caso tratteremo meglio grafici interattivi creati in Python nel capitolo applicazioni interattive[310]

[304] https://matplotlib.org/gallery/index.html
[305] https://github.com/juliangaal/python-cheat-sheet/blob/master/Matplotlib/Matplotlib.md
[306] https://github.com/juliangaal/python-cheat-sheet/blob/master/NumPy/NumPy.md
[307] https://app.rawgraphs.io/
[308] https://it.softpython.org/jm-templates/project-NAME-SURNAME-ID/markdown.html
[309] https://www.datawrapper.de/
[310] https://it.softpython.org/interactive/interactive-sol.ipynb

5.2.9 importazione in Jupyter come cella HTML

L'HTML è il codice con cui sono scritte le pagine web. Usando i comandi 'magici' di Jupyter `%%HTML` è possibile importare dei frammenti di codice HTML nelle celle. Qua riportiamo solo qualche esempio, vedremo meglio l'HTML nel tutorial sull'estrazione[311] .

ATTENZIONE: le celle HTML NON sono visualizzate nei PDF esportati!

Google Calendar

Se vai nelle impostazioni di un Google Calendar, vedrai che c'è una voce 'Incorpora codice' con dentro del codice che inizia con `<iframe>`.

Se copi quel codice in Jupyter, ricordandoti di mettere `%%HTML` nella prima cella vedrai il calendario.

[23]:
```
%%HTML

<iframe src="https://calendar.google.com/calendar/embed?src=h5tv130eddjl9mmgh55hr2ak7k
↪%40group.calendar.google.com&ctz=Europe%2FRome&dates=20180201%2F20180401" style=
↪"border: 0" width="800" height="600" frameborder="0" scrolling="no"></iframe>
```

```
<IPython.core.display.HTML object>
```

Video YouTube

Se in YouTube sotto un video clicchi su *CONDIVIDI* e poi scegli *incorpora*, vedrai del codice che inizia con `<iframe>`. Puoi incollare tale codice in Jupyter in una cella, basta che nella prima riga scrivi `%%HTML`

[24]:
```
%%HTML

<iframe width="560" height="315" src="https://www.youtube.com/embed/jeG49DxMsvw"␣
↪frameborder="0" allow="autoplay; encrypted-media" allowfullscreen></iframe>
```

```
<IPython.core.display.HTML object>
```

Mappa Umap

Sempre tramite è possibile inserire mappe, per esempio UMap - qua vediamo solo un'esempio, vedremo i dettagli nel tutorial sull'integrazione[312].

NOTA: Queste mappe permettono di essere cliccate e sono spesso più che sufficienti per scopi di viusalizzazione / browsing, ma non è possibile scrivere del codice Python che reagisca ai click. Se hai questa esigenza, bisogna usare sistemi più avanzati discussi nel tutorial Interfacce utente[313]

Questa che segue è la mappa Umap[314] dei Servizi di Rovereto (tutorial creazione mappa[315])

[311] https://it.softpython.org/extraction/extraction-sol.html
[312] https://it.softpython.org/integration/integration-sol.html
[313] https://it.softpython.org/gui/gui-sol.html#Mappe
[314] https://umap.openstreetmap.fr/it/map/servizi-rovereto_41127#14/45.8883/11.0500
[315] https://docs.google.com/presentation/d/1CWo9pFl6jcR1EmDAXOmNeOayfyjfLqLR5-h5U8zxrrk/edit?usp=sharing

[25]:
```
%%HTML

<iframe width="100%" height="300px" frameBorder="0" allowfullscreen src="https://umap.
↪openstreetmap.fr/it/map/servizi-rovereto_41127?scaleControl=false&miniMap=false&
↪scrollWheelZoom=false&zoomControl=true&allowEdit=false&moreControl=true&
↪searchControl=null&tilelayersControl=null&embedControl=null&datalayersControl=true&
↪onLoadPanel=undefined&captionBar=false#14/45.8883/11.0500"></iframe><p><a href=
↪"https://umap.openstreetmap.fr/it/map/servizi-rovereto_41127">Visualizza a schermo␣
↪intero</a></p>
```

```
<IPython.core.display.HTML object>
```

5.2.10 Esportare fogli Jupyter

Puoi esportare un singolo foglio Jupyter in diversi formati:

- formato PDF: `File->Download as-> PDF via Latex (.pdf)`
- sito a pagina singola in formato HTML: `File->Download as-> HTML (.html)`

Per esportare un insieme di fogli Jupyter a intero sito HTML / mega PDF, puoi usare NBSphinx[316] - usato anche per generare tutto il sito di SoftPython a partire da fogli Jupyter (vedi anche codice di SoftPython[317] su Github) !

5.3 Ricerca - espressioni regolari

5.3.1 Scarica zip esercizi

Naviga file online[318]

5.3.2 Introduzione

In questo capitolo parleremo di ricerca nei dati usando le cosidette **Espressioni regolari (regex)**:

Per compiere operazioni di ricerca su testo, quando il problema è relativamente semplice si possono usare alcuni metodi delle stringhe (`replace`, `search`, `index`, `upper`, `lower`, etc…), ma in casi più compicati, per evitare di scrivere tonnellate di codice ed `if` può essere più pratico utilizzare delle *espressioni regolari*.

In particolare, vedremo:

- un esempio dai dati dei trasporti
- filtreremo strade provinciali
- `re.search` e altro

Per capire velocemente cosa sono le regex, prova a giocare un po' con regexcrossword.com[319] (vedere istruzioni[320] e tutorial[321])

[316] https://nbsphinx.readthedocs.io
[317] https://github.com/DavidLeoni/softpython-it
[318] https://github.com/DavidLeoni/softpython-it/tree/master/search
[319] https://regexcrossword.com
[320] https://regexcrossword.com/howtoplay
[321] https://regexcrossword.com/challenges/tutorial/puzzles/1

Quando hai dubbi durante il tutorial, prova le regex online su regex101.com[322]

```
ATTENZIONE: ricordati di selezionare 'Python' nella barra a sinistra in FLAVOR !
```

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
search
    regex.ipynb
    regex-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `regex.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

5.3.3 1 Prendiamoci i dati

Per partire da casi concreti, come già fatto in precedenza andiamo a cercarci dei dati dal catalogo opendata `dati.trentino.it`. In questo caso sceglieremo un file dal dataset Trasporti pubblici del Trentino (formato GTFS)[323]. Sono dati dei trasporti, ma quello che impareremo vale per qualunque dataset che contenga del testo.

⊛ **DOMANDA 1.1**: Quale è la licenza del dataset? Possiamo farci tutto quello che vogliamo ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: La licenza è Creative Commons BY, quindi quando riusiamo dobbiamo citare l'autore

Il formato GTFS[324] è un formato pratico per gli orari e tracciati del trasporto pubblico. Questo formato ci descrive i campi che ci aspettiamo nei file. Ma i file, fisicamente, in che formato sono?

Nel dataset troviamo la risorsa GTFS Urbano TTE[325] che al suo interno contiene un link ad uno zip[326]. Se apriamo lo zip troveremo diversi `.txt` che se attentamente osservati rivelano essere in formato CSV (cominciate a notare l'utilità del formato ;-) ?

[322] https://regex101.com/
[323] http://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs
[324] https://developers.google.com/transit/gtfs/
[325] http://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs/resource/57869023-adfa-467e-8100-76403257d2d1
[326] http://www.ttesercizio.it/opendata/google_transit_urbano_tte.zip

Concentriamoci sul file stops.txt, di cui vediamo un estratto qui:

```
stop_id,stop_code,stop_name,stop_desc,stop_lat,stop_lon,zone_id,wheelchair_boarding
1,28105z,Baselga Del Bondone,,46.078317,11.046924,10110,2
2,28105x,Baselga Del Bondone,,46.078581,11.047541,10110,2
3,27105c,Belvedere,,46.044406,11.105342,10110,2
4,22220z,Lamar Ponte Avisio,,46.134620,11.110914,10110,2
5,28060z,Sp 85 Bivio Sopramonte,,46.085226,11.069313,10110,2
7,24405z,Maso Bolleri,,46.102485,11.124174,10110,2
8,24405x,Maso Bolleri,,46.102234,11.123940,10110,2
9,25205x,Borino,,46.067367,11.165050,10110,2
```

Come ci aspettiamo da un buon file CSV, nella prima riga costituisce le intestazioni e vediamo che i campi sono separati da virgole.

⊛ **DOMANDA 1.2**: Dove possiamo trovare il significato del file ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Pagina del dataset, cerca PDF MITT - Manuale OpenData - v.7: http://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs

5.3.4 2. Verifichiamo che i dati siano corretti

Come avrai intuito, il file `stops.txt` è un file CSV che contiene le informazioni riguardanti le stazioni degli autobus di Trento.

Supponiamo che tu voglia fare avere la lista di tutte le fermate su *strade provinciali* (sigla *SP*):

⊛ **DOMANDA 2.1**: Cerca un po' manualmente dentro il file completo: i dati sono sempre perfettamente regolari come ci piacerebbe? Riesci ad individuare dei criteri per filtrare le righe con strade provinciali ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: I dati non sono uniformi.`Sp` quando è riferito alle superstrade a volte è seguito da spazio e a volte da un punto:

```
5,28060z,Sp 85 Bivio Sopramonte,,46.085226,11.069313,10110,2
1339,32671x,Sp.2 Noriglio,,45.884263,11.070323,10101,2
```

e non sempre è all'inizio del nome di strada, in questo caso è alla fine:

```
2217,35021x,Brancolino Sp.90,,45.900719,11.020254,15091,1
```

In più, la sigla `Sp` si presenta anche in strade non provinciali:

```
2449,22607x,Spini Praga,,46.123666,11.097351,10110,2
```

E ci sono pure strade non provinciali contenenti `sp.` col punto:

```
2376,21385x,Grazioli "Osp. S.Camillo",,46.065677,11.132418,10110,2
```

⊛⊛ **ESERCIZIO 2.2**: Con quello che sai dalla lezione sui formati[327], prova ad aprire il CSV e stampare solo le linee che contengono qualcosa che assomiglia a strade provinciali.

NOTA: non serve che filtri per bene tutte, fai solo qualche tentativo per i casi più ovvi, usando funzioni sulle stringhe che già conosci. Ai casi più difficili ci penseremo in seguito con le regex!

SUGGERIMENTO 1: se devi usare diversi condizioni alternative in un `if`, separale con `or`

SUGGERIMENTO 2: usa il metodo `upper` delle stringhe:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[1]:
```
# scrivi qui

import csv

with open('stops.txt', encoding='utf-8', newline='') as f:
    reader = csv.reader(f, delimiter=',')
    for row in reader:
        name = row[2].upper()

        # mettere le condizioni su più righe non è necessario ma aumenta di
        # molto la leggibilità. Notate che per poterlo fare,
        # le espressioni devono essere racchiuse tra parentesi tonde

        if (name.find("SP ") >= 0
            or name.find("SP.") == 0
            or name.find(" SP.") >= 0):

            print(row)
```

```
['5', '28060z', 'Sp 85 Bivio Sopramonte', '', '46.085226', '11.069313', '10110', '2']
['48', '22080z', 'Sp 76 Carpenedi', '', '46.117195', '11.108678', '10110', '2']
['49', '22080x', 'Sp 76 Carpenedi', '', '46.117171', '11.108438', '10110', '2']
['104', '24040z', 'Sp 131 "Maso Pradiscola"', '', '46.083189', '11.135736', '10110',
↪'2']
['105', '24040x', 'Sp 131 "Maso Pradiscola"', '', '46.083255', '11.135874', '10110',
↪'1']
['109', '24350z', 'Sp 131 "Res. Silvana"', '', '46.097726', '11.126962', '10110', '2']
['110', '24350x', 'Sp 131 "Res. Silvana"', '', '46.097493', '11.126925', '10110', '2']
['115', '24355z', 'Sp 131 Al Maso Specchio', '', '46.104391', '11.123568', '10110', '2
↪']
['116', '24355x', 'Sp 131 Al Maso Specchio', '', '46.104353', '11.123420', '10110', '2
↪']
['131', '23040z', 'Sp 76 "Piac"', '', '46.126690', '11.114532', '10110', '1']
['133', '23035z', 'Sp 76 "Maregioli"', '', '46.130755', '11.121750', '10110', '2']
['134', '23035x', 'Sp 76 "Maregioli"', '', '46.130836', '11.121666', '10110', '1']
['139', '23045z', 'Sp 76 "Via Rossa"', '', '46.119858', '11.111416', '10110', '2']
['140', '23045x', 'Sp 76 "Via Rossa"', '', '46.119812', '11.111266', '10110', '2']
['143', '23055z', 'Sp 76 Dos di Lamar', '', '46.124939', '11.113463', '10110', '2']
['144', '23055x', 'Sp 76 Dos di Lamar', '', '46.124718', '11.113018', '10110', '1']
['154', '24520z', 'Sp 131 "Paganin"', '', '46.106288', '11.137610', '10110', '2']
['155', '24520x', 'Sp 131 "Paganin"', '', '46.106524', '11.137306', '10110', '2']
['245', '28045z', 'Sp 85 "Soraval"', '', '46.083258', '11.063955', '10110', '2']
```

(continues on next page)

[327] http://it.softpython.org/formats/formats-sol.html#File-CSV

(continued from previous page)

```
['246', '28045x', 'Sp 85 "Soraval"', '', '46.083094', '11.063877', '10110', '2']
['439', '27050z', 'Sp 90 "Maso Prudenza"', '', '46.028867', '11.111521', '10110', '2']
['440', '27050x', 'Sp 90 "Maso Prudenza"', '', '46.029290', '11.111771', '10110', '2']
['1296', '32651x', "Sp.2 Loc. Beccache'", '', '45.882087', '11.071643', '10101', '2']
['1334', '32651z', "Sp.2 Loc. Beccache'", '', '45.882063', '11.071733', '10101', '2']
['1338', '32661x', 'Sp.2 Fr. Campolongo', '', '45.886995', '11.063902', '10101', '2']
['1339', '32671x', 'Sp.2 Noriglio', '', '45.884263', '11.070323', '10101', '2']
['1340', '32671z', 'Sp.2 Noriglio', '', '45.884371', '11.069842', '10101', '2']
['1341', '32661z', 'Sp.2 Fr. Campolongo', '', '45.886988', '11.063994', '10101', '2']
['2217', '35021x', 'Brancolino Sp.90', '', '45.900719', '11.020254', '15091', '1']
['2218', '35021z', 'Brancolino Sp.90', '', '45.900853', '11.020410', '15091', '1']
['2285', '38601z', 'Sp.20 S.Sisinio', '', '45.924877', '11.021092', '18511', '2']
['2286', '38611z', 'Sp.20 Zisi', '', '45.931009', '11.022102', '18511', '2']
['2287', '38611x', 'Sp.20 Zisi', '', '45.931416', '11.022116', '18511', '2']
['2288', '38601x', 'Sp.20 S.Sisinio', '', '45.924981', '11.021093', '18511', '2']
['2363', '38591x', 'Sp.20 Maso Tiaf', '', '45.936511', '11.008597', '18511', '2']
['2364', '38621x', 'Sp.20 Bivio Per Bordala', '', '45.930910', '11.007172', '18511',
↪'2']
['2367', '38621z', 'Sp.20 Bivio Per Bordala', '', '45.930864', '11.007233', '18511',
↪'2']
['2368', '38591z', 'Sp.20 Maso Tiaf', '', '45.935890', '11.008594', '18511', '2']
['2377', '23050x', 'Sp 76 Bivio S.Lazzaro', '', '46.129006', '11.115500', '10110', '2
↪']
['2510', '32641z', 'Sp.2 Bivio Per Cisterna', '', '45.881281', '11.075365', '10101',
↪'2']
['2843', '35161x', 'Sp.20 Bivio Noarna', '', '45.915825', '11.016559', '15091', '2']
['2844', '35161z', 'Sp.20 Bivio Noarna', '', '45.915880', '11.016803', '15091', '2']
['2897', '32218x', 'Sp.23 "Nero Cubo"', '', '45.855421', '11.002179', '10101', '2']
['2936', '37551z', 'Sp.89 Maso Brentegam', '', '45.880439', '11.050882', '10101', '2']
```

[1]:
```
# scrivi qui
```

```
['5', '28060z', 'Sp 85 Bivio Sopramonte', '', '46.085226', '11.069313', '10110', '2']
['48', '22080z', 'Sp 76 Carpenedi', '', '46.117195', '11.108678', '10110', '2']
['49', '22080x', 'Sp 76 Carpenedi', '', '46.117171', '11.108438', '10110', '2']
['104', '24040z', 'Sp 131 "Maso Pradiscola"', '', '46.083189', '11.135736', '10110',
↪'2']
['105', '24040x', 'Sp 131 "Maso Pradiscola"', '', '46.083255', '11.135874', '10110',
↪'1']
['109', '24350z', 'Sp 131 "Res. Silvana"', '', '46.097726', '11.126962', '10110', '2']
['110', '24350x', 'Sp 131 "Res. Silvana"', '', '46.097493', '11.126925', '10110', '2']
['115', '24355z', 'Sp 131 Al Maso Specchio', '', '46.104391', '11.123568', '10110', '2
↪']
['116', '24355x', 'Sp 131 Al Maso Specchio', '', '46.104353', '11.123420', '10110', '2
↪']
['131', '23040z', 'Sp 76 "Piac"', '', '46.126690', '11.114532', '10110', '1']
['133', '23035z', 'Sp 76 "Maregioli"', '', '46.130755', '11.121750', '10110', '2']
['134', '23035x', 'Sp 76 "Maregioli"', '', '46.130836', '11.121666', '10110', '1']
['139', '23045z', 'Sp 76 "Via Rossa"', '', '46.119858', '11.111416', '10110', '2']
['140', '23045x', 'Sp 76 "Via Rossa"', '', '46.119812', '11.111266', '10110', '2']
['143', '23055z', 'Sp 76 Dos di Lamar', '', '46.124939', '11.113463', '10110', '2']
['144', '23055x', 'Sp 76 Dos di Lamar', '', '46.124718', '11.113018', '10110', '1']
['154', '24520z', 'Sp 131 "Paganin"', '', '46.106288', '11.137610', '10110', '2']
```

(continues on next page)

(continued from previous page)

```
['155', '24520x', 'Sp 131 "Paganin"', '', '46.106524', '11.137306', '10110', '2']
['245', '28045z', 'Sp 85 "Soraval"', '', '46.083258', '11.063955', '10110', '2']
['246', '28045x', 'Sp 85 "Soraval"', '', '46.083094', '11.063877', '10110', '2']
['439', '27050z', 'Sp 90 "Maso Prudenza"', '', '46.028867', '11.111521', '10110', '2']
['440', '27050x', 'Sp 90 "Maso Prudenza"', '', '46.029290', '11.111771', '10110', '2']
['1296', '32651x', "Sp.2 Loc. Beccache'", '', '45.882087', '11.071643', '10101', '2']
['1334', '32651z', "Sp.2 Loc. Beccache'", '', '45.882063', '11.071733', '10101', '2']
['1338', '32661x', 'Sp.2 Fr. Campolongo', '', '45.886995', '11.063902', '10101', '2']
['1339', '32671x', 'Sp.2 Noriglio', '', '45.884263', '11.070323', '10101', '2']
['1340', '32671z', 'Sp.2 Noriglio', '', '45.884371', '11.069842', '10101', '2']
['1341', '32661z', 'Sp.2 Fr. Campolongo', '', '45.886988', '11.063994', '10101', '2']
['2217', '35021x', 'Brancolino Sp.90', '', '45.900719', '11.020254', '15091', '1']
['2218', '35021z', 'Brancolino Sp.90', '', '45.900853', '11.020410', '15091', '1']
['2285', '38601z', 'Sp.20 S.Sisinio', '', '45.924877', '11.021092', '18511', '2']
['2286', '38611z', 'Sp.20 Zisi', '', '45.931009', '11.022102', '18511', '2']
['2287', '38611x', 'Sp.20 Zisi', '', '45.931416', '11.022116', '18511', '2']
['2288', '38601x', 'Sp.20 S.Sisinio', '', '45.924981', '11.021093', '18511', '2']
['2363', '38591x', 'Sp.20 Maso Tiaf', '', '45.936511', '11.008597', '18511', '2']
['2364', '38621x', 'Sp.20 Bivio Per Bordala', '', '45.930910', '11.007172', '18511',
↪'2']
['2367', '38621z', 'Sp.20 Bivio Per Bordala', '', '45.930864', '11.007233', '18511',
↪'2']
['2368', '38591z', 'Sp.20 Maso Tiaf', '', '45.935890', '11.008594', '18511', '2']
['2377', '23050x', 'Sp 76 Bivio S.Lazzaro', '', '46.129006', '11.115500', '10110', '2
↪']
['2510', '32641z', 'Sp.2 Bivio Per Cisterna', '', '45.881281', '11.075365', '10101',
↪'2']
['2843', '35161x', 'Sp.20 Bivio Noarna', '', '45.915825', '11.016559', '15091', '2']
['2844', '35161z', 'Sp.20 Bivio Noarna', '', '45.915880', '11.016803', '15091', '2']
['2897', '32218x', 'Sp.23 "Nero Cubo"', '', '45.855421', '11.002179', '10101', '2']
['2936', '37551z', 'Sp.89 Maso Brentegam', '', '45.880439', '11.050882', '10101', '2']
```

[2]:
```
"Ciao MONDO".upper()
```

[2]:
```
'CIAO MONDO'
```

[3]:
```
"SoftPython".upper()
```

[3]:
```
'SOFTPYTHON'
```

SUGGERIMENTO 3: usa anche il metodo `find` che ritorna la posizione di una sottostringa all'interno di un'altra.

[4]:
```
"ab cde".find('cd')
```

[4]:
```
3
```

[5]:
```
"ab cde".find(' c')
```

[5]:
```
2
```

Ricordati che gli indici delle stringhe iniziano da zero:

[6]:
```
"ab cde".find('a')
```

[6]:
```
0
```

Quando `find` non trova qualcosa ritorna `-1` per segnalarlo:

```
[7]: "ab cde".find('z')
```

```
[7]: -1
```

Ricordati che `find` distingue tra maiuscole / minuscole, quindi non troverà la `D` maiuscola:

```
[8]: "ab cde".find('D')
```

```
[8]: -1
```

Discussione

Prosegui la lettura solo dopo aver provato l'esercizio precedente.

Pur avendo analizzato un file piccolo, sono spuntati fuori parecchi casi da trattare. Per poter quindi filtrare agevolmente insiemi grandi di dati senza specificare mille casi particolari, è bene cominciare a pensare a tutte le caratteristiche comuni dell'*insieme* di stringhe che vogliamo ottenere. Si può poi implementare dei filtri in Python usando le *regex*.

5.3.5 3. Introduzione alle regex

Vediamo cosa sono queste **espressioni regolari**:

> **WIKIPEDIA**
>
> Una *espressione regolare* (in lingua inglese regular expression o, in forma abbreviata, regexp, regex o RE) è una sequenza di simboli (quindi una stringa) che identifica un insieme di stringhe: essa definisce una funzione che prende in ingresso una stringa, e restituisce in uscita un valore del tipo sì/no, a seconda che la stringa segua o meno un certo *pattern*.

L'utilizzo di una espressione regolare è sicuramente più veloce perché ci permette di cercare non solo una stringa bensì un intero insieme di stringhe, detto appunto *pattern*. Una considerazione ESERCIZIO è che nonostante i concetti riguardo le regex siano universali, alcune implementazioni si differenziano nel comportamento in alcuni casi particolari oppure aggiungendo funzionalità *non standard*. Quindi, se già avete usate le *regex* nel vostro linguaggio preferito, state attenti a controllare le eventuali differenze con Python!

Stringhe e sequenze di escape

Le *regex* si esprimono usando semplici stringhe Python, per cui è meglio spendere 5 minuti per capire meglio alcune peculiarità delle stringhe. Quando indichiamo una stringa in Python, possiamo inserire delle squenze speciali dette *sequenze di escape*, come per esempio `\n` in `"ciao\nSoftPython"` che dice a Python che quando stampiamo la stringa, dopo aver stampato la stringa `ciao` deve andare a capo e quindi stampare la seguente `SoftPython`:

```
[9]: print("ciao\nSoftPython")
```

```
ciao
SoftPython
```

Se non vogliamo che Python interpreti queste sequenze, perchè vogliamo che in fase di stampa sia invece proprio stampato il `\n`, possiamo aggiungere prima della stringa una `r` così :

```
[10]: print(r"ciao\nSoftPython")
```

```
ciao\nSoftPython
```

La `r` prima dell'inizio della stringa serve ad indicare a Python che la seguente è una raw string, cioè una stringa in cui non deve espandere le *sequenze di escape* (cioè `\` seguito da altri caratteri al fine di generare caratteri non stampabili, per esempio `\n` è il carattere di new-line).

⊛ **ESERCIZIO 3.1**: magari già conosci le *sequenze di escape*, si trovano in molti linguaggi. Se non le conosci, prova a scrivere i comandi qua sotto, sempre in nuove celle:

- `print("ciao mondo")`
- `print("ciao\tmondo")`
- `print("ciao\nmondo")`
- `print("ciao\rmondo")` (questo è strano…)
- `print("ciao\\mondo")`

Che differenze noti? E se metti il carattere `r` *davanti* alle stringhe (quindi subito prima del doppio apice `"`, come in `r"hello"`), che succede ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[11]:
```
# scrivi qui

print("ciao mondo")
print("ciao\tmondo")
print("ciao\nmondo")
print("ciao\\mondo")
print("ciao\rmondo")
print(r"ciao mondo")
print(r"ciao\tmondo")
print(r"ciao\nmondo")
print(r"ciao\\mondo")
print(r"ciao\rmondo")
```

```
ciao mondo
ciao    mondo
ciao
mondo
ciao\mondo
mondo
ciao mondo
ciao\tmondo
ciao\nmondo
ciao\\mondo
ciao\rmondo
```

[11]:
```
# scrivi qui
```

```
ciao mondo
ciao    mondo
ciao
mondo
ciao\mondo
mondo
ciao mondo
ciao\tmondo
ciao\nmondo
ciao\\mondo
ciao\rmondo
```

La nostra prima regex

Proviamo ad eseguire la nostra prima regex:

[12]:
```
# diciamo a Python che vogliamo usare il modulo per usare le regex, che si chiama 're'
# NOTA: per quanto breve,  're' è proprio il nome del modulo, non un comando speciale␣
↪!
import re

# usando la funzione search del modulo 're', effettuiamo una ricerca del pattern 'Sp'
# dentro 'Sp.89 Maso Brentegam'
re.search('Sp', 'Sp.89 Maso Brentegam')
```

[12]:
```
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
```

Osservando `span=(0, 2), match='Sp'` nel risultato, si vede che Python ha individuato la stringa `Sp` dicendoci che inizia alla posizione 0 (inclusa) e termina alla posizione 2 (esclusa) di `'Sp.89 Maso Brentegam'`

Possiamo ottenere i numeri delle posizioni con `start()` e `end()`:

[13]:
```
re.search('Sp', 'Sp.89 Maso Brentegam').start()
```

[13]:
```
0
```

[14]:
```
re.search('Sp', 'Sp.89 Maso Brentegam').end()
```

[14]:
```
2
```

Proviamo ora a cercare un'altra stringa che sappiamo essere presente, come `Maso`:

[15]:
```
re.search('Maso', 'Sp.89 Maso Brentegam')
```

[15]:
```
<_sre.SRE_Match object; span=(6, 10), match='Maso'>
```

Notiamo che la stringa è stata trovata tra il sesto carattere (incluso) e il decimo (escluso)

ESERCIZIO: Prova ad estrarre qua sotto le posizioni in cui la stringa sopra è stata trovata

[16]:
```
import re
# scrivi qui
```

Proviamo ora a cercare una stringa che sappiamo *non* esserci, come `'blabla'`:

[17]:
```
print(re.search('blabla', 'Sp.89 Maso Brentegam'))
```

```
None
```

Vediamo che Python ci restituisce l'oggetto None, per indicare che non ha trovato nulla.

ESERCIZIO: Perchè in questo caso abbiamo messo il `print`? Per capirlo, prova a scrivere qui sotto la chiamata alla `re.search` senza usare la `print`, e vedi che succede (non dovrebbe succedere proprio niente perchè Jupyter di default non ci mostra gli oggetti `None` a meno che non li stampiamo esplicitamente)

[18]:
```
import re
# scrivi qui
```

Maiuscole e minuscole

La ricerca nelle regex distingue tra maiuscole e minuscole, quindi se cerchiamo `SP` tutto maiuscolo in `'Sp.89 Maso Brentegam'` non vedremo nulla come risultato:

[19]:
```
re.search('SP', 'Sp.89 Maso Brentegam')
```

Per dire a Python di ignorare maiuscole/minuscole, si può aggiungere il parametro `re.I`:

[20]:
```
re.search('SP', 'Sp.89 Maso Brentegam', re.I)
```

[20]:
```
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
```

Notare come ora il parametro `match` sia più utile, perchè ci dice esattamente quali caratteri maiuscoli o minuscoli hanno fatto il match (i nquesto caso `Sp`).

Trovare tutti i match

`re.search` ritorna il primo match trovato. Se vogliamo trovarli tutti, bisogna usare `finditer`. Per esempio, possiamo provare a cercare la stringa `ab` in `abcabd`:

[21]:
```
for trovato in re.finditer('ab', 'abcabd'):
    print(trovato)
```

```
<_sre.SRE_Match object; span=(0, 2), match='ab'>
<_sre.SRE_Match object; span=(3, 5), match='ab'>
```

⊛ **ESERCIZIO 3.2**: Se provi a stampare direttamente il risultato restituito `re.finditer`, non otterrai molte informazioni. Conosci un modo per ottenere una bella lista senza usare il for ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[22]:
```
# scrivi qui

import re
list(re.finditer('ab', 'abcabd'))
```

[22]:
```
[<_sre.SRE_Match object; span=(0, 2), match='ab'>,
 <_sre.SRE_Match object; span=(3, 5), match='ab'>]
```

[22]:
```
# scrivi qui
```

[22]:
```
[<_sre.SRE_Match object; span=(0, 2), match='ab'>,
 <_sre.SRE_Match object; span=(3, 5), match='ab'>]
```

Il metacarattere punto

Proviamo ora una regex un po' più interessante, per esempio cerchiamo nella stringa `abcabd` tutte le stringhe che iniziano con `b` e sono seguite da un solo qualsiasi carattere. Come facciamo ad indicare che vogliamo un solo carattare, ma senza specificare quale ? Possiamo usare il *metacarattere* punto `.`, che ha un significato speciale nelle *regex* e agisce come jolly:

[23]:
```
for trovato in re.finditer('b.', 'abcabd'):
    print(trovato)
```

```
<_sre.SRE_Match object; span=(1, 3), match='bc'>
<_sre.SRE_Match object; span=(4, 6), match='bd'>
```

⊛ **ESERCIZIO 3.3**: Prova a cercare tutte le stringhe in `abcabd` che iniziano con `a` e sono seguite da due caratteri qualsiasi:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[24]:
```
import re

# scrivi qui

for trovato in re.finditer('a..', 'abcabd'):
    print(trovato)
```

```
<_sre.SRE_Match object; span=(0, 3), match='abc'>
<_sre.SRE_Match object; span=(3, 6), match='abd'>
```

[24]:
```
import re

# scrivi qui
```

```
<_sre.SRE_Match object; span=(0, 3), match='abc'>
<_sre.SRE_Match object; span=(3, 6), match='abd'>
```

⊛ **ESERCIZIO 3.4**: Prova a cercare tutte le stringhe in `abcabd` che iniziano con un carattere qualsiasi e sono seguite da `b`:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[25]:
```
import re
# scrivi qui

for trovato in re.finditer('.b', 'abcabd'):
    print(trovato)
```

```
<_sre.SRE_Match object; span=(0, 2), match='ab'>
<_sre.SRE_Match object; span=(3, 5), match='ab'>
```

[25]:
```
import re
# scrivi qui
```

```
<_sre.SRE_Match object; span=(0, 2), match='ab'>
<_sre.SRE_Match object; span=(3, 5), match='ab'>
```

Abbiamo capito che i punti sono caratteri speciali. E se volessimo cercare invece proprio un punto, per distinguire per esempio `Sp.89 Maso Brentegam` da `Sp 85 Bivio Sopramonte` che invece ha lo spazio dopo `Sp`? Per filtIn questo caso, prima del punto dovremmo usare il carattere di barra rovesciata `\` , detto anche `carattere di escape`:

[26]:
```
re.search('Sp\.', 'Sp.89 Maso Brentegam')
```

[26]:
```
<_sre.SRE_Match object; span=(0, 3), match='Sp.'>
```

Mettendo `Sp 85 Bivio Sopramonte` non dovrebbe trovare nulla (lo spazio non è un punto!):

[27]:
```
re.search('Sp\.', 'Sp 85 Bivio Sopramonte')
```

⊛⊛ **ESERCIZIO 3.5**: Prova a trovare tutte le stringhe che contengono un punto seguito da qualsiasi carattere nella stringa `'il.corso.soft.python'` (dovrebbe trovare match per `.c`, `.s` e `.p`):

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
import re

# scrivi qui

for trovato in re.finditer('\..', 'il.corso.soft.python'):
    print(trovato)
```

```
<_sre.SRE_Match object; span=(2, 4), match='.c'>
<_sre.SRE_Match object; span=(8, 10), match='.s'>
<_sre.SRE_Match object; span=(13, 15), match='.p'>
```

[28]:
```
import re

# scrivi qui
```

```
<_sre.SRE_Match object; span=(2, 4), match='.c'>
<_sre.SRE_Match object; span=(8, 10), match='.s'>
<_sre.SRE_Match object; span=(13, 15), match='.p'>
```

Scriviamoci una funzione di test

Dato che scrivere i comandi per stampare i risultati dei test può diventare ripetitivo e noioso, ci conviene creare una *funzione* con dentro delle istruzioni di stampa da eseguire automaticamente (per più info sulle funzioni, vedi capitolo 3 Pensare in Python[328]):

[29]:
```
import re

def test_regex(pattern):
    names = ['Baselga del Bondone',
             'Spini Bregenz',
             'Sp.89 Maso Brentegam',
             'sp 90 "Maso Prudenza"',
             'Brancolino Sp.90']
    for name in names:
        print(re.search(pattern, name, re.I))

test_regex("SP")
```

```
None
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
<_sre.SRE_Match object; span=(0, 2), match='sp'>
<_sre.SRE_Match object; span=(11, 13), match='Sp'>
```

In Python, le funzioni si dichiarano con la parola riservata `def` seguita dal nome della funzione che scegliamo arbitrariamente noi. In questo caso il nome scelto è `test_regex`:

```
def test_regex(pattern):
```

Poi, a questa funzione abbiamo deciso che bisognerà passare un parametro, che chiamiamo `pattern` (ma potremmo chiamarlo come ci pare, anche `pippo`).

NOTA: alla fine della prima riga, ci sono dei due punti `:` se dimentichi di metterli potresti trovarti con strani errori di syntassi !

La nostra funzione farà qualcosa con questa variabile che abbiamo chiamato `pattern`:

```
def test_regex(pattern):
    names = ['Baselga Del Bondone',
             'Spini Bregenz',
             'Sp.89 Maso Brentegam',
             'sp 90 "Maso Prudenza"',
             'Brancolino Sp.90']
    for name in names:
        print(re.search(pattern, name, re.I))
```

In questo caso, per ciascun nome, eseguirà questa riga (vedremo in seguito il contenuto della `print`):

```
print(re.search(pattern, name, re.I))
```

sfruttando la variabile `pattern` che passeremo al momento di chiamare la funzione:

[30]:
```
test_regex("SP")
```

```
None
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
```

(continues on next page)

[328] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2004.html

(continued from previous page)

```
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
<_sre.SRE_Match object; span=(0, 2), match='sp'>
<_sre.SRE_Match object; span=(11, 13), match='Sp'>
```

⊛ **ESERCIZIO 3.6**: Copia a mano la funzione di sopra qua sotto, ed eseguila con Control + Invio:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[31]:
```
# scrivi qui

import re

def test_regex(pattern):
    names = ['Spini Bregenz',
             'Sp.89 Maso Brentegam',
             'sp 90 "Maso Prudenza"',
             'Brancolino Sp.90']
    for name in names:
        print(re.search(pattern, name, re.I))

test_regex("SP")
```

```
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
<_sre.SRE_Match object; span=(0, 2), match='sp'>
<_sre.SRE_Match object; span=(11, 13), match='Sp'>
```

[31]:
```
# scrivi qui
```

```
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
<_sre.SRE_Match object; span=(0, 2), match='Sp'>
<_sre.SRE_Match object; span=(0, 2), match='sp'>
<_sre.SRE_Match object; span=(11, 13), match='Sp'>
```

Filtriamo bene gli sp

È ora di usare la nostra prima regex 'seria', per farlo dobbiamo guardare le differenze tra le stringhe corrette e quella errata: in questo caso sappiamo che ogni *strada provinciale* ha un numero. Proviamo con questa regex:

[32]:
```
test_regex(r"sp.\d\d")
```

```
None
<_sre.SRE_Match object; span=(0, 5), match='Sp.89'>
<_sre.SRE_Match object; span=(0, 5), match='sp 90'>
<_sre.SRE_Match object; span=(11, 16), match='Sp.90'>
```

Tornando alla regex `r"sp.\d\d"`:

- Come detto nella sezione precedente, la `r` prima dell'inizio della stringa serve ad indicare a Python che la seguente è una raw string, cioè una stringa in cui non deve espandere le *sequenze di escape* (cioè `\` seguito da altri caratteri al fine di generare caratteri non stampabili, per esempio `\n` è il carattere di new-line).

- Abbiamo visto che le lettere dell'alfabeto (e i numeri) hanno semplicemente il loro valore.
- Come detto in precedenza, il carattere `.` in questo caso è un *metacarattere* che in questo caso si comporta come un "jolly" e può identificare qualsiasi carattere ad eccezione de carattere di fine riga (solitamente).
- `\d` sono due caratteri ma ai fini della regex sono da considerarsi uno solo. Ogni volta che vediamo il carattere `\` è da considerarsi assieme al carattere successivo. Il significato in questo caso è una qualsiasi cifra tra `0` e `9`.

⊛⊛⊛ **ESERCIZIO 3.7**: Prova ad integrare l'esempio di regex qui sopra con quello di lettura del file stops.txt precedente, usando nell'`if` le regex invece delle `find` (se non lo hai risolto prendilo *dalle soluzioni*):

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[33]:
```
# scrivi qui

import csv
import re

with open('stops.txt', encoding='utf-8', newline='') as f:
    reader = csv.reader(f, delimiter=',')
    for row in reader:
        if re.search(r'sp.\d\d', row[2], re.I):
            print(row)
```

```
['5', '28060z', 'Sp 85 Bivio Sopramonte', '', '46.085226', '11.069313', '10110', '2']
['48', '22080z', 'Sp 76 Carpenedi', '', '46.117195', '11.108678', '10110', '2']
['49', '22080x', 'Sp 76 Carpenedi', '', '46.117171', '11.108438', '10110', '2']
['104', '24040z', 'Sp 131 "Maso Pradiscola"', '', '46.083189', '11.135736', '10110',
↪'2']
['105', '24040x', 'Sp 131 "Maso Pradiscola"', '', '46.083255', '11.135874', '10110',
↪'1']
['109', '24350z', 'Sp 131 "Res. Silvana"', '', '46.097726', '11.126962', '10110', '2']
['110', '24350x', 'Sp 131 "Res. Silvana"', '', '46.097493', '11.126925', '10110', '2']
['115', '24355z', 'Sp 131 Al Maso Specchio', '', '46.104391', '11.123568', '10110', '2
↪']
['116', '24355x', 'Sp 131 Al Maso Specchio', '', '46.104353', '11.123420', '10110', '2
↪']
['131', '23040z', 'Sp 76 "Piac"', '', '46.126690', '11.114532', '10110', '1']
['133', '23035z', 'Sp 76 "Maregioli"', '', '46.130755', '11.121750', '10110', '2']
['134', '23035x', 'Sp 76 "Maregioli"', '', '46.130836', '11.121666', '10110', '1']
['139', '23045z', 'Sp 76 "Via Rossa"', '', '46.119858', '11.111416', '10110', '2']
['140', '23045x', 'Sp 76 "Via Rossa"', '', '46.119812', '11.111266', '10110', '2']
['143', '23055z', 'Sp 76 Dos di Lamar', '', '46.124939', '11.113463', '10110', '2']
['144', '23055x', 'Sp 76 Dos di Lamar', '', '46.124718', '11.113018', '10110', '1']
['154', '24520z', 'Sp 131 "Paganin"', '', '46.106288', '11.137610', '10110', '2']
['155', '24520x', 'Sp 131 "Paganin"', '', '46.106524', '11.137306', '10110', '2']
['245', '28045z', 'Sp 85 "Soraval"', '', '46.083258', '11.063955', '10110', '2']
['246', '28045x', 'Sp 85 "Soraval"', '', '46.083094', '11.063877', '10110', '2']
['439', '27050z', 'Sp 90 "Maso Prudenza"', '', '46.028867', '11.111521', '10110', '2']
['440', '27050x', 'Sp 90 "Maso Prudenza"', '', '46.029290', '11.111771', '10110', '2']
['2217', '35021x', 'Brancolino Sp.90', '', '45.900719', '11.020254', '15091', '1']
['2218', '35021z', 'Brancolino Sp.90', '', '45.900853', '11.020410', '15091', '1']
['2285', '38601z', 'Sp.20 S.Sisinio', '', '45.924877', '11.021092', '18511', '2']
['2286', '38611z', 'Sp.20 Zisi', '', '45.931009', '11.022102', '18511', '2']
['2287', '38611x', 'Sp.20 Zisi', '', '45.931416', '11.022116', '18511', '2']
['2288', '38601x', 'Sp.20 S.Sisinio', '', '45.924981', '11.021093', '18511', '2']
['2363', '38591x', 'Sp.20 Maso Tiaf', '', '45.936511', '11.008597', '18511', '2']
```

(continues on next page)

(continued from previous page)

```
['2364', '38621x', 'Sp.20 Bivio Per Bordala', '', '45.930910', '11.007172', '18511',
↪'2']
['2367', '38621z', 'Sp.20 Bivio Per Bordala', '', '45.930864', '11.007233', '18511',
↪'2']
['2368', '38591z', 'Sp.20 Maso Tiaf', '', '45.935890', '11.008594', '18511', '2']
['2377', '23050x', 'Sp 76 Bivio S.Lazzaro', '', '46.129006', '11.115500', '10110', '2
↪']
['2843', '35161x', 'Sp.20 Bivio Noarna', '', '45.915825', '11.016559', '15091', '2']
['2844', '35161z', 'Sp.20 Bivio Noarna', '', '45.915880', '11.016803', '15091', '2']
['2897', '32218x', 'Sp.23 "Nero Cubo"', '', '45.855421', '11.002179', '10101', '2']
['2936', '37551z', 'Sp.89 Maso Brentegam', '', '45.880439', '11.050882', '10101', '2']
```

[33]:
```
# scrivi qui
```

```
['5', '28060z', 'Sp 85 Bivio Sopramonte', '', '46.085226', '11.069313', '10110', '2']
['48', '22080z', 'Sp 76 Carpenedi', '', '46.117195', '11.108678', '10110', '2']
['49', '22080x', 'Sp 76 Carpenedi', '', '46.117171', '11.108438', '10110', '2']
['104', '24040z', 'Sp 131 "Maso Pradiscola"', '', '46.083189', '11.135736', '10110',
↪'2']
['105', '24040x', 'Sp 131 "Maso Pradiscola"', '', '46.083255', '11.135874', '10110',
↪'1']
['109', '24350z', 'Sp 131 "Res. Silvana"', '', '46.097726', '11.126962', '10110', '2']
['110', '24350x', 'Sp 131 "Res. Silvana"', '', '46.097493', '11.126925', '10110', '2']
['115', '24355z', 'Sp 131 Al Maso Specchio', '', '46.104391', '11.123568', '10110', '2
↪']
['116', '24355x', 'Sp 131 Al Maso Specchio', '', '46.104353', '11.123420', '10110', '2
↪']
['131', '23040z', 'Sp 76 "Piac"', '', '46.126690', '11.114532', '10110', '1']
['133', '23035z', 'Sp 76 "Maregioli"', '', '46.130755', '11.121750', '10110', '2']
['134', '23035x', 'Sp 76 "Maregioli"', '', '46.130836', '11.121666', '10110', '1']
['139', '23045z', 'Sp 76 "Via Rossa"', '', '46.119858', '11.111416', '10110', '2']
['140', '23045x', 'Sp 76 "Via Rossa"', '', '46.119812', '11.111266', '10110', '2']
['143', '23055z', 'Sp 76 Dos di Lamar', '', '46.124939', '11.113463', '10110', '2']
['144', '23055x', 'Sp 76 Dos di Lamar', '', '46.124718', '11.113018', '10110', '1']
['154', '24520z', 'Sp 131 "Paganin"', '', '46.106288', '11.137610', '10110', '2']
['155', '24520x', 'Sp 131 "Paganin"', '', '46.106524', '11.137306', '10110', '2']
['245', '28045z', 'Sp 85 "Soraval"', '', '46.083258', '11.063955', '10110', '2']
['246', '28045x', 'Sp 85 "Soraval"', '', '46.083094', '11.063877', '10110', '2']
['439', '27050z', 'Sp 90 "Maso Prudenza"', '', '46.028867', '11.111521', '10110', '2']
['440', '27050x', 'Sp 90 "Maso Prudenza"', '', '46.029290', '11.111771', '10110', '2']
['2217', '35021x', 'Brancolino Sp.90', '', '45.900719', '11.020254', '15091', '1']
['2218', '35021z', 'Brancolino Sp.90', '', '45.900853', '11.020410', '15091', '1']
['2285', '38601z', 'Sp.20 S.Sisinio', '', '45.924877', '11.021092', '18511', '2']
['2286', '38611z', 'Sp.20 Zisi', '', '45.931009', '11.022102', '18511', '2']
['2287', '38611x', 'Sp.20 Zisi', '', '45.931416', '11.022116', '18511', '2']
['2288', '38601x', 'Sp.20 S.Sisinio', '', '45.924981', '11.021093', '18511', '2']
['2363', '38591x', 'Sp.20 Maso Tiaf', '', '45.936511', '11.008597', '18511', '2']
['2364', '38621x', 'Sp.20 Bivio Per Bordala', '', '45.930910', '11.007172', '18511',
↪'2']
['2367', '38621z', 'Sp.20 Bivio Per Bordala', '', '45.930864', '11.007233', '18511',
↪'2']
['2368', '38591z', 'Sp.20 Maso Tiaf', '', '45.935890', '11.008594', '18511', '2']
['2377', '23050x', 'Sp 76 Bivio S.Lazzaro', '', '46.129006', '11.115500', '10110', '2
↪']
```

(continues on next page)

(continued from previous page)

```
['2843', '35161x', 'Sp.20 Bivio Noarna', '', '45.915825', '11.016559', '15091', '2']
['2844', '35161z', 'Sp.20 Bivio Noarna', '', '45.915880', '11.016803', '15091', '2']
['2897', '32218x', 'Sp.23 "Nero Cubo"', '', '45.855421', '11.002179', '10101', '2']
['2936', '37551z', 'Sp.89 Maso Brentegam', '', '45.880439', '11.050882', '10101', '2']
```

5.3.6 4. Sintassi delle Python RegEx

Proviamo ora a guardare alcuni meta-caratteri importanti nelle regular expression in Python - scusate per l'orribile tabella ma fare tabelle in Jupyter è sempre problematico[329]

Metacaratteri

Modificatore	Descrizione
\	usato come *escape* (cioè segnalare che il carattere a seguire, nonostante sia un carattere speciale deve essere trattato come se non lo fosse) o per iniziare una *sequenza* (vedi sotto).
.	usato per rappresentare un qualsiasi carattere ad eccezione della nuova linea (con l'opzione `re.A` possiamo rimuovere anche questa eccezione).
^	usato per indicare l'inizio della riga
$	usato per indicare la fine della riga
[...]	usate per racchiudere l'insieme di caratteri che verificano questa espressione regolare
[^...]	usate per racchiudere l'insieme di caratteri che se presenti **NON** verificano questa espressione regolare
A|B	usato per indicare una rappresentazione alternativa, è valida sia che appaia `A`, sia che appaia `B`
()	usate come in matematica per indicare la precedenza sulle operazioni

Nel codice qui sotto definiamo la funzione `test_regex_num()` che è come quella già ma vista, ma usando dei numeri di telefono al posto dei nomi delle fermate; per ogni numero controlliamo se l'espressione regolare viene soddisfatta e se lo è lo stampiamo ed in fine testiamo varie proprietà di questi numeri di telefono.

[34]:
```
import re
def test_regex_num(pattern):
    numbers = ['3471234567',  #NOTA: i numeri sono in formato stringa !!!!
               '3303303367',
               '3232123323',
               '3383123222']
    for num in numbers:
        if re.search(pattern, num):
            print(num)
    print("-----")

print("Tutti i numeri che contengono 33")
test_regex_num("33")
print("Tutti i numeri che iniziano per 33")
test_regex_num("^33")
print("Tutti i numeri che hanno come penultima cifra 2")
test_regex_num("2.$")
print("Tutti i numeri che contengono 212 o 312")
test_regex_num("212|312")
```

(continues on next page)

[329] https://github.com/jupyter/notebook/issues/3024

(continued from previous page)

```
# Oppure
test_regex_num("[23]12")
# Oppure
test_regex_num("(2|3)12")
```

```
Tutti i numeri che contengono 33
3303303367
3232123323
3383123222
-----
Tutti i numeri che iniziano per 33
3303303367
3383123222
-----
Tutti i numeri che hanno come penultima cifra 2
3232123323
3383123222
-----
Tutti i numeri che contengono 212 o 312
3232123323
3383123222
-----
3232123323
3383123222
-----
3232123323
3383123222
-----
```

⊛⊛ **ESERCIZIO 4.1** Prova a scrivere qua sotto i pattern che soddisfano le proprietà richieste nelle chiamate a `test_regex_num`. Cerca di *non guardare* all'esercizio precedente:

[35]:
```
import re

def test_regex_num(pattern):
    numbers = ['3471234567',
               '3303303367',
               '3232123323',
               '3383123222']
    for num in numbers:
        if re.search(pattern, num):
            print(num)
    print("-----")

print("Tutti i numeri che contengono 32")
test_regex_num("")    # metti il pattern giusto

print("Tutti i numeri che finiscono per 67")
test_regex_num("")

print("Tutti i numeri che hanno come quarta cifra 3")
test_regex_num("")

print("Tutti i numeri che contengono 232 o 233 o 234")
test_regex_num("")
# Oppure
```

(continues on next page)

(continued from previous page)

```
test_regex_num("")
# Oppure
test_regex_num("")
```

```
Tutti i numeri che contengono 32
3471234567
3303303367
3232123323
3383123222
-----
Tutti i numeri che finiscono per 67
3471234567
3303303367
3232123323
3383123222
-----
Tutti i numeri che hanno come quarta cifra 3
3471234567
3303303367
3232123323
3383123222
-----
Tutti i numeri che contengono 232 o 233 o 234
3471234567
3303303367
3232123323
3383123222
-----
3471234567
3303303367
3232123323
3383123222
-----
3471234567
3303303367
3232123323
3383123222
-----
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[36]:
```
# SOLUZIONE

print("Tutti i numeri che contengono 32")
test_regex_num("32")

print("Tutti i numeri che finiscono per 67")
test_regex_num("67$")

print("Tutti i numeri che hanno come quarta cifra 3")
test_regex_num("^...3")

print("Tutti i numeri che contengono 232 o 233 o 234")
test_regex_num("232|233|234")
# Oppure
```

(continues on next page)

(continued from previous page)

```
test_regex_num("23[234]")
# Oppure
test_regex_num("23(2|3|4)")
# Oppure
#test_regex("23[2-4]")
```

```
Tutti i numeri che contengono 32
3232123323
3383123222
-----
Tutti i numeri che finiscono per 67
3471234567
3303303367
-----
Tutti i numeri che hanno come quarta cifra 3
3303303367
3383123222
-----
Tutti i numeri che contengono 232 o 233 o 234
3471234567
3232123323
3383123222
-----
3471234567
3232123323
3383123222
-----
3471234567
3232123323
3383123222
-----
```

[36]:

```
Tutti i numeri che contengono 32
3232123323
3383123222
-----
Tutti i numeri che finiscono per 67
3471234567
3303303367
-----
Tutti i numeri che hanno come quarta cifra 3
3303303367
3383123222
-----
Tutti i numeri che contengono 232 o 233 o 234
3471234567
3232123323
3383123222
-----
3471234567
3232123323
3383123222
-----
```

(continues on next page)

(continued from previous page)

```
3471234567
3232123323
3383123222
-----
```

Ripetizioni

Le espressioni regolari possono anche gestire delle ripetizioni di particolari pattern utilizzando altri caratteri speciali.

Modificatori	Descrizione
`{m, n}`	il carattere o gruppo a cui è riferito viene ripetuto almeno `m` volte fino ad un massimo di `n` volte.
`{m}`	il carrattere o il gruppo a cui è riferito viene ripetuto esattamente `m` volte
`?`	il carattere o il gruppo a cui è riferito viene ripetuto `0` o `1` volta. Equivale a `{,1}`.
`*`	il carattere o il gruppo a cui è riferito viene ripetuto `0` o più volte . Equivale a `{,}`.
`+`	il carattere o il gruppo a cui è riferito viene ripetuto `1` o più volte. Equivale a `{1,}`.

I caratteri o gruppi a cui si riferiscono i `modificatori delle ripetizioni` appena precedenti ad essi, vediamo un esempio

[37]:
```
print(re.search("a+b", "aaaabbb"))
print(re.search("a+b", "bab"))
print(re.search("a+b", "bbb"))
print(re.search("a+b", "aaaa"))
```

```
<_sre.SRE_Match object; span=(0, 5), match='aaaab'>
<_sre.SRE_Match object; span=(1, 3), match='ab'>
None
None
```

Come pui vedere nell'esempio qui sopra il pattern `a+b` indica `a` una o più volte, seguito da una `b`. Quando si ha un match puoi vedere nell'oggetto ritornato dal metodo `re.search()` la sottostringa che ha verificato la regex (usando il metodo `group()`) e gli indici della posizione di essa all'interno della stringa (chiamando il metodo `.span()`).

Funzione `pass_n_fail`

Qui sotto la funzione `pass_n_fail()` prende come parametri un `pattern` e due liste di stringhe: `pass_list` e `fail_list`. La funzione verifica se quelle che appartengono alla prima lista matchano l'espressione regolare e se quelle nella seconda non la matchano:

[38]:
```
def pass_n_fail(pattern, pass_list, fail_list):
    for p in pass_list:
        if not re.search(pattern, p):
            print("ERRORE: '{}' non matcha il pattern '{}' ma dovrebbe farlo!".
↪format(p, pattern))
    for f in fail_list:
        if re.search(pattern, f):
            print("ERRORE: '{}' matcha il pattern '{}' ma non dovrebbe farlo!".
↪format(f, pattern))
```

Guardiamo quest'esempio:

NOTA: il fatto che nell'output dell'esempio sia scritto `ERRORE` è voluto, lo scopo della funzione è proprio segnalarci che non abbiamo messo i parametri giusti !

[39]:
```
# ESEMPIO
pass_n_fail("c",
            ["aa","a"],  # espressioni che vorremmo matchassero il pattern "c"
            ["b","c"]    # espressioni che vorremmo NON matchassero il pattern "c"
           )
```

```
ERRORE: 'aa' non matcha il pattern 'c' ma dovrebbe farlo!
ERRORE: 'a' non matcha il pattern 'c' ma dovrebbe farlo!
ERRORE: 'c' matcha il pattern 'c' ma non dovrebbe farlo!
```

Quest'esempio qua invece non dovrebbe darci nessun output, perchè le stringhe `"aa"` che `"a` matchano il pattern `"a"`, e le stringhe `"b"` e `"c"` non matchano il pattern `"a"`:

[40]:
```
# ESEMPIO
pass_n_fail("a",
            ["aa","a"],  # espressioni che vorremmo matchassero il pattern "a"
            ["b","c"]    # espressioni che vorremmo NON matchassero il pattern "a"
           )
```

⊛⊛⊛ **ESERCIZIO 4.2**: Prova tu a fornire esempi nelle due liste qua sotto (non preoccuparti se vedi scritto `ERRORE` nell'output della cella prima ancora di cominciare, se metti esempi giusti ed esegui la cella i messaggi di errore dovrebbero sparire):

[41]:
```
pass_n_fail("a",
            ["",""], # metti esempi che matchano
            ["",""]  # metti esempi che non matchano
           )
```

```
ERRORE: '' non matcha il pattern 'a' ma dovrebbe farlo!
ERRORE: '' non matcha il pattern 'a' ma dovrebbe farlo!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[42]:
```
# SOLUZIONE

pass_n_fail("a",
            ["a","(aa)+b"],
            ["c","bb"]
           )
```

[42]:

[43]:
```
 # Come sopra, ma un po più difficile:
pass_n_fail("ab+",
            ["",""],
            ["",""]
           )
```

```
ERRORE: '' non matcha il pattern 'ab+' ma dovrebbe farlo!
ERRORE: '' non matcha il pattern 'ab+' ma dovrebbe farlo!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[44]:
```
# SOLUZIONE

pass_n_fail("ab+",
            ["abb","abbb"],
            ["bbb","aa"]
           )
```

[44]:

[45]:
```
# Come sopra, un po più difficile ancora:
pass_n_fail("ab*",
            ["",""],
            ["",""]
           )
```

```
ERRORE: '' non matcha il pattern 'ab*' ma dovrebbe farlo!
ERRORE: '' non matcha il pattern 'ab*' ma dovrebbe farlo!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[46]:
```
# SOLUZIONE

pass_n_fail("ab*",
            ["a","ba"],
            ["bb","b"]
           )
```

[46]:

[47]:
```
# OK! Ancora un paio
pass_n_fail("^[ab]+[^ab]$",
            ["",""],
            ["",""]
           )
```

```
ERRORE: '' non matcha il pattern '^[ab]+[^ab]$' ma dovrebbe farlo!
ERRORE: '' non matcha il pattern '^[ab]+[^ab]$' ma dovrebbe farlo!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[48]:
```
# SOLUZIONE

pass_n_fail("^[ab]+[^ab]$",
```

(continues on next page)

(continued from previous page)

```
            ["bbbbc","aaac"],
            ["acccca","cbac"]
           )
```

[48]:

```

```

[49]:

```
# L'ultima
pass_n_fail(".?\.{3}$",
            ["",""], #TODO
            ["",""]
           )
```

```
ERRORE: '' non matcha il pattern '.?\.{3}$' ma dovrebbe farlo!
ERRORE: '' non matcha il pattern '.?\.{3}$' ma dovrebbe farlo!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[50]:

```
# SOLUZIONE
pass_n_fail(".?\.{3}$",
            ["foo...","..."],
            ["bar..","... "]
           )
```

[50]:

```

```

5.3.7 5. Sequenze

A volte vogliamo considerare insiemi molto grandi di possibili simboli in una espressione regolare: per esempio se vogliamo validare la struttura di un **indirizzo email** vogliamo controllare che

- contenga almeno 3 caratteri
- una `@`
- altri 3 caratteri
- un punto
- e almeno altri 2 caratteri.

Il problema è che alcuni caratteri non possono essere presenti nelle email (come ad esempio `\|{}()[]` etc…), e se dovessimo scrivere un set di caratteri da escludere usando l'espressione `[^...]` ci costerebbe molto tempo e spazio, inoltre sarebbe facile dimenticarsi qualche simbolo e quasi impossibile da leggere.

Per ovviare a questo problema sono stati introdotte delle scorciatoie: delle *sequenze* di simboli che vanno a sostituire lunghi set di caratteri di comune utilizzo, eccone alcuni:

Sequenza	Descrizione
\A	Inizio della stringa (simile a ^)
\b	Valida la stringa vuota che delimita una parola
\d	Cifre da 0 a 9
\D	Tutto eccetto le cifre
\s	Spaziature
\S	Tutto eccetto le spaziature
\w	Tutti i caratteri alfanumerici e _
\W	Tutto eccetto i caratteri alfanumerici e l'underscore _
\Z	Fine della stringa (simile a $)

⊛⊛⊛ **ESERCIZIO 5.1**: Proviamo a variare l'esercizio precedente adesso quando matcherà sarai tu a *scrivere dei pattern* che verifichi le stringhe nella `pass_list` ed escluda quelle nella `fail_list`, dove troverai il pattern sarà come l'esercizio precedente (di nuovo non preoccuparti se vedi 'ERRORE' scritto sotto le celle prima ancora di iniziare, se metti i giusti pattern / esempi come richiesto le scritte 'ERRORE' dovrebbero scomparire)

[51]:
```
#Scegli il PATTERN giusto !
pass_n_fail(r"",
            ["3 ramarri", "2 carri"],  # queste devono matchare
            ["tre ramarri", "due carri", "3 ", "2 "]  # queste non devono matchare
           )
```

```
ERRORE: 'tre ramarri' matcha il pattern '' ma non dovrebbe farlo!
ERRORE: 'due carri' matcha il pattern '' ma non dovrebbe farlo!
ERRORE: '3 ' matcha il pattern '' ma non dovrebbe farlo!
ERRORE: '2 ' matcha il pattern '' ma non dovrebbe farlo!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[52]:
```
# SOLUZIONE

pass_n_fail(r"^\d\s\w+",
            ["3 ramarri", "2 carri"],
            ["tre ramarri", "due carri", "3 ", "2 "]
           )
```

[52]:

[53]:
```
#Scegli le STRINGHE
pass_n_fail(r"^(w{3}\.)?\w\w+\.\w\w+",
            ["",""],
            ["",""]
           )
```

```
ERRORE: '' non matcha il pattern '^(w{3}\.)?\w\w+\.\w\w+' ma dovrebbe farlo!
ERRORE: '' non matcha il pattern '^(w{3}\.)?\w\w+\.\w\w+' ma dovrebbe farlo!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[54]:
```
# SOLUZIONE

pass_n_fail(r"^(w{3}\.)?\w\w+\.\w\w+",
            ["www.google.it","youtube.com"],
            ["ciaociao"," www.google.com"]
           )
```

[54]:

[55]:
```
#Scegli il PATTERN
pass_n_fail(r"",
            ["21.12.2017", "11/01/2018", "16-12-89"],
            ["1621211003", "11/20/2010", "12-12-123"]
           )
```

```
ERRORE: '1621211003' matcha il pattern '' ma non dovrebbe farlo!
ERRORE: '11/20/2010' matcha il pattern '' ma non dovrebbe farlo!
ERRORE: '12-12-123' matcha il pattern '' ma non dovrebbe farlo!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[56]:
```
# SOLUZIONE

pass_n_fail(r"\d{2}[-/.][01]\d[-/.]\d\d(\d\d)?$",
            ["21.12.2017", "11/01/2018", "16-12-89"],
            ["1621211003", "11/20/2010", "12-12-123"]
           )
```

[56]:

5.3.8 6. Le funzioni della libreria `re`

Fino ad ora abbiamo usato una sola funzione della libreria `re` di Python, ossia `re.search()` ma sono presenti anche altre funzionalità, la più simile è il metodo `re.match()`:

[57]:
```
print(re.search('c', 'abcde'))
```

```
<_sre.SRE_Match object; span=(2, 3), match='c'>
```

[58]:
```
print(re.match('c', 'abcde'))
```

```
None
```

[59]:
```
print(re.match('a', 'abcde'))
```

```
<_sre.SRE_Match object; span=(0, 1), match='a'>
```

⊛⊛ **DOMANDA 6.1**: Riesci a capire la differenza? `help(re.match)` e `help(re.search)` possono tornarti utili.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Match cerca dal primo carattere.

5.3.9 7. Sostituzioni con `re.sub`

Le espressioni regolari possono anche essere utilizzate per sostituire del testo, un po' come il metodo `replace()` sulle stringhe. Quando chiamiamo il metodo `re.sub()`, per ricevere in output la stringa elaborata, dobbiamo passare come argomenti:

1. il pattern della regular expression
2. il testo da sostituire
3. la stringa su cui effettuare la ricerca

Tornando all'esempio delle fermate dell'autobus, adesso vogliamo sostituire tutto quel confusionario "Sp" in "strade provinciali", vediamo il codice:

[60]:
```
help(re.sub)
```

```
Help on function sub in module re:

sub(pattern, repl, string, count=0, flags=0)
    Return the string obtained by replacing the leftmost
    non-overlapping occurrences of the pattern in string by the
    replacement repl.  repl can be either a string or a callable;
    if a string, backslash escapes in it are processed.  If it is
    a callable, it's passed the match object and must return
    a replacement string to be used.
```

[61]:
```
import csv
import re
with open('stops.txt', newline='') as f:
    reader = csv.reader(f, delimiter=',')

    for row in reader:
        nuova = re.sub(r'Sp.(\d+)', r'Strada Provinciale \1',  row[2])
        if nuova != row[2]:
            print(row[2], 'DIVENTA', nuova)
```

```
Sp 85 Bivio Sopramonte DIVENTA Strada Provinciale 85 Bivio Sopramonte
Sp 76 Carpenedi DIVENTA Strada Provinciale 76 Carpenedi
Sp 76 Carpenedi DIVENTA Strada Provinciale 76 Carpenedi
Sp 131 "Maso Pradiscola" DIVENTA Strada Provinciale 131 "Maso Pradiscola"
Sp 131 "Maso Pradiscola" DIVENTA Strada Provinciale 131 "Maso Pradiscola"
Sp 131 "Res. Silvana" DIVENTA Strada Provinciale 131 "Res. Silvana"
Sp 131 "Res. Silvana" DIVENTA Strada Provinciale 131 "Res. Silvana"
Sp 131 Al Maso Specchio DIVENTA Strada Provinciale 131 Al Maso Specchio
Sp 131 Al Maso Specchio DIVENTA Strada Provinciale 131 Al Maso Specchio
Sp 76 "Piac" DIVENTA Strada Provinciale 76 "Piac"
Sp 76 "Maregioli" DIVENTA Strada Provinciale 76 "Maregioli"
Sp 76 "Maregioli" DIVENTA Strada Provinciale 76 "Maregioli"
Sp 76 "Via Rossa" DIVENTA Strada Provinciale 76 "Via Rossa"
```

(continues on next page)

(continued from previous page)

```
Sp 76 "Via Rossa" DIVENTA Strada Provinciale 76 "Via Rossa"
Sp 76 Dos di Lamar DIVENTA Strada Provinciale 76 Dos di Lamar
Sp 76 Dos di Lamar DIVENTA Strada Provinciale 76 Dos di Lamar
Sp 131 "Paganin" DIVENTA Strada Provinciale 131 "Paganin"
Sp 131 "Paganin" DIVENTA Strada Provinciale 131 "Paganin"
Sp 85 "Soraval" DIVENTA Strada Provinciale 85 "Soraval"
Sp 85 "Soraval" DIVENTA Strada Provinciale 85 "Soraval"
Sp 90 "Maso Prudenza" DIVENTA Strada Provinciale 90 "Maso Prudenza"
Sp 90 "Maso Prudenza" DIVENTA Strada Provinciale 90 "Maso Prudenza"
Sp.2 Loc. Beccache' DIVENTA Strada Provinciale 2 Loc. Beccache'
Sp.2 Loc. Beccache' DIVENTA Strada Provinciale 2 Loc. Beccache'
Sp.2 Fr. Campolongo DIVENTA Strada Provinciale 2 Fr. Campolongo
Sp.2 Noriglio DIVENTA Strada Provinciale 2 Noriglio
Sp.2 Noriglio DIVENTA Strada Provinciale 2 Noriglio
Sp.2 Fr. Campolongo DIVENTA Strada Provinciale 2 Fr. Campolongo
Brancolino Sp.90 DIVENTA Brancolino Strada Provinciale 90
Brancolino Sp.90 DIVENTA Brancolino Strada Provinciale 90
Sp.20 S.Sisinio DIVENTA Strada Provinciale 20 S.Sisinio
Sp.20 Zisi DIVENTA Strada Provinciale 20 Zisi
Sp.20 Zisi DIVENTA Strada Provinciale 20 Zisi
Sp.20 S.Sisinio DIVENTA Strada Provinciale 20 S.Sisinio
Sp.20 Maso Tiaf DIVENTA Strada Provinciale 20 Maso Tiaf
Sp.20 Bivio Per Bordala DIVENTA Strada Provinciale 20 Bivio Per Bordala
Sp.20 Bivio Per Bordala DIVENTA Strada Provinciale 20 Bivio Per Bordala
Sp.20 Maso Tiaf DIVENTA Strada Provinciale 20 Maso Tiaf
Sp 76 Bivio S.Lazzaro DIVENTA Strada Provinciale 76 Bivio S.Lazzaro
Sp.2 Bivio Per Cisterna DIVENTA Strada Provinciale 2 Bivio Per Cisterna
Sp.20 Bivio Noarna DIVENTA Strada Provinciale 20 Bivio Noarna
Sp.20 Bivio Noarna DIVENTA Strada Provinciale 20 Bivio Noarna
Sp.23 "Nero Cubo" DIVENTA Strada Provinciale 23 "Nero Cubo"
Sp.89 Maso Brentegam DIVENTA Strada Provinciale 89 Maso Brentegam
```

Bene, ma cosa succede? Conosciamo già tutto fino alla linea 3, poi cerchiamo il pattern `Sp.(\d+)` ; nota che ho messo delle parentesi dove sembrerebbe non servano ma fai attenzione al prossimo parametro (cioè la stringa sostitutiva): ti accorgerai di un carattere speciale di una sequenza particolare di cui non abbiamo parlato, ovvero `\1` . Questo tipo di sequenza è detta **gruppo di backreference** e viene definito nel pattern da cercare utilizzando le parentesi tonde appunto: tutto quello tra `(` and `)` diventa un gruppo e per richiamarlo è necessario soltando aggiungere un *backslash* seguito dal numero del gruppo, contando da sinistra a destra.

Nel nostro caso esiste un solo gruppo quindi non possiamo sbagliare: la ricerca prima esamina il testo per trovare `Sp` seguito da un carattere e poi (da un gruppo definito come) una o più cifre. La stringa di sostituzione riporterà prima la scritta `Strada Provinciale` poi uno spazio ed infine la cifra che ha verificato il gruppo definito nel pattern di ricerca.

⊛⊛⊛ **ESERCIZIO 7.1**: Prova a ricopiare il codice precedente qua sotto, e compi la stessa operazione ma questa volta con le *Strade Statali*. Riesci a scrivere pattern diversi di ricerca e sostituzione per farlo?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[62]:
```
# scrivi qui il metodo 1

with open('stops.txt', encoding='utf-8', newline='') as f:
    reader = csv.reader(f, delimiter=',')
    for row in reader:
        nuova = re.sub(r'Ss.(\d+)', r'Strada Statale \1', row[2])
```

(continues on next page)

(continued from previous page)

```
        if nuova != row[2]:  # NOTA: questo if non sarebbe strettamente necessario,
                             #       l'abbiamo messo per non stampare tutto il dataset
            row[2] = nuova
            print(row)
```

```
['64', '22075x', 'Gardolo Svincolo Strada Statale 12', '', '46.106227', '11.109072',
↪'10110', '1']
['72', '22230z', 'Strada Statale 12 "Zona Ind.Le Ftm"', '', '46.119309', '11.106123',
↪'10110', '2']
['94', '26040z', 'Strada Statale 12 "Casteller"', '', '46.025284', '11.132751', '10110
↪', '2']
['95', '26040x', 'Strada Statale 12 "Casteller"', '', '46.025707', '11.132791', '10110
↪', '2']
['99', '26045z', 'Strada Statale 12 "Le Caverne"', '', '46.030862', '11.132677',
↪'10110', '2']
['100', '26045x', 'Strada Statale 12 "Le Caverne"', '', '46.030338', '11.132911',
↪'10110', '1']
['158', '28055z', 'Strada Statale 45 "Montevideo"', '', '46.080499', '11.098333',
↪'10110', '2']
['214', '22230x', 'Strada Statale 12 "Zona Ind.Le Ftm"', '', '46.120242', '11.105822',
↪ '10110', '2']
['278', '21680z', 'Strada Statale 47 Muralta', '', '46.077059', '11.131773', '10110',
↪'2']
['286', '22710x', 'Strada Statale 12 "Pioppeto"', '', '46.108369', '11.108528', '10110
↪', '1']
['287', '22715x', 'Strada Statale 12 Bivio Meano', '', '46.116062', '11.106197',
↪'10110', '2']
['288', '22720x', 'Strada Statale 12 Canova', '', '46.098985', '11.111739', '10110',
↪'1']
['289', '22705x', 'Strada Statale 12 "Paludi"', 'via Bolzano incrocio via Noce', '46.
↪102643', '11.110490', '10110', '2']
['290', '22720z', 'Strada Statale 12 Canova', '', '46.100264', '11.111578', '10110',
↪'1']
['340', '25815z', 'Strada Statale 349 Pozzata', '', '46.039534', '11.144070', '10110',
↪ '2']
['1350', '31221z', 'Lizzanella Strada Statale 12', '', '45.875522', '11.031822',
↪'10101', '2']
['1351', '31221x', 'Lizzanella Strada Statale 12', '', '45.875686', '11.032030',
↪'10101', '2']
['1422', '32221z', 'Strada Statale 12 "Rovercenter"', '', '45.864490', '11.029338',
↪'10101', '2']
['2289', '32251x', 'Strada Statale 240 "Millennium Sud"', '', '45.868678', '11.014119
↪', '10101', '2']
['2291', '32251z', 'Strada Statale 240 "Millennium Sud"', '', '45.868802', '11.014170
↪', '10101', '2']
['2293', '32241z', 'Strada Statale 240 "Millennium Nord"', '', '45.870320', '11.016768
↪', '10101', '2']
['2294', '32241x', 'Strada Statale 240 "Millennium Nord"', '', '45.870588', '11.017358
↪', '10101', '2']
['2337', '39041z', 'Volano Strada Statale 12', '', '45.917871', '11.064024', '19011',
↪'2']
['2338', '39041x', 'Volano Strada Statale 12', '', '45.917904', '11.063866', '19011',
↪'2']
['2418', '22710z', 'Strada Statale 12 "Pioppeto"', '', '46.108623', '11.108717',
↪'10110', '1']
['2419', '22730z', 'Strada Statale 12 S.Anna', '', '46.112409', '11.107438', '10110',
↪'2']
```

(continues on next page)

(continued from previous page)

```
['2438', '22725z', 'Strada Statale 12 Talvera', '', '46.103288', '11.110545', '10110',
↪ '1']
['2500', '21685z', 'Strada Statale 45 "Scala"', '', '46.077367', '11.099792', '10110',
↪ '2']
['2514', '34101x', "Strada Statale 240 Ponte Sull'adige", '', '45.863191', '10.997289
↪', '14051', '2']
['2515', '34101z', "Strada Statale 240 Ponte Sull'adige", '', '45.863235', '10.997355
↪', '14051', '2']
['2699', '32221x', 'Strada Statale 12 "Rovercenter"', '', '45.864206', '11.029050',
↪'10101', '2']
['2701', '25815x', 'Strada Statale 349 Pozzata', '', '46.041779', '11.142383', '10110
↪', '2']
['2780', '21687z', 'Strada Statale 45 S.Giorgio', '', '46.078668', '11.099302', '10110
↪', '2']
['2806', '32236x', 'Strada Statale 240 "Alla Staffa"', '', '45.872760', '11.024323',
↪'10101', '2']
['2838', '39061z', 'Strada Statale 12 Fornaci', '', '45.919443', '11.085327', '19011',
↪ '2']
['2856', '36511z', 'Strada Statale 12 Castel Pietra', '', '45.921721', '11.091309',
↪'16501', '2']
['2857', '36511x', 'Strada Statale 12 Castel Pietra', '', '45.922938', '11.092225',
↪'16501', '2']
['2858', '36521x', 'Strada Statale 12 Campagnole', '', '45.928854', '11.093011',
↪'16501', '2']
['2859', '36521z', 'Strada Statale 12 Campagnole', '', '45.928851', '11.093153',
↪'16501', '2']
['2870', '32236z', 'Strada Statale 240 "Alla Staffa"', '', '45.872820', '11.024167',
↪'10101', '2']
['2877', '32256x', 'Strada Statale 240 "Meb"', '', '45.867858', '11.011826', '10101',
↪'2']
['2906', '36541z', 'Calliano Strada Statale 12', '', '45.934315', '11.096017', '16501
↪', '2']
['2907', '36541x', 'Calliano Strada Statale 12', '', '45.934334', '11.095870', '16501
↪', '2']
```

[62]:
```
# scrivi qui il metodo 1
```

```
['64', '22075x', 'Gardolo Svincolo Strada Statale 12', '', '46.106227', '11.109072',
↪'10110', '1']
['72', '22230z', 'Strada Statale 12 "Zona Ind.Le Ftm"', '', '46.119309', '11.106123',
↪'10110', '2']
['94', '26040z', 'Strada Statale 12 "Casteller"', '', '46.025284', '11.132751', '10110
↪', '2']
['95', '26040x', 'Strada Statale 12 "Casteller"', '', '46.025707', '11.132791', '10110
↪', '2']
['99', '26045z', 'Strada Statale 12 "Le Caverne"', '', '46.030862', '11.132677',
↪'10110', '2']
['100', '26045x', 'Strada Statale 12 "Le Caverne"', '', '46.030338', '11.132911',
↪'10110', '1']
['158', '28055z', 'Strada Statale 45 "Montevideo"', '', '46.080499', '11.098333',
↪'10110', '2']
['214', '22230x', 'Strada Statale 12 "Zona Ind.Le Ftm"', '', '46.120242', '11.105822',
↪ '10110', '2']
```

(continues on next page)

(continued from previous page)

```
['278', '21680z', 'Strada Statale 47 Muralta', '', '46.077059', '11.131773', '10110',
↪'2']
['286', '22710x', 'Strada Statale 12 "Pioppeto"', '', '46.108369', '11.108528', '10110
↪', '1']
['287', '22715x', 'Strada Statale 12 Bivio Meano', '', '46.116062', '11.106197',
↪'10110', '2']
['288', '22720x', 'Strada Statale 12 Canova', '', '46.098985', '11.111739', '10110',
↪'1']
['289', '22705x', 'Strada Statale 12 "Paludi"', 'via Bolzano incrocio via Noce', '46.
↪102643', '11.110490', '10110', '2']
['290', '22720z', 'Strada Statale 12 Canova', '', '46.100264', '11.111578', '10110',
↪'1']
['340', '25815z', 'Strada Statale 349 Pozzata', '', '46.039534', '11.144070', '10110',
↪ '2']
['1350', '31221z', 'Lizzanella Strada Statale 12', '', '45.875522', '11.031822',
↪'10101', '2']
['1351', '31221x', 'Lizzanella Strada Statale 12', '', '45.875686', '11.032030',
↪'10101', '2']
['1422', '32221z', 'Strada Statale 12 "Rovercenter"', '', '45.864490', '11.029338',
↪'10101', '2']
['2289', '32251x', 'Strada Statale 240 "Millennium Sud"', '', '45.868678', '11.014119
↪', '10101', '2']
['2291', '32251z', 'Strada Statale 240 "Millennium Sud"', '', '45.868802', '11.014170
↪', '10101', '2']
['2293', '32241z', 'Strada Statale 240 "Millennium Nord"', '', '45.870320', '11.016768
↪', '10101', '2']
['2294', '32241x', 'Strada Statale 240 "Millennium Nord"', '', '45.870588', '11.017358
↪', '10101', '2']
['2337', '39041z', 'Volano Strada Statale 12', '', '45.917871', '11.064024', '19011',
↪'2']
['2338', '39041x', 'Volano Strada Statale 12', '', '45.917904', '11.063866', '19011',
↪'2']
['2418', '22710z', 'Strada Statale 12 "Pioppeto"', '', '46.108623', '11.108717',
↪'10110', '1']
['2419', '22730z', 'Strada Statale 12 S.Anna', '', '46.112409', '11.107438', '10110',
↪'2']
['2438', '22725z', 'Strada Statale 12 Talvera', '', '46.103288', '11.110545', '10110',
↪ '1']
['2500', '21685z', 'Strada Statale 45 "Scala"', '', '46.077367', '11.099792', '10110',
↪ '2']
['2514', '34101x', "Strada Statale 240 Ponte Sull'adige", '', '45.863191', '10.997289
↪', '14051', '2']
['2515', '34101z', "Strada Statale 240 Ponte Sull'adige", '', '45.863235', '10.997355
↪', '14051', '2']
['2699', '32221x', 'Strada Statale 12 "Rovercenter"', '', '45.864206', '11.029050',
↪'10101', '2']
['2701', '25815x', 'Strada Statale 349 Pozzata', '', '46.041779', '11.142383', '10110
↪', '2']
['2780', '21687z', 'Strada Statale 45 S.Giorgio', '', '46.078668', '11.099302', '10110
↪', '2']
['2806', '32236x', 'Strada Statale 240 "Alla Staffa"', '', '45.872760', '11.024323',
↪'10101', '2']
['2838', '39061z', 'Strada Statale 12 Fornaci', '', '45.919443', '11.085327', '19011',
↪ '2']
['2856', '36511z', 'Strada Statale 12 Castel Pietra', '', '45.921721', '11.091309',
↪'16501', '2']
['2857', '36511x', 'Strada Statale 12 Castel Pietra', '', '45.922938', '11.092225',
↪'16501', '2']
```

(continues on next page)

(continued from previous page)

```
['2858', '36521x', 'Strada Statale 12 Campagnole', '', '45.928854', '11.093011',
↪'16501', '2']
['2859', '36521z', 'Strada Statale 12 Campagnole', '', '45.928851', '11.093153',
↪'16501', '2']
['2870', '32236z', 'Strada Statale 240 "Alla Staffa"', '', '45.872820', '11.024167',
↪'10101', '2']
['2877', '32256x', 'Strada Statale 240 "Meb"', '', '45.867858', '11.011826', '10101',
↪'2']
['2906', '36541z', 'Calliano Strada Statale 12', '', '45.934315', '11.096017', '16501
↪', '2']
['2907', '36541x', 'Calliano Strada Statale 12', '', '45.934334', '11.095870', '16501
↪', '2']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[63]:
```
# scrivi qui il metodo 2

with open('stops.txt', encoding='utf-8', newline='') as f:
    reader = csv.reader(f, delimiter=',')
    for row in reader:
        nuova = re.sub(r'Ss\b', r'Strada Statale', row[2])
        if nuova != row[2]:  # NOTA: questo if non sarebbe strettamente necessario,
                             #       l'abbiamo messo per non stampare tutto il dataset
            row[2] = nuova
            print(row)
```

```
['64', '22075x', 'Gardolo Svincolo Strada Statale.12', '', '46.106227', '11.109072',
↪'10110', '1']
['72', '22230z', 'Strada Statale 12 "Zona Ind.Le Ftm"', '', '46.119309', '11.106123',
↪'10110', '2']
['94', '26040z', 'Strada Statale 12 "Casteller"', '', '46.025284', '11.132751', '10110
↪', '2']
['95', '26040x', 'Strada Statale 12 "Casteller"', '', '46.025707', '11.132791', '10110
↪', '2']
['99', '26045z', 'Strada Statale 12 "Le Caverne"', '', '46.030862', '11.132677',
↪'10110', '2']
['100', '26045x', 'Strada Statale 12 "Le Caverne"', '', '46.030338', '11.132911',
↪'10110', '1']
['158', '28055z', 'Strada Statale 45 "Montevideo"', '', '46.080499', '11.098333',
↪'10110', '2']
['214', '22230x', 'Strada Statale 12 "Zona Ind.Le Ftm"', '', '46.120242', '11.105822',
↪ '10110', '2']
['278', '21680z', 'Strada Statale 47 Muralta', '', '46.077059', '11.131773', '10110',
↪'2']
['286', '22710x', 'Strada Statale 12 "Pioppeto"', '', '46.108369', '11.108528', '10110
↪', '1']
['287', '22715x', 'Strada Statale 12 Bivio Meano', '', '46.116062', '11.106197',
↪'10110', '2']
['288', '22720x', 'Strada Statale 12 Canova', '', '46.098985', '11.111739', '10110',
↪'1']
['289', '22705x', 'Strada Statale 12 "Paludi"', 'via Bolzano incrocio via Noce', '46.
↪102643', '11.110490', '10110', '2']
['290', '22720z', 'Strada Statale 12 Canova', '', '46.100264', '11.111578', '10110',
↪'1']
['340', '25815z', 'Strada Statale 349 Pozzata', '', '46.039534', '11.144070', '10110',
↪ '2']
```

(continues on next page)

(continued from previous page)

```
['1350', '31221z', 'Lizzanella Strada Statale.12', '', '45.875522', '11.031822',
↪'10101', '2']
['1351', '31221x', 'Lizzanella Strada Statale.12', '', '45.875686', '11.032030',
↪'10101', '2']
['1422', '32221z', 'Strada Statale.12 "Rovercenter"', '', '45.864490', '11.029338',
↪'10101', '2']
['2289', '32251x', 'Strada Statale.240 "Millennium Sud"', '', '45.868678', '11.014119
↪', '10101', '2']
['2291', '32251z', 'Strada Statale.240 "Millennium Sud"', '', '45.868802', '11.014170
↪', '10101', '2']
['2293', '32241z', 'Strada Statale.240 "Millennium Nord"', '', '45.870320', '11.016768
↪', '10101', '2']
['2294', '32241x', 'Strada Statale.240 "Millennium Nord"', '', '45.870588', '11.017358
↪', '10101', '2']
['2337', '39041z', 'Volano Strada Statale.12', '', '45.917871', '11.064024', '19011',
↪'2']
['2338', '39041x', 'Volano Strada Statale.12', '', '45.917904', '11.063866', '19011',
↪'2']
['2418', '22710z', 'Strada Statale 12 "Pioppeto"', '', '46.108623', '11.108717',
↪'10110', '1']
['2419', '22730z', 'Strada Statale 12 S.Anna', '', '46.112409', '11.107438', '10110',
↪'2']
['2438', '22725z', 'Strada Statale 12 Talvera', '', '46.103288', '11.110545', '10110',
↪ '1']
['2500', '21685z', 'Strada Statale 45 "Scala"', '', '46.077367', '11.099792', '10110',
↪ '2']
['2514', '34101x', "Strada Statale.240 Ponte Sull'adige", '', '45.863191', '10.997289
↪', '14051', '2']
['2515', '34101z', "Strada Statale.240 Ponte Sull'adige", '', '45.863235', '10.997355
↪', '14051', '2']
['2699', '32221x', 'Strada Statale.12 "Rovercenter"', '', '45.864206', '11.029050',
↪'10101', '2']
['2701', '25815x', 'Strada Statale 349 Pozzata', '', '46.041779', '11.142383', '10110
↪', '2']
['2780', '21687z', 'Strada Statale 45 S.Giorgio', '', '46.078668', '11.099302', '10110
↪', '2']
['2806', '32236x', 'Strada Statale.240 "Alla Staffa"', '', '45.872760', '11.024323',
↪'10101', '2']
['2838', '39061z', 'Strada Statale.12 Fornaci', '', '45.919443', '11.085327', '19011',
↪ '2']
['2856', '36511z', 'Strada Statale.12 Castel Pietra', '', '45.921721', '11.091309',
↪'16501', '2']
['2857', '36511x', 'Strada Statale.12 Castel Pietra', '', '45.922938', '11.092225',
↪'16501', '2']
['2858', '36521x', 'Strada Statale.12 Campagnole', '', '45.928854', '11.093011',
↪'16501', '2']
['2859', '36521z', 'Strada Statale.12 Campagnole', '', '45.928851', '11.093153',
↪'16501', '2']
['2870', '32236z', 'Strada Statale.240 "Alla Staffa"', '', '45.872820', '11.024167',
↪'10101', '2']
['2877', '32256x', 'Strada Statale.240 "Meb"', '', '45.867858', '11.011826', '10101',
↪'2']
['2906', '36541z', 'Calliano Strada Statale.12', '', '45.934315', '11.096017', '16501
↪', '2']
['2907', '36541x', 'Calliano Strada Statale.12', '', '45.934334', '11.095870', '16501
↪', '2']
```

[63]:
```
# scrivi qui il metodo 2
```

```
['64', '22075x', 'Gardolo Svincolo Strada Statale.12', '', '46.106227', '11.109072',
↪'10110', '1']
['72', '22230z', 'Strada Statale 12 "Zona Ind.Le Ftm"', '', '46.119309', '11.106123',
↪'10110', '2']
['94', '26040z', 'Strada Statale 12 "Casteller"', '', '46.025284', '11.132751', '10110
↪', '2']
['95', '26040x', 'Strada Statale 12 "Casteller"', '', '46.025707', '11.132791', '10110
↪', '2']
['99', '26045z', 'Strada Statale 12 "Le Caverne"', '', '46.030862', '11.132677',
↪'10110', '2']
['100', '26045x', 'Strada Statale 12 "Le Caverne"', '', '46.030338', '11.132911',
↪'10110', '1']
['158', '28055z', 'Strada Statale 45 "Montevideo"', '', '46.080499', '11.098333',
↪'10110', '2']
['214', '22230x', 'Strada Statale 12 "Zona Ind.Le Ftm"', '', '46.120242', '11.105822',
↪ '10110', '2']
['278', '21680z', 'Strada Statale 47 Muralta', '', '46.077059', '11.131773', '10110',
↪'2']
['286', '22710x', 'Strada Statale 12 "Pioppeto"', '', '46.108369', '11.108528', '10110
↪', '1']
['287', '22715x', 'Strada Statale 12 Bivio Meano', '', '46.116062', '11.106197',
↪'10110', '2']
['288', '22720x', 'Strada Statale 12 Canova', '', '46.098985', '11.111739', '10110',
↪'1']
['289', '22705x', 'Strada Statale 12 "Paludi"', 'via Bolzano incrocio via Noce', '46.
↪102643', '11.110490', '10110', '2']
['290', '22720z', 'Strada Statale 12 Canova', '', '46.100264', '11.111578', '10110',
↪'1']
['340', '25815z', 'Strada Statale 349 Pozzata', '', '46.039534', '11.144070', '10110',
↪ '2']
['1350', '31221z', 'Lizzanella Strada Statale.12', '', '45.875522', '11.031822',
↪'10101', '2']
['1351', '31221x', 'Lizzanella Strada Statale.12', '', '45.875686', '11.032030',
↪'10101', '2']
['1422', '32221z', 'Strada Statale.12 "Rovercenter"', '', '45.864490', '11.029338',
↪'10101', '2']
['2289', '32251x', 'Strada Statale.240 "Millennium Sud"', '', '45.868678', '11.014119
↪', '10101', '2']
['2291', '32251z', 'Strada Statale.240 "Millennium Sud"', '', '45.868802', '11.014170
↪', '10101', '2']
['2293', '32241z', 'Strada Statale.240 "Millennium Nord"', '', '45.870320', '11.016768
↪', '10101', '2']
['2294', '32241x', 'Strada Statale.240 "Millennium Nord"', '', '45.870588', '11.017358
↪', '10101', '2']
['2337', '39041z', 'Volano Strada Statale.12', '', '45.917871', '11.064024', '19011',
↪'2']
['2338', '39041x', 'Volano Strada Statale.12', '', '45.917904', '11.063866', '19011',
↪'2']
['2418', '22710z', 'Strada Statale 12 "Pioppeto"', '', '46.108623', '11.108717',
↪'10110', '1']
['2419', '22730z', 'Strada Statale 12 S.Anna', '', '46.112409', '11.107438', '10110',
↪'2']
```

(continues on next page)

(continued from previous page)

```
['2438', '22725z', 'Strada Statale 12 Talvera', '', '46.103288', '11.110545', '10110',
↪ '1']
['2500', '21685z', 'Strada Statale 45 "Scala"', '', '46.077367', '11.099792', '10110',
↪ '2']
['2514', '34101x', "Strada Statale.240 Ponte Sull'adige", '', '45.863191', '10.997289
↪', '14051', '2']
['2515', '34101z', "Strada Statale.240 Ponte Sull'adige", '', '45.863235', '10.997355
↪', '14051', '2']
['2699', '32221x', 'Strada Statale.12 "Rovercenter"', '', '45.864206', '11.029050',
↪'10101', '2']
['2701', '25815x', 'Strada Statale 349 Pozzata', '', '46.041779', '11.142383', '10110
↪', '2']
['2780', '21687z', 'Strada Statale 45 S.Giorgio', '', '46.078668', '11.099302', '10110
↪', '2']
['2806', '32236x', 'Strada Statale.240 "Alla Staffa"', '', '45.872760', '11.024323',
↪'10101', '2']
['2838', '39061z', 'Strada Statale.12 Fornaci', '', '45.919443', '11.085327', '19011',
↪ '2']
['2856', '36511z', 'Strada Statale.12 Castel Pietra', '', '45.921721', '11.091309',
↪'16501', '2']
['2857', '36511x', 'Strada Statale.12 Castel Pietra', '', '45.922938', '11.092225',
↪'16501', '2']
['2858', '36521x', 'Strada Statale.12 Campagnole', '', '45.928854', '11.093011',
↪'16501', '2']
['2859', '36521z', 'Strada Statale.12 Campagnole', '', '45.928851', '11.093153',
↪'16501', '2']
['2870', '32236z', 'Strada Statale.240 "Alla Staffa"', '', '45.872820', '11.024167',
↪'10101', '2']
['2877', '32256x', 'Strada Statale.240 "Meb"', '', '45.867858', '11.011826', '10101',
↪'2']
['2906', '36541z', 'Calliano Strada Statale.12', '', '45.934315', '11.096017', '16501
↪', '2']
['2907', '36541x', 'Calliano Strada Statale.12', '', '45.934334', '11.095870', '16501
↪', '2']
```

NOTA In realtà l'utilizzo delle backreferences non è limitato alla funzione di sostituzione `re.sub` ma può essere usato anche nei pattern: il pattern `^(a+)=\1$` per esempio significa che la stringa deve iniziare `^` con una o più `a` (`a+`), e deve essere seguito da un uguale (`=`) e dalla stessa stringa che ha verificato il *gruppo 1* (`\1`) che nel nostro caso equivale a quanto stato verificato da `a+` precedentemente.

⊛⊛⊛ **ESERCIZIO 7.2** Prova a inserire delle stringhe che verificano i pattern inseriti:

[64]:
```
pass_n_fail(r'^(a+)=\1$',
            ['', '',''], # Inserisci qui almeno 3 elementi
            [])
```

```
ERRORE: '' non matcha il pattern '^(a+)=\1$' ma dovrebbe farlo!
ERRORE: '' non matcha il pattern '^(a+)=\1$' ma dovrebbe farlo!
ERRORE: '' non matcha il pattern '^(a+)=\1$' ma dovrebbe farlo!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[65]:
```
# SOLUZIONE

pass_n_fail(r'^(a+)=\1$',
```

(continues on next page)

(continued from previous page)

```
            ['a=a', 'aaa=aaa','aaaaaa=aaaaaa'], #TODO Inserisci almeno 3 elementi
            [])
```

[65]:

```

```

5.3.10 8. Dividere le stringhe

Uno degli altri task ai quali le espressioni regolari possono essere utile è quello di spezzare le stringhe, magari per isolare parti specifiche. Anche in questo caso esiste un'analogia con il metodo `split()` per le stringhe visto che entrambe le istruzioni compiono una funzione simile ma `re.split()` ha due parametri obbligatori, il primo è il `pattern` dell'espressione regolare che quando verificata spezza il testo e il secondo è la stringa da spezzare.

Nell'esempio qui sotto separare tutte le frasi in un testo, prendiamo il file, lo apriamo e lo leggiamo: `f.readlines()` legge tutte le linee e restituisce una lista di linee, il metodo `"\n".join()` usa il carattere di nuova riga `\n` per unire gli elementi della lista (in questo caso le varie linee, formando il testo).

A questo punto chiamiamo il metodo `re.split()` usando come pattern di separazione punti, punto e virgola, due punti, etc… presenti almeno una volta e seguiti da almeno uno spazio seguito da opzionalmente da una serie di acapo; abbiamo anche aggiunto il flag `re.MULTILINE` per permettere le regex su righe multiple.

[66]:

```
import re

with open('psposi1.txt', encoding='utf-8') as f:
    testo = "\n".join(f.readlines())

frasi = re.split(r'[.;:?!-]+\s+\n*', testo, flags=re.MULTILINE)

frasi[:10]  # con la slice [:10] mostriamo solo le prime 10 frasi
```

[66]:

```
['Quel ramo del lago di Como, che volge a mezzogiorno, tra due catene [1] non␣
↪interrotte di monti, tutto a seni e a golfi, a seconda dello sporgere e del␣
↪rientrare di quelli, vien, quasi a un tratto, a ristringersi, e a prender corso e␣
↪figura di fiume, tra un promontorio a destra, e un'ampia costiera dall'altra parte',
 'e il ponte [2], che ivi congiunge le due rive, par che renda ancor più sensibile␣
↪all'occhio questa trasformazione, e segni il punto in cui il lago cessa, e l'Adda␣
↪rincomincia, per ripigliar poi nome di lago dove le rive, allontanandosi di nuovo,␣
↪lascian l'acqua distendersi e rallentarsi in nuovi golfi e in nuovi seni',
 'La costiera, formata dal deposito di tre grossi torrenti [3], scende appoggiata a␣
↪due monti contigui, l'uno detto di san Martino, l'altro, con voce lombarda, il␣
↪Resegone, dai molti suoi cocuzzoli in fila, che in vero lo fanno somigliare a una␣
↪sega',
 'talché non è chi, al primo vederlo, purché sia di fronte, come per esempio di su le␣
↪mura di Milano che guardano a settentrione, non lo discerna tosto, a un tal␣
↪contrassegno, in quella lunga e vasta giogaia, dagli altri monti di nome più oscuro␣
↪e di forma più comune',
 'Per un buon pezzo, la costa sale con un pendìo lento e continuo',
 'poi si rompe in poggi e in valloncelli, in erte e in ispianate, secondo l'ossatura␣
↪de' due monti, e il lavoro dell'acque',
 'Il lembo estremo, tagliato dalle foci de' torrenti, è quasi tutto ghiaia e␣
↪ciottoloni',
 'il resto, campi e vigne, sparse di terre, di ville, di casali',
 'in qualche parte boschi, che si prolungano su per la montagna',
 'Lecco, la principale di quelle terre, e che dà nome al territorio, giace poco␣
↪discosto dal ponte, alla riva del lago, anzi viene in parte a trovarsi nel lago␣
↪stesso, quando questo ingrossa']
```

(continues on next page)

(continued from previous page)

5.3.11 9. Cercare pattern nel testo

A volte vogliamo trovare delle informazioni all'interno del testo e sappiamo che queste appaiono in un certo pattern, possiamo scrivere una espressione regolare e utilizzare il metodo `re.findall()` per estrarle. Questo comando è molto simile al comando `re.find()` ma al contrario di questo non si ferma al primo match ma prosegue estraendo una lista delle stringhe che hanno verificato la regex oppure, se sono presenti delle backreferences, con *una lista di tuple avente i valori di tutti i gruppi di backreference* che hanno verificato il pattern.

Proviamo a cercare tutte le parole che seguono la parola `"terra"` o `"terre"` : la regular expression cerca prima una stringa che inizi per *terr* e che abbia una *a* oppure una *e*, uno spazio ed infine una stringa alfanumerica di uno o più caratteri ed un delimitatore di fine parola:

[67]:
```
import re

with open('psposi1.txt', encoding='utf-8') as f:
    testo = "\n".join(f.readlines())

re.findall(r'terr[ae]\s\w+\b', testo)
```

[67]:
```
['terre accennate', 'terra cotta', 'terra con']
```

Come vedi ci viene restituita una lista di stringhe, infatti non ci sono gruppi all'interno della regex e quindi ci viene restituito tutto il matching.

⊛⊛ **ESERCIZIO 9.1**: Prova a copiare la linea di sopra qua sotto e ad aggiungere uno o più gruppi nel pattern e vedi cosa succede:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[68]:
```
# scrivi qui

import re

with open('psposi1.txt', encoding='utf-8') as f:
    testo = "\n".join(f.readlines())

#Aggiungo un gruppo attorno a terr[ae]
print(re.findall(r'(terr[ae])\s\w+\b', testo))
#Ne aggiungo un altro attorno a \w+
print(re.findall(r'(terr[ae])\s(\w+)\b', testo))
```

```
['terre', 'terra', 'terra']
[('terre', 'accennate'), ('terra', 'cotta'), ('terra', 'con')]
```

[68]:
```
# scrivi qui
```

```
['terre', 'terra', 'terra']
[('terre', 'accennate'), ('terra', 'cotta'), ('terra', 'con')]
```

[ ]:

5.4 Integrazione dati

5.4.1 Scarica zip esercizi

Naviga file online[330]

5.4.2 Introduzione

In questo tutorial parleremo di integrazione di dati, presentando diversi argomenti / tool:

- OpenStreetMap
- UMap
- Formato JSON
- Web API

In particolare mapperemo degli agritur del Trentino su Umap, seguendo grossomodo il tutorial sugli agritur di CoderDojoTrento[331]. La differenza sostanziale sarà che in questo caso invece dei Google spreadsheet useremo Python.

Scaletta:

- 0 Presentazione OpenStreetMap e UMap
- 1 prendere dati agritur da dati.trentino.it
- 2 leggere file CSV in Python
- 3 cercare automaticamente coordinate geografiche usando le web api di MapQuest / OpenStreetMap Nominatim
  - parsing formato json
- 4 scrivere il nuovo file CSV con i campi latitudine e longitudine riempiti
- 5 importare il file CSV in Umap
- 6 Inserire la mappa in Jupyter

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
integration
    integration.ipynb
    integration-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

[330] https://github.com/DavidLeoni/softpython-it/tree/master/integration
[331] https://www.coderdojotrento.it/risorse/openstreetmap-e-agritur/

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `integration.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

5.4.3 0 Presentazione OpenStreetMap e UMap

0.1 OpenStreetMap

Conosciamo tutti le Google Maps. Se le usiamo come privati cittadini, sono gratute, ma se siamo delle aziende Google potrebbe porci dei vincoli al riuso - sicuramente per scaricare tutto il database del mondo dovremmo chiedere permessi e pagare soldoni, sempre che Google ce lo conceda ! Esistono alternative? Sappiamo che esiste un enciclopedia realizzata completamente da volontari che si chiama Wikipedia, e per le mappe? Per nostra fortuna esiste OpenStreetMap[332] (detta anche OSM), che è una mappa del mondo realizzata da volontari, completamente gratuita, nonchè scaricabile e riusabile con licenza OpenDatabase License[333]. La licenza è piuttosto libera, il peggio che può capitare è che in alcuni casi si sia costretti a ripubblicare le proprie mappe costruite usando dati presi da OpenStreetMap.

⊛ **ESERCIZIO 0.1.1**: Per capire un po' come navigare OpenStreetMap, puoi guardare il tutorial di CoderDojo Trento Inseriamo un punto in OpenStreetMap[334]. Non serve che inserisci punti in OpenStreetMap, ma naturalmente se vuoi arricchire la mappa la comunità te ne sarà grata !

⊛ **ESERCIZIO 0.1.2** Per capire le potenzialità di OpenStreetMap e le differenze con Google Maps, guarda anche queste diverse visualizzazioni di OpenStreetMap che evidenziano alcune categorie di punti che si trovano in OpenStreetMap:

OsmHydrant[335]: Mappa di idranti, notare il raggio d'azione utile degli idranti

WheelMap[336]: mappa dei luoghi accessibili ai disabili

TagInfo[337] Ad ogni oggetto in OpenStreetMap si possono aggiungere tag. TagInfo mostra le tag più utilizzate.

0.1 UMap

UMap[338] è un tool online per creare mappe in cui si sovrappongono punti che vogliamo noi a OpenStreetMap.

⊛ **ESERCIZIO 0.2.1**: Se non l'hai già fatto precedentemente, prova a fare il tutorial di CoderDojo Trento sui Servizi di Rovereto e Umap[339]. E' molto semplice e non serve Python, basta usare tool online. Mostra come prendere da dati.trentino.it i servizi di Rovereto in formato JSON georeferenziati, convertirli in CSV e importarli in UMap

[332] http://openstreetmap.org
[333] https://it.okfn.org/odbl-riassunto/
[334] https://www.coderdojotrento.it/materiale/webmapping/tutorial/osm-inseriamo-un-punto/tutorial-osm-inseriamo-un-punto.pdf
[335] https://www.osmhydrant.org/en/
[336] https://wheelmap.org/it/map#/?lat=46.0900194916288&lon=11.123485565185547&q=trento&zoom=15
[337] https://taginfo.openstreetmap.org
[338] https://umap.openstreetmap.fr
[339] https://docs.google.com/presentation/d/1CWo9pFl6jcR1EmDAXOmNeOayfyjfLqLR5-h5U8zxrrk/edit#slide=id.g518a59eb3_0_0

5.4.4 1 Prendiamoci i dati

Concentriamoci adesso sugli Agritur. Andiamo a cercarci dei dati dal catalogo opendata `dati.trentino.it`. In questo caso sceglieremo un file dal dataset Agritur del Trentino[340].

⊛ **DOMANDA 1.1**: Quale è la licenza del dataset? Possiamo farci tutto quello che vogliamo ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: E' CC-BY, basta citare la fonte

Nel dataset troviamo la risorsa Elenco Agritur[341] che al suo interno contiene un link ad un file CSV.

ATTENZIONE: Per questo esercizio, **NON** usate il CSV dal sito, ma scaricate invece agritur16_10_2014.csv a questo link che come dal nome contiene un file del 2014.

⊛ **DOMANDA 1.2**: notate differenze tra il file del 2014 e quello corrente sul sito ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: al momento (Marzo 2018) neanche si può scaricare, precedentemente a Novembre 2017 rispetto al file del 2014 erano apparsi cambiamenti nel formato delle colonne

⊛ **DOMANDA 1.3**: Se realizzaste un programma per leggere questi file degli agriturismi scaricandoli periodicamente dal sito, a lungo andare quali problemi potrebbero insorgere (pensate anche al contenuto del file) ?

Vediamo qua una anterprima del file agritur16_10_2014.csv (NOTA: i campi a destra sono tagliati) :

```
N_prog;Num_archivio;data_rilascio_prima_autorizzazione;Nome_Impresa_agricola;A ...
1;10;11/11/1986;DALLAGO LUCIANO;no;38060;Aldeno;Via S. D'Acquisto n. 4;29251/9 ...
2;26;16/12/1986;ARMAN CRISTINA;no;38010;Faedo;Loc. Pineta - Maso Nello;137876/ ...
3;37;22/12/1986;INAMA FRANCESCO;no;38010;Sanzeno;Via Casalini n. 74;130041/96; ...
4;49;15/01/1987;MONTIBELLER VALTER;no;38050;Roncegno;Via Prose n. 1;138559/96; ...
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Bisogna sempre prepararsi al peggio: sparizione di colonne, aggiunta di nuove colonne, cambiamento dei valori attesi dentro le colonne, sparizione di tutto il file…

⊛ **DOMANDA 1.4**: Ci sono le intestazioni ? Qual'è il separatore ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: ci sono le intestazioni, separate dal punto e virgola.

[340] http://dati.trentino.it/dataset/agriturismi-del-trentino
[341] http://dati.trentino.it/dataset/agriturismi-del-trentino/resource/bb9f3185-602d-44c5-9f06-fc1f7ae25038

⊛ **DOMANDA 1.5**: Quanti indirizzi ci sono nel file ? Se volessimo posizionare gli agritur su una mappa, quali indirizzi useremmo ? Ci sono le coordinate geografiche?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

indirizzi:

```
indirizzo_impresa_agricola
Indirizzo_agriturismo
```

coordinate geografiche: Latitudine e Longitudine

La posizione degli agritur è in Indirizzo_agriturismo, ma le coordinate geografiche sono vuote.

⊛ **DOMANDA 1.6**: Prova ad aprire il file in LibreOffice Calc o Excel, facendo attenzione a specificare i separatori corretti e l'encoding (guarda l'header 'Società' , in particolare il carattere finale !)

ATTENZIONE SE USI EXCEL!

Facendo direttamente `File->Apri in Excel`, probabilmente Excel cercherà di immaginarsi da solo come intabellare il CSV, e sbaglierà metterà tutto le righe in una colonna. Per ovviare al problema, dobbiamo dire ad Excel di mostrare un pannello per chiederci come vogliamo aprire il CSV, facendo così:

- In Excel vecchi, cerca `File-> Importa`
- In Excel recenti, clicca la scheda `Dati` e poi seleziona `Da testo`. Per ulteriori riferimenti su Excel, vedere guida di Salvatore Aranzulla[342]

5.4.5 2. Leggiamo il CSV

⊛ **ESERCIZIO 2.1**: Prova a caricare le prime 10 righe del file CSV in Python usando le istruzioni già viste nella capitolo sui formati[343]

Fai attenzione al delimitatore e specifica `encoding='utf-8'` come parametro nella `open` !

Per ottenere 10 righe, puoi usare un ciclo `while` e ottenere ciascuna riga con un istruzione del tipo

```
row = next(lettore)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[1]:
```
# scrivi qui

import csv
with open('agritur16_10_2014.csv', encoding='utf-8', newline='') as f:
    lettore = csv.reader(f, delimiter=';')
    i = 0
```

(continues on next page)

[342] https://www.aranzulla.it/come-aprire-file-csv-672903.html
[343] http://it.softpython.org/formats/formats-sol.html#File-CSV

(continued from previous page)

```
    while i < 10:
        row = next(lettore)
        print(row)
        i += 1
```

```
['N_prog', 'Num_archivio', 'data_rilascio_prima_autorizzazione', 'Nome_Impresa_
↪agricola', 'Azienda_zootecnica', 'CAP', 'sede_impresa_agricola', 'indirizzo_impresa_
↪agricola', 'CCIAA', 'pref', 'tel', 'società', 'malga', 'PEC', 'indirizzo_e_mail',
↪'Altitudine', 'Latitudine', 'Longitudine', 'Comune_Sede_Agriturismo', 'Indirizzo_
↪agriturismo', 'Denominazione_Agriturismo', 'N_appartamenti', 'N_stanze_in_
↪appartamento', 'Tot_posti_letto_in_appartamento', 'N_bagni_appartamento', 'N_Stanze
↪', 'Tot_posti_letto_in_stanze', 'N_bagni_in_stanze', 'N_tot_stanze_e_stanze_in_
↪appartamento', 'Tot_posti_letto', 'Alloggio_in_appartamenti', 'Alloggio_in_stanze',
↪'Prima_colazione', 'Ristorazione_solo_per_alloggiati', 'Servizio_Ristorante_per_
↪passanti', 'N_posti_tavola_interni_a_struttura', 'N_posti_tavola_esterni_alla_
↪struttura', 'N_tot_posti_tavola', 'N_tot_Bagni', 'Agricampeggio', 'N_piazzole', 'N_
↪tot_posti_in_campeggio', 'Fattoria_didattica', 'Altre_attività_ricreative',
↪'Degustazione', 'Classificazione_punteggio_tabella_1', 'Classificazione_punteggio_
↪tabella_2', 'Classificazione_punteggio_tabella_3', 'ASSEGNAZIONE_MARGHERITE']
['1', '10', '11/11/1986', 'DALLAGO LUCIANO', 'no', '38060', 'Aldeno', "Via S. D
↪'Acquisto n. 4", '29251/97', '0461', '842629', 'no', 'no', 'luciano.dallago@pec.
↪agritel.it', 'gastone.dallago@smach.it ', '810', '', '', 'Garniga Terme', 'Loc. al␣
↪Lago n. 15', 'DALLAGO LUCIANO', '2', '5', '10', '2', '0', '0', '0', '5', '10', 'si',
↪ 'no', 'no', 'no', 'no', '0', '0', '0', '0', 'no', '0', '0', 'no', '', 'no', '0', '0
↪', '0', '']
['2', '26', '16/12/1986', 'ARMAN CRISTINA', 'no', '38010', 'Faedo', 'Loc. Pineta -␣
↪Maso Nello', '137876/96', '0461', '650384', 'no', 'no', 'cristina.arman@pec.agritel.
↪it', 'masonello@hotmail.com ', '595', '', '', 'Faedo', 'Loc. Pineta - Maso Nello',
↪'MASO NELLO', '1', '2', '4', '1', '4', '8', '4', '6', '12', 'si', 'si', 'si', 'no',
↪'si', '30', '0', '30', '1', 'no', '0', '0', 'si', '', 'si', '31', '40', '10', '3']
['3', '37', '22/12/1986', 'INAMA FRANCESCO', 'no', '38010', 'Sanzeno', 'Via Casalini␣
↪n. 74', '130041/96', '0463', '434072', 'no', 'no', 'inama.francesco@cia.legalmail.it
↪', '', '641', '', '', 'Sanzeno', 'Via Casalini n. 74', 'ANAUNIA', '0', '0', '0', '0
↪', '5', '10', '5', '5', '10', 'no', 'si', 'si', 'no', 'no', '0', '0', '0', '0', 'no
↪', '0', '0', 'no', '', 'no', '22', '19', '6', '2']
['4', '49', '15/01/1987', 'MONTIBELLER VALTER', 'no', '38050', 'Roncegno', 'Via Prose␣
↪n. 1', '138559/96', '0461', '773349', 'no', 'no', 'AZ.MONTIBELLER@PEC.IT', '', '535
↪', '', '', 'Roncegno', 'Via Prose n. 1', 'MONTIBELLER', '6', '6', '8', '6', '4', '8
↪', '4', '10', '16', 'si', 'si', 'si', 'no', 'si', '43', '0', '43', '2', 'si', '15',
↪'30', 'si', 'bagni di fieno', 'si', '32', '24', '10', '2']
['5', '52', '20/01/1987', 'BRESCIANI ITALO', 'no', '38060', 'Tenno', 'Via Diaz n. 40 -
↪ Fraz. Cologna', '138928/96', '0464', '521701', 'no', 'no', 'italo.bresciani@pec.
↪agritel.it', '', '428', '', '', 'Tenno', 'Loc. Fontanelle', 'PIZACOL DI BRESCIANI␣
↪ITALO', '0', '0', '0', '0', '0', '0', '0', '0', '0', 'no', 'no', 'no', 'no', 'si',
↪'35', '0', '35', '2', 'no', '0', '0', 'no', '', 'no', '7', '11', '2', '2']
['6', '61', '30/01/1987', 'FONTANARI ETTORE', 'no', '38057', 'Pergine Valsugana',
↪'Via Chimelli n. 25', '38575/97', '0461', '530023', 'no', 'no', 'ettore.
↪fontanari@pec.agritel.it', '', '482', '', '', 'Pergine Valsugana', 'Via Chimelli n.␣
↪25', 'FONTANARI ETTORE', '4', '6', '9', '4', '0', '0', '0', '6', '9', 'si', 'no',
↪'no', 'no', 'no', '0', '0', '0', '0', 'no', '0', '0', 'no', '', 'no', '8', '14', '4
↪', '2']
['7', '68', '30/01/1987', 'MARINCONZ GINO', 'no', '38010', 'Coredo', 'Via G. Inama n.␣
↪21', '61840/97', '0463', '536328', 'no', 'no', 'GINO.MARINCONZ@PEC.IT', '', '831', '
↪', '', 'Coredo', 'Via G. Inama n. 21', 'MARINCONZ GINO', '3', '6', '11', '3', '0',
↪'0', '0', '6', '11', 'si', 'no', 'no', 'no', 'no', '0', '0', '0', '0', 'no', '0', '0
↪', 'no', '', 'no', '9', '12', '4', '2']
```

(continues on next page)

(continued from previous page)

```
['8', '74', '06/02/1987', 'BERNARDI ARMANDA BORTOLOTTI', 'no', '38057', 'Pergine␣
↪Valsugana', 'Via Montesei n. 2', '141055/96', '0461', '530125', 'no', 'no',
↪'AGRITUR.BORTOLOTTI@PEC.CGN.IT', 'agritur.bortolotti@tin.it', '482', '', '',
↪'Pergine Valsugana', 'Via Montesei n. 4', 'AGRITUR BORTOLOTTI', '2', '2', '8', '2',
↪'6', '14', '6', '8', '22', 'si', 'si', 'si', 'no', 'si', '25', '0', '25', '2', 'no',
↪ '0', '0', 'si', '', 'no', '31', '26', '11', '3']
['9', '94', '06/05/1987', 'ZAMBONI PIA', 'no', '38040', 'Bosentino', 'Maso Fosina n. 5
↪', '129820/96', '0461', '848468', 'no', 'no', 'pia.zamboni@pec.agritel.it', '', '700
↪', '', '', 'Calceranica al Lago', 'Maso Marini', 'MASO MARINI', '3', '5', '8', '3',
↪'0', '0', '0', '5', '8', 'si', 'no', 'no', 'no', 'no', '0', '0', '0', '0', 'no', '0
↪', '0', 'no', '', 'no', '27', '39', '15', '4']
```

[1]:
```
# scrivi qui
```

```
['N_prog', 'Num_archivio', 'data_rilascio_prima_autorizzazione', 'Nome_Impresa_
↪agricola', 'Azienda_zootecnica', 'CAP', 'sede_impresa_agricola', 'indirizzo_impresa_
↪agricola', 'CCIAA', 'pref', 'tel', 'società', 'malga', 'PEC', 'indirizzo_e_mail',
↪'Altitudine', 'Latitudine', 'Longitudine', 'Comune_Sede_Agriturismo', 'Indirizzo_
↪agriturismo', 'Denominazione_Agriturismo', 'N_appartamenti', 'N_stanze_in_
↪appartamento', 'Tot_posti_letto_in_appartamento', 'N_bagni_appartamento', 'N_Stanze
↪', 'Tot_posti_letto_in_stanze', 'N_bagni_in_stanze', 'N_tot_stanze_e_stanze_in_
↪appartamento', 'Tot_posti_letto', 'Alloggio_in_appartamenti', 'Alloggio_in_stanze',
↪'Prima_colazione', 'Ristorazione_solo_per_alloggiati', 'Servizio_Ristorante_per_
↪passanti', 'N_posti_tavola_interni_a_struttura', 'N_posti_tavola_esterni_alla_
↪struttura', 'N_tot_posti_tavola', 'N_tot_Bagni', 'Agricampeggio', 'N_piazzole', 'N_
↪tot_posti_in_campeggio', 'Fattoria_didattica', 'Altre_attività_ricreative',
↪'Degustazione', 'Classificazione_punteggio_tabella_1', 'Classificazione_punteggio_
↪tabella_2', 'Classificazione_punteggio_tabella_3', 'ASSEGNAZIONE_MARGHERITE']
['1', '10', '11/11/1986', 'DALLAGO LUCIANO', 'no', '38060', 'Aldeno', "Via S. D
↪'Acquisto n. 4", '29251/97', '0461', '842629', 'no', 'no', 'luciano.dallago@pec.
↪agritel.it', 'gastone.dallago@smach.it ', '810', '', '', 'Garniga Terme', 'Loc. al␣
↪Lago n. 15', 'DALLAGO LUCIANO', '2', '5', '10', '2', '0', '0', '0', '5', '10', 'si',
↪ 'no', 'no', 'no', 'no', '0', '0', '0', '0', 'no', '0', '0', 'no', '', 'no', '0', '0
↪', '0', '']
['2', '26', '16/12/1986', 'ARMAN CRISTINA', 'no', '38010', 'Faedo', 'Loc. Pineta -␣
↪Maso Nello', '137876/96', '0461', '650384', 'no', 'no', 'cristina.arman@pec.agritel.
↪it', 'masonello@hotmail.com ', '595', '', '', 'Faedo', 'Loc. Pineta - Maso Nello',
↪'MASO NELLO', '1', '2', '4', '1', '4', '8', '4', '6', '12', 'si', 'si', 'si', 'no',
↪'si', '30', '0', '30', '1', 'no', '0', '0', 'si', '', 'si', '31', '40', '10', '3']
['3', '37', '22/12/1986', 'INAMA FRANCESCO', 'no', '38010', 'Sanzeno', 'Via Casalini␣
↪n. 74', '130041/96', '0463', '434072', 'no', 'no', 'inama.francesco@cia.legalmail.it
↪', '', '641', '', '', 'Sanzeno', 'Via Casalini n. 74', 'ANAUNIA', '0', '0', '0', '0
↪', '5', '10', '5', '5', '10', 'no', 'si', 'si', 'no', 'no', '0', '0', '0', '0', 'no
↪', '0', '0', 'no', '', 'no', '22', '19', '6', '2']
['4', '49', '15/01/1987', 'MONTIBELLER VALTER', 'no', '38050', 'Roncegno', 'Via Prose␣
↪n. 1', '138559/96', '0461', '773349', 'no', 'no', 'AZ.MONTIBELLER@PEC.IT', '', '535
↪', '', '', 'Roncegno', 'Via Prose n. 1', 'MONTIBELLER', '6', '6', '8', '6', '4', '8
↪', '4', '10', '16', 'si', 'si', 'si', 'no', 'si', '43', '0', '43', '2', 'si', '15',
↪'30', 'si', 'bagni di fieno', 'si', '32', '24', '10', '2']
['5', '52', '20/01/1987', 'BRESCIANI ITALO', 'no', '38060', 'Tenno', 'Via Diaz n. 40 -
↪ Fraz. Cologna', '138928/96', '0464', '521701', 'no', 'no', 'italo.bresciani@pec.
↪agritel.it', '', '428', '', '', 'Tenno', 'Loc. Fontanelle', 'PIZACOL DI BRESCIANI␣
↪ITALO', '0', '0', '0', '0', '0', '0', '0', '0', '0', 'no', 'no', 'no', 'no', 'si',
↪'35', '0', '35', '2', 'no', '0', '0', 'no', '', 'no', '7', '11', '2', '2']
```

(continues on next page)

(continued from previous page)

```
['6', '61', '30/01/1987', 'FONTANARI ETTORE', 'no', '38057', 'Pergine Valsugana',
↪'Via Chimelli n. 25', '38575/97', '0461', '530023', 'no', 'no', 'ettore.
↪fontanari@pec.agritel.it', '', '482', '', '', 'Pergine Valsugana', 'Via Chimelli n.␣
↪25', 'FONTANARI ETTORE', '4', '6', '9', '4', '0', '0', '0', '6', '9', 'si', 'no',
↪'no', 'no', 'no', '0', '0', '0', '0', 'no', '0', '0', 'no', '', 'no', '8', '14', '4
↪', '2']
['7', '68', '30/01/1987', 'MARINCONZ GINO', 'no', '38010', 'Coredo', 'Via G. Inama n.␣
↪21', '61840/97', '0463', '536328', 'no', 'no', 'GINO.MARINCONZ@PEC.IT', '', '831', '
↪', '', 'Coredo', 'Via G. Inama n. 21', 'MARINCONZ GINO', '3', '6', '11', '3', '0',
↪'0', '0', '6', '11', 'si', 'no', 'no', 'no', 'no', '0', '0', '0', '0', 'no', '0', '0
↪', 'no', '', 'no', '9', '12', '4', '2']
['8', '74', '06/02/1987', 'BERNARDI ARMANDA BORTOLOTTI', 'no', '38057', 'Pergine␣
↪Valsugana', 'Via Montesei n. 2', '141055/96', '0461', '530125', 'no', 'no',
↪'AGRITUR.BORTOLOTTI@PEC.CGN.IT', 'agritur.bortolotti@tin.it', '482', '', '',
↪'Pergine Valsugana', 'Via Montesei n. 4', 'AGRITUR BORTOLOTTI', '2', '2', '8', '2',
↪'6', '14', '6', '8', '22', 'si', 'si', 'si', 'no', 'si', '25', '0', '25', '2', 'no',
↪ '0', '0', 'si', '', 'no', '31', '26', '11', '3']
['9', '94', '06/05/1987', 'ZAMBONI PIA', 'no', '38040', 'Bosentino', 'Maso Fosina n. 5
↪', '129820/96', '0461', '848468', 'no', 'no', 'pia.zamboni@pec.agritel.it', '', '700
↪', '', '', 'Calceranica al Lago', 'Maso Marini', 'MASO MARINI', '3', '5', '8', '3',
↪'0', '0', '0', '5', '8', 'si', 'no', 'no', 'no', 'no', '0', '0', '0', '0', 'no', '0
↪', '0', 'no', '', 'no', '27', '39', '15', '4']
```

5.4.6 3. Geocoding con webapi

3.1 MapQuest / OpenStreetMap Nominatim

Possiamo leggere i valori dal CSV, ma purtroppo notiamo che mancano le coordinate geografiche. Per ottenerle, possiamo usare i servizi di MapQuest, che ci offre gratutitamente un cosiddetto servizio di *geocoding* : Dati degli indirizzi, ci ritornerà le loro coordinate geografiche usando OpenStreetMap (detto OSM per gli amici) come riferimento.

Per capire cosa potremmo avere indietro, cerchiamo su OpenStreetMap un agritur, tipo il Montibeller di Roncegno:

- Sito di OpenStreetMap, stringa di ricerca `Montibeller, Roncegno`:

  http://www.openstreetmap.org/search?query=Montibeller%2C%20Roncegno#map=19/46.04691/11.41157

- Sito di OpenStreetMap, stringa di ricerca `Via Prose n. 1, Roncegno`:

  http://www.openstreetmap.org/search?query=via%20Prose%20n.%201%2C%20Roncegno#map=18/46.04698/11.41101

⊛ **DOMANDA 3.1.1**: Le due stringhe trovano risultati diversi. Come mai? Manca forse qualche dato ad OpenStreetMap?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Mancano i numeri civici, ma c'è il nome esatto dell'agritur. Quindi la query con il numero civico viene semplicemente centrata sulla strada ma non sull'agritur

3.2 Webapi JSON

Oggigiorno, tantissimi portali offrono la possibilità di leggere e scrivere informazioni programmaticamente tramite cosiddette 'API REST'. `API` significa `Application Programming Interface`, ed è una serie di specifiche su come accedere programmaticamente ai dati di un sito. Di solito, le API disponibili vengono descritte nella sezione sviluppatori.

⊛ **DOMANDA 3.2.2**: prova ad andare sul sito di dati.gov.it[344] e cerca dove sono le API. Provane qualcuna dal browser cercando di capire cosa viene ritornato.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Ci sono API alla sezione Sviluppatori[345] tipo questa che ritorna tutti i dataset (chiamati anche *packages* in CKAN) :

http://www.dati.gov.it/api/3/action/package_list

Webapi geografiche

Abbiamo visto una bella rappresentazione grafica del punto sulla mappa. Però ci piacerebbe ottenere quell'informazione in Python. Come fare? openstreetmap.org[346] offre delle API che potremmo usare, ma teniamo presente che OpenStreetMap è un servizio gratuito gestito principalmente su base volontaristica con risorse limitate.

In alternativa, conviene rivolgersi a servizi offerti da aziende commerciali che possono sostenere un traffico più elevato. Di interessante c'è MapQuest, che oltre a mappe commerciali, offre anche lo stesso identico servizio di OpenStreetMap (può farlo perchè le condizioni di licenza di OSM, molto libere, lo permettono). L'unico vincolo è che per usare il servizio bisogna prima registrare una cosiddetta 'Api key' da passare al servizio ogni volta che lo usiamo. Questo consente a MapQuest di monitorare eventuali abusi del servizio (per più info, vedere le condizioni di licenza di MapQuest[347]. Quando ci connettiamo a indirizzi che iniziano con `open.mapquestapi`, vuol dire che stiamo usando mappe di OpenStreetMap.

Specifichiamo un paio di parametri importanti delle nostre chiamate web:

[2]:
```
api_key = "Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG"  # Usate questa key SOLO per fare questi␣
↪esercizi!
url_base = "http://open.mapquestapi.com/nominatim/v1/search"
```

Provate a fare copia e incolla nel vostro broswer dell'indirizzo seguente:

[3]:
```
print(url_base)
```

```
http://open.mapquestapi.com/nominatim/v1/search
```

MapQuest dovrebbe rispondervi così:

```
The AppKey submitted with this request is invalid.
```

Si è offeso perchè non gli abbiamo passato una api key.

> **IMPORTANTE:** Per oggi, la api key ve la diamo noi, ma se usate il servizio per i vostri progetti, registratevene una[348]!!!!**

[344] https://www.dati.gov.it/
[345] https://www.dati.gov.it/content/sviluppatori
[346] http://openstreetmap.org
[347] https://developer.mapquest.com/documentation/open/

Proviamo ad aggiungere la api key, mettendo il tutto nel browser dovremmo vedere dei bottoni (notate che il primo parametro è sempre preceduto da il punto di domanda `?`) :

[4]:
```
url_with_key = url_base + "?key=" + api_key
print(url_with_key)
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG
```

Se siamo riusciti a vedere dei bottoni, vuol dire che abbiamo superato il primo ostacolo. Ma Python di bottoni non ci capisce un tubo! Dobbiamo fornire a pyhton un formato più 'digeribile':

[5]:
```
url_json = url_base + "?key=" + api_key + "&format=json"
```

Se proviamo nel browser questa nuova url, vedremo che è sparito tutto - al più vedrai due parentesi quadre vuote `[]`. Un JSON vuoto non è per niente interessante, ma almeno sono spariti i bottoni:

[6]:
```
print(url_json)
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json
```

Siamo finalmente pronti per eseguire la nostra prima query via web api ! Nella nostra query, scriveremo in linguaggio naturale `Montibeller,Roncegno`:

[7]:
```

query = "Montibeller,Roncegno"

url_complete = url_json = url_base + "?key=" + api_key + "&format=json"  + "&q=" +␣
↪query

print(url_complete)
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&q=Montibeller,Roncegno
```

Adesso dovremmo vedere un risultato tipo questo:

```
[{"place_id":"6560673","licence":"Data \u00a9 OpenStreetMap contributors, ODbL 1.0.␣
↪http:\/\/www.openstreetmap.org\/copyright","osm_type":"node","osm_id":"673194565",
↪"boundingbox":["46.0469105","46.0469105","11.4115734","11.4115734"],"lat":"46.
↪0469105","lon":"11.4115734","display_name":"Montibeller, Via Prose, Alps, Salembis,␣
↪Roncegno Terme, Comunit\u00e0 Valsugana e Tesino, Provincia autonoma di Trento,␣
↪Trentino-Alto Adige - S\u00fcdtirol, 38051, Italy","class":"tourism","type":"hotel",
↪"importance":0.211,"icon":"http:\/\/ip-10-98-183-183.mq-us-east-1.ec2.aolcloud.net:
↪8000\/nominatim\/v1\/images\/mapicons\/accommodation_hotel2.p.20.png"}]
```

prova a copia e incollare il risultato in un editor che supporta i JSON, salva il file come `.json` e prova a dire all'editor di riformattare il documento. Se tutto va bene, dovrebbe venire fuori un bell'albero ordinato così:

```
[
    {
        "place_id": "6560673",
        "licence": "Data \u00a9 OpenStreetMap contributors, ODbL 1.0. http:\/\/www.
↪openstreetmap.org\/copyright",
        "osm_type": "node",
        "osm_id": "673194565",
        "boundingbox": [
```

(continues on next page)

[348] https://developer.mapquest.com/plan_purchase/steps/business_edition/business_edition_free/register

(continued from previous page)

```
            "46.0469105",
            "46.0469105",
            "11.4115734",
            "11.4115734"
        ],
        "lat": "46.0469105",
        "lon": "11.4115734",
        "display_name": "Montibeller, Via Prose, Alps, Salembis, Roncegno Terme,␣
↪Comunit\u00e0 Valsugana e Tesino, TN, Trentino-Alto Adige - S\u00fcdtirol, 38051,␣
↪Italy",
        "class": "tourism",
        "type": "hotel",
        "importance": 0.211,
        "icon": "http:\/\/ip-10-98-183-183.mq-us-east-1.ec2.aolcloud.net:8000\/
↪nominatim\/v1\/images\/mapicons\/accommodation_hotel2.p.20.png"
    }
]
```

Mmm.. non sembra tanto diversa da una combinazione di liste e dizionari Python… Forse possiamo riuscire ad estrarre quel `lat` e `lon` senza neanche troppa fatica …

3.3. Requests in Python

Per chiamare le webapi da Python, installiamo la libreria `requests`:

- Anaconda: `conda install requests`
- Linux/Mac : `python3 -m pip install --user requests`

[8]:
```
# importiamo il modulo per la libreria:
import requests

query = "Montibeller,Roncegno"

url_complete = url_json = url_base + "?key=" + api_key + "&format=json"  + "&q=" +␣
↪query


# effettuiamo una chiamata HTTP GET:

r = requests.get(url_complete)
```

Stampando direttamente `r`, vedremo qual'è stato il codice di risposta. Se è `200`, vuol dire che è andato tutto bene. er altri possibili codici di risposta, puoi guardare Wikipedia[349]

[9]:
```
print(r)
```

```
<Response [200]>
```

Possiamo accedere al contenuto testuale della risposta con `r.text`:

[10]:
```
r.text
```

[10]:
```
'[{"place_id":"6471026","licence":"Data © OpenStreetMap contributors, ODbL 1.0. https:
↪\\/\\/www.openstreetmap.org\\/copyright","osm_type":"node","osm_id":"673194565",
↪"boundingbox":["46.0468605","46.0469605","11.4115234","11.4116234"],"lat":"46.
↪0469105","lon":"11.4115734","display_name":"Montibeller, Via Prose, Maso Vazzena,␣
↪Larganza, Roncegno Terme, Comunità Valsugana e Tesino, TN, TAA, 38051, Italia",
↪"class":"tourism","type":"hotel","importance":0.211,"icon":"http:\\/\\/ip-10-98-165-
↪39.mq-us-east-1.ec2.aolcloud.net\\/nominatim\\/images\\/mapicons\\/accommodation_
↪hotel2.p.20.png"}]'
```

(continues on next page)

[349] https://it.wikipedia.org/wiki/Codici_di_stato_HTTP

(continued from previous page)

```
[11]: type(r.text)
[11]: str
```

Gli headers `content-type` ci dicono il tipo di formato e l'encoding dichiarato dal server (NOTA: purtroppo l'encoding dichiarato dal server non sempre corrisponde a quello effettivo !):

```
[12]: r.headers['content-type']
[12]: 'application/json; charset=UTF-8'
```

Possiamo anche ottenere l'encoding direttamente:

```
[13]: r.encoding
[13]: 'UTF-8'
```

Dalle ispezioni fatte sinora, abbiamo capito che abbiamo ottenuto una stringa in formato json. `requests` mette a disposizione un comodo metodo che interpreta la stringa come json, e ritorna delle strutture dati Python per accedere facilmente ai campi interni del json. Quali strutture? Come avrete notato, il formato del json è molto simile a strutture dati che già abbiamo in python, come stringhe, numeri interi, float, liste e dizionari. L'unica differenza sono i campi `null` in json che diventano `None` in Python. Quindi la conversione a Python è quasi sempre facile e indolore:

```
[14]: r.json()
[14]: [{'place_id': '6471026',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'node',
  'osm_id': '673194565',
  'boundingbox': ['46.0468605', '46.0469605', '11.4115234', '11.4116234'],
  'lat': '46.0469105',
  'lon': '11.4115734',
  'display_name': 'Montibeller, Via Prose, Maso Vazzena, Larganza, Roncegno Terme,␣
↪Comunità Valsugana e Tesino, TN, TAA, 38051, Italia',
  'class': 'tourism',
  'type': 'hotel',
  'importance': 0.211,
  'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/accommodation_hotel2.p.20.png'}]
```

Notiamo che abbiamo ricevuto una lista di dizionari:

```
[15]: type(r.json())
[15]: list
```

Prendiamo il primo dizionario:

```
[16]: r.json()[0]
[16]: {'place_id': '6471026',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'node',
 'osm_id': '673194565',
 'boundingbox': ['46.0468605', '46.0469605', '11.4115234', '11.4116234'],
```

(continues on next page)

(continued from previous page)

```
 'lat': '46.0469105',
 'lon': '11.4115734',
 'display_name': 'Montibeller, Via Prose, Maso Vazzena, Larganza, Roncegno Terme,␣
↪Comunità Valsugana e Tesino, TN, TAA, 38051, Italia',
 'class': 'tourism',
 'type': 'hotel',
 'importance': 0.211,
 'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/accommodation_hotel2.p.20.png'}
```

[17]:
```
type(r.json()[0])
```

[17]:
```
dict
```

Prendiamo il campo `lat` dal primo dizionario:

[18]:
```
r.json()[0]['lat']
```

[18]:
```
'46.0469105'
```

[19]:
```
r.json()[0]['lon']
```

[19]:
```
'11.4115734'
```

⊛ **DOMANDA 3.3.1**: Come è stato convertito in Python il campo `lat`. Numero o qualcos'altro? In cosa potremmo convertirlo?:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[20]:
```
# scrivi qui
# è stato convertito in stringa, quindi si potrebbe pensare di convertirlo in `float`

print(type('11.4115734'))
print(float('11.4115734'))
```

```
<class 'str'>
11.4115734
```

[20]:
```
# scrivi qui
```

```
<class 'str'>
11.4115734
```

⊛ **ESERCIZIO 3.3.2**: prova a chiamare `requests.get` passandogli una URL sbagliata, come `boh` o parzialmente giusta come `http://open.mapquestapi.com/BLA`. Cosa ottieni di ritorno per i vari campi di `r`? Il codice HTTP di ritorno[350] (successo / errore) ti sembra consistente con il risultato che ottieni?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[350] https://it.wikipedia.org/wiki/Codici_di_stato_HTTP

[21]:
```
# scrivi qui

import requests
requests.get('http://open.mapquestapi.com/BLA')
```

[21]:
```
<Response [200]>
```

[21]:
```
# scrivi qui
```

[21]:
```
<Response [200]>
```

Notiamo che pur richiedendo una pagina inesistente, il sito di mapquest (a Marzo 2018) ci ritorna un codice http 200 che indicherebbe 'successo'. Questo comportamento può essere molto pericoloso, perchè può indurre programmi che ricevono la pagina a ritenere di aver ottenuto effettivamente quello che chiedevano, quando invece si dovrebbe essere verificato un errore. A peggiorare le cose, se avete TIM a volte ci si mettono di mezzo pure loro: a Ottobre 2016, se voi richiedete un indirizzo inesistente (http://bla), la TIM vi rimanda ad una pagina di cortesia che dice `Spiacenti, l'indirizzo digitato nonesiste`, ma mandava indietro al browser un codice di successo dal valore 200 !

3.4 Funzioni geocode per requests

Proviamo a scriverci delle funzioni comode per effettuare delle chiamate semplicemente passando un indirizzo

[22]:
```
def geocode_generic(address):
    # 'payload' è una variabile che ci definiamo noi, per metterci più comodamente i␣
↪parametri
    # dentro un dizionario
    payload = {'key': api_key,  # Questa è la chiave lunga tipo Er38Wk... che abbiamo␣
↪definito più sopra
               'format': 'json',
               'q' : address}
    r = requests.get(url_base, params=payload)  # qua passiamo il dizionario 'payload
↪' alla libreria requests
    print(r.url) # stampa l'url che requests ha usato
    return r.json()
```

Facciamo una prova:

[23]:
```
geocode_generic("Montibeller, Roncegno")
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&q=Montibeller%2C+Roncegno
```

[23]:
```
[{'place_id': '6471026',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'node',
  'osm_id': '673194565',
  'boundingbox': ['46.0468605', '46.0469605', '11.4115234', '11.4116234'],
  'lat': '46.0469105',
  'lon': '11.4115734',
  'display_name': 'Montibeller, Via Prose, Maso Vazzena, Larganza, Roncegno Terme,␣
↪Comunità Valsugana e Tesino, TN, TAA, 38051, Italia',
  'class': 'tourism',
```

(continues on next page)

(continued from previous page)

```
  'type': 'hotel',
  'importance': 0.211,
  'icon': 'http://ip-10-98-178-30.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/accommodation_hotel2.p.20.png'}]
```

A volte essere precisi non aiuta:

[24]:
```
json = geocode_generic("Montibeller, Via Prose n. 1, Roncegno")
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&q=Montibeller%2C+Via+Prose+n.+1%2C+Roncegno
```

[25]:
```
print(json)
```

```
[]
```

Per verificare se abbiamo trovato o meno qualcosa, possiamo controllare che la lunghezza della lista ritornata sia zero con `len`:

[26]:
```
if len(json) == 0:
    print("non ho trovato niente!")
else:
    print("ho trovato !")
```

```
non ho trovato niente!
```

[27]:
```
geocode_generic("Via Prose n. 1, Roncegno")
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&q=Via+Prose+n.+1%2C+Roncegno
```

[27]:
```
[{'place_id': '106260827',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'way',
  'osm_id': '149123476',
  'boundingbox': ['46.0464048', '46.047558', '11.4100616', '11.4119591'],
  'lat': '46.0469731',
  'lon': '11.4109589',
  'display_name': 'Via Prose, Maso Vazzena, Larganza, Roncegno Terme, Comunità␣
↪Valsugana e Tesino, TN, TAA, 38051, Italia',
  'class': 'highway',
  'type': 'residential',
  'importance': 0.525}]
```

Con query generiche è possibile che vengano ritornati parecchi risultati:

⊛ **DOMANDA 3.4.1**: Qual'è il risultato più rilevante secondo Nominatim (ricordiamo che Nominatim è il search engine di OpenStreetMap)? E quanto è rilevante ? Qual'è la rilevanza minimima? Qual'è la massima ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Il primo, i risultati sono ordinati secondo il campo `importance`, che può andare da 0 a 1.0. Molte API hanno ordinamenti di questo tipo (es: rilevanza risultati ricerca in Google)

[28]:
```
geocode_generic("Trento")
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&q=Trento
```

[28]:
```
[{'place_id': '186698302',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'relation',
  'osm_id': '46663',
  'boundingbox': ['45.9775306', '46.1530112', '11.0224735', '11.1948226'],
  'lat': '46.0664228',
  'lon': '11.1257601',
  'display_name': "Trento, Territorio Val d'Adige, TN, TAA, Italia",
  'class': 'place',
  'type': 'city',
  'importance': 0.26364591679333,
  'icon': 'http://ip-10-98-173-122.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_place_city.p.20.png'},
 {'place_id': '187948156',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'relation',
  'osm_id': '3870471',
  'boundingbox': ['7.9378151', '8.292381', '126.00048', '126.355208'],
  'lat': '8.114415',
  'lon': '126.158888603496',
  'display_name': 'Trento, Agusan del Sur, Caraga, 8505, Philippines',
  'class': 'boundary',
  'type': 'administrative',
  'importance': 0.2225,
  'icon': 'http://ip-10-98-173-122.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_boundary_administrative.p.20.png'},
 {'place_id': '652743',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'node',
  'osm_id': '248804218',
  'boundingbox': ['46.0721752', '46.0722752', '11.1189539', '11.1190539'],
  'lat': '46.0722252',
  'lon': '11.1190039',
  'display_name': "Trento, Piazzetta Filippo Foti e Edoardo Martini, Centro storico␣
↪Trento, Trento, Territorio Val d'Adige, TN, TAA, 38122, Italia",
  'class': 'railway',
  'type': 'stop',
  'importance': 0.18947959270135},
 {'place_id': '43377314',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'node',
  'osm_id': '3134911289',
  'boundingbox': ['44.9456976', '44.9856976', '11.4473224', '11.4873224'],
  'lat': '44.9656976',
  'lon': '11.4673224',
  'display_name': 'Trento, RO, VEN, Italia',
  'class': 'place',
  'type': 'hamlet',
  'importance': 0.17875,
```

(continues on next page)

(continued from previous page)

```
 'icon': 'http://ip-10-98-173-122.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_place_village.p.20.png'},
{'place_id': '531699',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'node',
 'osm_id': '198505670',
 'boundingbox': ['8.0059463', '8.0859463', '126.0214264', '126.1014264'],
 'lat': '8.0459463',
 'lon': '126.0614264',
 'display_name': 'Trento, Agusan del Sur, Caraga, 8505, Philippines',
 'class': 'place',
 'type': 'town',
 'importance': 0.16650761976897,
 'icon': 'http://ip-10-98-173-122.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_place_town.p.20.png'},
{'place_id': '105325154',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'way',
 'osm_id': '147148070',
 'boundingbox': ['44.1482269', '44.1507838', '4.8476098', '4.850579'],
 'lat': '44.14948815',
 'lon': '4.84974947145189',
 'display_name': "Trento, La Baussenque, Orange, Carpentras, Vaucluse, Provence-
↪Alpes-Côte d'Azur, France métropolitaine, 84100, France",
 'class': 'landuse',
 'type': 'industrial',
 'importance': 0.16},
{'place_id': '166654422',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'way',
 'osm_id': '431617802',
 'boundingbox': ['46.0782422', '46.0784452', '11.1421592', '11.1427322'],
 'lat': '46.0783806',
 'lon': '11.1424334',
 'display_name': "Trento, Marnighe, Trento, Territorio Val d'Adige, TN, TAA, 38122,␣
↪Italia",
 'class': 'highway',
 'type': 'pedestrian',
 'importance': 0.135},
{'place_id': '132505515',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'way',
 'osm_id': '252075211',
 'boundingbox': ['-32.9847715', '-32.9825163', '-60.6614742', '-60.6609117'],
 'lat': '-32.9836455',
 'lon': '-60.6611934',
 'display_name': 'Trento, Villa Moreno, Domingo Matheu, Distrito Sur, Rosario,␣
↪Municipio de Rosario, Departamento Rosario, Sta. Fe, S2000, Argentina',
 'class': 'highway',
 'type': 'residential',
 'importance': 0.135},
{'place_id': '89398989',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
```

(continues on next page)

(continued from previous page)

```
  'osm_type': 'way',
  'osm_id': '73838593',
  'boundingbox': ['-32.9893919', '-32.985958', '-60.6626152', '-60.6617997'],
  'lat': '-32.9882128',
  'lon': '-60.6623171',
  'display_name': 'Trento, La Guardia, Distrito Sur, Rosario, Municipio de Rosario,␣
↪Departamento Rosario, Sta. Fe, S2000, Argentina',
  'class': 'highway',
  'type': 'residential',
  'importance': 0.135},
 {'place_id': '89720595',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'way',
  'osm_id': '80595940',
  'boundingbox': ['45.7466148', '45.7469574', '9.2305622', '9.2319169'],
  'lat': '45.7468391',
  'lon': '9.2310522',
  'display_name': 'Trento, Inverigo, CO, LOM, 22044, Italia',
  'class': 'highway',
  'type': 'residential',
  'importance': 0.135}]
```

⊛⊛⊛ **ESERCIZIO 3.4.2**: guardando anche la documentazione di sorted[351] scrivere del codice python per riordinare i risultati precedenti dal meno rilevante al più rilevante. Ci sono vari modi per farlo, ma il più sintetico è con funzioni lambda. Riuscite a usarlo?

SUGGERIMENTO: Per ottenere per es. il campo `'osm_id'` di un dizionario `miodiz`, si può chiamare il metodo `miodiz.get('odm_id')`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[29]:
```
# scrivi qui

sorted(geocode_generic("Trento"), key=lambda res: res.get('importance'))
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&q=Trento
```

[29]:
```
[{'place_id': '166654422',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'way',
  'osm_id': '431617802',
  'boundingbox': ['46.0782422', '46.0784452', '11.1421592', '11.1427322'],
  'lat': '46.0783806',
  'lon': '11.1424334',
  'display_name': "Trento, Marnighe, Trento, Territorio Val d'Adige, TN, TAA, 38122,␣
↪Italia",
  'class': 'highway',
  'type': 'pedestrian',
  'importance': 0.135},
 {'place_id': '132505515',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
```

(continues on next page)

[351] https://docs.python.org/3/howto/sorting.html

(continued from previous page)

```
 'osm_type': 'way',
 'osm_id': '252075211',
 'boundingbox': ['-32.9847715', '-32.9825163', '-60.6614742', '-60.6609117'],
 'lat': '-32.9836455',
 'lon': '-60.6611934',
 'display_name': 'Trento, Villa Moreno, Domingo Matheu, Distrito Sur, Rosario,␣
↪Municipio de Rosario, Departamento Rosario, Sta. Fe, S2000, Argentina',
 'class': 'highway',
 'type': 'residential',
 'importance': 0.135},
{'place_id': '89398989',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'way',
 'osm_id': '73838593',
 'boundingbox': ['-32.9893919', '-32.985958', '-60.6626152', '-60.6617997'],
 'lat': '-32.9882128',
 'lon': '-60.6623171',
 'display_name': 'Trento, La Guardia, Distrito Sur, Rosario, Municipio de Rosario,␣
↪Departamento Rosario, Sta. Fe, S2000, Argentina',
 'class': 'highway',
 'type': 'residential',
 'importance': 0.135},
{'place_id': '89720595',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'way',
 'osm_id': '80595940',
 'boundingbox': ['45.7466148', '45.7469574', '9.2305622', '9.2319169'],
 'lat': '45.7468391',
 'lon': '9.2310522',
 'display_name': 'Trento, Inverigo, CO, LOM, 22044, Italia',
 'class': 'highway',
 'type': 'residential',
 'importance': 0.135},
{'place_id': '105325154',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'way',
 'osm_id': '147148070',
 'boundingbox': ['44.1482269', '44.1507838', '4.8476098', '4.850579'],
 'lat': '44.14948815',
 'lon': '4.84974947145189',
 'display_name': "Trento, La Baussenque, Orange, Carpentras, Vaucluse, Provence-
↪Alpes-Côte d'Azur, France métropolitaine, 84100, France",
 'class': 'landuse',
 'type': 'industrial',
 'importance': 0.16},
{'place_id': '531699',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'node',
 'osm_id': '198505670',
 'boundingbox': ['8.0059463', '8.0859463', '126.0214264', '126.1014264'],
 'lat': '8.0459463',
 'lon': '126.0614264',
 'display_name': 'Trento, Agusan del Sur, Caraga, 8505, Philippines',
```

(continues on next page)

(continued from previous page)

```
 'class': 'place',
 'type': 'town',
 'importance': 0.16650761976897,
 'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_place_town.p.20.png'},
{'place_id': '43377314',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'node',
 'osm_id': '3134911289',
 'boundingbox': ['44.9456976', '44.9856976', '11.4473224', '11.4873224'],
 'lat': '44.9656976',
 'lon': '11.4673224',
 'display_name': 'Trento, RO, VEN, Italia',
 'class': 'place',
 'type': 'hamlet',
 'importance': 0.17875,
 'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_place_village.p.20.png'},
{'place_id': '652743',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'node',
 'osm_id': '248804218',
 'boundingbox': ['46.0721752', '46.0722752', '11.1189539', '11.1190539'],
 'lat': '46.0722252',
 'lon': '11.1190039',
 'display_name': "Trento, Piazzetta Filippo Foti e Edoardo Martini, Centro storico␣
↪Trento, Trento, Territorio Val d'Adige, TN, TAA, 38122, Italia",
 'class': 'railway',
 'type': 'stop',
 'importance': 0.18947959270135},
{'place_id': '187948156',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'relation',
 'osm_id': '3870471',
 'boundingbox': ['7.9378151', '8.292381', '126.00048', '126.355208'],
 'lat': '8.114415',
 'lon': '126.158888603496',
 'display_name': 'Trento, Agusan del Sur, Caraga, 8505, Philippines',
 'class': 'boundary',
 'type': 'administrative',
 'importance': 0.2225,
 'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_boundary_administrative.p.20.png'},
{'place_id': '186698302',
 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
 'osm_type': 'relation',
 'osm_id': '46663',
 'boundingbox': ['45.9775306', '46.1530112', '11.0224735', '11.1948226'],
 'lat': '46.0664228',
 'lon': '11.1257601',
 'display_name': "Trento, Territorio Val d'Adige, TN, TAA, Italia",
 'class': 'place',
 'type': 'city',
```

(continues on next page)

(continued from previous page)

```
  'importance': 0.26364591679333,
  'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_place_city.p.20.png'}]
```

[29]:
```
# scrivi qui
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&q=Trento
```

[29]:
```
[{'place_id': '166654422',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'way',
  'osm_id': '431617802',
  'boundingbox': ['46.0782422', '46.0784452', '11.1421592', '11.1427322'],
  'lat': '46.0783806',
  'lon': '11.1424334',
  'display_name': "Trento, Marnighe, Trento, Territorio Val d'Adige, TN, TAA, 38122,␣
↪Italia",
  'class': 'highway',
  'type': 'pedestrian',
  'importance': 0.135},
 {'place_id': '132505515',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'way',
  'osm_id': '252075211',
  'boundingbox': ['-32.9847715', '-32.9825163', '-60.6614742', '-60.6609117'],
  'lat': '-32.9836455',
  'lon': '-60.6611934',
  'display_name': 'Trento, Villa Moreno, Domingo Matheu, Distrito Sur, Rosario,␣
↪Municipio de Rosario, Departamento Rosario, Sta. Fe, S2000, Argentina',
  'class': 'highway',
  'type': 'residential',
  'importance': 0.135},
 {'place_id': '89398989',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'way',
  'osm_id': '73838593',
  'boundingbox': ['-32.9893919', '-32.985958', '-60.6626152', '-60.6617997'],
  'lat': '-32.9882128',
  'lon': '-60.6623171',
  'display_name': 'Trento, La Guardia, Distrito Sur, Rosario, Municipio de Rosario,␣
↪Departamento Rosario, Sta. Fe, S2000, Argentina',
  'class': 'highway',
  'type': 'residential',
  'importance': 0.135},
 {'place_id': '89720595',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'way',
  'osm_id': '80595940',
  'boundingbox': ['45.7466148', '45.7469574', '9.2305622', '9.2319169'],
```

(continues on next page)

(continued from previous page)

```
  'lat': '45.7468391',
  'lon': '9.2310522',
  'display_name': 'Trento, Inverigo, CO, LOM, 22044, Italia',
  'class': 'highway',
  'type': 'residential',
  'importance': 0.135},
 {'place_id': '105325154',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'way',
  'osm_id': '147148070',
  'boundingbox': ['44.1482269', '44.1507838', '4.8476098', '4.850579'],
  'lat': '44.14948815',
  'lon': '4.84974947145189',
  'display_name': "Trento, La Baussenque, Orange, Carpentras, Vaucluse, Provence-
↪Alpes-Côte d'Azur, France métropolitaine, 84100, France",
  'class': 'landuse',
  'type': 'industrial',
  'importance': 0.16},
 {'place_id': '531699',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'node',
  'osm_id': '198505670',
  'boundingbox': ['8.0059463', '8.0859463', '126.0214264', '126.1014264'],
  'lat': '8.0459463',
  'lon': '126.0614264',
  'display_name': 'Trento, Agusan del Sur, Caraga, 8505, Philippines',
  'class': 'place',
  'type': 'town',
  'importance': 0.16650761976897,
  'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_place_town.p.20.png'},
 {'place_id': '43377314',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'node',
  'osm_id': '3134911289',
  'boundingbox': ['44.9456976', '44.9856976', '11.4473224', '11.4873224'],
  'lat': '44.9656976',
  'lon': '11.4673224',
  'display_name': 'Trento, RO, VEN, Italia',
  'class': 'place',
  'type': 'hamlet',
  'importance': 0.17875,
  'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_place_village.p.20.png'},
 {'place_id': '652743',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'node',
  'osm_id': '248804218',
  'boundingbox': ['46.0721752', '46.0722752', '11.1189539', '11.1190539'],
  'lat': '46.0722252',
  'lon': '11.1190039',
  'display_name': "Trento, Piazzetta Filippo Foti e Edoardo Martini, Centro storico␣
↪Trento, Trento, Territorio Val d'Adige, TN, TAA, 38122, Italia",
```

(continues on next page)

(continued from previous page)

```
  'class': 'railway',
  'type': 'stop',
  'importance': 0.18947959270135},
 {'place_id': '187948156',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'relation',
  'osm_id': '3870471',
  'boundingbox': ['7.9378151', '8.292381', '126.00048', '126.355208'],
  'lat': '8.114415',
  'lon': '126.158888603496',
  'display_name': 'Trento, Agusan del Sur, Caraga, 8505, Philippines',
  'class': 'boundary',
  'type': 'administrative',
  'importance': 0.2225,
  'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_boundary_administrative.p.20.png'},
 {'place_id': '186698302',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'relation',
  'osm_id': '46663',
  'boundingbox': ['45.9775306', '46.1530112', '11.0224735', '11.1948226'],
  'lat': '46.0664228',
  'lon': '11.1257601',
  'display_name': "Trento, Territorio Val d'Adige, TN, TAA, Italia",
  'class': 'place',
  'type': 'city',
  'importance': 0.26364591679333,
  'icon': 'http://ip-10-98-165-99.mq-us-east-1.ec2.aolcloud.net/nominatim/images/
↪mapicons/poi_place_city.p.20.png'}]
```

La API di Nominatim[352] (ricordiamo che Nominatim è il search engine di OpenStreetMap) ci permette di essere più specifici nei parametri che passiamo. Per esempio, si possono passare i parameteri `street` e `county`:

[30]:
```
def geocode_street_county(street, county):
    # 'payload' è una variabile che ci definiamo noi, per metterci più comodamente i␣
↪parametri
    payload = {'key': api_key,  # api_key è la chiave lunga tipo Er38Wk... che␣
↪abbiamo definito più sopra
               'format': 'json',
               'street' : street,
               'county' : county}
    r = requests.get(url_base, params=payload)  # qua passiamo il 'payload' alla␣
↪libreria requests
    print(r.url) # stampa l'url che requests ha usato
    return r.json()
```

[31]:
```
geocode_street_county("Via Prose n. 1", "Roncegno")
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Via+Prose+n.+1&county=Roncegno
```

[352] https://wiki.openstreetmap.org/wiki/Nominatim#Special_Keywords

[31]:
```
[{'place_id': '106260827',
  'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.
↪org/copyright',
  'osm_type': 'way',
  'osm_id': '149123476',
  'boundingbox': ['46.0464048', '46.047558', '11.4100616', '11.4119591'],
  'lat': '46.0469731',
  'lon': '11.4109589',
  'display_name': 'Via Prose, Maso Vazzena, Larganza, Roncegno Terme, Comunità␣
↪Valsugana e Tesino, TN, TAA, 38051, Italia',
  'class': 'highway',
  'type': 'residential',
  'importance': 0.525}]
```

5.4.7 4. Scriviamo un nuovo CSV con i campi lat e long

Adesso che abbiamo capito come prenderci le coordinate, possiamo

1. aprire in scrittura un nuovo CSV chiamato `agritur_mini_latlon.csv`
2. leggere il CSV originale `agritur_mini.csv`
3. per ogni riga del CSV originale `agritur_mini.csv`, leggere `lat` e `lon` da MapQuest
4. scrivere nel nuovo CSV `agritur_mini_latlon.csv` i campi vecchi più le nuove coordinate

Attenzione: Per non sovraccaricare MapQuest, negli esercizi seguenti useremo **SOLO** il file agritur_mini.csv che contiene i primi 7 Agritur. Per un paio di questi sarà possibile trovare una georeferenziazione. Il CSV 'soluzione' lo potete vedere nel file agritur_soluzione.csv (NON modificatelo !)

Intanto scriviamo un programmino per leggere dall'input `agritur_mini.csv` e copiare tutto quanto letto in un file di output che chiameremo `agritur_mini_latlon.csv`

[32]:
```
import csv

# apriamo il file `agritur_mini_latlon`in scrittura (fouput è un nome  scelto da noi)
with open('agritur_mini_latlon.csv', 'w', encoding='utf-8') as foutput:
    scrittore = csv.writer(foutput)  # Ci serve creare un'oggetto 'scrittore'

    # apriamo il file `agritur_mini`in scrittura (finput è un nome  scelto da noi)
    with open('agritur_mini.csv', encoding='utf-8', newline='') as finput:
        lettore = csv.reader(finput, delimiter=';')  # delimitatore ';'
        for riga in lettore:
            scrittore.writerow(riga)  # chiamiamo l'oggetto scrittore dicendogli di␣
↪scrivere la riga appena letta
```

⊛ **ESERCIZIO 4.1**: Copia a mano qua sotto il codice qua sopra, e usa `Control+Invio` per eseguirlo

[33]:
```
# scrivi il codice
```

⊛ **ESERCIZIO 4.2**: prova a cancellare il file `agritur_mini_latlon.csv`, eseguire la cella qua sopra e verificare che il programma effettivamente crei il file

⊛ **DOMANDA 4.3**: di default, lo `scrittore` che separatori usa? Guarda il file di risultato.

SOLUZIONE: Usa la virgola

⊛⊛ **ESERCIZIO 4.4**: Prova ad aggiungere un contatore per verificare a che riga siamo, poi tanto per capire dove sta la latitudine, prova a modificare `row` prima che venga scritta, in modo che il campo `Latitudine` (`row[16]`) e `Longitudine` (`row[17]`) siano messi rispettivamente a `123` e `456`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[34]:
```
# scrivi qui

import csv

# apriamo il file `agritur_mini_latlon`in scrittura (fouput è un nome  scelto da noi)
with open('agritur_mini_latlon.csv', 'w', encoding='utf-8') as foutput:
    scrittore = csv.writer(foutput)  # Ci serve creare un'oggetto 'scrittore'

    # apriamo il file `agritur_mini`in lettura (finput è un nome  scelto da noi)
    with open('agritur_mini.csv', encoding='utf-8', newline='') as finput:
        lettore = csv.reader(finput, delimiter=';')  # delimitatore ';'
        i = 0
        for riga in lettore:
            if i > 0:
                riga[16] = 123
                riga[17] = 456
            scrittore.writerow(riga)  #chiamiamo l'oggetto scrittore dicendogli di␣
↪scrivere la riga appena letta
            i += 1
```

[34]:
```
# scrivi qui
```

⊛⊛ **ESERCIZIO 4.5**: Mentre leggi il CSV, adesso setta latitudine e longitudine usando risultati ottenuti chiamando `geocode_street_county` definita precedentemente. Per gli input, puoi usare questi indici:

- `Indirizzo_agriturismo : riga[19]`
- `Comune_Sede_Agriturismo : riga[18]`

(ci sarebbe la `Denominazione_Agriturismo : riga[20]` ma come visto prima funziona solo se l'agriturismo è già in OpenStreetMap, usando solo la via abbiamo qualche probabilità in più di successo)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[35]:
```
import requests

def geocode_street_county(street, county):
    # 'payload' è una variabile che ci definiamo noi, per metterci più comodamente i␣
↪parametri
    payload = {'key': api_key,  # api_key è la chiave lunga tipo Er38Wk... che␣
↪abbiamo definito più sopra
               'format': 'json',
```

(continues on next page)

(continued from previous page)

```
                'street' : street,
                'county' : county}
    r = requests.get(url_base, params=payload)  # qua passiamo il 'payload' alla␣
↪libreria requests
    print(r.url) # stampa l'url che requests ha usato
    return r.json()

# scrivi qui

import csv

# apriamo il file `agritur_mini_latlon`in scrittura (fouput è un nome  scelto da noi)
with open('agritur_mini_latlon.csv', 'w', encoding='utf-8') as foutput:
    scrittore = csv.writer(foutput)  # Ci serve creare un'oggetto 'scrittore'

    # apriamo il file `agritur_mini`in lettura (finput è un nome  scelto da noi)
    with open('agritur_mini.csv', encoding='utf-8', newline='') as finput:
        lettore = csv.reader(finput, delimiter=';')  # delimitatore ';'
        i = 0
        for riga in lettore:
            if i > 0:
                json = geocode_street_county(riga[19], riga[18])
                if len(json) > 0:
                    print('trovato !')
                    riga[16] = json[0]['lat']
                    riga[17] = json[0]['lon']
            scrittore.writerow(riga)  # chiamiamo l'oggetto scrittore dicendogli di␣
↪scrivere la riga appena letta
            i += 1
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Loc.+al+Lago+n.+15&county=Garniga+Terme
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Loc.+Pineta+-+Maso+Nello&county=Faedo
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Via+Casalini+n.+74&county=Sanzeno
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Via+Prose+n.+1&county=Roncegno
trovato !
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Loc.+Fontanelle&county=Tenno
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Via+Chimelli+n.+25&county=Pergine+Valsugana
trovato !
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Via+G.+Inama+n.+21&county=Coredo
trovato !
```

[35]:

```
import requests

def geocode_street_county(street, county):
    # 'payload' è una variabile che ci definiamo noi, per metterci più comodamente i␣
↪parametri
    payload = {'key': api_key,  # api_key è la chiave lunga tipo Er38Wk... che␣
↪abbiamo definito più sopra
```

(continues on next page)

(continued from previous page)

```
                'format': 'json',
                'street' : street,
                'county' : county}
    r = requests.get(url_base, params=payload)  # qua passiamo il 'payload' alla␣
↪libreria requests
    print(r.url) # stampa l'url che requests ha usato
    return r.json()

# scrivi qui
```

```
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Loc.+al+Lago+n.+15&county=Garniga+Terme
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Loc.+Pineta+-+Maso+Nello&county=Faedo
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Via+Casalini+n.+74&county=Sanzeno
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Via+Prose+n.+1&county=Roncegno
trovato !
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Loc.+Fontanelle&county=Tenno
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Via+Chimelli+n.+25&county=Pergine+Valsugana
trovato !
http://open.mapquestapi.com/nominatim/v1/search?key=Er38WkJVmeOl5AvFIAzM6lBBq4uEdgvG&
↪format=json&street=Via+G.+Inama+n.+21&county=Coredo
trovato !
```

5.4.8 5. Importiamo in UMap

⊛ **ESERCIZIO 5.1** Importa il CSV in Umap, seguendo le indicazioni di del tutorial sui Servizi di Rovereto[353] visto precedentemente al punto 0.2

Il risultato finale dovrebbe essere così: Mappa Agritur mini[354] . Nota che abbiamo scritto `%%HTML` che è un comando speciale di Jupyter e poi incollato sotto il codice HTML ricavato da Umap:

[36]:
```
%%HTML

<iframe width="100%" height="300px" frameBorder="0" src="https://umap.openstreetmap.
↪fr/en/map/mia-mappa-agritur_182055?scaleControl=false&miniMap=false&
↪scrollWheelZoom=false&zoomControl=true&allowEdit=false&moreControl=true&
↪searchControl=null&tilelayersControl=null&embedControl=null&datalayersControl=true&
↪onLoadPanel=undefined&captionBar=false#11/46.0966/11.4024"></iframe><p><a href=
↪"http://umap.openstreetmap.fr/en/map/mia-mappa-agritur_182055">See full screen</a></
↪p>
```

```
<IPython.core.display.HTML object>
```

⊛ **ESERCIZIO 5.2**: Crea una nuova cella qua sotto, e prova ad incorporare nel foglio Jupyter la tua mappa come fatto qua sopra:

[353] https://docs.google.com/presentation/d/1CWo9pFl6jcR1EmDAXOmNeOayfyjfLqLR5-h5U8zxrrk/edit
[354] http://umap.openstreetmap.fr/en/map/agritur-mini_181977#10/46.0594/11.3805

```
[ ]:
```

5.5 Estrazione dati

5.5.1 Scarica zip esercizi

Naviga file online[355]

5.5.2 Introduzione

In questo tutorial affronteremo il tema dell'estrazione di dati semi-strutturati, focalizzandoci in particolare sull'HTML. I file in questo formato seguono (o dovrebbero seguire !) le regole dell'XML, quindi guardando dei file HTML possiamo anche imparare qualcosa sul più generico XML.

Scaletta:

0. Impariamo due cose di HTML creando una paginetta web seguendo un semplice tutorial CoderDojoTrento (senza Python).
1. Chiedersi se vale la pena estrarre informazioni dall'HTML
2. Estrazione eventi del Trentino da visittrentino.info[356] usando Python e BeautifulSoup 4
3. Per ogni evento, estrarremo nome, data, luogo, tipo, descrizione e li metteremo in una lista di dizionari Python così:

```
[{'data': '14/12/2017',
  'descrizione': 'Al Passo Costalunga sfida tra i migliori specialisti del mondo',
  'luogo': 'Passo Costalunga',
  'nome': 'Coppa del Mondo di Snowboard',
  'tipo': 'Sport, TOP EVENTI SPORT'},
 {'data': '18/12/2017',
  'descrizione': 'Lunedì 18 dicembre la Coppa Europa fa tappa in Val di Fassa',
  'luogo': 'Pozza di Fassa',
  'nome': 'Coppa Europa di sci alpino maschile - slalom speciale',
  'tipo': 'Sport, TOP EVENTI SPORT'},
  ....
]
```

4. Infine scriveremo un file CSV `eventi.csv` usando la lista di dizionari generata al punto precedente.

[355] https://github.com/DavidLeoni/softpython-it/tree/master/extraction
[356] https://www.visittrentino.info/it/guida/eventi

Che fare

1. Per avere un'idea di cosa sia l'HTML, prova a crearti una paginetta web seguendo il tutorial 1[357] di CoderDojoTrento (in questa parte non serve Python, puoi fare tutto online saltando la registrazione su Thimble)
2. Adesso puoi passare ad usare Python e Jupyter come al solito: scarica lo zip con esercizi e soluzioni

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
extraction
    extraction.ipynb
    extraction-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `extraction.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

5.5.3 1. Guardiamo l'html

Indipendentemente dalla domanda posta al punto precedente, per fini didattici proseguiremo con lo *scraping* html.

All'interno dello zip degli esercizi, c'è un file chiamato eventi.html. Dopo aver scompattato lo zip, apri il file nel tuo browser.

IMPORTANTE: usa solo il file eventi.html nella cartella del progetto Jupyter!

Il file eventi.html è stato salvato nel 2017 ed è un po' più brutto da vedere della versione online sul sito di visttrentino (mancano immagini, etc). Se sei curioso di provare quella online[358], per vedere gli eventi come lista ricordati di cliccare sulla relativa icona:

[357] http://coderdojotrento.it/web1

⊛ **1.1 ESERCIZIO**: Dopo aver aperto il file nel browser, visualizza l'HTML all'interno (premendo per es `Ctrl+U` in Firefox/Chrome/Safari). Familiarizzati un po' con il file sorgente, cercando all'interno alcuni valori del primo evento , come per esempio il nome `Coppa del Mondo di Snowboard`, la data `14/12/2017`, il luogo `Passo Costalunga`, il tipo `Sport, TOP EVENTI SPORT`, e la descrizione `Al Passo Costalunga sfida tra i migliori specialisti del mondo`.

NOTA 1: Per questo esercizio e tutti i seguenti, usa solo il file eventi.html indicato, NON usare l'html dalla versione live[359] proveniente dalla di visittrentino, che può essere soggetta a cambiamenti !

NOTA 2: Evita di usare il browser Internet Explorer, in alternativa cerca di usare uno tra i seguenti (in quest'ordine): Firefox, Chrome, Safari.

Per maggiori informazioni sul formato XML, di cui l'HTML è una incarnazione, puoi consultare il libro Immersione in Python al capitolo 12[360],

[358] https://www.visittrentino.info/it/guida/eventi
[359] https://www.visittrentino.info/it/guida/eventi
[360] http://gpiancastelli.altervista.org/dip3-it/xml.html

5.5.4 2. Estrazione con BeautifulSoup

Installiamo le librerie `beautifulsoup4` e il parser `lxml`:

- Anaconda:
  - `conda install beautifulsoup4`
  - `conda install lxml`
- Linux/Mac (`--user` installa nella propria home):
  - `python3 -m pip install --user beautifulsoup4`
  - `python3 -m pip install --user lxml`

I parser

Quanto installato sopra è quanto basta per questo esercizio. In particolare abbiamo installato il parser `lxml`, che è il componente che permette a BeautifulSoup di leggere l'HTML in modo veloce e *lenient*, cioè tollerando possibili errori di formattazione che potrebbero essere presenti nell'HTML (NOTA: la maggior parte dei documenti html nel mondo reale ha problemi di formattazione !). Prendendo pagine a caso da internet, `lxml` potrebbe non essere adatto. Se hai problemi a leggere pagine html, potresti provare a sostituire `lxml` con `html5lib`, o altri parser. Per una lista di possibili parser, vedere la documentazione di BeautifulSoup[361]

2.1 Estraiamo i nomi

⊛ **ESERCIZIO 2.1.1** Cerca nell'HTML la stringa `Coppa del Mondo di Snowboard`, cercando di capire bene in quali blocchi appare.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

Compare in 4 blocchi: h2, script Javascript, script semantici `ld+json` e blocchi `h4`, li ricapitoliamo qua sotto:

RISPOSTA 2.1.1 - blocchi h2 : Compare in blocchi `h2` come `<h2 class="text-secondary">Coppa del Mondo di Snowboard</h2>` qua:

```
<div class="col col-sm-9 arrange__item">
                    <a href="https://www.visittrentino.info/it/guida/eventi/coppa-del-
↪mondo-di-snowboard_e_165375" class="text-primary list-teaser__link"><span class=
↪"icon icon-circle-arrow fz30"></span></a>
                    <div class="teaser__body">
                                                        <span class="text-secondary fz14␣
↪text-uppercase strong">Sport, TOP EVENTI SPORT</span>
                                                    <h2 class="text-secondary">Coppa del␣
↪Mondo di Snowboard</h2>
                        <ul class="list-unstyled list-inline list__teaser__list mb15␣
↪mt10">
                                                                <li>
                                    <a class="fz14 text-uppercase strong text-primary
↪"><span class="icon icon-map-view mr10"></span>Passo Costalunga</a>
                                </li>
                                                            <li>
                                <a class="fz14 text-uppercase strong text-primary">
↪<span class="icon icon-map-view mr10"></span>14/12/2017</a>
```

(continues on next page)

[361] https://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser

(continued from previous page)

```
                            </li>
                        </ul>
                        <p>Al Passo Costalunga sfida tra i migliori specialisti del␣
↪mondo</p>
                    </div>
                </div>
```

RISPOSTA 2.1.1 - script Javascript: Compare in blocchi script Javascript come qua alla riga `class=\"fz24 text-blue\">Coppa del Mondo di Snowboard<\/`:

```
<script>
    var mapData = [{"id":165375,"lat":46.4021014,"lng":11.6110718,"poiIcon":"\/static\
↪/img\/content\/pois\/poi.png","poiIconActive":"\/static\/img\/content\/pois\/poi-
↪active.png","infoBox":"<div class=\"row\"><div class=\"col-xs-5\"><img src=\"\/
↪website\/var\/tmp\/image-thumbnails\/0\/9500\/thumb__moodboardmap\/snowparks-and-
↪snowboard-brenta-dolomites_1.jpeg\"><\/div><div class=\"col-xs-7\"><h2 class=\"fz24␣
↪text-blue\">Coppa del Mondo di Snowboard<\/h2><p class=\"fz14\">14\/12 A Passo␣
↪Costalunga sfida tra i migliori specialisti del mondo<\/p><a href=\"\/it\/guida\/
↪eventi\/coppa-del-mondo-di-snowboard_e_165375\" class=\"btn btn-primary\">Maggiori␣
↪info<\/a><\/div><\/div>"},{"id":309611,"lat":46.403707118232,"lng":11.60915851593,
↪"poiIcon":"\/static\/img\/content\/pois\/poi.png","poiIconActive":"\/static\/img\/
↪content\/pois\/poi-active.png","infoBox":"<div class=\"row\"><div class=\"col-xs-5\
↪"><img src=\"\/website\/var\/tmp\/image-thumbnails\/110000\/119958\/thumb__
↪moodboardmap\/foto-3_2.jpeg\"><\/div><div class=\"col-xs-7\"><h2 class=\"fz24 text-
↪blue\">Coppa Europa di sci alpino maschile - slalom speciale<\/h2><p class=\"fz14\">
↪18 dicembre in Val di Fassa<\/p><a href=\"\/it\/guida\/eventi\/coppa-europa-di-sci-
↪alpino-maschile-slalom-speciale_e_309611\" class=\"btn btn-primary\">Maggiori info<\
↪/a><\/div><\/div>"},

....
....
</script>
```

RISPOSTA 2.1.1 - blocchi script ld+json: Compare in altri blocchi script semantici jsonld nella parte `"name": "Coppa del Mondo di Snowboard"` come qua. Questi dati sono intesi per le macchine (come il motore di ricerca Google quando indicizza il contenuto delle pagine), quindi sarebbero particolarmente appetibili per i nostri scopi, ma per questo esercizio li ignoreremo:

```
 <script type="application/ld+json">
{"@context":"http://schema.org","@type":"Event","name":"Coppa del Mondo di Snowboard",
↪"description":"Al Passo Costalunga sfida tra i migliori specialisti del mondo",
↪"startDate":"2017-12-14T00:00:00+01:00","endDate":"2017-12-14T00:00:00+01:00",
↪"location":{"@type":"Place","name":"Passo Costalunga","address":{"@type":
↪"PostalAddress","addressCountry":"Italy","postalCode":"39056","streetAddress":""}},
↪"image":{"@type":"ImageObject","url":"https://www.visittrentino.info/website/var/
↪tmp/image-thumbnails/0/9500/thumb__contentgallery/snowparks-and-snowboard-brenta-
↪dolomites_1.jpeg","height":396,"width":791},"performer":"Trentino Marketing S.r.l.",
↪"url":"https://www.visittrentino.info/it/guida/eventi/coppa-del-mondo-di-snowboard_
↪e_165375","offers":{"@type":"AggregateOffer","lowPrice":"\u20ac0","offerCount":"000
↪","url":"https://www.visittrentino.info/it/guida/eventi/coppa-del-mondo-di-
↪snowboard_e_165375"}}
</script>
```

Cerchiamo con Python

Se hai fatto l'esercizio precedente, dovresti aver trovato vari blocchi che contengono `Coppa del Mondo di Snow-`

board. Per un elenco e discussione, *guarda la soluzione* Per questo esercizio, considereremo principalmente i blocchi h4, in cui il nome Coppa del Mondo di Snowboard compare in questa riga:

```
<h4 class="moodboard__item-headline">Coppa del Mondo di Snowboard</h4>
```

vediamo cosa c'è intorno:

```
<div class="moodboard__item-text text-white">
    <div>
        <h4 class="moodboard__item-headline">Coppa del Mondo di Snowboard</h4>
                    <span class="moodboard__item-subline strong fz14 text-uppercase d-
↪b">14/12/2017</span>
                            <span class="moodboard__item-subline strong fz14 d-b">
↪<span class="icon icon-map-view fz20"></span> Passo Costalunga</span>
            </div>
    <div class="moodboard__item-text__link text-right">
        <a href="https://www.visittrentino.info/it/guida/eventi/coppa-del-mondo-di-
↪snowboard_e_165375" role="link"><span class="icon icon-circle-arrow fz30"></span></
↪a>
    </div>
</div>
```

Torniamo alla riga:

```
<h4 class="moodboard__item-headline">Coppa del Mondo di Snowboard</h4>
```

Notiamo che:

- inizia con il tag <h4>
- finisce simmetricamente con il tag di chiusura </h4>
- il tag di apertura ha un parametro class="moodboard__item-headline", ma al momento non ci interessa
- Il testo che cerchiamo Coppa del Mondo di Snowboard è incluso tra i due tag

Per estrarre solo i nomi degli eventi dal documento, possiamo quindi cercare i tag h4. Eseguiamo la nostra prima estrazione in Python:

[1]:
```
# Importa l'oggetto BeatifulSoup dal modulo bs4:
from bs4 import BeautifulSoup

# apriamo il file degli eventi:
# - specifichiamo l'encoding come 'utf-8' ATTENZIONE: MAI DIMENTICARE L'ENCODING !!
# - lo chiamiamo con f, un nome che scegliamo noi

with open("eventi.html", encoding='utf-8') as f:
    soup = BeautifulSoup(f, "lxml")     # creiamo un oggetto che chiamiamo 'soup',␣
↪usando il parser lxml
# soup ci permette di chiamare il metodo select, per selezionare per esempio solo tag
↪'h4':
# il metodo ritorna una lista di tag trovate nel documento. Per ogni tag ritornata,␣
↪la stampiamo:
for tag in soup.select("h4"):
    print(tag)
```

```
<h4 class="moodboard__item-headline">Coppa del Mondo di Snowboard</h4>
<h4 class="moodboard__item-headline">Coppa Europa di sci alpino maschile - slalom␣
↪speciale</h4>
```

(continues on next page)

(continued from previous page)

```
<h4 class="moodboard__item-headline">3Tre - AUDI FIS Ski World Cup</h4>
<h4 class="moodboard__item-headline">La Marcialonga di Fiemme e Fassa </h4>
<h4 class="moodboard__item-headline">TrentinoSkiSunrise: sulle piste alla luce␣
↪dell'alba</h4>
<h4 class="moodboard__item-headline">Tour de Ski</h4>
<h4 class="moodboard__item-headline">La mia nuvola</h4>
<h4 class="moodboard__item-headline">Dormire sotto un cielo di stelle</h4>
<h4 class="moodboard__item-headline">Mercatini di Natale di Canale di Tenno e Rango </
↪h4>
<h4 class="moodboard__item-headline">La Ciaspolada</h4>
<h4 class="moodboard__item-headline">Mercatini di Natale a Trento</h4>
<h4 class="moodboard__item-headline">Mercatini di Natale di Rovereto</h4>
<h4 class="moodboard__item-headline">Mercatino di Natale asburgico di Levico Terme</
↪h4>
<h4 class="moodboard__item-headline">Magnifico Mercatino di Cavalese</h4>
<h4 class="moodboard__item-headline">Mercatino di Natale ad Arco</h4>
<h4 class="moodboard__item-headline">El paès dei presepi</h4>
```

Benissimo, abbiamo filtrato le tag coi nomi degli eventi. Ma noi vogliamo solo i nomi. Tipicamente dalle nostre tag ci interessa estrarre solo il testo. A tal fine. possiamo usare l'attributo `text` delle tag:

[2]:
```
for tag in soup.select("h4"):
    print(tag.text)
```

```
Coppa del Mondo di Snowboard
Coppa Europa di sci alpino maschile - slalom speciale
3Tre - AUDI FIS Ski World Cup
La Marcialonga di Fiemme e Fassa
TrentinoSkiSunrise: sulle piste alla luce dell'alba
Tour de Ski
La mia nuvola
Dormire sotto un cielo di stelle
Mercatini di Natale di Canale di Tenno e Rango
La Ciaspolada
Mercatini di Natale a Trento
Mercatini di Natale di Rovereto
Mercatino di Natale asburgico di Levico Terme
Magnifico Mercatino di Cavalese
Mercatino di Natale ad Arco
El paès dei presepi
```

Casomai volessimo, si può anche estrarre un attributo, per esempio la `class`

[3]:
```
for tag in soup.select('h4'):
    print(tag['class'])
```

```
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
```

(continues on next page)

(continued from previous page)

```
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
['moodboard__item-headline']
```

La lista nomi

Intanto, cerchiamo di mettere tutti i nomi in una lista che chiameremo `nomi`

[4]:
```
nomi = []
for tag in soup.select("h4"):
    nomi.append(tag.text)
```

[5]:
```
nomi
```

[5]:
```
['Coppa del Mondo di Snowboard',
 'Coppa Europa di sci alpino maschile - slalom speciale',
 '3Tre - AUDI FIS Ski World Cup',
 'La Marcialonga di Fiemme e Fassa ',
 'TrentinoSkiSunrise: sulle piste alla luce dell'alba',
 'Tour de Ski',
 'La mia nuvola',
 'Dormire sotto un cielo di stelle',
 'Mercatini di Natale di Canale di Tenno e Rango ',
 'La Ciaspolada',
 'Mercatini di Natale a Trento',
 'Mercatini di Natale di Rovereto',
 'Mercatino di Natale asburgico di Levico Terme',
 'Magnifico Mercatino di Cavalese',
 'Mercatino di Natale ad Arco',
 'El paès dei presepi']
```

Una struttura dati per il CSV

Cominciamo a costruirci la struttura dati che modellerà il CSV che vogliamo creare. Un CSV può essere visto come una lista di dizionari. Ogni dizionario rappresenterà un evento. Passo passo, vorremmo arrivare ad avere una forma simile:

```
[{'data': '14/12/2017',
  'descrizione': 'Al Passo Costalunga sfida tra i migliori specialisti del mondo',
  'luogo': 'Passo Costalunga',
  'nome': 'Coppa del Mondo di Snowboard',
  'tipo': 'Sport, TOP EVENTI SPORT'},
 {'data': '18/12/2017',
  'descrizione': 'Lunedì 18 dicembre la Coppa Europa fa tappa in Val di Fassa',
  'luogo': 'Pozza di Fassa',
  'nome': 'Coppa Europa di sci alpino maschile - slalom speciale',
  'tipo': 'Sport, TOP EVENTI SPORT'},
  ....
]
```

Inizializziamo la lista delle righe:

[6]:
```
righe = []
```

[7]:
```
righe
```

[7]:
```
[]
```

Dobbiamo aggiungere dei dizionari vuoti, ma quanti ce ne servono ? Vediamo quanti titoli abbiamo trovato con la funzione `len`:

[8]:
```
len(nomi)
```

[8]:
```
16
```

Procediamo quindi ad aggiungere 16 dizionari aggiungendo un dizionario vuoto alla volta a `rows` con la funzione `append` :

[9]:
```
for i in range(16):
    righe.append({})
```

[10]:
```
righe
```

[10]:
```
[{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}]
```

Popoliamo i dizionari con i nomi

Adesso, in ogni dizionario, mettiamo un campo 'nome' usando il valore corrispendente trovato nella lista `nomi`:

[11]:
```
i = 0
for stringa in nomi:
    righe[i]['nome'] = stringa  # metti stringa nella riga i-esima, al campo 'nome' ␣
↪del dizionario
    i += 1
```

Dovremmo cominciare a vedere i dizionari in `righe` popolati con i nomi. Verifichiamo:

[12]:
```
righe
```

[12]:
```
[{'nome': 'Coppa del Mondo di Snowboard'},
 {'nome': 'Coppa Europa di sci alpino maschile - slalom speciale'},
 {'nome': '3Tre - AUDI FIS Ski World Cup'},
 {'nome': 'La Marcialonga di Fiemme e Fassa '},
 {'nome': 'TrentinoSkiSunrise: sulle piste alla luce dell'alba'},
 {'nome': 'Tour de Ski'},
 {'nome': 'La mia nuvola'},
 {'nome': 'Dormire sotto un cielo di stelle'},
 {'nome': 'Mercatini di Natale di Canale di Tenno e Rango '},
 {'nome': 'La Ciaspolada'},
 {'nome': 'Mercatini di Natale a Trento'},
 {'nome': 'Mercatini di Natale di Rovereto'},
 {'nome': 'Mercatino di Natale asburgico di Levico Terme'},
 {'nome': 'Magnifico Mercatino di Cavalese'},
 {'nome': 'Mercatino di Natale ad Arco'},
 {'nome': 'El paès dei presepi'}]
```

La funzione `aggiungi_campo`

Visto che i campi da aggiungere saranno diversi, definiamoci una funzione per aggiugere comodamente i campi alla variabile `righe`:

[13]:
```
# 'attributo' potrebbe essere la stringa 'nome'
# 'lista' potrebbe essere la lista dei nomi degli eventi ['Coppa del Mondo di␣
↪Snowboard', 'Coppa Europa', ...]
def aggiungi_campo(attributo, lista):
    i = 0
    for stringa in lista:
        righe[i][attributo] = stringa
        i += 1
    return righe   # ritorniamo righe per stampare immediatamente il risultato in␣
↪Jupyter
```

Chiamando `aggiungi_campo` con i nomi, non dovrebbe cambiare nulla perchè andremo semplicemente a riscrivere i campi `nome` dentro i dizionari della lista `righe` :

[14]:
```
aggiungi_campo('nome', nomi)
```

[14]:
```
[{'nome': 'Coppa del Mondo di Snowboard'},
 {'nome': 'Coppa Europa di sci alpino maschile - slalom speciale'},
 {'nome': '3Tre - AUDI FIS Ski World Cup'},
 {'nome': 'La Marcialonga di Fiemme e Fassa '},
 {'nome': 'TrentinoSkiSunrise: sulle piste alla luce dell'alba'},
 {'nome': 'Tour de Ski'},
 {'nome': 'La mia nuvola'},
 {'nome': 'Dormire sotto un cielo di stelle'},
 {'nome': 'Mercatini di Natale di Canale di Tenno e Rango '},
 {'nome': 'La Ciaspolada'},
 {'nome': 'Mercatini di Natale a Trento'},
 {'nome': 'Mercatini di Natale di Rovereto'},
 {'nome': 'Mercatino di Natale asburgico di Levico Terme'},
 {'nome': 'Magnifico Mercatino di Cavalese'},
 {'nome': 'Mercatino di Natale ad Arco'},
 {'nome': 'El paès dei presepi'}]
```

2.2 Estraiamo le date

Bene, è tempo di aggiungere un altro campo, per esempio per la data. Dove trovarlo?

⊛ **DOMANDA 2.2.1**: Cerca dentro l'HTML il valore della data della Coppa del Mondo di Snowboard `14/12/2017`. Quanti valori trovi? Intorno alle date, trovi del testo che potrebbero permetterci di filtrare tutte e sole le date ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Guardiamo l'HTML intorno alle posizioni `h4`:

```
<div class="moodboard__item-text text-white">
    <div>
        <h4 class="moodboard__item-headline">Coppa del Mondo di Snowboard</h4>
                    <span class="moodboard__item-subline strong fz14 text-uppercase d-
↪b">14/12/2017</span>
                            <span class="moodboard__item-subline strong fz14 d-b">
↪<span class="icon icon-map-view fz20"></span> Passo Costalunga</span>
```

(continues on next page)

(continued from previous page)

```
            </div>
    <div class="moodboard__item-text__link text-right">
        <a href="https://www.visittrentino.info/it/guida/eventi/coppa-del-mondo-di-
↪snowboard_e_165375" role="link"><span class="icon icon-circle-arrow fz30"></span></
↪a>
    </div>
</div>
```

Vediamo che le date sono dentro attributi `span`, questi attributi semplicemente indicano una sequenza di testo.

⊛ **ESERCIZIO 2.2.2**: Prova a filtrare qui sotto in Python le tag `span`, stampando i risultati. Trovi solo le date ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[15]:
```
# scrivi qui il codice

#  Le span sono troppe!

#for tag in soup.select("span"):
#    print(tag)
```

[15]:
```
# scrivi qui il codice
```

Filtri più selettivi

Noi dobbiamo filtrare solo le tag `span` che contengono l'attributo `class` con un lungo valore criptico:

```
class="moodboard__item-subline strong fz14 text-uppercase d-b"
```

NOTA: Non farti spaventare da strani nomi che non conosci. Quando si traffica nei file HTML, si può trovare di tutto e bisogna 'navigare a vista'. Per fortuna, spesso non occorre sapere troppi dettagli tecnici per estrarre testo rilevante.

Per estrarre le nostre date, possiamo sfruttare il fatto che alla funzione `soup.select` possiamo passare non solo tag ma qualunque espressione di selezione CSS[362]

Tra le prime ne troviamo una che ci dice che possiamo scrivere l'attributo e il valore dentro a parentesi quadre dopo il nome della tag, come qua:

[16]:
```
for tag in soup.select('span[class="moodboard__item-subline strong fz14 text-
↪uppercase d-b"]'):
    print(tag.text)
```

[362] https://www.mrwebmaster.it/css/selettori-css3_11011.html

```
14/12/2017
18/12/2017
22/12/2017
28/01/2018
06/01/2018 - 07/04/2018
06/01/2018 - 07/01/2018
01/10/2017 - 30/11/2017
31/10/2017 - 20/12/2017
19/11/2017 - 30/12/2017
06/01/2018
18/11/2017 - 06/01/2018
25/11/2017 - 06/01/2018
25/11/2017 - 06/01/2018
01/12/2017 - 06/01/2018
17/11/2017 - 07/01/2018
08/12/2017 - 07/01/2018
```

Ecco le nostre date! In questo caso particolare, il `"moodboard__item-subline strong fz14 text-uppercase d-b"` è così identificativo che non serve nemmeno specificare `span`:

[17]:
```
            # nota che 'span' è rimosso:
for tag in soup.select('[class="moodboard__item-subline strong fz14 text-uppercase d-b
↪"]'):
    print(tag.text)
```

```
14/12/2017
18/12/2017
22/12/2017
28/01/2018
06/01/2018 - 07/04/2018
06/01/2018 - 07/01/2018
01/10/2017 - 30/11/2017
31/10/2017 - 20/12/2017
19/11/2017 - 30/12/2017
06/01/2018
18/11/2017 - 06/01/2018
25/11/2017 - 06/01/2018
25/11/2017 - 06/01/2018
01/12/2017 - 06/01/2018
17/11/2017 - 07/01/2018
08/12/2017 - 07/01/2018
```

⊛ **DOMANDA 2.2.3**: possiamo scrivere direttamente senza nient'altro la stringa `moodboard__item-subline strong fz14 text-uppercase d-b` nella `select` ? Scrivi qui sotto i tuoi esperimenti.

[18]:
```
# scrivi qui
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: No, le regole CSS sono abbastanza rigide e non consentono ricerca di testo libero.

Come fatto prima, possiamo crearci una lista per le date:

[19]:
```
date_eventi = []
for tag in soup.select('[class="moodboard__item-subline strong fz14 text-uppercase d-b
↪"]'):
    date_eventi.append(tag.text)
```

[20]:
```
date_eventi
```

[20]:
```
['14/12/2017',
 '18/12/2017',
 '22/12/2017',
 '28/01/2018',
 '06/01/2018 - 07/04/2018',
 '06/01/2018 - 07/01/2018',
 '01/10/2017 - 30/11/2017',
 '31/10/2017 - 20/12/2017',
 '19/11/2017 - 30/12/2017',
 '06/01/2018',
 '18/11/2017 - 06/01/2018',
 '25/11/2017 - 06/01/2018',
 '25/11/2017 - 06/01/2018',
 '01/12/2017 - 06/01/2018',
 '17/11/2017 - 07/01/2018',
 '08/12/2017 - 07/01/2018']
```

Per sincerarci di aver rastrellato tutte le date necessarie, possiamo controllare quante sono:

[21]:
```
len(date_eventi)
```

[21]:
```
16
```

Adesso possiamo sfruttare la funzione di prima `aggiungi_campo` per aggiornare `righe`:

[22]:
```
aggiungi_campo('data', date_eventi)
```

[22]:
```
[{'nome': 'Coppa del Mondo di Snowboard', 'data': '14/12/2017'},
 {'nome': 'Coppa Europa di sci alpino maschile - slalom speciale',
  'data': '18/12/2017'},
 {'nome': '3Tre - AUDI FIS Ski World Cup', 'data': '22/12/2017'},
 {'nome': 'La Marcialonga di Fiemme e Fassa ', 'data': '28/01/2018'},
 {'nome': 'TrentinoSkiSunrise: sulle piste alla luce dell'alba',
  'data': '06/01/2018 - 07/04/2018'},
 {'nome': 'Tour de Ski', 'data': '06/01/2018 - 07/01/2018'},
 {'nome': 'La mia nuvola', 'data': '01/10/2017 - 30/11/2017'},
 {'nome': 'Dormire sotto un cielo di stelle',
  'data': '31/10/2017 - 20/12/2017'},
 {'nome': 'Mercatini di Natale di Canale di Tenno e Rango ',
  'data': '19/11/2017 - 30/12/2017'},
 {'nome': 'La Ciaspolada', 'data': '06/01/2018'},
 {'nome': 'Mercatini di Natale a Trento', 'data': '18/11/2017 - 06/01/2018'},
 {'nome': 'Mercatini di Natale di Rovereto',
  'data': '25/11/2017 - 06/01/2018'},
 {'nome': 'Mercatino di Natale asburgico di Levico Terme',
  'data': '25/11/2017 - 06/01/2018'},
 {'nome': 'Magnifico Mercatino di Cavalese',
  'data': '01/12/2017 - 06/01/2018'},
 {'nome': 'Mercatino di Natale ad Arco', 'data': '17/11/2017 - 07/01/2018'},
 {'nome': 'El paès dei presepi', 'data': '08/12/2017 - 07/01/2018'}]
```

2.3 Estraiamo i luoghi

E' tempo di aggiungere il luogo.

⊛⊛ **ESERCIZIO 2.3.1**: Cerca nell'HTML il luogo `Passo Costalunga` dell'evento Coppa di Snowboard. Con quale criterio possiamo filtrare i luoghi ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

Possiamo usare la classe `class="moodboard__item-subline strong fz14 d-b"` nelle righe:

```
<div class="moodboard__item-text text-white">
    <div>
        <h4 class="moodboard__item-headline">Coppa del Mondo di Snowboard</h4>
                    <span class="moodboard__item-subline strong fz14 text-uppercase d-
↪b">14/12/2017</span>
                            <span class="moodboard__item-subline strong fz14 d-b">
↪<span class="icon icon-map-view fz20"></span> Passo Costalunga</span>
            </div>
    <div class="moodboard__item-text__link text-right">
        <a href="https://www.visittrentino.info/it/guida/eventi/coppa-del-mondo-di-
↪snowboard_e_165375" role="link"><span class="icon icon-circle-arrow fz30"></span></
↪a>
    </div>
</div>
```

⊛ **ESERCIZIO 2.3.2**: Scrivi il codice Python per estrarre i luoghi e metterlo nella lista `luoghi` (usa sempre l'html intorno ai blocchi `h4`)

SUGGERIMENTO: Non importa se il testo che cerchi è contenuto in uno span all'interno di un'altro span identificabile

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
# scrivi qui

for tag in soup.select('[class="moodboard__item-subline strong fz14 d-b"]'):
    print(tag.text)
luoghi = []
for tag in soup.select('[class="moodboard__item-subline strong fz14 d-b"]'):
    luoghi.append(tag.text)

luoghi
```

```
Passo Costalunga
Pozza di Fassa
Madonna di Campiglio
Val di Fassa
Tesero
Pozza di Fassa
Pozza di Fassa
Tenno
Fondo
```

(continues on next page)

(continued from previous page)

```
Trento
Rovereto
Levico Terme
Cavalese
Arco
Baselga di Piné
```

[23]:
```
[' Passo Costalunga',
 ' Pozza di Fassa',
 ' Madonna di Campiglio',
 ' Val di Fassa ',
 ' Tesero',
 ' Pozza di Fassa',
 ' Pozza di Fassa',
 ' Tenno',
 ' Fondo',
 ' Trento',
 ' Rovereto',
 ' Levico Terme',
 ' Cavalese',
 ' Arco',
 ' Baselga di Piné']
```

[23]:
```
# scrivi qui
```

```
Passo Costalunga
Pozza di Fassa
Madonna di Campiglio
Val di Fassa
Tesero
Pozza di Fassa
Pozza di Fassa
Tenno
Fondo
Trento
Rovereto
Levico Terme
Cavalese
Arco
Baselga di Piné
```

[23]:
```
[' Passo Costalunga',
 ' Pozza di Fassa',
 ' Madonna di Campiglio',
 ' Val di Fassa ',
 ' Tesero',
 ' Pozza di Fassa',
 ' Pozza di Fassa',
 ' Tenno',
 ' Fondo',
 ' Trento',
 ' Rovereto',
 ' Levico Terme',
 ' Cavalese',
```

(continues on next page)

(continued from previous page)

```
 ' Arco',
 ' Baselga di Piné']
```

2.3 Sistemiamo i luoghi

⊛ **DOMANDA 2.3.3**: Quanti luoghi hai trovato ? Dovrebbero essere 16, ma se ne hai trovato 15 allora manca un dato! Che dato è? Guarda la pagina e cerca se qualche evento non ha un luogo. In che posizione è nella lista ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: L'evento incriminato è `TrentinoSkiSunrise: sulle piste alla luce dell'alba`, guardando dentro l'html vediamo che manca proprio lo `span` con la classe:

```
<div class="moodboard__item-text text-white">
    <div>
        <h4 class="moodboard__item-headline">TrentinoSkiSunrise: sulle piste alla␣
↪luce dell'alba</h4>
                    <span class="moodboard__item-subline strong fz14 text-uppercase d-
↪b">06/01/2018 - 07/04/2018</span>
                    </div>
    <div class="moodboard__item-text__link text-right">
        <a href="https://www.visittrentino.info/it/guida/eventi/trentinoskisunrise-
↪sulle-piste-alla-luce-dell-alba_e_318133" role="link"><span class="icon icon-circle-
↪arrow fz30"></span></a>
    </div>
</div>
```

⊛ **ESERCIZIO 2.3.4**: Come rimediare al problema? Ci sono metodo più furbi, ma per stavolta semplicemente possiamo inserire una stringa vuota alla posizione della lista `luoghi` in cui ci dovrebbe essere il campo mancante - e cioè dopo `Val di Fassa`. Per farlo, puoi usare il metodo `insert`. Qua puoi vedere qualche esempio di utilizzo.

NOTA: Ricordati che gli indici iniziano da 0 !

[24]:
```
prova = ['a','b','c','d']
prova.insert(0, 'x') # inserisce 'x' all'inizio in posizione 0
prova
```

[24]:
```
['x', 'a', 'b', 'c', 'd']
```

[25]:
```
prova = ['a','b','c','d']
prova.insert(2, 'x') # inserisce 'x' a metà
prova
```

[25]:
```
['a', 'b', 'x', 'c', 'd']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[26]:
```
# scrivi qui la soluzione

luoghi.insert(4, '')
luoghi
```

[26]:
```
[' Passo Costalunga',
 ' Pozza di Fassa',
 ' Madonna di Campiglio',
 ' Val di Fassa ',
 '',
 ' Tesero',
 ' Pozza di Fassa',
 ' Pozza di Fassa',
 ' Tenno',
 ' Fondo',
 ' Trento',
 ' Rovereto',
 ' Levico Terme',
 ' Cavalese',
 ' Arco',
 ' Baselga di Piné']
```

[26]:
```
# scrivi qui la soluzione
```

[26]:
```
[' Passo Costalunga',
 ' Pozza di Fassa',
 ' Madonna di Campiglio',
 ' Val di Fassa ',
 '',
 ' Tesero',
 ' Pozza di Fassa',
 ' Pozza di Fassa',
 ' Tenno',
 ' Fondo',
 ' Trento',
 ' Rovereto',
 ' Levico Terme',
 ' Cavalese',
 ' Arco',
 ' Baselga di Piné']
```

⊛⊛ **ESERCIZIO 2.3.5**: Prima di procedere, verifica che la lunghezza della lista `len(luoghi)` sia corretta (= 16). Adesso, osserva attentamente le stringhe dei luoghi. Dovresti vedere che iniziano tutte con uno spazio inutile: sistema la lista `luoghi` togliendo gli spazi extra usando una *list comprehension* (vedere Capitolo 19.2 Pensare in Python[363]). Per eliminare gli spazi puoi usare il metodo delle stringhe `.strip()`

NOTA: quando crei una *list comprehension*, ne generi una nuova, la lista originale *non* viene modificata!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[27]:
```
# scrivi qui

luoghi = [luogo.strip() for luogo in luoghi]
```

[363] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2020.html#sec227

[27]:
```
# scrivi qui
```

2.4 Tipo dell'evento

Proseguiamo con il tipo dell'evento. Per esempio, per la Coppa del Mondo di Snowboard, la tipologia dell'evento sarebbe `Sport, TOP EVENTI SPORT`

⊛ **ESERCIZIO 2.4.1**: Cerca la stringa `Sport, TOP EVENTI SPORT` nell'HTML. Quante occorrenze ci sono ? E' possibile filtrare il tipo eventi in modo univoco ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

Questa volta il tipo eventi appare solo nei blocchi intorno agli `h2`, in particolare notiamo la riga

`<span class="text-secondary fz14 text-uppercase strong">Sport, TOP EVENTI SPORT</span>`:

```
<div class="col col-sm-9 arrange__item">
                    <a href="https://www.visittrentino.info/it/guida/eventi/coppa-del-
↪mondo-di-snowboard_e_165375" class="text-primary list-teaser__link"><span class=
↪"icon icon-circle-arrow fz30"></span></a>
                    <div class="teaser__body">
                                                        <span class="text-secondary fz14␣
↪text-uppercase strong">Sport, TOP EVENTI SPORT</span>
                                                    <h2 class="text-secondary">Coppa del␣
↪Mondo di Snowboard</h2>
                        <ul class="list-unstyled list-inline list__teaser__list mb15␣
↪mt10">
                                                                <li>
                                    <a class="fz14 text-uppercase strong text-primary
↪"><span class="icon icon-map-view mr10"></span>Passo Costalunga</a>
                                </li>
                                                            <li>
                                <a class="fz14 text-uppercase strong text-primary">
↪<span class="icon icon-map-view mr10"></span>14/12/2017</a>
                            </li>
                        </ul>
                        <p>Al Passo Costalunga sfida tra i migliori specialisti del␣
↪mondo</p>
                    </div>
                </div>
```

⊛⊛ **ESERCIZIO 2.4.2**: Scrivi il codice Python per estrarre la lista degli eventi, e chiamala `tipo_evento`. Per farlo, puoi usare il codice già visto in precedenza, ma questa volta, per creare la lista sforzati di usare una *list comprehension*. Verifica poi che la lunghezza della stringa sia 16.

SUGGERIMENTO: La lista originale da cui prelevare il testo delle le tag sarà creata dalla chiamata a `soup.select`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
# scrivi qui

for tag in soup.select('[class="text-secondary fz14 text-uppercase strong"]'):
    print(tag.text)

tipi_evento = [tag.text for tag in soup.select('[class="text-secondary fz14 text-
↪uppercase strong"]')]

tipi_evento
```

```
Sport, TOP EVENTI SPORT
Sport, TOP EVENTI SPORT
Sport, TOP EVENTI SPORT
Sport, TOP EVENTI SPORT
Enogastronomia, Intrattenimento della Località, Sport
Sport, Intrattenimento della Località, Enogastronomia
Wellness
Wellness
Intrattenimento della Località, Mercati e mercatini
Sport, TOP EVENTI SPORT
Intrattenimento della Località, Mercati e mercatini
Intrattenimento della Località, Mercati e mercatini
Mercati e mercatini, Intrattenimento della Località
Intrattenimento della Località, Mercati e mercatini
Intrattenimento della Località, Mercati e mercatini
Mercati e mercatini, Folklore
```

[28]:
```
['Sport, TOP EVENTI SPORT',
 'Sport, TOP EVENTI SPORT',
 'Sport, TOP EVENTI SPORT',
 'Sport, TOP EVENTI SPORT',
 'Enogastronomia, Intrattenimento della Località, Sport',
 'Sport, Intrattenimento della Località, Enogastronomia',
 'Wellness',
 'Wellness',
 'Intrattenimento della Località, Mercati e mercatini',
 'Sport, TOP EVENTI SPORT',
 'Intrattenimento della Località, Mercati e mercatini',
 'Intrattenimento della Località, Mercati e mercatini',
 'Mercati e mercatini, Intrattenimento della Località',
 'Intrattenimento della Località, Mercati e mercatini',
 'Intrattenimento della Località, Mercati e mercatini',
 'Mercati e mercatini, Folklore']
```

[28]:
```
# scrivi qui
```

```
Sport, TOP EVENTI SPORT
Sport, TOP EVENTI SPORT
Sport, TOP EVENTI SPORT
Sport, TOP EVENTI SPORT
Enogastronomia, Intrattenimento della Località, Sport
Sport, Intrattenimento della Località, Enogastronomia
Wellness
Wellness
Intrattenimento della Località, Mercati e mercatini
```

(continues on next page)

(continued from previous page)

```
Sport, TOP EVENTI SPORT
Intrattenimento della Località, Mercati e mercatini
Intrattenimento della Località, Mercati e mercatini
Mercati e mercatini, Intrattenimento della Località
Intrattenimento della Località, Mercati e mercatini
Intrattenimento della Località, Mercati e mercatini
Mercati e mercatini, Folklore
```

[28]:
```
['Sport, TOP EVENTI SPORT',
 'Sport, TOP EVENTI SPORT',
 'Sport, TOP EVENTI SPORT',
 'Sport, TOP EVENTI SPORT',
 'Enogastronomia, Intrattenimento della Località, Sport',
 'Sport, Intrattenimento della Località, Enogastronomia',
 'Wellness',
 'Wellness',
 'Intrattenimento della Località, Mercati e mercatini',
 'Sport, TOP EVENTI SPORT',
 'Intrattenimento della Località, Mercati e mercatini',
 'Intrattenimento della Località, Mercati e mercatini',
 'Mercati e mercatini, Intrattenimento della Località',
 'Intrattenimento della Località, Mercati e mercatini',
 'Intrattenimento della Località, Mercati e mercatini',
 'Mercati e mercatini, Folklore']
```

2.5 descrizione

Rimane da aggiungere la descrizione, per esempio per la Coppa del Mondo di Snowboard la descrizione sarebbe "Al Passo Costalunga sfida tra i migliori specialisti del mondo".

⊛ **ESERCIZIO 2.5.1**: Cerca manualmente nell'HTML la stringa `Al Passo Costalunga sfida tra i migliori specialisti del mondo`. In quante posizioni appare ? In base a quali criteri potremo filtrare i tipi degli eventi ? **NOTA**: In base a quanto sai finora, ti sarà impossibile trovare un selettore che possa filtrare quello che serve, quindi per ora pensa solo a trovare stringhe ricorrenti intorno alle descrizioni

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Vediamo che compare in un blocco html regolare e come già successo prima in un blocco `script`. Concentriamoci sul blocco html. Vediamo che la descrizione sta solo vicino ai blocchi `h2`, in particolare alla linea `<p>Al Passo Costalunga sfida tra i migliori specialisti del mondo</p>`:

```
<div class="col col-sm-9 arrange__item">
                    <a href="https://www.visittrentino.info/it/guida/eventi/coppa-del-
↪mondo-di-snowboard_e_165375" class="text-primary list-teaser__link"><span class=
↪"icon icon-circle-arrow fz30"></span></a>
                    <div class="teaser__body">
                                                <span class="text-secondary fz14␣
↪text-uppercase strong">Sport, TOP EVENTI SPORT</span>
                                              <h2 class="text-secondary">Coppa del␣
↪Mondo di Snowboard</h2>
                        <ul class="list-unstyled list-inline list__teaser__list mb15␣
↪mt10">
                                                          <li>
                                    <a class="fz14 text-uppercase strong text-primary
↪"><span class="icon icon-map-view mr10"></span>Passo Costalunga</a>
```

(continues on next page)

(continued from previous page)

```
                                    </li>
                                                                    <li>
                                    <a class="fz14 text-uppercase strong text-primary">
↪<span class="icon icon-map-view mr10"></span>14/12/2017</a>
                                </li>
                            </ul>
                            <p>Al Passo Costalunga sfida tra i migliori specialisti del␣
↪mondo</p>
                        </div>
                    </div>
```

Stavolta il filtraggio non è così semplice come prima, perchè per `<p>` non c'è nessun comodo attributo come `class`. Possiamo però cercare se qualche attributo prima è più facilmente identificabile. Guardiamo per esempio la tag `<ul>` immediatamente precedente:

```
<ul class="list-unstyled list-inline list__teaser__list mb15 mt10">
                                    <li>
            <a class="fz14 text-uppercase strong text-primary"><span class="icon icon-
↪map-view mr10"></span>Passo Costalunga</a>
        </li>
                                    <li>
        <a class="fz14 text-uppercase strong text-primary"><span class="icon icon-map-
↪view mr10"></span>14/12/2017</a>
    </li>
</ul>
<p>Al Passo Costalunga sfida tra i migliori specialisti del mondo</p>
```

Vediamo che la tag in questione `<ul class="list-unstyled list-inline list__teaser__list mb15 mt10">` contiene una classe, se la cerchiamo nel documento troveremo che tutte queste `ul` con questa classe compaiono sempre e solo prima del `<p>` che vogliamo noi.

Trovare la descrizione in Python

Guarda la *soluzione dell'esercizio precedente*. Troverai scritto che la tag `<ul class="list-unstyled list-inline list__teaser__list mb15 mt10">` contiene una classe, se la cerchiamo nel documento scopriremo che tutte queste `ul` con questa classe compaiono sempre e solo prima del `<p>` con le descrizioni che vogliamo noi. Non ci resta quindi che trovare un selettore CSS che

- trovi le tag `ul` con `class=list-unstyled list-inline list__teaser__list mb15 mt10`
- selezioni la tag successiva `p`

Per fare ciò possiamo usare il selettore col simbolo +, quindi nella select basterà scrivere:

```
[29]: soup.select('ul[class="list-unstyled list-inline list__teaser__list mb15 mt10"] + p')
```

```
[29]: [<p>Al Passo Costalunga sfida tra i migliori specialisti del mondo</p>,
 <p>Lunedì 18 dicembre la Coppa Europa fa tappa in Val di Fassa</p>,
 <p>Torna il mitico slalom in notturna: vivi l'evento da VIP!</p>,
 <p>La "regina" delle granfondo compie 45 anni</p>,
 <p>L'emozione di sciare all'alba e prima di tutti. Dopo, una colazione da campioni!</
↪p>,
 <p>Coppa del Mondo Sci di Fondo
 </p>,
```

(continues on next page)

(continued from previous page)

```
 <p>Contest fotografico della Nuvola del Benessere</p>,
 <p>Fra le nuvole ai piedi delle Dolomiti</p>,
 <p>L'incanto genuino del Natale</p>,
 <p>La regina delle ciaspole nel giorno della Befana</p>,
 <p>Profumi, luci, atmosfera di festa </p>,
 <p>La forza della tradizione con un omaggio all'arte</p>,
 <p>Parco secolare degli Asburgo </p>,
 <p>Musica, zelten, vin brulé e spumante Trentodoc per assaporare l'atmosfera␣
↪prenatalizia</p>,
 <p>Quasi 40 casette negli angoli più suggestivi del centro</p>,
 <p>Tradizioni e mercatini natalizi di Pinè</p>]
```

Benissimo, abbiamo ottenuto la lista dei <p> che vogliamo. Adesso possiamo mettere il testo interno dei <p> dentro una lista che chiameremo `descrizioni`. Possiamo farlo in un colpo solo usando una *list comprehension*:

```
[30]: descrizioni = [tag.text for tag in soup.select('ul[class="list-unstyled list-inline␣
↪list__teaser__list mb15 mt10"] + p')]
```

```
[31]: descrizioni
```

```
[31]: ['Al Passo Costalunga sfida tra i migliori specialisti del mondo',
 'Lunedì 18 dicembre la Coppa Europa fa tappa in Val di Fassa',
 "Torna il mitico slalom in notturna: vivi l'evento da VIP!",
 'La “regina” delle granfondo compie 45 anni',
 "L'emozione di sciare all'alba e prima di tutti. Dopo, una colazione da campioni!",
 'Coppa del Mondo Sci di Fondo\n',
 'Contest fotografico della Nuvola del Benessere',
 'Fra le nuvole ai piedi delle Dolomiti',
 "L'incanto genuino del Natale",
 'La regina delle ciaspole nel giorno della Befana',
 'Profumi, luci, atmosfera di festa ',
 "La forza della tradizione con un omaggio all'arte",
 'Parco secolare degli Asburgo ',
 'Musica, zelten, vin brulé e spumante Trentodoc per assaporare l'atmosfera␣
↪prenatalizia',
 'Quasi 40 casette negli angoli più suggestivi del centro',
 'Tradizioni e mercatini natalizi di Pinè']
```

Come al solito, verifichiamo che la lista delle descrizioni sia di 16 elementi:

```
[32]: len(descrizioni)
```

```
[32]: 16
```

2.6 Riempiamo la variabile `righe`

Ricapitolando tutto quanto visto precedentemente, popoliamo `righe` con i dizionari e tutti i campi per assicurarci di avere la struttura dati corretta. In questa riscrittura, usiamo di più le *list comprehnesion* per avere codice più sintetico:

```
[33]: righe = []

for i in range(len(soup.select("h4"))):
    righe.append({})

aggiungi_campo('nome', [tag.text for tag in soup.select("h4")])
```

(continues on next page)

(continued from previous page)

```
aggiungi_campo('data', [tag.text for tag in soup.select('[class="moodboard__item-
↪subline strong fz14 text-uppercase d-b"]')])

luoghi = [tag.text.strip() for tag in soup.select('[class="moodboard__item-subline␣
↪strong fz14 d-b"]')]
luoghi.insert(4, '')
aggiungi_campo('luogo', luoghi)

aggiungi_campo('tipo', [tag.text for tag in soup.select('[class="text-secondary fz14␣
↪text-uppercase strong"]')])

aggiungi_campo('descrizione',
               [tag.text for tag in soup.select('ul[class="list-unstyled list-inline␣
↪list__teaser__list mb15 mt10"] + p')])
```

[33]:
```
[{'nome': 'Coppa del Mondo di Snowboard',
  'data': '14/12/2017',
  'luogo': 'Passo Costalunga',
  'tipo': 'Sport, TOP EVENTI SPORT',
  'descrizione': 'Al Passo Costalunga sfida tra i migliori specialisti del mondo'},
 {'nome': 'Coppa Europa di sci alpino maschile - slalom speciale',
  'data': '18/12/2017',
  'luogo': 'Pozza di Fassa',
  'tipo': 'Sport, TOP EVENTI SPORT',
  'descrizione': 'Lunedì 18 dicembre la Coppa Europa fa tappa in Val di Fassa'},
 {'nome': '3Tre - AUDI FIS Ski World Cup',
  'data': '22/12/2017',
  'luogo': 'Madonna di Campiglio',
  'tipo': 'Sport, TOP EVENTI SPORT',
  'descrizione': "Torna il mitico slalom in notturna: vivi l'evento da VIP!"},
 {'nome': 'La Marcialonga di Fiemme e Fassa ',
  'data': '28/01/2018',
  'luogo': 'Val di Fassa',
  'tipo': 'Sport, TOP EVENTI SPORT',
  'descrizione': 'La “regina” delle granfondo compie 45 anni'},
 {'nome': 'TrentinoSkiSunrise: sulle piste alla luce dell’alba',
  'data': '06/01/2018 - 07/04/2018',
  'luogo': '',
  'tipo': 'Enogastronomia, Intrattenimento della Località, Sport',
  'descrizione': "L'emozione di sciare all’alba e prima di tutti. Dopo, una colazione␣
↪da campioni!"},
 {'nome': 'Tour de Ski',
  'data': '06/01/2018 - 07/01/2018',
  'luogo': 'Tesero',
  'tipo': 'Sport, Intrattenimento della Località, Enogastronomia',
  'descrizione': 'Coppa del Mondo Sci di Fondo\n'},
 {'nome': 'La mia nuvola',
  'data': '01/10/2017 - 30/11/2017',
  'luogo': 'Pozza di Fassa',
  'tipo': 'Wellness',
  'descrizione': 'Contest fotografico della Nuvola del Benessere'},
 {'nome': 'Dormire sotto un cielo di stelle',
  'data': '31/10/2017 - 20/12/2017',
  'luogo': 'Pozza di Fassa',
  'tipo': 'Wellness',
  'descrizione': 'Fra le nuvole ai piedi delle Dolomiti'},
 {'nome': 'Mercatini di Natale di Canale di Tenno e Rango ',
```

(continues on next page)

(continued from previous page)

```
 'data': '19/11/2017 - 30/12/2017',
 'luogo': 'Tenno',
 'tipo': 'Intrattenimento della Località, Mercati e mercatini',
 'descrizione': "L'incanto genuino del Natale"},
{'nome': 'La Ciaspolada',
 'data': '06/01/2018',
 'luogo': 'Fondo',
 'tipo': 'Sport, TOP EVENTI SPORT',
 'descrizione': 'La regina delle ciaspole nel giorno della Befana'},
{'nome': 'Mercatini di Natale a Trento',
 'data': '18/11/2017 - 06/01/2018',
 'luogo': 'Trento',
 'tipo': 'Intrattenimento della Località, Mercati e mercatini',
 'descrizione': 'Profumi, luci, atmosfera di festa '},
{'nome': 'Mercatini di Natale di Rovereto',
 'data': '25/11/2017 - 06/01/2018',
 'luogo': 'Rovereto',
 'tipo': 'Intrattenimento della Località, Mercati e mercatini',
 'descrizione': "La forza della tradizione con un omaggio all'arte"},
{'nome': 'Mercatino di Natale asburgico di Levico Terme',
 'data': '25/11/2017 - 06/01/2018',
 'luogo': 'Levico Terme',
 'tipo': 'Mercati e mercatini, Intrattenimento della Località',
 'descrizione': 'Parco secolare degli Asburgo '},
{'nome': 'Magnifico Mercatino di Cavalese',
 'data': '01/12/2017 - 06/01/2018',
 'luogo': 'Cavalese',
 'tipo': 'Intrattenimento della Località, Mercati e mercatini',
 'descrizione': 'Musica, zelten, vin brulé e spumante Trentodoc per assaporare␣
↪l'atmosfera prenatalizia'},
{'nome': 'Mercatino di Natale ad Arco',
 'data': '17/11/2017 - 07/01/2018',
 'luogo': 'Arco',
 'tipo': 'Intrattenimento della Località, Mercati e mercatini',
 'descrizione': 'Quasi 40 casette negli angoli più suggestivi del centro'},
{'nome': 'El paès dei presepi',
 'data': '08/12/2017 - 07/01/2018',
 'luogo': 'Baselga di Piné',
 'tipo': 'Mercati e mercatini, Folklore',
 'descrizione': 'Tradizioni e mercatini natalizi di Pinè'}]
```

5.5.5 3. Scriviamo il CSV

⊛⊛ **ESERCIZIO 3.1**: Prova a scrivere un file CSV contenente tutti i dati raccolti, con quest'ordine di colonne:

`nome, data, luogo, tipo, descrizione`

Diversamente da quanto fatto finora, in cui abbiamo solo considerato righe come liste, in questo caso dato che abbiamo dizionari ci servirà un oggetto di tipo `DictWriter`. Leggi la documentazione Python su DictWriter[364] e copiandone l'esempio prova a scrivere il csv in un file chiamato `eventi.csv`.

NOTA: Per testare il tutto, fai attenzione a che nella cartella di questo progetto non sia già presente il file `eventi.csv`, in caso eliminalo a mano.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra

[364] https://docs.python.org/3/library/csv.html#csv.DictWriter

soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[34]:
```
# scrivi qui

import csv

# apriamo il file `eventi.csv`in scrittura :
# 'fouput' è un nome  scelto da noi)
# 'w' indica a Python che vogliamo scrivere il file
#     se il file esiste già, verrà sovrascritto.
#     NOTA:  FATE SEMPRE MOLTA ATTENZIONE A NON SOVRASCRIVERE FILE PER SBAGLIO !!!!!
with open('eventi.csv', 'w', encoding='utf-8') as foutput:
    # andremo a scrivere dizionari, ma nei dizionari le chiavi sono in ordine casuale:
    # per avere una serie di colonne ordinata in modo prevedibile, siamo quindi␣
↪obbligati
    # a fornire una lista di intestazioni:
    nomi_campi = ['nome', 'data', 'luogo', 'tipo', 'descrizione']
    writer = csv.DictWriter(foutput,fieldnames=nomi_campi)  # Ci serve creare un
↪'oggetto 'scrittore' di tipo DictWriter
    writer.writeheader()
    for diz in righe: # stavolta le righe contengono dizionari
        writer.writerow(diz)  # chiamiamo l'oggetto scrittore dicendogli di scrivere␣
↪il dizionario corrente
```

[34]:
```
# scrivi qui
```

⊛⊛⊛ **ESERCIZIO 3.2**: Abbiamo dei luoghi estratti dall'HTML. Nella lezione sull'integrazione dati[365], abbiamo imparato a usare Webapi di MapQuest per ottenere delle coordinate geografiche a partire da nomi di luoghi. Riusciresti quindi a:

- usare la libreria `requests` e webapi MapQuest per arricchire prima i dizionari di `righe` con `lat` e `lon`
- aggiungere `lat` e `lon` al CSV generato
- Caricare il CSV in Umap

[35]:
```
# scrivi qui
```

5.6 Pandas

5.6.1 Scarica zip esercizi

Naviga file online[366]

[365] http://it.softpython.org/data-integration.html
[366] https://github.com/DavidLeoni/softpython-it/tree/master/pandas

5.6.2 1. Introduzione

Oggi affronteremo il tema dell'analisi dati con Pandas:

- analisi dati con libreria Pandas
- plotting con MatPlotLib
- Esempi con dataset AstroPi
- Esercizi con dataset meteotrentino
- mappa regioni italiane con GeoPandas

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
pandas
    pandas.ipynb
    pandas-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `analytics.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

Python mette a disposizione degli strumenti potenti per l'analisi dei dati:

Uno di questi è Pandas[367], che fornisce strutture di dati veloci e flessibili, soprattutto per l'analisi dei dati in tempo reale.

Installiamolo subito:

Anaconda:

- `conda install pandas`

Linux/Mac (`--user` installa nella propria home):

- `python3 -m pip install --user pandas`

5.6.3 2. Analisi dei dati di Astro Pi

Proviamo ad analizzare i dati registrati dal Raspberry presente sulla Stazione Spaziale Internazionale, scaricati da qui:

raspberrypi.org/learning/astro-pi-flight-data-analysis/worksheet[368]

su cui è possibile trovare la descrizione dettagliata dei dati raccolti dai sensori, nel mese di febbraio 2016 (un record ogni 10 secondi).

DOMANDA: Che licenza hanno i dati ? Riesci a trovarla da qualche parte ?

[367] https://pandas.pydata.org/
[368] https://www.raspberrypi.org/learning/astro-pi-flight-data-analysis/worksheet/

Il metodo `read_csv` importa i dati da un file CSV e li memorizza in una struttura DataFrame.

In questo esercizio useremo il file Columbus_Ed_astro_pi_datalog.csv

[1]:
```
import pandas as pd    # importiamo pandas e per comodità lo rinominiamo in  'pd'
import numpy as np     # importiamo numpy e per comodità lo rinominiamo in 'np'

# ricordatevi l'encoding !
df = pd.read_csv('Columbus_Ed_astro_pi_datalog.csv', encoding='UTF-8')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110869 entries, 0 to 110868
Data columns (total 20 columns):
ROW_ID        110869 non-null int64
temp_cpu      110869 non-null float64
temp_h        110869 non-null float64
temp_p        110869 non-null float64
humidity      110869 non-null float64
pressure      110869 non-null float64
```

(continues on next page)

(continued from previous page)

```
pitch         110869 non-null float64
roll          110869 non-null float64
yaw           110869 non-null float64
mag_x         110869 non-null float64
mag_y         110869 non-null float64
mag_z         110869 non-null float64
accel_x       110869 non-null float64
accel_y       110869 non-null float64
accel_z       110869 non-null float64
gyro_x        110869 non-null float64
gyro_y        110869 non-null float64
gyro_z        110869 non-null float64
reset         110869 non-null int64
time_stamp    110869 non-null object
dtypes: float64(17), int64(2), object(1)
memory usage: 16.9+ MB
```

Possiamo vedere rapidamente righe e colonne del dataframe con l'attributo `shape`:

NOTA: `shape` non è seguito da parentesi tonde !

[2]:
```
df.shape
```

[2]:
```
(110869, 20)
```

Il metodo `describe` vi da al volo tutta una serie di dati di riepilogo:

- il conteggio delle righe
- la media
- la deviazione standard[369]
- i quartili[370]
- minimo e massimo

[3]:
```
df.describe()
```

[3]:
```
              ROW_ID       temp_cpu         temp_h         temp_p  \
count  110869.000000  110869.000000  110869.000000  110869.000000
mean    55435.000000      32.236259      28.101773      25.543272
std     32005.267835       0.360289       0.369256       0.380877
min         1.000000      31.410000      27.200000      24.530000
25%     27718.000000      31.960000      27.840000      25.260000
50%     55435.000000      32.280000      28.110000      25.570000
75%     83152.000000      32.480000      28.360000      25.790000
max    110869.000000      33.700000      29.280000      26.810000

            humidity       pressure          pitch           roll  \
count  110869.000000  110869.000000  110869.000000  110869.000000
mean       46.252005    1008.126788       2.770553      51.807973
std         1.907273       3.093485      21.848940       2.085821
min        42.270000    1001.560000       0.000000      30.890000
25%        45.230000    1006.090000       1.140000      51.180000
50%        46.130000    1007.650000       1.450000      51.950000
75%        46.880000    1010.270000       1.740000      52.450000
```

(continues on next page)

[369] https://it.wikipedia.org/wiki/Scarto_quadratico_medio
[370] https://it.wikipedia.org/wiki/Quantile

(continued from previous page)

```
max        60.590000    1021.780000     360.000000     359.400000

               yaw          mag_x          mag_y          mag_z  \
count  110869.00000  110869.000000  110869.000000  110869.000000
mean      200.90126     -19.465265      -1.174493      -6.004529
std        84.47763      28.120202      15.655121       8.552481
min         0.01000     -73.046240     -43.810030     -41.163040
25%       162.43000     -41.742792     -12.982321     -11.238430
50%       190.58000     -21.339485      -1.350467      -5.764400
75%       256.34000       7.299000      11.912456      -0.653705
max       359.98000      33.134748      37.552135      31.003047

            accel_x        accel_y        accel_z        gyro_x  \
count  110869.000000  110869.000000  110869.000000  1.108690e+05
mean       -0.000630       0.018504       0.014512 -8.959493e-07
std         0.000224       0.000604       0.000312  2.807614e-03
min        -0.025034      -0.005903      -0.022900 -3.037930e-01
25%        -0.000697       0.018009       0.014349 -2.750000e-04
50%        -0.000631       0.018620       0.014510 -3.000000e-06
75%        -0.000567       0.018940       0.014673  2.710000e-04
max         0.018708       0.041012       0.029938  2.151470e-01

             gyro_y        gyro_z          reset
count  110869.000000  1.108690e+05  110869.000000
mean        0.000007 -9.671594e-07       0.000180
std         0.002456  2.133104e-03       0.060065
min        -0.378412 -2.970800e-01       0.000000
25%        -0.000278 -1.200000e-04       0.000000
50%        -0.000004 -1.000000e-06       0.000000
75%         0.000271  1.190000e-04       0.000000
max         0.389499  2.698760e-01      20.000000
```

DOMANDA: Manca qualche campo alla tabella prodotta da describe? Perchè non l'ha incluso ?

Per limitare `describe` ad una sola colonna come `humidity`, puoi scrivere così:

[4]:
```
df['humidity'].describe()
```

[4]:
```
count    110869.000000
mean         46.252005
std           1.907273
min          42.270000
25%          45.230000
50%          46.130000
75%          46.880000
max          60.590000
Name: humidity, dtype: float64
```

Ancora più comodamente, puoi usare la notazione con il punto:

[5]:
```
df.humidity.describe()
```

[5]:
```
count    110869.000000
mean         46.252005
std           1.907273
min          42.270000
25%          45.230000
50%          46.130000
```

(continues on next page)

(continued from previous page)

```
75%          46.880000
max          60.590000
Name: humidity, dtype: float64
```

ATTENZIONE: Nel caso il nome del campo avesse degli spazi (es. `'rotazioni frullatore'`), **non** potreste usare la notazione con il punto ma sareste costretti ad usare la notazione con le quadre vista sopra (es: `df.['rotazioni frullatore'].describe()`)

Il metodo `head` restituisce i primi dataset:

```
[6]: df.head()
```

```
[6]:    ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll     yaw  \
0       1     31.88   27.57   25.01     44.94   1001.68   1.49  52.25  185.21
1       2     31.79   27.53   25.01     45.12   1001.72   1.03  53.73  186.72
2       3     31.66   27.53   25.01     45.12   1001.72   1.24  53.57  186.21
3       4     31.69   27.52   25.01     45.32   1001.69   1.57  53.63  186.03
4       5     31.66   27.54   25.01     45.18   1001.71   0.85  53.66  186.46

       mag_x     mag_y      mag_z   accel_x   accel_y   accel_z    gyro_x  \
0 -46.422753 -8.132907 -12.129346 -0.000468  0.019439  0.014569  0.000942
1 -48.778951 -8.304243 -12.943096 -0.000614  0.019436  0.014577  0.000218
2 -49.161878 -8.470832 -12.642772 -0.000569  0.019359  0.014357  0.000395
3 -49.341941 -8.457380 -12.615509 -0.000575  0.019383  0.014409  0.000308
4 -50.056683 -8.122609 -12.678341 -0.000548  0.019378  0.014380  0.000321

     gyro_y    gyro_z  reset           time_stamp
0  0.000492 -0.000750     20  2016-02-16 10:44:40
1 -0.000005 -0.000235      0  2016-02-16 10:44:50
2  0.000600 -0.000003      0  2016-02-16 10:45:00
3  0.000577 -0.000102      0  2016-02-16 10:45:10
4  0.000691  0.000272      0  2016-02-16 10:45:20
```

Il metodo `tail` restituisce gli ultimi dataset:

```
[7]: df.tail()
```

```
[7]:        ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll  \
110864  110865     31.56   27.52   24.83     42.94   1005.83   1.58  49.93
110865  110866     31.55   27.50   24.83     42.72   1005.85   1.89  49.92
110866  110867     31.58   27.50   24.83     42.83   1005.85   2.09  50.00
110867  110868     31.62   27.50   24.83     42.81   1005.88   2.88  49.69
110868  110869     31.57   27.51   24.83     42.94   1005.86   2.17  49.77

           yaw      mag_x      mag_y     mag_z   accel_x   accel_y   accel_z  \
110864  129.60 -15.169673 -27.642610  1.563183 -0.000682  0.017743  0.014646
110865  130.51 -15.832622 -27.729389  1.785682 -0.000736  0.017570  0.014855
110866  132.04 -16.646212 -27.719479  1.629533 -0.000647  0.017657  0.014799
110867  133.00 -17.270447 -27.793136  1.703806 -0.000835  0.017635  0.014877
110868  134.18 -17.885872 -27.824149  1.293345 -0.000787  0.017261  0.014380

          gyro_x    gyro_y    gyro_z  reset           time_stamp
110864 -0.000264  0.000206  0.000196      0  2016-02-29 09:24:21
110865  0.000143  0.000199 -0.000024      0  2016-02-29 09:24:30
110866  0.000537  0.000257  0.000057      0  2016-02-29 09:24:41
110867  0.000534  0.000456  0.000195      0  2016-02-29 09:24:50
110868  0.000459  0.000076  0.000030      0  2016-02-29 09:25:00
```

La proprietà `colums` restituisce le intestazioni di colonna:

[8]:
```
df.columns
```

[8]:
```
Index(['ROW_ID', 'temp_cpu', 'temp_h', 'temp_p', 'humidity', 'pressure',
       'pitch', 'roll', 'yaw', 'mag_x', 'mag_y', 'mag_z', 'accel_x', 'accel_y',
       'accel_z', 'gyro_x', 'gyro_y', 'gyro_z', 'reset', 'time_stamp'],
      dtype='object')
```

Nota: Come si vede qua sopra, il tipo dell'oggetto ritornato non è una lista, ma un contenitore speciale definito da pandas:

[9]:
```
type(df.columns)
```

[9]:
```
pandas.core.indexes.base.Index
```

Ciononostante, possiamo accedere agli elementi di questo contenitore usando indici dentro le parentesi quadre:

[10]:
```
df.columns[0]
```

[10]:
```
'ROW_ID'
```

[11]:
```
df.columns[1]
```

[11]:
```
'temp_cpu'
```

2.1 ESERCIZIO: Creare un nuovo dataframe chiamato `meteo` importando i dati dal file meteo.csv, che contiene i dati meteo di Trento di novembre 2017 (fonte: https://www.meteotrentino.it). **IMPORTANTE**: assegna il dataframe ad una variabile chiamata `meteo` (così evitiamo confusione con il dataframe dell'AstroPi)

2.2 ESERCIZIO: Visualizzare le informazioni relative a questo Dataframe.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
# scrivi qui - crea il dataframe

meteo = pd.read_csv('meteo.csv', encoding='UTF-8')
print("COLUMNS:")
print()
print(meteo.columns)
print()
print("INFO:")
print(meteo.info())
print()
print("HEAD():")

meteo.head()
```

```
COLUMNS:

Index(['Data', 'Pressione', 'Pioggia', 'Temp'], dtype='object')

INFO:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2878 entries, 0 to 2877
Data columns (total 4 columns):
Data         2878 non-null object
Pressione    2878 non-null float64
Pioggia      2878 non-null float64
```

(continues on next page)

(continued from previous page)

```
Temp         2878 non-null float64
dtypes: float64(3), object(1)
memory usage: 90.1+ KB
None

HEAD():
```

[12]:
```
               Data  Pressione  Pioggia  Temp
0  01/11/2017 00:00      995.4      0.0   5.4
1  01/11/2017 00:15      995.5      0.0   6.0
2  01/11/2017 00:30      995.5      0.0   5.9
3  01/11/2017 00:45      995.7      0.0   5.4
4  01/11/2017 01:00      995.7      0.0   5.3
```

[12]:
```
# scrivi qui - crea il dataframe
```

```
COLUMNS:

Index(['Data', 'Pressione', 'Pioggia', 'Temp'], dtype='object')

INFO:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2878 entries, 0 to 2877
Data columns (total 4 columns):
Data         2878 non-null object
Pressione    2878 non-null float64
Pioggia      2878 non-null float64
Temp         2878 non-null float64
dtypes: float64(3), object(1)
memory usage: 90.1+ KB
None

HEAD():
```

[12]:
```
               Data  Pressione  Pioggia  Temp
0  01/11/2017 00:00      995.4      0.0   5.4
1  01/11/2017 00:15      995.5      0.0   6.0
2  01/11/2017 00:30      995.5      0.0   5.9
3  01/11/2017 00:45      995.7      0.0   5.4
4  01/11/2017 01:00      995.7      0.0   5.3
```

5.6.4 3. Indicizzazione, filtraggio e ordinamento

Per ottenere la i-esima serie si utilizza il metodo `iloc[i]` (qui riusiamo il dataset dell'AstroPI) :

[13]:
```
df.iloc[6]
```

[13]:
```
ROW_ID                       7
temp_cpu                 31.68
temp_h                   27.53
temp_p                   25.01
humidity                 45.31
pressure                1001.7
```

(continues on next page)

(continued from previous page)

```
pitch                        0.63
roll                        53.55
yaw                         186.1
mag_x                    -50.4473
mag_y                    -7.93731
mag_z                    -12.1886
accel_x                  -0.00051
accel_y                  0.019264
accel_z                  0.014528
gyro_x                  -0.000111
gyro_y                   0.00032
gyro_z                   0.000222
reset                           0
time_stamp    2016-02-16 10:45:41
Name: 6, dtype: object
```

È possibile selezionare un dataframe di posizioni contigue, utilizzando lo *slicing* (vedere libro Pensare in Python[371]).

Qua per esempio selezioniamo le righe dalla 5 *inclusa* alla 7 *esclusa* :

[14]:
```
df.iloc[5:7]
```

[14]:
```
   ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll     yaw  \
5       6     31.69   27.55   25.01     45.12   1001.67   0.85  53.53  185.52
6       7     31.68   27.53   25.01     45.31   1001.70   0.63  53.55  186.10

       mag_x     mag_y      mag_z   accel_x   accel_y   accel_z    gyro_x  \
5 -50.246476 -8.343209 -11.938124 -0.000536  0.019453  0.014380  0.000273
6 -50.447346 -7.937309 -12.188574 -0.000510  0.019264  0.014528 -0.000111

     gyro_y    gyro_z  reset           time_stamp
5  0.000494 -0.000059      0  2016-02-16 10:45:30
6  0.000320  0.000222      0  2016-02-16 10:45:41
```

È possibile filtrare i dati in base al soddisfacimento di una condizione.

Possiamo scoprire che tipo di dato è ad esempio `df.ROW_ID >= 6`:

[15]:
```
type(df.ROW_ID >= 6)
```

[15]:
```
pandas.core.series.Series
```

Cosa contiene questo oggetto `Series` ? Se proviamo a stamparlo vedremo che si tratta di una serie di valori Vero o Falso, a seconda se il valore di ROW_ID è maggiore o uguale a 6:

[16]:
```
df.ROW_ID >= 6
```

[16]:
```
0         False
1         False
2         False
3         False
4         False
          …
110864     True
110865     True
110866     True
110867     True
```

(continues on next page)

[371] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2009.html#sec98

(continued from previous page)

```
110868     True
Name: ROW_ID, Length: 110869, dtype: bool
```

In modo analogo `(df.ROW_ID >= 6) & (df.ROW_ID <= 10)` è una serie di valori Vero o Falso, se ROW_ID è contemporaneamente maggiore o uguale a 6 e minore e uguale a 10.

```
[17]: type((df.ROW_ID >= 6) & (df.ROW_ID <= 10))
```

```
[17]: pandas.core.series.Series
```

Se vogliamo le righe complete del dataframe che soddisfano la condizione, possiamo scrivere così:

> **IMPORTANTE**: usiamo `df` all'*esterno* dell'espressione `df[ ]` iniziando e chiudendo con le parentesi quadrate per dire a Python che vogliamo filtrare sul dataframe `df`, e usiamo di nuovo `df` all'*interno* delle quadre per indicare su *quali colonne* e *quali righe* vogliamo filtrare

```
[18]: df[  (df.ROW_ID >= 6) & (df.ROW_ID <= 10)  ]
```

```
[18]:    ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll     yaw  \
5       6     31.69   27.55   25.01     45.12   1001.67   0.85  53.53  185.52
6       7     31.68   27.53   25.01     45.31   1001.70   0.63  53.55  186.10
7       8     31.66   27.55   25.01     45.34   1001.70   1.49  53.65  186.08
8       9     31.67   27.54   25.01     45.20   1001.72   1.22  53.77  186.55
9      10     31.67   27.54   25.01     45.41   1001.75   1.63  53.46  185.94

       mag_x     mag_y      mag_z   accel_x   accel_y   accel_z    gyro_x  \
5 -50.246476 -8.343209 -11.938124 -0.000536  0.019453  0.014380  0.000273
6 -50.447346 -7.937309 -12.188574 -0.000510  0.019264  0.014528 -0.000111
7 -50.668232 -7.762600 -12.284196 -0.000523  0.019473  0.014298 -0.000044
8 -50.761529 -7.262934 -11.981090 -0.000522  0.019385  0.014286  0.000358
9 -51.243832 -6.875270 -11.672494 -0.000581  0.019390  0.014441  0.000266

     gyro_y    gyro_z  reset           time_stamp
5  0.000494 -0.000059      0  2016-02-16 10:45:30
6  0.000320  0.000222      0  2016-02-16 10:45:41
7  0.000436  0.000301      0  2016-02-16 10:45:50
8  0.000651  0.000187      0  2016-02-16 10:46:01
9  0.000676  0.000356      0  2016-02-16 10:46:10
```

Quindi se cerchiamo il record in cui la pressione è massima, utilizziamo la proprietà `values` della serie su cui calcoliamo il valore massimo:

```
[19]: df[  (df.pressure == df.pressure.values.max())  ]
```

```
[19]:       ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll  \
77602   77603     32.44   28.31   25.74     47.57   1021.78    1.1  51.82

          yaw     mag_x      mag_y      mag_z   accel_x  accel_y   accel_z  \
77602  267.39 -0.797428  10.891803 -15.728202 -0.000612  0.01817  0.014295

         gyro_x    gyro_y    gyro_z  reset           time_stamp
77602 -0.000139 -0.000179 -0.000298      0  2016-02-25 12:13:20
```

Il metodo `sort_values` ritorna un dataframe ordinato in base a una o più colonne:

[20]:
```
df.sort_values('pressure',ascending=False).head()
```

[20]:
```
       ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll  \
77602   77603     32.44   28.31   25.74     47.57   1021.78   1.10  51.82
77601   77602     32.45   28.30   25.74     47.26   1021.75   1.53  51.76
77603   77604     32.44   28.30   25.74     47.29   1021.75   1.86  51.83
77604   77605     32.43   28.30   25.74     47.39   1021.75   1.78  51.54
77608   77609     32.42   28.29   25.74     47.36   1021.73   0.86  51.89

          yaw     mag_x      mag_y      mag_z   accel_x   accel_y   accel_z  \
77602  267.39 -0.797428  10.891803 -15.728202 -0.000612  0.018170  0.014295
77601  266.12 -1.266335  10.927442 -15.690558 -0.000661  0.018357  0.014533
77603  268.83 -0.320795  10.651441 -15.565123 -0.000648  0.018290  0.014372
77604  269.41 -0.130574  10.628383 -15.488983 -0.000672  0.018154  0.014602
77608  272.77  0.952025  10.435951 -16.027235 -0.000607  0.018186  0.014232

         gyro_x    gyro_y    gyro_z  reset           time_stamp
77602 -0.000139 -0.000179 -0.000298      0  2016-02-25 12:13:20
77601  0.000152  0.000459 -0.000298      0  2016-02-25 12:13:10
77603  0.000049  0.000473 -0.000029      0  2016-02-25 12:13:30
77604  0.000360  0.000089 -0.000002      0  2016-02-25 12:13:40
77608 -0.000260 -0.000059 -0.000187      0  2016-02-25 12:14:20
```

La properietà `loc` ci permette di filtrare righe secondo una proprietà e selezionare una colonna, che può essere anche nuova. In questo caso, per le righe dove la temperatura cpu è eccessiva, scriviamo il valore `True` nei campi della colonna con intestazione `'Too hot'`:

[21]:
```
df.loc[(df.temp_cpu > 31.68),'Too hot'] = True
```

Vediamo la tabella risultante (scorri fino in fondo per vedere la nuova colonna). Notiamo come i valori delle righe che non abbiamo filtrato vengono rappresentati con `NaN`, che letteralmente significa *not a number* :

[22]:
```
df.head()
```

[22]:
```
   ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll     yaw  \
0       1     31.88   27.57   25.01     44.94   1001.68   1.49  52.25  185.21
1       2     31.79   27.53   25.01     45.12   1001.72   1.03  53.73  186.72
2       3     31.66   27.53   25.01     45.12   1001.72   1.24  53.57  186.21
3       4     31.69   27.52   25.01     45.32   1001.69   1.57  53.63  186.03
4       5     31.66   27.54   25.01     45.18   1001.71   0.85  53.66  186.46

       mag_x  …      mag_z   accel_x   accel_y   accel_z    gyro_x  \
0 -46.422753  … -12.129346 -0.000468  0.019439  0.014569  0.000942
1 -48.778951  … -12.943096 -0.000614  0.019436  0.014577  0.000218
2 -49.161878  … -12.642772 -0.000569  0.019359  0.014357  0.000395
3 -49.341941  … -12.615509 -0.000575  0.019383  0.014409  0.000308
4 -50.056683  … -12.678341 -0.000548  0.019378  0.014380  0.000321

     gyro_y    gyro_z  reset           time_stamp Too hot
0  0.000492 -0.000750     20  2016-02-16 10:44:40    True
1 -0.000005 -0.000235      0  2016-02-16 10:44:50    True
2  0.000600 -0.000003      0  2016-02-16 10:45:00     NaN
3  0.000577 -0.000102      0  2016-02-16 10:45:10    True
4  0.000691  0.000272      0  2016-02-16 10:45:20     NaN

[5 rows x 21 columns]
```

Pandas è una libreria molto flessibile, e fornisce diversi modi per ottenere gli stessi obbiettivi. Per esempio, possiamo effettuare la stessa operazione di sopra con il comando `np.where` come qua sotto. Ad esempio, aggiungiamo una

colonna che mi dice se la pressione è sopra o sotto la media.

[23]:
```
pressione_media = df.pressure.values.mean()
df['check_p'] = np.where(df.pressure <= pressione_media, 'sotto', 'sopra')
```

ESERCIZIO 3.1: Analizzare i dati del Dataframe `meteo` e trovare:

- i valori di pressione media, minima e massima
- la temperatura media
- le date delle giornate di pioggia

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[24]:
```
# scrivi qui
print("Media pressione : %s" % meteo.Pressione.values.mean())
print("Minimo pressione : %s" % meteo.Pressione.values.min())
print("Massimo pressione : %s" % meteo.Pressione.values.max())
print("Media temperatura : %s" % meteo.Temp.values.mean())
meteo[(meteo.Pioggia > 0)]
```

```
Media pressione : 986.3408269631689
Minimo pressione : 966.3
Massimo pressione : 998.3
Media temperatura : 6.410701876302988
```

[24]:
```
                  Data  Pressione  Pioggia  Temp
433   05/11/2017 12:15      979.2      0.2   8.6
435   05/11/2017 12:45      978.9      0.2   8.4
436   05/11/2017 13:00      979.0      0.2   8.4
437   05/11/2017 13:15      979.1      0.8   8.2
438   05/11/2017 13:30      979.0      0.6   8.2
…                  …          …        …    …
2754  29/11/2017 17:15      976.1      0.2   0.9
2755  29/11/2017 17:30      975.9      0.2   0.9
2802  30/11/2017 05:15      971.3      0.2   1.3
2803  30/11/2017 05:30      971.3      0.2   1.1
2804  30/11/2017 05:45      971.5      0.2   1.1

[107 rows x 4 columns]
```

[24]:
```
# scrivi qui
```

```
Media pressione : 986.3408269631689
Minimo pressione : 966.3
Massimo pressione : 998.3
Media temperatura : 6.410701876302988
```

[24]:
```
                  Data  Pressione  Pioggia  Temp
433   05/11/2017 12:15      979.2      0.2   8.6
435   05/11/2017 12:45      978.9      0.2   8.4
436   05/11/2017 13:00      979.0      0.2   8.4
437   05/11/2017 13:15      979.1      0.8   8.2
438   05/11/2017 13:30      979.0      0.6   8.2
```

(continues on next page)

(continued from previous page)

```
...                  ...          ...       ...    ...
2754  29/11/2017 17:15      976.1      0.2   0.9
2755  29/11/2017 17:30      975.9      0.2   0.9
2802  30/11/2017 05:15      971.3      0.2   1.3
2803  30/11/2017 05:30      971.3      0.2   1.1
2804  30/11/2017 05:45      971.5      0.2   1.1

[107 rows x 4 columns]
```

5.6.5 4. Rivediamo MatPlotLib

Abbiamo già visto MatplotLib nella parte sulla visualizzazione[372], e oggi lo useremo Matplotlib[373] per disegnare i dati.

Riprendiamo un esempio, usando l'approccio in *stile Matlab*. Plotteremo una retta passando due liste di coordinate, una per le x e una per le y:

[25]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
```

[26]:
```
x = [1,2,3,4]
y = [2,4,6,8]
plt.plot(x, y) # possiamo direttamente passare liste per le x e y
plt.title('Qualche numero')
plt.show()
```

Possiamo anche creare serie con numpy. Proviamo a fare una parabola:

[27]:
```
x = np.arange(0.,5.,0.1)
#  '**' è l'operatore di elevamento a potenza in Python, NON '^'
y = x**2
```

[372] http://it.softpython.org/visualization/visualization-sol.html
[373] http://matplotlib.org

Utilizziamo la funzione `type` per capire che tipo di dati sono x e y:

[28]:
```
type(x)
```

[28]:
```
numpy.ndarray
```

[29]:
```
type(y)
```

[29]:
```
numpy.ndarray
```

Si tratta quindi di vettori di NumPy.

[30]:
```
plt.title('La parabola')
plt.plot(x,y);
```

Se vogliamo che le unità dell'asse x siano della stessa dimensione di quelle dell'asse y, possiamo utilizzare la funzione gca[374]

Per settare i limiti delle x e delle y, possiamo usare `xlim` e `ylim`:

[31]:
```
plt.xlim([0, 5])
plt.ylim([0,10])
plt.title('La parabola')

plt.gca().set_aspect('equal')
plt.plot(x,y);
```

[374] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.gca.html?highlight=matplotlib%20pyplot%20gca#matplotlib.pyplot.gca

Grafici matplotlib da strutture pandas

Si possono ricavare grafici direttamente da strutture pandas, sempre usando lo *stile matlab*. Qua c'è la documentazione di DataFrame.plot[375] Facciamo un esempio. In caso di un numero molto elevato di dati, può essere utile avere un'idea qualitativa dei dati, mettendoli in grafico:

[32]:
```
df.humidity.plot(label="Humidity", legend=True)
# con secondary_y=True facciamo apparire i numeri per l'asse delle y
# del secondo grafico sulla destra
df.pressure.plot(secondary_y=True, label="Pressure", legend=True);
```

Proviamo a mettere valori di pressione sull'asse orizzontale, e vedere quali valori di umidità sull'asse verticale corrispondono ad una certa pressione:

[33]:
```
plt.plot(df['pressure'], df['humidity'])
```

[33]:
```
[<matplotlib.lines.Line2D at 0x7f62a33b2518>]
```

[375] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html

Selezioniamo nel nuovo dataframe `df2` le righe tra la 12500esima (inclusa) e la 15000esima (esclusa):

[34]:
```
df2=df.iloc[12500:15000]
```

[35]:
```
plt.plot(df2['pressure'], df2['humidity'])
```

[35]:
```
[<matplotlib.lines.Line2D at 0x7f629f4100b8>]
```

[36]:
```
df2.humidity.plot(label="Humidity", legend=True)
df2.pressure.plot(secondary_y=True, label="Pressure", legend=True)
```

[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f629f3ff630>
```

Il metodo `corr` permette di calcolare la correlazione tra le colonne del DataFrame.

[37]:
```
df2.corr()
```

[37]:
```
            ROW_ID  temp_cpu    temp_h    temp_p  humidity  pressure  \
ROW_ID    1.000000  0.561540  0.636899  0.730764  0.945210  0.760732
temp_cpu  0.561540  1.000000  0.591610  0.670043  0.488038  0.484902
temp_h    0.636899  0.591610  1.000000  0.890775  0.539603  0.614536
temp_p    0.730764  0.670043  0.890775  1.000000  0.620307  0.650015
humidity  0.945210  0.488038  0.539603  0.620307  1.000000  0.750000
pressure  0.760732  0.484902  0.614536  0.650015  0.750000  1.000000
pitch     0.005633  0.025618  0.022718  0.019178  0.012247  0.037081
roll      0.266995  0.165540  0.196767  0.192621  0.231316  0.225112
yaw       0.172192  0.056950 -0.024700  0.007474  0.181905  0.070603
mag_x    -0.108713 -0.019815 -0.151336 -0.060122 -0.108781 -0.246485
mag_y     0.057601 -0.028729  0.031512 -0.039648  0.131218  0.194611
mag_z    -0.270656 -0.193077 -0.260633 -0.285640 -0.191957 -0.173808
accel_x   0.015936 -0.021093 -0.009408 -0.034348  0.040452  0.085183
accel_y   0.121838  0.108878  0.173037  0.187457  0.069717 -0.032049
accel_z   0.075160  0.065628  0.129074  0.144595  0.021627 -0.068296
gyro_x   -0.014346 -0.019478 -0.005255 -0.010679  0.005625 -0.014838
gyro_y   -0.026012 -0.007527 -0.017054 -0.016674 -0.001927 -0.008821
gyro_z    0.011714 -0.006737 -0.016113 -0.017010  0.014431  0.032056
reset          NaN       NaN       NaN       NaN       NaN       NaN

             pitch      roll       yaw     mag_x     mag_y     mag_z  \
ROW_ID    0.005633  0.266995  0.172192 -0.108713  0.057601 -0.270656
temp_cpu  0.025618  0.165540  0.056950 -0.019815 -0.028729 -0.193077
temp_h    0.022718  0.196767 -0.024700 -0.151336  0.031512 -0.260633
temp_p    0.019178  0.192621  0.007474 -0.060122 -0.039648 -0.285640
humidity  0.012247  0.231316  0.181905 -0.108781  0.131218 -0.191957
pressure  0.037081  0.225112  0.070603 -0.246485  0.194611 -0.173808
pitch     1.000000  0.068880  0.030448 -0.008220 -0.002278 -0.019085
roll      0.068880  1.000000 -0.053750 -0.281035 -0.479779 -0.665041
yaw       0.030448 -0.053750  1.000000  0.536693  0.300571  0.394324
mag_x    -0.008220 -0.281035  0.536693  1.000000  0.046591  0.475674
mag_y    -0.002278 -0.479779  0.300571  0.046591  1.000000  0.794756
mag_z    -0.019085 -0.665041  0.394324  0.475674  0.794756  1.000000
accel_x   0.024460  0.057330 -0.028267 -0.097520  0.046693  0.001699
accel_y  -0.053634 -0.049233  0.078585  0.168764 -0.035111 -0.020016
```

(continues on next page)

(continued from previous page)

```
accel_z  -0.029345 -0.153524  0.068321  0.115423 -0.022579 -0.006496
gyro_x    0.040685  0.139427 -0.021071 -0.017739 -0.084045 -0.092749
gyro_y    0.041674  0.134319 -0.009650 -0.006722 -0.061460 -0.060097
gyro_z   -0.024081 -0.078113  0.064290  0.008456  0.115327  0.101276
reset           NaN       NaN       NaN       NaN       NaN       NaN

           accel_x   accel_y   accel_z    gyro_x    gyro_y    gyro_z  reset
ROW_ID    0.015936  0.121838  0.075160 -0.014346 -0.026012  0.011714    NaN
temp_cpu -0.021093  0.108878  0.065628 -0.019478 -0.007527 -0.006737    NaN
temp_h   -0.009408  0.173037  0.129074 -0.005255 -0.017054 -0.016113    NaN
temp_p   -0.034348  0.187457  0.144595 -0.010679 -0.016674 -0.017010    NaN
humidity  0.040452  0.069717  0.021627  0.005625 -0.001927  0.014431    NaN
pressure  0.085183 -0.032049 -0.068296 -0.014838 -0.008821  0.032056    NaN
pitch     0.024460 -0.053634 -0.029345  0.040685  0.041674 -0.024081    NaN
roll      0.057330 -0.049233 -0.153524  0.139427  0.134319 -0.078113    NaN
yaw      -0.028267  0.078585  0.068321 -0.021071 -0.009650  0.064290    NaN
mag_x    -0.097520  0.168764  0.115423 -0.017739 -0.006722  0.008456    NaN
mag_y     0.046693 -0.035111 -0.022579 -0.084045 -0.061460  0.115327    NaN
mag_z     0.001699 -0.020016 -0.006496 -0.092749 -0.060097  0.101276    NaN
accel_x   1.000000 -0.197363 -0.174005 -0.016811 -0.013694 -0.017850    NaN
accel_y  -0.197363  1.000000  0.424272 -0.023942 -0.054733  0.014870    NaN
accel_z  -0.174005  0.424272  1.000000  0.006313 -0.011883 -0.015390    NaN
gyro_x   -0.016811 -0.023942  0.006313  1.000000  0.802471 -0.012705    NaN
gyro_y   -0.013694 -0.054733 -0.011883  0.802471  1.000000 -0.043332    NaN
gyro_z   -0.017850  0.014870 -0.015390 -0.012705 -0.043332  1.000000    NaN
reset           NaN       NaN       NaN       NaN       NaN       NaN    NaN
```

5.6.6 5. Calcolare nuove colonne

E' possibile ottenere nuove colonne effettuando calcoli da campi di altri colonne in modo molto naturale. Per esempio, qua ricaviamo la nuova colonna `mag_tot`, cioè il campo magnetico assoluto rilevato dalla stazione spaziale ricavandolo a partire da `mag_x`, `mag_y`, e `mag_z`, e poi la plottiamo:

[38]:
```
df['mag_tot'] = df['mag_x']**2 + df['mag_y']**2 + df['mag_z']**2
df.mag_tot.plot()
```

[38]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f629f33b4a8>
```

Troviamo dove il campo magnetico era al massimo:

[39]:
```
df['time_stamp'][(df.mag_tot == df.mag_tot.values.max())]
```

[39]:
```
96156    2016-02-27 16:12:31
Name: time_stamp, dtype: object
```

Inserendo il valore trovato sul sito isstracker.com/historical[376], possiamo rilevare le posizioni in cui il campo magnetico è più forte.

5.6.7 6. Filtrare per valori testuali

Ci sono vari modi per filtrare in base a testo (vedi documentazione[377]), qua facciamo solo un esempio. Quando vogliamo filtrare per valori testuali, possiamo usare `.str.contains`, qua per esempio selezioniamo tutte le rilevazioni degli ultimi giorni di febbraio (che hanno quindi il timestamp che contiene `2016-02-2`) :

[40]:
```
df[  df['time_stamp'].str.contains('2016-02-2')  ]
```

[40]:
```
        ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll  \
30442    30443     32.30   28.12   25.59     45.05   1008.01   1.47  51.82
30443    30444     32.25   28.13   25.59     44.82   1008.02   0.81  51.53
30444    30445     33.07   28.13   25.59     45.08   1008.09   0.68  51.69
30445    30446     32.63   28.10   25.60     44.87   1008.07   1.42  52.13
30446    30447     32.55   28.11   25.60     44.94   1008.07   1.41  51.86
…        …       …      …      …       …       …    …     …
110864  110865     31.56   27.52   24.83     42.94   1005.83   1.58  49.93
110865  110866     31.55   27.50   24.83     42.72   1005.85   1.89  49.92
110866  110867     31.58   27.50   24.83     42.83   1005.85   2.09  50.00
110867  110868     31.62   27.50   24.83     42.81   1005.88   2.88  49.69
110868  110869     31.57   27.51   24.83     42.94   1005.86   2.17  49.77

           yaw      mag_x  …   accel_y   accel_z    gyro_x    gyro_y  \
30442    51.18   9.215883  …  0.018792  0.014558 -0.000042  0.000275
30443    52.21   8.710130  …  0.019290  0.014667  0.000260  0.001011
30444    57.36   7.383435  …  0.018714  0.014598  0.000299  0.000343
30445    59.95   7.292313  …  0.018857  0.014565  0.000160  0.000349
30446    61.83   6.699141  …  0.018871  0.014564 -0.000608 -0.000381
…        …        …  …       …       …       …       …
110864  129.60 -15.169673  …  0.017743  0.014646 -0.000264  0.000206
110865  130.51 -15.832622  …  0.017570  0.014855  0.000143  0.000199
110866  132.04 -16.646212  …  0.017657  0.014799  0.000537  0.000257
110867  133.00 -17.270447  …  0.017635  0.014877  0.000534  0.000456
110868  134.18 -17.885872  …  0.017261  0.014380  0.000459  0.000076

          gyro_z  reset           time_stamp  Too hot  check_p      mag_tot
30442   0.000157      0  2016-02-20 00:00:00     True    sotto   269.091903
30443   0.000149      0  2016-02-20 00:00:10     True    sotto   260.866157
30444  -0.000025      0  2016-02-20 00:00:41     True    sotto   265.421154
30445  -0.000190      0  2016-02-20 00:00:50     True    sotto   269.572476
30446  -0.000243      0  2016-02-20 00:01:01     True    sotto   262.510966
…          …    …                   …       …       …          …
110864  0.000196      0  2016-02-29 09:24:21      NaN    sotto   996.676408
110865 -0.000024      0  2016-02-29 09:24:30      NaN    sotto  1022.779594
110866  0.000057      0  2016-02-29 09:24:41      NaN    sotto  1048.121268
110867  0.000195      0  2016-02-29 09:24:50      NaN    sotto  1073.629703
```

(continues on next page)

[376] http://www.isstracker.com/historical
[377] https://pandas.pydata.org/pandas-docs/stable/text.html

(continued from previous page)

```
110868  0.000030      0  2016-02-29 09:25:00      NaN    sotto  1095.760426

[80427 rows x 23 columns]
```

5.6.8 7. Esercizi col meteo

ESERCIZIO 7.1: Mettere in grafico l'andamento delle temperature del dataframe *meteo*

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[41]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

# scrivi qui

meteo.Temp.plot()
```

[41]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6291f3b2e8>
```

[41]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

# scrivi qui
```

[41]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6291f3b2e8>
```

ESERCIZIO 7.2: Nello stesso grafico mostrare l'andamento della pressione e la quantità delle precipitazioni.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[42]:
```
# scrivi qui

meteo.Temp.plot(label="Temperatura", legend=True)
meteo.Pioggia.plot(label="Pioggia", legend=True)
meteo.Pressione.plot(secondary_y=True, label="Pressione", legend=True);
```

[42]:
```
# scrivi qui
```

⊛⊛⊛ **ESERCIZIO 7.3**: Calcolare la temperatura media giornaliera e mostrarla nel grafico.

SUGGERIMENTO 1: aggiungere la colonna `'Giorno'` estraendo solo il giorno dalla data. Per farlo usare la funzione `.str` applicata a tutta la colonna.

SUGGERIMENTO 2: Successivamente usare un ciclo `for` per ciclare sui giorni. Tipicamente usare un `for` non è una buona idea con Pandas, perchè con dataset larghi ci può voler molto ad eseguire gli aggiornamenti. Comunque, dato che questo dataset è piccolo a sufficienza, puoi provare ad usare un `for` per ciclare sui giorni e dovresti ottenere i risultati in un tempo ragionevole

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[43]:
```
# scrivi qui

meteo = pd.read_csv('meteo.csv', encoding='UTF-8')
meteo['Giorno'] = meteo['Data'].str[0:10]
print()
print("CON GIORNO")
print(meteo.head())
for giorno in meteo['Giorno']:
    temp_media_giorno = meteo[(meteo.Giorno == giorno)].Temp.values.mean()
    meteo.loc[(meteo.Giorno == giorno),'Temp_media_giorno']= temp_media_giorno
print()
print("CON TEMPERATURA MEDIA")
print(meteo.head())
meteo.Temp.plot(label="Temperatura", legend=True)
meteo.Temp_media_giorno.plot(label="Temperatura media", legend=True)
```

```
CON GIORNO
                Data  Pressione  Pioggia  Temp      Giorno
0  01/11/2017 00:00      995.4      0.0   5.4  01/11/2017
1  01/11/2017 00:15      995.5      0.0   6.0  01/11/2017
2  01/11/2017 00:30      995.5      0.0   5.9  01/11/2017
3  01/11/2017 00:45      995.7      0.0   5.4  01/11/2017
4  01/11/2017 01:00      995.7      0.0   5.3  01/11/2017

CON TEMPERATURA MEDIA
```

(continues on next page)

(continued from previous page)

```
                Data  Pressione  Pioggia  Temp      Giorno  Temp_media_giorno
0  01/11/2017 00:00      995.4      0.0   5.4  01/11/2017           7.983333
1  01/11/2017 00:15      995.5      0.0   6.0  01/11/2017           7.983333
2  01/11/2017 00:30      995.5      0.0   5.9  01/11/2017           7.983333
3  01/11/2017 00:45      995.7      0.0   5.4  01/11/2017           7.983333
4  01/11/2017 01:00      995.7      0.0   5.3  01/11/2017           7.983333
```

[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6291df2400>
```

[43]:
```
# scrivi qui
```

```
CON GIORNO
                Data  Pressione  Pioggia  Temp      Giorno
0  01/11/2017 00:00      995.4      0.0   5.4  01/11/2017
1  01/11/2017 00:15      995.5      0.0   6.0  01/11/2017
2  01/11/2017 00:30      995.5      0.0   5.9  01/11/2017
3  01/11/2017 00:45      995.7      0.0   5.4  01/11/2017
4  01/11/2017 01:00      995.7      0.0   5.3  01/11/2017

CON TEMPERATURA MEDIA
                Data  Pressione  Pioggia  Temp      Giorno  Temp_media_giorno
0  01/11/2017 00:00      995.4      0.0   5.4  01/11/2017           7.983333
1  01/11/2017 00:15      995.5      0.0   6.0  01/11/2017           7.983333
2  01/11/2017 00:30      995.5      0.0   5.9  01/11/2017           7.983333
3  01/11/2017 00:45      995.7      0.0   5.4  01/11/2017           7.983333
4  01/11/2017 01:00      995.7      0.0   5.3  01/11/2017           7.983333
```

[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6291df2400>
```

5.6.9 8. Esercizi Inquinanti aria

⊛ **ESERCIZIO 8.1** Caricare in pandas il file aria.csv, che riporta i dati orari delle stazioni di monitoraggio della qualità dell'aria della Provincia Autonoma di Trento validati dall'Agenzia per l'ambiente

IMPORTANTE: metti il dataframe nella variabile `aria`, così da non confonderlo coi dataframe precedenti

Fonte: dati.trentino.it[378]

IMPORTANTE: metti come encoding `'latin-1'` (altrimenti a seconda del tuo sistema operativo potrebbe non caricarlo dando strani messaggi d'errore)

IMPORTANTE: se ricevi altri strani messaggi d'errore, aggiungi anche il parametro `engine=python`

[44]:
```
import pandas as pd    # importiamo pandas e per comodità lo rinominiamo in  'pd'
import numpy as np     # importiamo numpy e per comodità lo rinominiamo in 'np'

# ricordatevi l'encoding !
aria = pd.read_csv('aria.csv', encoding='latin-1')
aria.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20693 entries, 0 to 20692
Data columns (total 6 columns):
Stazione           20693 non-null object
Inquinante         20693 non-null object
Data               20693 non-null object
Ora                20693 non-null int64
Valore             20693 non-null float64
Unità di misura    20693 non-null object
dtypes: float64(1), int64(1), object(4)
memory usage: 970.1+ KB
```

⊛ **ESERCIZIO 8.2** Trova la media dei valori di inquinanti `PM10` al `Parco S. Chiara` (media su tutte le giornate)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[378] https://dati.trentino.it/dataset/qualita-dell-aria-rilevazioni-delle-stazioni-monitoraggio

[45]:
```
# scrivi qui

aria[(aria.Stazione == 'Parco S. Chiara') & (aria.Inquinante == 'PM10')].Valore.
↪values.mean()
```

[45]:
```
11.385752688172044
```

[45]:
```
# scrivi qui
```

[45]:
```
11.385752688172044
```

[46]:
```
aria[(aria.Stazione == 'Parco S. Chiara') & (aria.Inquinante == 'PM10')]
```

[46]:
```
            Stazione Inquinante        Data  Ora  Valore Unità di misura
0    Parco S. Chiara       PM10  2019-05-04    1    17.0           µg/mc
1    Parco S. Chiara       PM10  2019-05-04    2    19.0           µg/mc
2    Parco S. Chiara       PM10  2019-05-04    3    17.0           µg/mc
3    Parco S. Chiara       PM10  2019-05-04    4    15.0           µg/mc
4    Parco S. Chiara       PM10  2019-05-04    5    13.0           µg/mc
..               …          …          …   …     …              …
739  Parco S. Chiara       PM10  2019-06-03   20    33.0           µg/mc
740  Parco S. Chiara       PM10  2019-06-03   21    40.0           µg/mc
741  Parco S. Chiara       PM10  2019-06-03   22    31.0           µg/mc
742  Parco S. Chiara       PM10  2019-06-03   23    31.0           µg/mc
743  Parco S. Chiara       PM10  2019-06-03   24    31.0           µg/mc

[744 rows x 6 columns]
```

[47]:
```
aria[(aria.Stazione == 'Parco S. Chiara') & (aria.Inquinante == 'PM10') & (aria.Data␣
↪== '2019-05-07')]
```

[47]:
```
           Stazione Inquinante        Data  Ora  Valore Unità di misura
72  Parco S. Chiara       PM10  2019-05-07    1     7.0           µg/mc
73  Parco S. Chiara       PM10  2019-05-07    2     9.0           µg/mc
74  Parco S. Chiara       PM10  2019-05-07    3    10.0           µg/mc
75  Parco S. Chiara       PM10  2019-05-07    4     9.0           µg/mc
76  Parco S. Chiara       PM10  2019-05-07    5    12.0           µg/mc
77  Parco S. Chiara       PM10  2019-05-07    6    16.0           µg/mc
78  Parco S. Chiara       PM10  2019-05-07    7    19.0           µg/mc
79  Parco S. Chiara       PM10  2019-05-07    8    20.0           µg/mc
80  Parco S. Chiara       PM10  2019-05-07    9    18.0           µg/mc
81  Parco S. Chiara       PM10  2019-05-07   10    12.0           µg/mc
82  Parco S. Chiara       PM10  2019-05-07   11     9.0           µg/mc
83  Parco S. Chiara       PM10  2019-05-07   12     9.0           µg/mc
84  Parco S. Chiara       PM10  2019-05-07   13    11.0           µg/mc
85  Parco S. Chiara       PM10  2019-05-07   14    12.0           µg/mc
86  Parco S. Chiara       PM10  2019-05-07   15    14.0           µg/mc
87  Parco S. Chiara       PM10  2019-05-07   16    16.0           µg/mc
88  Parco S. Chiara       PM10  2019-05-07   17    15.0           µg/mc
89  Parco S. Chiara       PM10  2019-05-07   18    16.0           µg/mc
90  Parco S. Chiara       PM10  2019-05-07   19    18.0           µg/mc
91  Parco S. Chiara       PM10  2019-05-07   20    21.0           µg/mc
92  Parco S. Chiara       PM10  2019-05-07   21    22.0           µg/mc
```

(continues on next page)

(continued from previous page)

```
93  Parco S. Chiara       PM10  2019-05-07   22    20.0          µg/mc
94  Parco S. Chiara       PM10  2019-05-07   23    22.0          µg/mc
95  Parco S. Chiara       PM10  2019-05-07   24    22.0          µg/mc
```

⊛ **ESERCIZIO 8.3**: Usando `plt.plot` come visto in un *esempio precedente* (quindi passandogli direttamente le serie rilevanti di Pandas), mostra in un grafico l'andamento dei valori di inquinanti `PM10` nella giornata del 7 Maggio 2019

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[48]:
```
# scrivi qui

import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

filtrato = aria[(aria.Stazione == 'Parco S. Chiara') & (aria.Inquinante == 'PM10') &␣
↪(aria.Data == '2019-05-07')]

plt.plot(filtrato['Ora'], filtrato['Valore'] )
plt.title('PM10 7 Maggio 2019')
plt.xlabel('Ora')
plt.show()
```

[48]:
```
# scrivi qui
```

5.6.10 9. GeoPandas

ATTENZIONE: Questa parte del tutorial è SPERIMENTALE, a breve aggiungeremo commenti

Pandas è anche molto comodo per gestire dati geografici, con l'estensione GeoPandas[379]

Installiamola subito:

Anaconda:

```
conda install geopandas
```

e poi

```
conda install -c conda-forge descartes
```

Linux/Mac (`--user` installa nella propria home):

- `python3 -m pip install --user geopandas descartes`

5.6.11 Un esempio semplice con GeoPandas

Faremo un esempio mostrando regioni italiane colorate in base alla popolazione residente:

[379] http://geopandas.org/

Quando si parla di mappe, tipicamente vogliamo mostrare delle regioni o nazioni colorate in base ad un valore associato ad ogni zona. Quindi servono sempre almeno due cose:

1. le forme geometriche delle zone da raffigurare
2. i valori da associare ad ogni zona da far corrispondere alle gradazioni di colore

Tipicamente questi dati vengono presi da almeno due dataset diversi, uno geografico e uno di statistiche, ma vi troverete spesso con il problema che nel dataset geografico le zone vengono chiamate con un nome o codice diverso da quello del dataset con le statistiche.

Divideremo l'esempio in due parti:

- nella prima, useremo tabelle già ripulite che trovate nella stessa cartella di questo foglio. Questo ci permetterà di comprendere i meccanismi di base di GeoPandas e del *fuzzy matching*
- nella seconda parte, proporremo di risolvere un esercizio completo che prevede lo scaricamento online del file html e pulizia

Vediamo il nostro esempio, in cui le zone geografiche vengono prese dal sito dell'istat da file geografici in formato shapefile. Il file è già salvato nella cartella qui: reg2011/reg2011_g.shp , se volete vedere dove era online guardate basi territoriali qua: https://www.istat.it/it/archivio/104317

Leggere shapefiles in GeoPandas

Leggiamo con geopandas lo shapefile:

[49]:
```
import geopandas as gpd

df_regioni = gpd.read_file(filename="reg2011/reg2011_g.shp")
df_regioni.head()
```

[49]:
```
   COD_REG                                          NOME_REG    SHAPE_Leng  \
0        1                                          PIEMONTE  1.236869e+06
1        2  VALLE D'AOSTA/VALLÉE D'AOSTE\r\nVALLE D'AOSTA/…  3.111651e+05
2        3                                         LOMBARDIA  1.411265e+06
3        4                       TRENTINO-ALTO ADIGE/SUDTIROL  8.005341e+05
4        5                                            VENETO  1.057856e+06

     SHAPE_Area                                           geometry
0  2.539410e+10  POLYGON ((457832.312 5145701.000, 458745.249 5…
```

(continues on next page)

(continued from previous page)

```
1  3.259041e+09  POLYGON ((390734.999 5091965.001, 390830.999 5…
2  2.386270e+10  MULTIPOLYGON (((595736.187 5163715.001, 596126…
3  1.360802e+10  POLYGON ((743386.080 5219948.900, 743472.190 5…
4  1.840550e+10  POLYGON ((768209.001 5175597.001, 768220.251 5…
```

Oltre alla solita tabella di Pandas, notiamo che tra le colonne ci sono dei codice `COD_REG` per identificare le regioni, i loro nomi `NOME_REG` e la geometria `geometry`. Chiamando `plot()` sul dataframe di geopandas possiamo vedere la cartina risultante:

[50]:
```
%matplotlib inline
df_regioni.plot()
```

[50]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62d45b27b8>
```

Prendiamo statistiche da visualizzare

Nel nostro esempio, estraiamo statistiche sulla popolazione delle regioni italiane da una pagina HTML. Metteremo poi i dati estratti in un dataframe Pandas (non GeoPandas) chiamato `df_popolazione`. Per comodità abbiamo salvato tale pagina nel file popolazione.html (se volete vedere la versione online, andate su questo sito: https://www.tuttitalia.it/regioni/popolazione)

> **ATTENZIONE**: Per il momento puoi ignorare il codice che segue, ci serve solo per caricare i dati nel dataframe `df_popolazione`

[51]:
```
import pandas as pd


# prende la riga di una tabella html, e ritorna un dizionario con i dati estratti
def estrai_dizionario(riga_html):
    colonne = riga_html.select('td')
    return dict(name=colonne[1].text,
                population=colonne[2].text.replace('.', '').replace(',', '.'),
                area=colonne[3].text.replace('.', '').replace(',', '.'))

# Estrae la popolazione per regione da popolazione.html, e restituisce un dataframe␣
↪Pandas (non GeoPandas)
```

(continues on next page)

(continued from previous page)

```
def estrai_popolazione():
    from bs4 import BeautifulSoup
    with open('popolazione.html', encoding='utf-8') as f:
        testo = f.read()
        listona = []    # listona di dizionari, ogni dizionario rappresenta una riga
        # usiamo il parser html5lib invece di lxml perchè il sito è complesso
        soup = BeautifulSoup(testo, 'html5lib')
        righe_html = soup.select('table.ut tr')[1:21]
        for riga_html in righe_html:
            listona.append(estrai_dizionario(riga_html))
        return pd.DataFrame(listona)
```

Vediamo qui il contenuto del file:

[52]:
```
df_popolazione = estrai_popolazione()
df_popolazione
```

[52]:

	name	population	area
0	Lombardia	10019166	23863.65
1	Lazio	5898124	17232.29
2	Campania	5839084	13670.95
3	Sicilia	5056641	25832.39
4	Veneto	4906210	18345.35
5	Em.-Romagna	4448841	22452.78
6	Piemonte	4392526	25387.07
7	Puglia	4063888	19540.90
8	Toscana	3742437	22987.04
9	Calabria	1965128	15221.90
10	Sardegna	1653135	24100.02
11	Liguria	1565307	5416.21
12	Marche	1538055	9401.38
13	Abruzzo	1322247	10831.84
14	Friuli VG	1219191	7924.36
15	Trentino-AA	1062860	13605.50
16	Umbria	888908	8464.33
17	Basilicata	570365	10073.32
18	Molise	310449	4460.65
19	V. d'Aosta	126883	3260.90

Se compariamo i nomi in questa tabella con il dataframe della prima, notiamo subito che parecchi nomi non sono identici. Per esempio, nello shapefile troviamo `TRENTINO-ALTO ADIGE/SUDTIROL` mentre nelle statistiche c'è `Trentino-AA`. Volendo creare una tabella unica, occorrerà quindi fare integrazione dati cercando di ottenere un *matching* tra le righe dei due dataset. Per venti regioni potremmo farla a mano ma chiaramente farlo per migliaia di righe sarebbe estremamente oneroso. Per agevolare questa operazione, ci conviene eseguire una cosiddetta *fuzzy join*, che cerca stringhe simili nei due dataset e in base ad un misura di similarità tra stringhe stabilisce come associare righe della prima tabella a righe della seconda.

Per

[53]:
```
def fuzzy_join(df_geo, df_right, name_left, name_right):
    """ Prende:
        - un data frame di geo pandas df_geo che contiene una colonna chiamata name_
↪left
        - un'altro dataframe generico df_right che contiene una colonna chiamata␣
↪name_right
```

(continues on next page)

(continued from previous page)

```
        Ritorna :
        - un nuovo dataframe che è la join dei due dataframe in base alla similirità␣
↪tra
          le colonne name_left e name_right

          ATTENZIONE: a volte l'agoritmo di similarità può confondersi e considerare␣
↪uguale due nomi
                      che invece dovrebbero essere distinti !
                      Per quanto possibile, verificare sempre i risultati manualmente.
    """
    from functools import partial
    from itertools import product
    import difflib
    import heapq
    #from pprint import pprint

    df1 = df_geo.set_index(name_left)
    df1.index = df1.index.str.lower()
    df2 = df_right.set_index(name_right)
    df2.index = df2.index.str.lower()


    def get_matcher_smart(dfl, dfr):
        heap = []
        for l, r in product(dfl.index, dfr.index):
            sm = difflib.SequenceMatcher(lambda x: ' .\n\t', l, r)
            heapq.heappush(heap, (1. - sm.quick_ratio(), l, r))
        ass_l, ass_r, ass_map = set(), set(), {}
        while len(ass_map) < len(dfl):
            score, l, r = heapq.heappop(heap)
            if not (l in ass_l or r in ass_r):
                ass_map[l] = r
                ass_l.add(l)
                ass_r.add(r)
        #pprint(ass_map)
        return dfl.index.map(lambda x: ass_map[x])

    df1.index = get_matcher_smart(df1, df2)

    return df1.join(df2)
```

[54]:
```
tabellona = fuzzy_join(df_regioni, df_popolazione, 'NOME_REG', 'name')

tabellona
```

[54]:
```
             COD_REG    SHAPE_Leng    SHAPE_Area  \
NOME_REG
piemonte           1  1.236869e+06  2.539410e+10
v. d'aosta         2  3.111651e+05  3.259041e+09
lombardia          3  1.411265e+06  2.386270e+10
trentino-aa        4  8.005341e+05  1.360802e+10
veneto             5  1.057856e+06  1.840550e+10
friuli vg          6  6.674897e+05  7.864294e+09
liguria            7  8.342245e+05  5.415465e+09
em.-romagna        8  1.164723e+06  2.245147e+10
```

(continues on next page)

(continued from previous page)

```
toscana            9  1.316658e+06  2.298443e+10
umbria            10  6.203152e+05  8.464008e+09
marche            11  6.292090e+05  9.401178e+09
lazio             12  1.055355e+06  1.722762e+10
abruzzo           13  6.145137e+05  1.082910e+10
molise            14  4.338747e+05  4.461149e+09
campania          15  8.923791e+05  1.366399e+10
puglia            16  1.176243e+06  1.953708e+10
basilicata        17  6.142192e+05  1.007326e+10
calabria          18  8.381944e+05  1.521668e+10
sicilia           19  1.339974e+06  2.582478e+10
sardegna          20  1.460657e+06  2.409417e+10

                                                      geometry population  \
NOME_REG
piemonte     POLYGON ((457832.312 5145701.000, 458745.249 5…    4392526
v. d'aosta   POLYGON ((390734.999 5091965.001, 390830.999 5…     126883
lombardia    MULTIPOLYGON (((595736.187 5163715.001, 596126…   10019166
trentino-aa  POLYGON ((743386.080 5219948.900, 743472.190 5…    1062860
veneto       POLYGON ((768209.001 5175597.001, 768220.251 5…    4906210
friuli vg    MULTIPOLYGON (((852211.994 5080672.916, 852270…    1219191
liguria      MULTIPOLYGON (((400403.625 4851436.938, 400257…    1565307
em.-romagna  MULTIPOLYGON (((760714.748 4937319.399, 760723…    4448841
toscana      MULTIPOLYGON (((593650.250 4867988.000, 593553…    3742437
umbria       MULTIPOLYGON (((771407.451 4833282.073, 771402…     888908
marche       MULTIPOLYGON (((863162.283 4840139.829, 863195…    1538055
lazio        MULTIPOLYGON (((802704.568 4594643.932, 802540…    5898124
abruzzo      POLYGON ((901880.250 4760558.000, 901910.750 4…    1322247
molise       POLYGON ((984090.000 4670910.250, 985154.250 4…     310449
campania     MULTIPOLYGON (((925294.173 4528798.912, 925329…    5839084
puglia       MULTIPOLYGON (((1141891.750 4588535.750, 11418…    4063888
basilicata   MULTIPOLYGON (((1074503.688 4446135.938, 10744…     570365
calabria     MULTIPOLYGON (((1084888.411 4414364.607, 10848…    1965128
sicilia      MULTIPOLYGON (((803299.052 4188096.575, 803275…    5056641
sardegna     MULTIPOLYGON (((432678.313 4492760.313, 432811…    1653135

                 area
NOME_REG
piemonte     25387.07
v. d'aosta    3260.90
lombardia    23863.65
trentino-aa  13605.50
veneto       18345.35
friuli vg     7924.36
liguria       5416.21
em.-romagna  22452.78
toscana      22987.04
umbria        8464.33
marche        9401.38
lazio        17232.29
abruzzo      10831.84
molise        4460.65
campania     13670.95
puglia       19540.90
basilicata   10073.32
calabria     15221.90
sicilia      25832.39
```

(continues on next page)

(continued from previous page)

```
sardegna     24100.02
```

[55]:
```
tabellona.plot(column='population', cmap='OrRd', edgecolor='k', legend=False)
```

[55]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62bb900668>
```

5.6.12 Esempio di integrazione

ATTENZIONE: QUESTA PARTE E' INCOMPLETA

Vediamo l'esempio di integrazione completo. Ti serviranno anche `requests`, `beautifulsoup4`, e `html5lib`. Installali così:

Anaconda:

- `conda install requests beautifulsoup4 html5lib`

Linux/Mac (`--user` installa nella propria home):

- `python3 -m pip install --user requests beautifulsoup4 html5lib`

Per fare un esempio di integrazione, useremo una pagina HTML con i dati delle regioni italiane:

- https://www.tuttitalia.it/regioni/popolazione/

Per capire come estrarre la popoloziona dall'HTML, guarda il tutorial sull'estrazione[380]

Nel menu basi territoriali qua invece abbiamo dei file geografici in formato shapefile delle regioni:

- basi territoriali https://www.istat.it/it/archivio/104317

[56]:
```
# Scarica la pagina HTML della popolazione, e la salva nel file 'popolazione.html'
def scarica_popolazione():
    from bs4 import BeautifulSoup
    import requests

    r = requests.get("https://www.tuttitalia.it/regioni/popolazione/")
```

(continues on next page)

[380] https://it.softpython.org/extraction/extraction-sol.html

(continued from previous page)

```
    if r.status_code == 200:
        testo = r.text
        with open('popolazione.html', 'w', encoding='utf-8') as f:
            f.write(testo)
            print("Ho salvato il file 'popolazione.html'")
    else:
        # se il codice non è 200, qualcosa è probabilmente andato storto
        # e blocchiamo l'esecuzione dello script
        raise Exception('Errore durante lo scaricamento : %s' % r)

# scarica_popolazione()
```

[ ]:

```

```

5.7 Information retrieval

5.7.1 Scarica zip esercizi

Naviga file online[381]

> **ATTENZIONE**: tutorial in progress !
>
> Questo tutorial è pensato per dare un idea rapida dei problemi principali dell'information retrieval ma è ancora incompleto e spesso presenta i concetti in modo molto semplificato. Se l'argomento t'appassiona non mancano buoni libri sul tema!

Information retrieval è l'attività che svolgono i sistemi informatici quando devono recuperare *rapidamente* delle informazioni *rilevanti* . Perchè questi due aggettivi?

Al giorno d'oggi molte aziende hanno a che fare con una mole di dati rilevante - i famosi *big data*. Sia i giganti dell'*Information Technology* (*IT*) come Google e Amazon che aziende di dimensioni più modeste per varie ragioni devono fornire ai clienti dei sistemi di ricerca efficienti. Nel caso di Google i clienti vogliono trovare tutte le pagine nel web che contengono un certo testo. Per Amazon, il cliente desidera conoscere tutti i prodotti in una certa categoria, o prodotti *simili* ad uno già cercato.

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
information-retrieval
    information-retrieval.ipynb
    information-retrieval-sol.ipynb
    jupman.py
```

> **ATTENZIONE**: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

[381] https://github.com/DavidLeoni/softpython-it/tree/master/information-retrieval

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `information-retrieval.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

5.7.2 Costruiamo il nostro motore di ricerca

Consideriamo in particolare il problema che risolvono i motori di ricerca come Google.

Supponiamo di avere tante pagine web (tutto il web !): per ogni pagina abbiamo l'indirizzo della pagina (come http://it.softpython.org) e il testo che contiene in formato HTML, per esempio:

```
<h2>Presentazione</h2>
<p>Il corso SoftPython fornisce un'introduzione al processamento dati usando Python,␣
↪un linguaggio di programmazione popolare sia nell'industria che nell'ambito della␣
↪ricerca. Fra le varie cose tratteremo Matplotlib.</p>
```

Se hai guardato il tutorial sull'estrazione dati[382] dovresti già avere un'idea di cosa è l'HTML. Se non l'hai fatto niente paura: puoi tranquillamente ignorare le strane sequenze racchiuse tra minore e maggiore come `<h2>`, tieni solo presente che si chiamano *tag HTML*. Parlando in genere, le useremo come esempio del fatto che quando si cerca dentro del testo si possono trovare cose strane o indesiderate, e un sistema di ricerca fatto bene deve essere pronto alle sorprese.

L'utente vuole trovare tutte le pagine contenti un certo testo, per esempio sul sito di SoftPython potrebbe voler cercare dove ci sono spiegazioni su `grafici con matplotlib`

- INPUT: stringa di ricerca (per es: `grafici con matplotlib`
- OUTPUT: tutte le pagine che contengono la stringa (in questo caso `grafici con matplotlib`

Se hai seguito qualcuno dei tutorial sul nostro sito, forse ti potrebbe venire in mente già una o due soluzioni a questo problema.

DOMANDA: Prima di proseguire, pensa a

- come rappresenteresti i dati delle pagine in Python?
- che funzione Python definiresti per fare la ricerca?
  - quali sarebbero i parametri della funzione?
  - cosa varia ad ogni chiamata e cosa invece è più 'statico' ?

Non ti preoccupare delle *performance* - quando si progetta un *algoritmo* all'inizio conviene capire bene cosa si vuole. Solo dopo aver realizzato un prototipo e nel caso sia troppo lento, ci si comincia a preoccupare delle performance !

[382] https://it.softpython.org/extraction/extraction-sol.html

Una prima soluzione

Vediamone qua una possibile soluzione al problema.

Rappresentare una pagina

Cominciamo a rappresentare le pagine web. Per farlo, possiamo usare unsemplice dizionario con le chiavi `url` (l'indirizzo web) e `contenuto` per il testo:

```
{
    'url': 'http://it.softpython.org',
    'contenuto' : '<h2>Presentazione</h2><p>Il corso SoftPython fornisce␣
↪un'introduzione al processamento dati usando Python, un linguaggio di␣
↪programmazione popolare sia nell'industria che nell'ambito della ricerca. Fra le␣
↪varie cose tratteremo Matplotlib.</p>'
}
```

Un database per le pagine

Il web ha tante pagine, quindi vorremo mettere la collezione di pagine in un contenitore. Ai fini dell'esempio proviamo ad usare una lista. La popoleremo con due pagine, quella principale[383] e la pagina chiamata Visualizzazione[384]. Infine assegnamo la lista alla variabile `pagine` (ricordiamo che il testo che mettiamo è semplificato rispetto a quello effettivo che trovate sul sito)

[1]:
```
pagine = [
            {
                'url': 'http://it.softpython.org',
                'contenuto' : '<h2>Presentazione</h2><p>Il corso SoftPython fornisce␣
↪un'introduzione al processamento dati usando Python, un linguaggio di␣
↪programmazione popolare sia nell'industria che nell'ambito della ricerca. Fra le␣
↪varie cose tratteremo Matplotlib.</p>'
            },
            {
                'url': 'https://it.softpython.org/visualization/visualization-sol.html
↪',
                'contenuto' : '<h2>Visualizzazione</h2> <p>Excel ci permette di␣
↪creare molti tipi di visualizzazione ma a volte è limitato, perciò useremo Python␣
↪facendo grafici in Matplotlib. Matplotlib è una libreria eccezionale che consente␣
↪di creare qualunque visualizzazione desideriamo. Procediamo dunque a importare␣
↪matplotlib in Python:</p>'
            }
        ]
```

Supponiamo che tutto il web sia costituito dalle due pagine precedenti. Supponiamo anche che le pagine non cambino mai nel tempo. Abbiamo creato il nostro primo rudimentale *database* del web!

[383] https://it.softpython.org
[384] https://it.softpython.org/visualization/visualization-sol.html

Implementiamo la funzione di ricerca

Ma come possiamo implementare la ricerca?

[2]:
```
def ricerca(stringa):
    risultati = []              # all'inizio non sappiamo quali pagine contengono le␣
↪parole cercate
    for pagina in pagine:                     # scorri le pagine
        if stringa in pagina['contenuto']:    # se la stringa cercata è nel contenuto␣
↪della pagina ...
            risultati.append(pagina)          # aggiungi la pagina ai risultati␣
↪collezionati finora
    return risultati                          # ritorna le pagine collezionate finora
```

Proviamola con una sola parola chiave `grafici`, dovrebbe ritornarci la pagina 1-esima sulla visualizzazione:

[3]:
```
ricerca("grafici")
```

[3]:
```
[{'contenuto': '<h2>Visualizzazione</h2> <p>Excel ci permette di creare molti tipi di␣
↪visualizzazione ma a volte è limitato, perciò useremo Python facendo grafici in␣
↪Matplotlib. Matplotlib è una libreria eccezionale che consente di creare qualunque␣
↪visualizzazione desideriamo. Procediamo dunque a importare matplotlib in Python:</p>
↪',
  'url': 'https://it.softpython.org/visualization/visualization-sol.html'}]
```

Funziona ! Baldanzosi col risultato del nostro primo esperimento, proviamo a cerca la stringa completa che ci eravamo prefissati all'inizio, `grafici con matplotlib`:

[4]:
```
ricerca("grafici con matplotlib")
```

[4]:
```
[]
```

Nulla ! Come mai?

DOMANDA: Prima di proseguire, osserva bene il testo semplificato della pagina Visualizzazione e prova a dare una risposta. Ci sono due problemi principali. Come li risolveresti ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

Se osserviamo bene il testo della pagina Visualizzazione, vediamo che riporta in fondo `"come fare grafici in Matplotlib"`

Saltano fuori diversi problemi:

1. il nostro utente sta cercando `"grafici con matplotlib"` usando `"con"` ma nel testo della pagina c'è scritto `"grafici in Matplotlib"` con `"in"`
2. l'utente cerca `"matplotlib"` in minuscolo, ma nella pagina è riportato `"Matplotlib"` in maiuscolo.

DOMANDA: Come risolveresti i problemi qua sopra?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

1. Preposizioni come `"con"`, `"in"`, `"di"`, `"fra"` etc. molto spesso non sono rilevanti ai fini di una ricerca. Nel gergo dell'Information retrieval, vengono chiamate stopwords[385]. Per garantire una ricerca più efficace, potresti eliminarle dalla stringa di ricerca dell'utente. Ai fini della ricerca, forse avrebbe senso eliminarle anche dal testo delle pagine ?
2. Per risolvere problemi di maiuscole/minuscole, spesso conviene convertire la stringa di ricerca tutta in minuscolo, e fare lo stesso con il testo della pagina. Per farlo, puoi usare il metodo `lower()` delle stringhe. **NOTA**: `lower()`, come *tutti* i metodi delle stringhe, ritorna una **nuova** stringa e non modifica l'originale !!

Una ricerca più efficace

Accediamo al contenuto della prima pagina:

```
[5]: pagine[0]["contenuto"]
```

```
[5]: '<h2>Presentazione</h2><p>Il corso SoftPython fornisce un'introduzione al␣
↪processamento dati usando Python, un linguaggio di programmazione popolare sia␣
↪nell'industria che nell'ambito della ricerca. Fra le varie cose tratteremo␣
↪Matplotlib.</p>'
```

Se cerco `"matplotlib"` in maiuscolo non lo troverò:

```
[6]: "matplotlib" in pagine[0]["contenuto"]
```

```
[6]: False
```

ma se metto tutto in minuscolo:

ATTENZIONE: `lower()`, come *tutti* i metodi delle stringhe, ritorna una **nuova** stringa e non modifica loriginale !!

```
[7]: pagine[0]["contenuto"].lower()
```

```
[7]: '<h2>presentazione</h2><p>il corso softpython fornisce un'introduzione al␣
↪processamento dati usando python, un linguaggio di programmazione popolare sia␣
↪nell'industria che nell'ambito della ricerca. fra le varie cose tratteremo␣
↪matplotlib.</p>'
```

adesso, `matplotlib` in minuscolo verrà trovato nella stringa generata dall'espressione qua sopra:

```
[8]: "matplotlib" in pagine[0]["contenuto"].lower()
```

```
[8]: True
```

Per essere sempre sicuri di trovare quello che l'utente cerca, conviene chiamare `.lower()` anche sulla stringa dell'utente:

```
[9]: "matplotlib".lower() in pagine[0]["contenuto"].lower()
```

```
[9]: True
```

```
[10]: "Matplotlib".lower() in pagine[0]["contenuto"].lower()
```

[385] http://comunicaresulweb.com/seo/stop-words-italiane/

[10]:
```
True
```

Proviamo a scrivere una ricerca più furba `ricerca2` che applica quanto visto sopra:

[11]:
```
def ricerca2(stringa):
    risultati = []
    for pagina in pagine:
        if stringa.lower() in pagina['contenuto'].lower():
            risultati.append(pagina)
    return risultati
```

[12]:
```
ricerca2("matplotlib")
```

[12]:
```
[{'contenuto': '<h2>Presentazione</h2><p>Il corso SoftPython fornisce un'introduzione␣
↪al processamento dati usando Python, un linguaggio di programmazione popolare sia␣
↪nell'industria che nell'ambito della ricerca. Fra le varie cose tratteremo␣
↪Matplotlib.</p>',
  'url': 'http://it.softpython.org'},
 {'contenuto': '<h2>Visualizzazione</h2> <p>Excel ci permette di creare molti tipi di␣
↪visualizzazione ma a volte è limitato, perciò useremo Python facendo grafici in␣
↪Matplotlib. Matplotlib è una libreria eccezionale che consente di creare qualunque␣
↪visualizzazione desideriamo. Procediamo dunque a importare matplotlib in Python:</p>
↪',
  'url': 'https://it.softpython.org/visualization/visualization-sol.html'}]
```

[13]:
```
ricerca2("Matplotlib")
```

[13]:
```
[{'contenuto': '<h2>Presentazione</h2><p>Il corso SoftPython fornisce un'introduzione␣
↪al processamento dati usando Python, un linguaggio di programmazione popolare sia␣
↪nell'industria che nell'ambito della ricerca. Fra le varie cose tratteremo␣
↪Matplotlib.</p>',
  'url': 'http://it.softpython.org'},
 {'contenuto': '<h2>Visualizzazione</h2> <p>Excel ci permette di creare molti tipi di␣
↪visualizzazione ma a volte è limitato, perciò useremo Python facendo grafici in␣
↪Matplotlib. Matplotlib è una libreria eccezionale che consente di creare qualunque␣
↪visualizzazione desideriamo. Procediamo dunque a importare matplotlib in Python:</p>
↪',
  'url': 'https://it.softpython.org/visualization/visualization-sol.html'}]
```

Performance

DOMANDA:

- Se abbiamo miliardi di pagine, conviene cercarle in una lista? Se la stringa che cerchiamo sta nell'ultima pagina, l'algoritmo di ricerca quante pagine guarda prima di trovare quella d'interesse?
- Che strutture dati alternative potremmo usare per la ricerca?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA: Ci converra *indicizzare* le pagine, cioè avere un indice che ci dice per ogni parola di ricerca in quali pagine possiamo trovarla.

- Per implementare questa corrispondenza in Python possiamo usare i dizionari.
- Per convenienza, ci conviene indicare le pagine con il numero che abbiamo usato quando le abbiamo messe nella listona.

- Di nuovo per convenienza, metteremo solo parole in minuscolo.

Quindi potrebbe essere una cosa del genere:

```
{
    "linguaggio":  [0],     # la stringa "linguaggio" sta alla pagina zeresima della␣
↪listona
    "matplotlib" : [0,1], # la stringa "matplotlib" sta nelle pagine zeresima e␣
↪unesima  della listona
    "importare" :  [1]      # "importare" sta solo nella pagina sulla visualizzazione
}
```

Ordinamento

Osserva di nuovo bene il testo delle pagine. Noti delle particolarità / differenze tra le due pagine ?

[14]:
```
pagine = [
            {
                'url': 'http://it.softpython.org',
                'contenuto' : '<h2>Presentazione</h2><p>Il corso SoftPython fornisce␣
↪un'introduzione al processamento dati usando Python, un linguaggio diprogrammazione␣
↪popolare sia nell'industria che nell'ambito della ricerca. Fra le varie cose␣
↪tratteremo Matplotlib.</p>'
            },
            {
                'url': 'https://it.softpython.org/visualization/visualization-sol.html
↪',
                'contenuto' : '<h2>Visualizzazione</h2> <p>Excel ci permette di␣
↪creare molti tipi di visualizzazione ma a volte è limitato, perciò useremo Python␣
↪facendo grafici in Matplotlib. Matplotlib è una libreria eccezionale che consente␣
↪di creare qualunque visualizzazione desideriamo. Procediamo dunque a importare␣
↪matplotlib in Python:</p>'
            }
        ]
```

Se l'utente cerca solo la stringa `"matplotlib"`, visto che è presente in entrambe le pagine ci aspettiamo che vengano ritornate entrambe:

[15]:
```
ricerca2("matplotlib")
```

[15]:
```
[{'contenuto': '<h2>Presentazione</h2><p>Il corso SoftPython fornisce un'introduzione␣
↪al processamento dati usando Python, un linguaggio diprogrammazione popolare sia␣
↪nell'industria che nell'ambito della ricerca. Fra le varie cose tratteremo␣
↪Matplotlib.</p>',
  'url': 'http://it.softpython.org'},
 {'contenuto': '<h2>Visualizzazione</h2> <p>Excel ci permette di creare molti tipi di␣
↪visualizzazione ma a volte è limitato, perciò useremo Python facendo grafici in␣
↪Matplotlib. Matplotlib è una libreria eccezionale che consente di creare qualunque␣
↪visualizzazione desideriamo. Procediamo dunque a importare matplotlib in Python:</p>
↪',
  'url': 'https://it.softpython.org/visualization/visualization-sol.html'}]
```

DOMANDA: Pensiamo all'ordine delle pagine. E' il migliore che possiamo fornire all'utente? Secondo te, qual'è la pagina che contiene più informazione riguardo Matplotlib e che andrebbe messa per prima?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra

risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

Ovviamente, noi umani sappiamo la pagina specifica sulla Visualizzazione sarà quella preferita dall'utente. Per capirlo automaticamente con un algoritmo possiamo notare che la frequenza di `"Matplotlib"` è mggiore nella seconda pagina rispetto alla prima. Possiamo farci dire la frequenza da Python con il comando `count`:

[16]:
```
pagine[0]["contenuto"].lower().count("Matplotlib".lower())
```

[16]:
```
1
```

[17]:
```
pagine[1]["contenuto"].lower().count("Matplotlib".lower())
```

[17]:
```
3
```

ESERCIZIO: Assegna ad ogni pagina un valore di rilevanza (*relevance*) , che permetta di determinare l'ordine (*rank*) in cui ritornare le pagine. Le pagine con rilevanza più alta devono essere presentate per prime nella lista ritornata

[18]:
```
# scrivi qui
```

5.7.3 Prendiamo le distanze

Un mondo di errori

Purtroppo, gli utenti non conoscono a priori il contenuto delle pagine che stanno cercando, e non sono nemmeno tutti dei pignoli letterati. Finchè cerchiamo parole che sono esattamente contenute nel testo va tutto bene:

[19]:
```
ricerca2('grafici')
```

[19]:
```
[{'contenuto': '<h2>Visualizzazione</h2> <p>Excel ci permette di creare molti tipi di␣
↪visualizzazione ma a volte è limitato, perciò useremo Python facendo grafici in␣
↪Matplotlib. Matplotlib è una libreria eccezionale che consente di creare qualunque␣
↪visualizzazione desideriamo. Procediamo dunque a importare matplotlib in Python:</p>
↪',
  'url': 'https://it.softpython.org/visualization/visualization-sol.html'}]
```

Ma se hai provato a scrivere le tue funzioni di ricerca, ti sarai reso ben presto conto di un problema di usabilità serio: cosa succede se l'utente inserisce una stringa contenente errori di battitura, come `grufici` (nota la `u`) ? In presenza di una stringa con errori il nostro motore fallirà la ricerca:

[20]:
```
ricerca2('grufici')
```

[20]:
```
[]
```

Questo fenomeno degli errori è molto comune, pensiamo a quanti ne commettiamo specialmente quando scriviamo da dispositivi come gli smartphone

Varie forme

In una lingua come l'italiano, il tema della presenza degli errori si interseca parzialmente col fatto che le parole possono contenere desinenze, avere forma singolare/plurale, maschile / femminile etc. Quindi se cercassimo `"grafico"` al singolare avremmo pure dei problemi:

[21]:
```
ricerca2('grafico')
```

[21]:
```
[]
```

Possiamo quindi chiederci cosa nella nostra mente ci fa interpretare `grufici` come qualcosa di *simile* a `grafici`. Proviamo a scrivere una serie di casistiche.

- una lettera in meno: `grfici`
- una lettera in più : `graficii`
- una lettera diversa: `grufici`
- due lettere in meno: `grfc`
- due lettera in più : `grraficii`
- due lettera diversa: `grufbci`
- una lettera in meno e una in più: `grficii`
- una lettera diversa e una in più: `gruficii`
- una lettera in meno, una diversa e una in più: `grufbcii`
- …

DOMANDA: Quali ti sembrano le forme più *simili* ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

Sicuramente, `grufici` ci sembra più *simile* a `grafici` di `grufbicii`. Ci è bastato contare quante lettere sono state cambiate, aggiunte o tolte per passare da una forma a l'altra.

Un mondo di distanze

Guardando gli esempi sopra, ci rendiamo presto conto di come date due parole, anche molto diverse tra loro, possiamo trovare quante e quali operazioni si possono compiere per *trasformarle* l'una nell'altra. Minore è il numero di operazioni, maggiore ci apparirà la *similarità* tra le due parole.

ESERCIZIO: Immagina di piazzare una decina parole sul pavimento di una stanza, e fai un disegno (sì, con la cara e vecchia carta): prova a mettere parole simili vicine, quindi `grufici` sarà più vicina a `grafici`, mentre `grufbcii` sarà più lontana. `grufbcii` e `grufici` saranno un po' più vicine tra loro. Quando la *distanza* è piccola, la *similarità* tra le parole deve essere più grande.

Se hai fatto il disegno, hai appena creato uno *spazio*. Però di spazi ce ne sono tanti. Tu hai disegnato su foglio, immaginando che fosse il pavimento di una stanza, che è bidimensionale. Forse avrai avuto difficoltà a piazzare alcune parole, perchè magari avevano senso vicine ad alcune ma fatalmente si trovavano vicine ad altre da cui erano molto diverse.

DOMANDA: E se potessimo appendere le parole con dei fili al soffitto ti aiuterebbe (supponi di poter scegliere per ogni parola la lunghezza del filo) ? Potrebbe servire avere più 'spazio'?

Quale spazio

Sicuramente questa cosa delle distanze tra parole ti ha sorpreso un po'. Noi siamo abituati alle distanze nel nostro mondo fisico, per esempio sappiamo dire se un oggetto è fisicamente vicino ad un altro. Però spesso anche nella quotidianità usiamo la distanza in modo figurativo, per esempio diciamo se una persona è *vicina* ad un'altra quando ne condivide i sentimenti, magari in un momento di difficoltà.

Quando si parla di distanze, bisogna imperativamente considerare tre cose:

1. gli oggetti che vogliamo considerare
2. lo spazio in cui misuriamo
3. un sistema di misura con cui per l'appunto misuriamo le distanze. Esempi:

Distanza tra stringhe

Quella che abbiamo visto sulle stringhe si chiama anche *distanza di edit*. Vediamo le caratteristiche:

1. oggetti: stringhe
2. spazio: tutte le possibile stringhe (anche quelle senza senso, e anche la stringa vuota)
3. misura di distanza: numero di caratteri da modificare per trasforamare una stringa in un'altra

Distanza fisica tra oggetti nel piano

Se abbiamo un oggetto nella stanza e vogliamo arrivare ad un oggetto B in un altro punto, possiamo facilmente calcolare quanti passi ci servono. Se abbiamo un metro, possiamo anche essere più precisi.

1. oggetti: cose del mondo fisico (bisogna decidere quali, si potrebbe considerare solo quelli con peso > 1 Kg)
2. spazio: spazio euclideo[386]
3. misura di distanza: distanza euclidea[387]

[386] https://it.wikipedia.org/wiki/Spazio_euclideo
[387] https://it.wikipedia.org/wiki/Distanza_euclidea

Distanza nel mondo

Qua le cose si fanno più complicate.

DOMANDA: Se sei in Italia e vuoi arrivare in Giappone, vai in linea retta? Attenzione: se rispondi *sì* ti servirà una trivella molto potente :-)

1. oggetti: cose del mondo fisico (bisogna decidere quali, si potrebbe considerare solo quelli con peso > 1 Kg)
2. spazio: spazio sferico[388] (2D o 3D)
3. misura di distanza: curva geodetica[389]

Quando si considerano distanze nel mondo, bisogna deve tenere conto che la Terra è sferica, lo sanno bene navigatori e piloti. Non si può andare in linea retta, ma invece muoversi lungo la superficie della sfera (vedi Wikipedia[390]). Questo di fatto aumenta le distanze:

[388] https://it.wikipedia.org/wiki/Geometria_sferica
[389] https://it.wikipedia.org/wiki/Geodetica
[390] https://en.wikipedia.org/wiki/Great-circle_distance

5.7.4 Ricerca a prova di errore

Possiamo migliorare il nostro motore di ricerca per renderlo più usabile anche in presenza di errori da parte dell'utente?

DOMANDA: supponi di avere bella pronta una funzione `distanza_stringhe` che date due stringhe ti dice quanti caratteri occorrono per trasformare una stringa in un altra. Sfruttando questa funzione, riusciresti a migliorare il motore di ricerca ?

La distanza di Levenshtein

Il matematico Russo Vladimir Levenshtein ha trovato il modo di calcolare rapidamente la distanza tra stringhe che ora prende il suo nome. Fortunamente per noi, ci sono parecchie implementazioni Python già pronte. Non ci addentreremo nell'implementazione, per capirla bene servirebbe un corso apposito di algoritmi:

[22]:
```
# ATTENZIONE: NON E' NECESSARIO CHE COMPRENDI COME FUNZIONA QUESTO CODICE !
#             CI LIMITEREMO SOLO A UTILIZZARE QUESTA FUNZIONE !!
#             tratto da https://stackabuse.com/levenshtein-distance-and-text-
↪similarity-in-python/

def distanza_stringhe(s, t):
    """
        iterative_levenshtein(s, t) -> ldist
        ldist is the Levenshtein distance between the strings
        s and t.
        For all i and j, dist[i,j] will contain the Levenshtein
        distance between the first i characters of s and the
        first j characters of t
    """
    rows = len(s)+1
    cols = len(t)+1
    dist = [[0 for x in range(cols)] for x in range(rows)]
    # source prefixes can be transformed into empty strings
    # by deletions:
    for i in range(1, rows):
        dist[i][0] = i
    # target prefixes can be created from an empty source string
    # by inserting the characters
    for i in range(1, cols):
        dist[0][i] = i

    for col in range(1, cols):
        for row in range(1, rows):
            if s[row-1] == t[col-1]:
                cost = 0
            else:
                cost = 1
            dist[row][col] = min(dist[row-1][col] + 1,      # deletion
                                 dist[row][col-1] + 1,      # insertion
                                 dist[row-1][col-1] + cost) # substitution
    #debug
    #for r in range(rows):
    #    print(dist[r])


    return dist[row][col]
```

Proviamola:

[23]:
```
distanza_stringhe("grafici", "grufici")
```
[23]:
```
1
```

[24]:
```
distanza_stringhe("grafici", "grfici")
```
[24]:
```
1
```

[25]:
```
distanza_stringhe("grafici", "gruficii")
```
[25]:
```
2
```

[26]:
```
distanza_stringhe("grafici", "grufbcii")
```
[26]:
```
3
```

Ricerca per distanza sintattica

La funzione `distanza_stringhe` pare fare al caso nostro !

ESERCIZIO: Prendendo codice dalla funzione `ricerca2`, crea una funzione `ricerca3` che restituisca risultati anche con stringhe di input aventi una distanza di Levenshtein <= 2 dalle parole corrette nel testo delle pagine.

- supponi che in input siano date parole singole
- ricordati delle pagine in cui hai già trovato una corrispondenza usando un insieme di url con la struttura dati set[391]
- Separa le parole nel testo delle pagine con la funzione split:

[27]:
```
"il testo della mia pagina".split(" ")
```
[27]:
```
['il', 'testo', 'della', 'mia', 'pagina']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[28]:
```
# scrivi qui

def ricerca3(stringa):
    url_trovate = set()
    risultati = []
    for pagina in pagine:
        for parola in pagina['contenuto']:
            minuscola = stringa.lower()
            parole_testo = pagina['contenuto'].lower().split(" ")
            for parola in parole_testo:
                if not pagina["url"] in url_trovate:
                    if distanza_stringhe(stringa, parola) <= 2:
                        risultati.append(pagina)
                        url_trovate.add(pagina["url"])
    return risultati
```

[391] https://www.python-course.eu/python3_sets_frozensets.php

[28]:
```
# scrivi qui
```

[29]:
```
ricerca3('grafico')
```

[29]:
```
[{'contenuto': '<h2>Visualizzazione</h2> <p>Excel ci permette di creare molti tipi di␣
↪visualizzazione ma a volte è limitato, perciò useremo Python facendo grafici in␣
↪Matplotlib. Matplotlib è una libreria eccezionale che consente di creare qualunque␣
↪visualizzazione desideriamo. Procediamo dunque a importare matplotlib in Python:</p>
↪',
  'url': 'https://it.softpython.org/visualization/visualization-sol.html'}]
```

[30]:
```
ricerca3('grufici')
```

[30]:
```
[{'contenuto': '<h2>Visualizzazione</h2> <p>Excel ci permette di creare molti tipi di␣
↪visualizzazione ma a volte è limitato, perciò useremo Python facendo grafici in␣
↪Matplotlib. Matplotlib è una libreria eccezionale che consente di creare qualunque␣
↪visualizzazione desideriamo. Procediamo dunque a importare matplotlib in Python:</p>
↪',
  'url': 'https://it.softpython.org/visualization/visualization-sol.html'}]
```

[31]:
```
ricerca3('gruficii')
```

[31]:
```
[{'contenuto': '<h2>Visualizzazione</h2> <p>Excel ci permette di creare molti tipi di␣
↪visualizzazione ma a volte è limitato, perciò useremo Python facendo grafici in␣
↪Matplotlib. Matplotlib è una libreria eccezionale che consente di creare qualunque␣
↪visualizzazione desideriamo. Procediamo dunque a importare matplotlib in Python:</p>
↪',
  'url': 'https://it.softpython.org/visualization/visualization-sol.html'}]
```

[32]:
```
ricerca3('grufbcii') # distanza = 3   > 2
```

[32]:
```
[]
```

5.7.5 Similarità semantica

Il calcolo della similarità di testi è uno strumento potente per costruire rapidamente sistemi di ricerca intelligenti.

Finora abbiamo visto la *similarità sintattica*, cioè abbiamo guardato i singoli caratteri. Ma ovviamente un testo non è solo una sequenza di caratteri, per noi è una sequenza di parole con *significato*. Il ramo della Data science che si occupa del *significato* delle parole è la *semantica* . Nel contesto del web, possiamo parlare di Semantic web[392]

Vediamo di sfruttare i significati nel testo. Per esempio, pensiamo ad un'impresa che dopo aver trovato un bando di gara interessante pubblicato dal Comune di Trento. potrebbe voler cercare bandi o avvisi simili , per esempio per capire quali sono stati i criteri di selezione e per individuare quanti e quali potenziali aziende concorrenti hanno partecipato al bando. Per comprendere con metodi automatici quanto il testo di un bando assomigli ad un'altro, si potrebbero usare i servizi di text similarity offerti da Dandelion[393] dell'azienda SpazioDati[394].

Un servizio di text similarity prende due testi e calcola un valore di similarità come un numero da `0.0` a `1.0`. In questo esempio[395] viene mostrato come venga rilevata una qualche somiglianza tra due testi che trattano di pavimentazione stradale.

[392] https://it.wikipedia.org/wiki/Web_semantico
[393] https://dandelion.eu
[394] https://spaziodati.eu
[395] https://dandelion.eu/semantic-text/text-similarity-demo/?text1=Lavori+di+sistemazione+pavimentazione+rampa+di+accesso+al+Parco+di+Mattarello+-+AggiudicazioneVerbale+di+gara+-+Provvedimento+che+determina+le+esclusioni+dall&text2=Le+opere+che+formano+oggetto+dell%27appalto+possono+riassumersi+in+via+puramente+indicativa+come+di+seguito%3A+pavimentazioni%3B-+reti+idrauliche%

Ricerca veloce

Al fine di avere un servizio di ricerca di similarità veloce, ogni volta che viene effettuata una ricerca non conviene confrontare il testo con tutti gli altri `n-1` , perché potrebbe via potenzialmente parecchio tempo e l'utente nel frattempo potrebbe stufarsi. Una ottimizzazione potrebbe essere precalcolare tutte le similarità necessarie al momento dell'ottenimento i dati, costruendo una tabella di similarità. Così, quando un utente ci indica un testo, possiamo cercare rapidamente nella nostra tabella quali sono i testi maggiormente simili, senza più nemmeno usare il servizio di similarità. La tabella può avere la forma di una matrice diagonale come la seguente. Supponendo di avere `n=4` testi, chiameremo il servizio di similarità per riempire solo le celle nella parte superiore così:

3B-+impianto+di+illuminazione+pubblica&lang=auto&exec=true

	testo 1	testo 2	testo 3	testo 4
testo 1		0.2	0.5	0.7
testo 2			0.9	0.8
testo 3				0.3
testo 4				

Per riempire la prima riga serviranno `n - 1` confronti, per la seconda `n - 2`, per la terza `n - 3` e così via.

In totale le chiamate al servizio di text similarity saranno

$$\frac{((n-1)^2+(n-1))}{2} = \frac{(n^2-2n+1+n-1)}{2} = \frac{(n^2-n)}{2}$$

Una volta ottenuta la tabella sopra, i restanti valori si possono calcolare rapidamente sapendo che un testo ha sempre similarità `1.0` con se stesso e gli altri valori si possono ricavare per simmetria lungo la diagonale. Qua li abbiamo scritti per motivi di chiarezza, ma di fatto non serve nemmeno materializzarli in memoria:

	testo 1	testo 2	testo 3	testo 4
testo 1	1.0	0.2	0.5	0.7
testo 2	0.2	1.0	0.9	0.8
testo 3	0.5	0.9	1.0	0.3
testo 4	0.7	0.8	0.3	1.0

Il prezzo del successo

Fatta la startup, dopo la geniale idea della tabella per avere ricerche veloci, abbiamo trovato i primi utenti in Trentino e sono entusiasti del nostro servizio. Convinciamo anche dei venture capitalist a investire denaro sonante nella nostra impresa. Ebbri di successo, pensiamo al futuro radioso: è tempo di conquistare il mondo intero.

Mentre pensiamo beati al prossimo modello di *Lambo* da comprare, all'improvviso una domanda sfreccia nella nostra testa come un fulmine, incenerendo senza pietà l'auto dei nostri sogni:

Ma l'algoritmo scala?

Una volta che abbiamo la tabella possiamo fare ricerche molto più rapide, ma se avessimo un miliardo di bandi da analizzare, quante celle ci sarebbero nella tabella risultante ? Quali sarebbero le conseguenze?

Per oggi, ci fermiamo qui con l'ottimizzazione. Quanto segue è solo un invito a pensare ai big data e a cosa implicano.

Di solito si analizzano il tempo e lo spazio:

Tempo: supponendo che ci voglia 1 secondo al servizio di similarità per calcolare ogni similarità, quanto tempo ci metterebbe a riempire la tabella superiore in minuti, ore, giorni, ….?

Spazio: Supponendo per semplicità che ogni cella occupi un byte, e sapendo che un gigabyte sono un miliardo di byte (1.000.000.000):

- quanti byte ci servirebbero?
- Che unità di misura useremmo? Puoi consultare Wikipedia[396] per trovare il nome giusto.
- Secondo te sarebbe realistico salvare la tabella in un solo server?
- Se usassimo solo la parte superiore della tabella, ridurremmo sostanzialmente lo spazio occupato ?

DOMANDA: (solo da pensare, niente implementazione !!): Riesci ad immaginare qualche alternativa alla tabellona ? Suggerimento: Se un pino è simile ad un abete nano, e l'abete nano è simile ad un cespuglio, possiamo dire qualcosa sul grado di similarità tra il pino e il cespuglio ?

```
[ ]:
```

5.8 Computer vision

5.8.1 Scarica zip esercizi

Naviga file online[397]

5.8.2 Introduzione

ATTENZIONE: Ciò che segue è solo una bozza MOLTO IN-PROGRESS !!!!!

Installazione

Se hai con Anaconda:

- apri Anaconda Navigator
- seleziona tab `Environments` sulla sinistra
- seleziona l'ambiente (se non hai idea di cosa scegliere, seleziona `base (root)`)
- sulla destra, seleziona `Not installed`
- cerca `opencv` e installalo

[396] https://en.wikipedia.org/wiki/Unit_prefix
[397] https://github.com/DavidLeoni/softpython-it/tree/master/computer-vision

Se hai Linux / Ubuntu:

```
sudo apt update
sudo apt install python3-opencv
```

Riferimenti

Mettiamo qua una lista di riferimenti per lavorare con Python e le immagini:

- PyImageSearch[398]
- Python Imaging Library Handbook[399]
- Hands on Image Processing[400]

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
computer-vision
    computer-vision.ipynb
    computer-vision-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `computer-vision.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

[398] https://www.pyimagesearch.com/
[399] http://effbot.org/imagingbook/pil-index.htm
[400] https://www.researchgate.net/profile/Sandipan_Dey/publication/329610370_Hands-on_Image_Processing_in_Python/links/5c120f09299bf139c754a03c/Hands-on-Image-Processing-in-Python.pdf

5.8.3 1. Anonimizzazione facce

Proponiamo un semplice programmino per anonimizzare volti.

Facendo partire la cella seguente, dovrebbe caricare una foto, mostrare i rettangoli sulle facce e anonimizzarla, salvandola in `'results/simple_output.png'`. Non è detto che trovi tutti i volti, potreste dover aggiustare dei parametri. Alcune foto di test sono fornite nella cartella `images`, di quelle nei campi da calcio a seconda della foto i volti vengono riconosciuti oppure no.

[1]:
```
# A simple algorithm for face detection which also allows the user to automatically␣
↪hide the detected faces. Moreover, in the case the detection missed some face,␣
↪those can be manually selected and obscured.

# At the moment the implemented hiding methods are four:
#   - *Blur*
#   - *Pixel Shuffle*
#   - *Image Swap*
#   - *Negative*



import numpy as np
import cv2

# fonte:  https://en.wikipedia.org/wiki/File:Amateur_Football_in_Kilkenny-Ireland.jpg
#filename = 'images/football/Amateur_Football_in_Kilkenny-Ireland.jpg'

# fonte: https://commons.wikimedia.org/wiki/File:Ronaldo_-_Manchester_United_vs_
↪Chelsea.jpg
#filename = 'images/football/800px-Ronaldo_-_Manchester_United_vs_Chelsea.jpg'

# fonte: https://commons.wikimedia.org/wiki/File:6_a_side_football_-_5_a_side_
↪football_-_7_a_side_football_-_Ballerz_league.jpg
#filename = 'images/football/6_a_side_football_-_5_a_side_football_-_7_a_side_
↪football_-_Ballerz_league.jpg'
# fonte: https://commons.wikimedia.org/wiki/File:2018_IBSA_Blind_Football_Madrid_
↪Argentina.JPG
filename = 'images/football/800px-2018_IBSA_Blind_Football_Madrid_Argentina.jpeg'

#filename = "images/square_faces.jpg"
grrr = "images/grrr_reaction.png"

# Loading image
img_raw = cv2.imread(filename)
img_raw_copy = cv2.imread(filename)
grrr_image = cv2.imread(grrr)
# print(img_raw[1:10,1:10])

# Converting image to grey-scale
img_raw_grey = cv2.cvtColor(img_raw, cv2.COLOR_BGR2GRAY)
# Create and load the CascadeClassifier for face detection
haar_cascade_face = cv2.CascadeClassifier()
loaded = haar_cascade_face.load('resources/haarcascade_frontalface_default.xml')

'''
Returns a rectangle around the face.

@param scale_factor: compensate the distance of the face from the camera
```

(continues on next page)

(continued from previous page)

```
@param minNeighbors: number of neighbors for a rectangle to be considered as a face
'''
#  scaleFactor: 1 - 1.5 2.0 massimo
#  minNeighbors = tra 1 e 6  NON è il numero di facce
faces_rects = haar_cascade_face.detectMultiScale(img_raw_grey, scaleFactor = 1.2,␣
↪minNeighbors = 3);
print('facce trovate: ', len(faces_rects))
for (x,y,w,h) in faces_rects:
    # print(x, y, w,h)
    cv2.rectangle(img_raw_copy, (x, y), (x+w, y+h), (0, 255, 0), 2)
    # UNCOMMENT THE DESIRED HIDING METHOD
    # 1] BLUR
    img_raw[y:y+h, x:x+w] = cv2.blur(img_raw[y:y+h, x:x+w], (40, 40))
    # 2] SHUFFLING PIXELS
    # np.random.shuffle(img_raw[y:y+h, x:x+w].flat)
    # 3] NEGATIVE
    #img_raw[y:y+h, x:x+w] = cv2.bitwise_not(img_raw[y:y+h, x:x+w])
    # 4] EMOJI
    #grrr_resize = cv2.resize(grrr_image, (w, h))
    #img_raw[y:y+h, x:x+w] = grrr_resize
    pass
cv2.namedWindow("img_copy")
cv2.imshow("img_copy", img_raw_copy)
# togli i commenti per permettere all'utente di selezionare manualmente un riquadro␣
↪attorno al volto
#fromCenter = False
#roi = cv2.selectROI("", img_raw, fromCenter)
#img_raw[roi[1]:roi[1]+roi[3], roi[0]:roi[0]+roi[2]] = cv2.blur(img_raw[roi[1]:
↪roi[1]+roi[3], roi[0]:roi[0]+roi[2]], (40, 40))

out_filename = 'results/simple_output.png'
cv2.imwrite('results/simple_output.png', img_raw)
print("immagine anonimizzata salvata in:   " + out_filename)

# mostra l'immagine in Jupyter:
# ATTENZIONE: a volte blocca l'esecuzione della cella e bisogna far ripartire il␣
↪kernel Jupyter.
#cv2.namedWindow("img")
#cv2.imshow("img", img_raw)
```

```
facce trovate:  3
immagine anonimizzata salvata in:   results/simple_output.png
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[2]:
```
# SOLUZIONE
```

[2]:

[ ]:

CHAPTER

SIX

C - APPLICAZIONI

6.1 Interfacce grafiche

6.1.1 Scarica zip esercizi

Naviga file online[401]

6.1.2 Introduzione

In questo tutoral affronteremo il tema delle interfacce grafiche (GUI: Graphical User Interfaces), usando:

- **i widget di Jupyter**. Li abbiamo scelti perchè sono sorprendentemente flessibili e intuitivi. Come riferimento, seguiremo l'ottima User Guide[402] ufficiale di Jupyter (in inglese)
- **bqplot** per fare grafici interattivi

Questo tutorial non può certo essere un corso intero di Human Computer Interaction, ma dovrebbe permettervi di avere un'idea di come sviluppare delle interfacce rudimentali

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
gui
    gui.ipynb
    gui-sol.ipynb
    my-webapp.ipynb
    gui-maps.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `interactive.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

[401] https://github.com/DavidLeoni/softpython-it/tree/master/gui
[402] http://ipywidgets.readthedocs.io/en/stable/user_guide.html

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

Perchè fare interfacce grafiche ?

Per quanto le interfacce grafiche possano sembrare attrattive, prima di lanciarsi a crearle bisogna sempre farsi alcune domande fondamentali:

Qual'è lo scopo?

- sperimentazione?
- creare prototipi ?
- prodotti per utenti finali?

Chi è l'utente?

- Tu?
- Altre persone ?
  - Che conoscenze hanno ?
- dove viene usata l'interfaccia ?
  - a casa?
  - fuori casa?
    - c'è poco / tanto sole ?

Scelte di stile

- interfaccia semplice o complessa ?
- stile in-progress (mockup) o prodotto finito ?

Scelte tecniche

Una volta identificati i requisiti, si possono fare le scelte tecniche, che riguardano linguaggi e architettura. Ci sono tantissime combinazioni possibili, ne menzioniamo solo alcune:

sito online, solo client

- niente server
- client browser con Javascript, HTML, CSS
- client mobile app
  - Android (in Java) ?
  - iPhone (in Object-C) ?

sito online, client / server

- server in Python (magari in Django[403])
- client Javascript,HTML,CSS in browser
- client in browser in Python transpilato a Javascript
- client mobile app

offline, desktop (applicazioni native Python)

Risposte per oggi:

scopo: sperimentazione

utente: Tu !

architettura: client/server

- server: Jupyter, con funzioni definite in Python dentro Jupyter
- client: browser
  - browser gestito automaticamente dal server Jupyter
  - niente Javascript / HTML, pochissimo CSS

Morale: se un giorno dovrete fare applicazioni grafiche sul serio per utenti finali, prima leggetevi un buon libro di Human Computer Interaction e testate spesso le vostre interfacce con amici & parenti !

6.1.3 Installazione ipywidgets

Prima di avviare Jupyter, bisogna installare la libreria `ipywidgets` ed eventualmente abilitarla, a seconda del sistema operativo:

Anaconda:

- `conda install -c conda-forge ipywidgets`
- Installare `ipywidgets` con `conda` abiliterà automaticamente l'estensione per te

Linux/Mac:

- installa ipywidgets (`--user` installa nella propria home):

```
python3 -m pip install --user ipywidgets
```

- abilita l'estensione così:

```
jupyter nbextension enable  --py   widgetsnbextension
```

Adesso prova ad aprire Jupyter ed incollare il seguente codice in una cella, eseguendolo dovrebbe apparirti il widget dello slider sotto la cella:

[1]:
```
import ipywidgets as widgets
widgets.IntSlider()
```

```
IntSlider(value=0)
```

[403] https://www.djangoproject.com/

[2]:
```
# copia incolla qua sotto:
```

6.1.4 Facciamo uno slider

Prima abbiamo dato l'istruzione a Jupyter di creare un widget slider, e Jupyter ci ha mostrato subito il risultato della creazione. Che succede se salviamo il risultato in una variabile ?

[3]:
```
w = widgets.IntSlider()
```

Vediamo che apparentemente non accade nulla. Cosa c'è adesso nella variabile `w`? Vediamolo con `type`:

[4]:
```
type(w)
```

[4]:
```
ipywidgets.widgets.widget_int.IntSlider
```

Vediamo che in `w` abbiamo un istanza di uno `IntSlider`. Ma come facciamo a dire a Python di mostrarlo? Possiamo usare la funzione `display`, importandola dal modulo `IPython.display` (Nota che il `display` dopo il punto è il nome del modulo, in questo caso particolare il modulo contiene anche una funzione che si chiama come il modulo):

[5]:
```
from IPython.display import display
display(w)
```

```
IntSlider(value=0)
```

Proviamo a cambiare un po' di proprietà dello slider:

[6]:
```
w.description = "ciao" # scritta a sinistra
```

[7]:
```
w.value
```

[7]:
```
0
```

[8]:
```
w.value = 30
```

Per una lista di variabili, usa `keys`:

[9]:
```
w.keys
```

[9]:
```
['_dom_classes',
 '_model_module',
 '_model_module_version',
 '_model_name',
 '_view_count',
 '_view_module',
 '_view_module_version',
 '_view_name',
 'continuous_update',
 'description',
 'description_tooltip',
 'disabled',
 'layout',
 'max',
 'min',
 'orientation',
```

(continues on next page)

(continued from previous page)

```
 'readout',
 'readout_format',
 'step',
 'style',
 'value']
```

ESERCIZIO: prova a settare `min`, `max`, `step`. Prova a settare valori di `min` più grandi del valore che vedi correntemente, che succede ?

ESERCIZIO: Prova qualche altro widget dal sito di Jupyter[404]

6.1.5 Model View Controller

Una cosa interessante è chiamare ripetutamente display:

[10]:
```
w = widgets.IntSlider()
display(w)
```

```
IntSlider(value=0)
```

[11]:
```
display(w)
```

```
IntSlider(value=0)
```

ESERCIZIO: prova a scorrere il secondo slider, e guarda cosa succede al primo

Ogni chiamata a `display` genera una vista (*view*) del widget, ma il modello sottostante (*model*) che contiene i dati con il numero della posizione rimane lo stesso. Ogni volta che clicchiamo su una vista e trasciniamo lo slider, il browser manda dei segnali al kernel Python per comunicargli che deve cambiare il valore nel modello dati. Il kernel controlla cosa succede (*controller*) e a sua volta può reagire ai segnali attuando altri comportamenti, come per esempio aggiornare un grafico nel browser.

[404] http://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html

6.1.6 Interact

`interact` è un modo semplice per creare delle funzioni che reagiscono a cambiamenti di componenti grafici. Per esempio, se vogliamo creare uno slider alla cui posizione è associata una variabile `k`: ogni volta che muoviamo lo slider, ci piacerebbe stampare il doppio di `k`.

Per cominciare, possiamo definire una nostra funzione `aggiorna`, che prende in input il numero `k`, calcola il nuovo numero e stampa:

NOTA: `aggiorna` prende un parametro `k` che definiamo noi. Potremmo anche chiamarlo `pippo`.

[12]:
```
from ipywidgets import interact

def aggiorna(k = 3):
    print("il doppio è " + str(k*2))    # con str convertiamo il numero a stringa,␣
↪altrimenti Python si offende

interact(aggiorna)
```

```
interactive(children=(IntSlider(value=3, description='k', max=9, min=-3), Output()), _
↪dom_classes=('widget-int…
```

[12]:
```
<function __main__.aggiorna(k=3)>
```

Abbiamo creato una semplice funzione Python, niente di speciale fin qui. Proviamo a chiamarla noi:

[13]:
```
aggiorna(5)
```

```
il doppio è 10
```

[14]:
```
aggiorna(7)
```

```
il doppio è 14
```

Ora non ci resta che dire a Jupyther di creare uno slider e chiamare `aggiorna` ogni volta che lo slider viene spostato. Possiamo farlo con la funzione di Jupyter `interact`.

NOTA: chiamando la funzione di Jupyter `interact`, come parametro gli passiamo *la funzione* aggiorna, NON il risultato della funzione aggiorna ! Dato che nella dichiarazione di aggiorna abbiamo messo un parametro k inizializzato da un intero 3, Python capisce magicamente che siamo interessati a visualizzare un widget in grado di modificare valori interi, e in questo caso creerà un bello slider!

[15]:
```
interact(aggiorna)
```

```
interactive(children=(IntSlider(value=3, description='k', max=9, min=-3), Output()), _
↪dom_classes=('widget-int…
```

[15]:
```
<function __main__.aggiorna(k=3)>
```

Abbiamo detto che `interact` è intelligente e crea il widget giusto in base al tipo del parametro iniziale. Proviamo con un boolean:

[16]:
```
from ipywidgets import interact


def aggiorna(k = True):
    if k:
        print("spuntata")
    else:
        print("non spuntata")

interact(aggiorna)
```

```
interactive(children=(Checkbox(value=True, description='k'), Output()), _dom_classes=(
↪'widget-interact',))
```

[16]:
```
<function __main__.aggiorna(k=True)>
```

Vediamo che ci viene creata una casella checkbox. Si possono anche passare più parametri per ottenere più widget:

[17]:
```
from ipywidgets import interact


def aggiorna(i=3, k = True):

    if i > 3:
        print('grande')
    elif i == 3:
        print('medio')
    else:
        print('piccolo')

    if k:
        print("spuntata")
    else:
        print("non spuntata")

interact(aggiorna)
```

```
interactive(children=(IntSlider(value=3, description='i', max=9, min=-3),␣
↪Checkbox(value=True, description='k'…
```

[17]:
```
<function __main__.aggiorna(i=3, k=True)>
```

Possiamo anche crearci il widget direttamente associandolo alla stessa variabile che usiamo in `aggiorna`. Qua per esempio associamo noi uno slider con valore iniziale 10 alla variabile `k`. Notare che il `10` è definito durante la creazione dell'`IntSlider` e non in `aggiorna`:

[18]:
```
from ipywidgets import interact

def aggiorna(k):
    print('Il numero è ' + str(k))

interact(aggiorna, k=widgets.IntSlider(min=-10,max=30,step=1,value=10));
```

```
interactive(children=(IntSlider(value=10, description='k', max=30, min=-10),␣
↪Output()), _dom_classes=('widget-…
```

6.1.7 Riusare il widget con interactive

Se vogliamo accedere programmaticamente agli oggetti widget creati da interact, non dobbiamo usare `interact` ma `interactive`:

[19]:
```
from ipywidgets import interactive
from IPython.display import display

def aggiorna(k):
    print('Il numero è ' + str(k))

slider = interactive(aggiorna, k=widgets.IntSlider(min=-10,max=30,step=1,value=10));
display(slider)
```

```
interactive(children=(IntSlider(value=10, description='k', max=30, min=-10),␣
↪Output()), _dom_classes=('widget-…
```

6.1.8 Eventi

E se volessimo accedere ai valori di un widget con logiche più complesse, per esempio per comandarne un altro? In questi casi conviene gestire eventi con `observe`. Per cominciare, vediamo come funziona sul buon vecchio `IntSlider`.

- Oltre a trascinare lo slider, prova anche a farlo saltare da un estremo all'altro, e osserva i valori stampati.
- Per far sparire tutte le stampe, basta rieseguire la cella.

> **ATTENZIONE**: attento ai bottoni !
>
> `.observe` va bene per widget in genere, ma per i bottoni ad Agosto 2018 non sembra funzionare. Per quest'ultimi usa gli *eventi click* con il metodo `.on_click`.

[20]:
```
import ipywidgets as widgets
from ipywidgets import IntSlider
from IPython.display import display
```

(continues on next page)

(continued from previous page)

```
slider1 = IntSlider()
display(slider1)

# notare che stavolta dichiariamo la variabile 'change', che NON è un solo valore␣
↪come con interact
# ma un oggetto con diversi valori riguardanti il cambiamento generato dal click dell
↪'utente,
# il più interessante dei quali è `new`. Prova a togliere i commenti
# da print(change) per vedere l'oggetto change completo
def aggiorna(change):
    print("Il nuovo valore è " + str(change.new))
    #print(type(change))
    print(change)

#NOTA: adesso passiamo anche il parametro names=['value'] per filtrare i tipi di␣
↪change che riceviamo
slider1.observe(aggiorna, names=['value'])
```

```
IntSlider(value=0)
```

ESERCIZIO: Perchè abbiamo aggiunto quel `names=['value']`? Ricopia il codice di sopra qua sotto togliendo `, names=['value']` scoprirai che vengono stampate un sacco di cose in più, che nella maggior parte dei casi sono inutili.

[21]:
```
# scrivi qui
```

Controlliamo una label con uno slider

Proviamo adesso ad usare il widget dello slider per comandare dei widget di tipo Label:

[22]:
```
import ipywidgets as widgets
from ipywidgets import IntSlider, Label
from IPython.display import display

slider1 = IntSlider()
display(slider1)
label_new = Label("nuovo valore = ")
display(label_new)

# notare che stavolta dichiariamo la variabile 'change', che NON è un solo valore␣
↪come con interact
# ma un oggetto con diversi valori riguardanti il cambiamento generato dal click dell
↪'utente,
# il più interessante dei quali è `new`. Prova a togliere i commenti
# da print(change) per vedere l'oggetto change completo
def aggiorna(change):
    label_new.value = "nuovo valore = " + str(change.new) # convertiamo il numero in␣
↪stringa

#NOTA: adesso passiamo anche il parametro names=['value'] per filtrare i tipi di␣
↪change che riceviamo
slider1.observe(aggiorna, names=['value'])
```

```
IntSlider(value=0)
```

```
Label(value='nuovo valore = ')
```

ESERCIZIO: Riscrivi qua sotto l'esempio di sopra, aggiungiendo in più una etichetta che mostri anche il vecchio valore

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[23]:
```
# scrivi qui

import ipywidgets as widgets
from ipywidgets import IntSlider, Label
from IPython.display import display

slider1 = IntSlider()
display(slider1)
label_new = Label("nuovo valore = ")
display(label_new)
label_old = Label("vecchio valore = ")
display(label_old)

# notare che stavolta dichiariamo la variabile 'change', che NON è un solo valore␣
↪come con interact
# ma un oggetto con diversi valori riguardanti il cambiamento generato dal click dell
↪'utente,
# il più interessante dei quali è `new`. Prova a togliere i commenti
# da print(change) per vedere l'oggetto change completo
def aggiorna(change):
    label_new.value = "nuovo valore = " + str(change.new) # convertiamo il numero in␣
↪stringa
    label_old.value = "vecchio valore = " + str(change.old) # convertiamo il numero␣
↪in stringa

#NOTA: adesso passiamo anche il parametro names=['value'] per filtrare i tipi di␣
↪change che riceviamo
slider1.observe(aggiorna, names=['value'])
```

```
IntSlider(value=0)
```

```
Label(value='nuovo valore = ')
```

```
Label(value='vecchio valore = ')
```

[23]:
```
# scrivi qui
```

```
IntSlider(value=0)
```

```
Label(value='nuovo valore = ')
```

```
Label(value='vecchio valore = ')
```

Controlliamo uno slider con un'altro slider

Proviamo adesso ad usare il widget dello slider per comandare un altro slider: quando cambiamo i valori del primo slider, vogliamo che il secondo mostri lo stesso valore raddoppiato.

[24]:
```
import ipywidgets as widgets
from ipywidgets import IntSlider
from IPython.display import display

slider1 = widgets.IntSlider(max=100)  # specifichiamo il limite estremo
display(slider1)
slider2 = widgets.IntSlider(max=200)  # ricordiamoci di specificare il limite doppio␣
↪del primo
display(slider2)


# notare che stavolta dichiariamo la variabile 'change', che NON è un solo valore␣
↪come con interact
# ma un oggetto con diversi valori riguardanti il cambiamento generato dal click dell
↪'utente,
# il più interessante dei quali è `new`. Prova a togliere i commenti
# da print(change) per vedere l'oggetto change completo
def aggiorna1(change):
    #print("Il nuovo valore è " + str(change.new))
    #print(type(change))
    slider2.value = change.new * 2

#NOTA: adesso passiamo anche il parametro names=['value'] per filtrare i tipi di␣
↪change che riceviamo
slider1.observe(aggiorna1, names=['value'])
```

```
IntSlider(value=0)
```

```
IntSlider(value=0, max=200)
```

ESERCIZIO: Cosa succede se muoviamo il secondo slider? Il primo si aggiorna? Scrivi qua sotto una versione modificata del codice sopra, in cui quando si muove il secondo slider al primo viene fatto mostrare un valore che è la metà del secondo.

SUGGERIMENTO: crea una seconda funzione `aggiorna2` che aggiorna il primo slider, e collegala allo `slider2`

NOTA: l'operatore di divisione intera è `//`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[25]:
```
# scrivi qui

slider1 = widgets.IntSlider(max=100)  # specifichiamo il limite estremo
display(slider1)
slider2 = widgets.IntSlider(max=200)  # ricordiamoci di specificare il limite doppio␣
↪del primo
display(slider2)


# notare che stavolta dichiariamo la variabile 'change', che NON è un solo valore␣
↪come con interact
# ma un oggetto con diversi valori riguardanti il cambiamento generato dal click dell
↪'utente,
```

(continues on next page)

(continued from previous page)

```
# il più interessante dei quali è `new`. Prova a togliere i commenti
# da print(change) per vedere l'oggetto change completo
def aggiorna1(change):
    #print("Il nuovo valore è " + str(change.new))
    #print(type(change))
    slider2.value = change.new * 2

def aggiorna2(change):
    #print("Il nuovo valore è " + str(change.new))
    #print(type(change))
    slider1.value = change.new // 2 # divisione intera !


#NOTA: adesso passiamo anche il parametro names=['value'] per filtrare i tipi di␣
↪change che riceviamo
slider1.observe(aggiorna1, names=['value'])
slider2.observe(aggiorna2, names=['value'])
```

```
IntSlider(value=0)
```

```
IntSlider(value=0, max=200)
```

[25]:
```
# scrivi qui
```

```
IntSlider(value=0)
```

```
IntSlider(value=0, max=200)
```

ESERCIZIO: Dopo aver svolto l'esercizio precedente, prova a farne un'altro dove gli slider sono `FloatSlider` e il secondo slider rappresenta il quadrato del primo slider.

- **NOTA 1**: Ricordati di importare il `FloatSlider`
- **NOTA 2**: l'operatore di esponenziazione in Python è `**`: `5 ** 2` è cinque al quadrato
- **NOTA 3**: per la radice quadrata, usa `** (1 / 2)`. Qual'è il risultato di `1 / 2`?

DOMANDA: se hai svolto l'esercizio precedente, probabilmente avrai notato che gli slider sembrano 'ballare' un po' più del caso intero. Hai idea del perchè ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[26]:
```
import ipywidgets as widgets
from ipywidgets import FloatSlider, Label
from IPython.display import display


# scrivi qui
slider1 = widgets.FloatSlider(max=100)  # specifichiamo il limite estremo
display(slider1)
slider2 = widgets.FloatSlider(max=10000)  # ricordiamoci di specificare il limite␣
↪doppio del primo
```

(continues on next page)

(continued from previous page)

```
display(slider2)


# notare che stavolta dichiariamo la variabile 'change', che NON è un solo valore␣
↪come con interact
# ma un oggetto con diversi valori riguardanti il cambiamento generato dal click dell
↪'utente,
# il più interessante dei quali è `new`. Prova a togliere i commenti
# da print(change) per vedere l'oggetto change completo
def aggiorna1(change):
    #print("Il nuovo valore è " + str(change.new))
    #print(type(change))
    slider2.value = change.new ** 2  # potenza

def aggiorna2(change):
    #print("Il nuovo valore è " + str(change.new))
    #print(type(change))
    slider1.value = change.new ** (1/2) # divisione float !


#NOTA: adesso passiamo anche il parametro names=['value'] per filtrare i tipi di␣
↪change che riceviamo
slider1.observe(aggiorna1, names=['value'])
slider2.observe(aggiorna2, names=['value'])
```

```
FloatSlider(value=0.0)
```

```
FloatSlider(value=0.0, max=10000.0)
```

[26]:

```
import ipywidgets as widgets
from ipywidgets import FloatSlider, Label
from IPython.display import display


# scrivi qui
```

```
FloatSlider(value=0.0)
```

```
FloatSlider(value=0.0, max=10000.0)
```

Eventi di widget a selezione multipla

Vediamo adesso le `change` di un widget di selezione multipla:

[27]:

```
sm = widgets.SelectMultiple(
    options=['Apples', 'Oranges', 'Pears'],
    value=['Oranges'],
    #rows=10,
    description='Fruits',
    disabled=False
)
```

(continues on next page)

(continued from previous page)

```
def aggiorna(change):
    #print("Il nuovo valore è " + str(change['new']))
    #print(type(change))
    print(change)

#NOTA: adesso passiamo anche il parametro names=['value'] per filtrare i tipi di␣
↪change che riceviamo
sm.observe(aggiorna, names=['value'])

display(sm)
```

```
SelectMultiple(description='Fruits', index=(1,), options=('Apples', 'Oranges', 'Pears
↪'), value=('Oranges',))
```

Eventi click

Attento ai bottoni ! `.observe` va bene per widget in genere, ma per i bottoni ad Agosto 2018 non sembra funzionare. Per quest'ultimi usa gli eventi click con il metodo `.on_click` così:

[28]:
```
import ipywidgets as widgets
from IPython.display import display
from ipywidgets import Button

def cliccato(b):
    # nota che on_click passa il widget cliccato

    print("Questo bottone è stato cliccato:\n%s" % b)

bottone = widgets.Button(
        description='Stampa',
        disabled=False,
        button_style='',
        tooltip='stampa qualcosa',
        icon='check'
    )

bottone.on_click(cliccato)
display(bottone)
```

```
Button(description='Stampa', icon='check', style=ButtonStyle(), tooltip='stampa␣
↪qualcosa')
```

6.1.9 Layout e stili

Quando si vuole posizionare widget, si parla generalmente di layout. Ci sono tanti modi di posizionarli, vediamo i principali:

Layout: HBox

Possiamo usare HBox per mettere i widget uno dopo l'altro in orizzontale:

[29]:
```
from ipywidgets import Button, IntSlider, HBox, VBox, Label

HBox([IntSlider(), Button(description='hello')])
```

```
HBox(children=(IntSlider(value=0), Button(description='hello', style=ButtonStyle())))
```

Alcuni widget hanno parametri appositi per associare testo al widget, ma talvolta il testo viene accorciato senza il nostro consenso. Per ovviare a ciò possiamo usare il widget Label, che consente di forzare la visualizzazione di testo lungo, in combinazione con HBox:

[30]:
```
HBox([widgets.Label('A too long description:'),  widgets.IntSlider()])
```

```
HBox(children=(Label(value='A too long description:'), IntSlider(value=0)))
```

Layout: VBox and HBox

Ovviamente c'è anche il layout verticale `VBox`. Per ottenere configurazioni a griglia, è possibile usare un misto di `HBox` e `VBox` :

[31]:
```
from ipywidgets import Button, HBox, VBox

left_box = VBox([Button(description='alto a sinistra'), Button(description='basso a␣
↪sinistra')])
right_box = VBox([Button(description='alto a destra'), Button(description='basso a␣
↪destra')])
HBox([left_box, right_box])
```

```
HBox(children=(VBox(children=(Button(description='alto a sinistra',␣
↪style=ButtonStyle()), Button(description='…
```

Flexbox

Se hai esigenze di layout complesse, raccomandiamo di cuore i FlexBox, che sono praticamente l'unico sistema di layout decente del modello CSS (Cascading Style Sheet, di cui avete provato il linguaggio di query nel capitolo sull'estrazione dati da HTML[405]). Per fortuna Jupyter li supporta nativamente e li potete programmare direttamente in Python. Per una descrizione completa rimandiamo al sito di Jupyter Widgets[406]

[405] http://it.softpython.org/extraction/extraction-sol.html
[406] http://ipywidgets.readthedocs.io/en/stable/examples/Widget%20Styling.html#The-Flexbox-layout

Stile

E' possibile applicare vari stili ai bottoni, di nuovo facendo riferimento alle proprietà CSS[407] che sono supportate da Jupyter. Esempio

[32]:
```
from ipywidgets import Button, Layout

b = Button(description='Bottone con stile applicato ',
           layout=Layout(width='50%', height='80px'))
b
```

```
Button(description='Bottone con stile applicato ', layout=Layout(height='80px', width=
↪'50%'), style=ButtonStyl…
```

6.1.10 Grafici interattivi con bqplot

Bqplot è una libreria molto potente per realizzare grafici in Jupyter. In particolare bqplot :

- riprende i comandi già visti per matplot (*stile matlab*), che quindi possono riusati pari pari
- si integra bene con `ipywidgets`
- permette di esportare i grafici in codice interattivo HTML, facilmente inseribile in siti web, blog, etc..

Installazione bqplot

Anaconda:

Apri Anaconda Prompt (per istruzioni su come trovarlo o se non hai idea di cosa sia, prima di proseguire leggi sezione interprete Python nell'introduzione[408]) ed esegui:

`conda install -c conda-forge bqplot`

Installare `bqplot` con `conda` abiliterà automaticamente l'estensione per te in Jupyter

Linux/Mac:

- installa ipywidgets (`--user` installa nella propria home):

```
python3 -m pip install --user bqplot
```

- abilita l'estensione così:

```
jupyter nbextension enable --py bqplot
```

[407] http://ipywidgets.readthedocs.io/en/stable/examples/Widget%20Styling.html#Layout-and-Styling-of-Jupyter-widgets
[408] https://it.softpython.org/intro/intro-sol.html#L'interprete-Python

bqplot - il primo grafico

Adesso prova ad aprire Jupyter ed incollare il seguente codice in una cella, eseguendolo dovrebbe apparirti un grafico

Abbiamo parlato di grafici modificabili interattivamente, proviamo a crearne uno. Intanto creiamo un semplice grafico con `bqplot`, riprendendo istruzioni dal tutorial sulla Visulazzazione[409]

> **ATTENZIONE al plt!**
>
> Il `pyplot` che vedete qui sotto, che viene importato con il nome di `plt` proviene dalla libreria di bqplot, *non* è lo stesso `pyplot` di matplotlib !!
>
> Gli autori di bqplot hanno adottato lo stesso nome e convenzioni per permettervi di riusare facilmente esempi che già conoscete di matplotlib, ma **NON E' AFFATTO DETTO CHE TUTTI GLI ESEMPI DI MATPLOTLIB FUNZIONINO ANCHE CON BQPLOT !!**

[33]:
```
# !!!!  IMPORTANTE !!!!
# Il 'pyplot' che vedete qui sotto, che viene importato con il nome di 'plt'
# proviene dalla libreria di bqplot, quindi NON E' lo stesso pyplot di matplotlib !!
# Gli autori di bqplot hanno adottato lo stesso nome e convenzioni per permettervi
# di riusare facilmente esempi che già conoscete di matplotlib

from bqplot import pyplot as plt

plt.figure(title='Grafico in bqplot')
x = [1,2,3,4,5]
y = [2,4,8,16,32]
plt.plot(x, y, 'bo') # b=blue o=punti (sostuendo 'o' con 'l' farà delle linee)
plt.show()
```

```
VBox(children=(Figure(axes=[Axis(scale=LinearScale()), Axis(orientation='vertical',␣
↪scale=LinearScale())], fig…
```

bqplot - variare parametri

Visto un esempio che già conosciamo bene, proviamo a plottare una funzione un po' più complessa, come per esempio un coseno:

[34]:
```
# !!!!  IMPORTANTE !!!!
# Il 'pyplot' che vedete qui sotto, che viene importato con il nome di 'plt'
# proviene dalla libreria di bqplot, NON E' lo stesso pyplot di matplotlib !!
# Gli autori di bqplot hanno adottato lo stesso nome e convenzioni per permettervi
# di riusare facilmente esempi che già conoscete di matplotlib

from bqplot import pyplot as plt
import numpy as np


x = np.linspace(0, 2 * np.pi, 50)  # mette nel vettore x 50 punti tra 0 e 2 pi greco␣
↪(inclusi)
fig = plt.figure()                 # genera la figure
```

(continues on next page)

[409] http://it.softpython.org/visualization/visualization-sol.html

(continued from previous page)

```
# plottiamo un coseno
# plot ritorna una lista di linee Line2D, ma all'interno in questo caso ne contiene␣
↪una sola che estraiamo:
lines = plt.plot(x, np.cos(x))
plt.title('Grafico in bqplot')
plt.show()
```

```
VBox(children=(Figure(axes=[Axis(scale=LinearScale()), Axis(orientation='vertical',␣
↪scale=LinearScale())], fig…
```

Abbiamo creato il grafico, usando solo bqplot (*non* matplotlib, vedi commenti nel codice!). Per poter variare il grafico, dovremo indicare a bqplot dei nuovi valori per le `y`. Come fare? Se avete notato, prima abbiamo salvato il risultato di `plot` nella variabile `lines`. Ma cos'è esattamente? Che campi contiene? Scopriamolo:

[35]:
```
type(lines)
```

[35]:
```
bqplot.marks.Lines
```

[36]:
```
lines
```

[36]:
```
Lines(colors=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b', '
↪#e377c2', '#7f7f7f', '#bcbd22', '#17becf'], interactions={'hover': 'tooltip'},␣
↪scales={'x': LinearScale(), 'y': LinearScale()}, scales_metadata={'x': {'orientation
↪': 'horizontal', 'dimension': 'x'}, 'y': {'orientation': 'vertical', 'dimension': 'y
↪'}, 'color': {'dimension': 'color'}}, tooltip_style={'opacity': 0.9}, x=array([0.  ␣
↪     , 0.12822827, 0.25645654, 0.38468481, 0.51291309,
       0.64114136, 0.76936963, 0.8975979 , 1.02582617, 1.15405444,
       1.28228272, 1.41051099, 1.53873926, 1.66696753, 1.7951958 ,
       1.92342407, 2.05165235, 2.17988062, 2.30810889, 2.43633716,
       2.56456543, 2.6927937 , 2.82102197, 2.94925025, 3.07747852,
       3.20570679, 3.33393506, 3.46216333, 3.5903916 , 3.71861988,
       3.84684815, 3.97507642, 4.10330469, 4.23153296, 4.35976123,
       4.48798951, 4.61621778, 4.74444605, 4.87267432, 5.00090259,
       5.12913086, 5.25735913, 5.38558741, 5.51381568, 5.64204395,
       5.77027222, 5.89850049, 6.02672876, 6.15495704, 6.28318531]), y=array([ 1.    ␣
↪    ,  0.99179001,  0.96729486,  0.92691676,  0.8713187 ,
        0.80141362,  0.71834935,  0.6234898 ,  0.51839257,  0.40478334,
        0.28452759,  0.1595999 ,  0.03205158, -0.09602303, -0.22252093,
       -0.34536505, -0.46253829, -0.57211666, -0.67230089, -0.76144596,
       -0.8380881 , -0.90096887, -0.94905575, -0.98155916, -0.99794539,
       -0.99794539, -0.98155916, -0.94905575, -0.90096887, -0.8380881 ,
       -0.76144596, -0.67230089, -0.57211666, -0.46253829, -0.34536505,
       -0.22252093, -0.09602303,  0.03205158,  0.1595999 ,  0.28452759,
        0.40478334,  0.51839257,  0.6234898 ,  0.71834935,  0.80141362,
        0.8713187 ,  0.92691676,  0.96729486,  0.99179001,  1.        ]))
```

Il campo che ci interessa è `y`:

```
y=array([ 1.        ,  0.99179001,  0.96729486,  0.92691676,  0.8713187 ,
```

Proviamo a cambiarlo, variando per esempio la frequenza del coseno moltiplicando `x` per 5:

```
lines.y = np.cos(5 * x)
```

ESERCIZIO: Prova a incollare il codice di sopra qui sotto, eseguilo e poi guarda se il grafico di prima è cambiato. Prova anche a variare il numero davanti a `x`

[37]:
```
# copia qui
```

Adesso ci piacerebbe aggiungere uno slider per cambiare interattivamente la frequenza dell'onda, indicheremo tale frequenza con la variabile `k`. Per fare ciò, possiamo definire una nostra funzione `aggiorna`, che prende in input la frequenza `k`, e ricalcola le `lines.y`

NOTA: `aggiorna` prende un parametro `k` che definiamo noi. Potremmo anche chiamarlo `pippo`.

[38]:
```
def aggiorna(k = 1.0):
    lines.y = np.cos(k * x)
```

Ora non ci resta che dire a Jupyter di creare uno slider e chiamare `aggiorna` ogni volta che lo slider viene spostato. Possiamo farlo con la funzione di Jupyter `interact`.

NOTA: chiamando la funzione di Jupyter `interact`, come parametro gli passiamo *la funzione* aggiorna, NON il risultato della funzione aggiorna ! Dato che nella dichiarazione di aggiorna abbiamo messo un parametro `k` inizializzato da un float 1.0, Python capisce magicamente che siamo interessati a visualizzare un widget in grado di modificare valori float, e in questo caso creerà un bello slider!

[39]:
```
from ipywidgets import interact

interact(aggiorna);
```

```
interactive(children=(FloatSlider(value=1.0, description='k', max=3.0, min=-1.0),␣
↪Output()), _dom_classes=('wi…
```

Ricapitolando, ecco tutto il codice completo:

[40]:
```
# !!!!  IMPORTANTE !!!!
# Il 'pyplot' che vedete qui sotto, che viene importato con il nome di 'plt'
# proviene dalla libreria di bqplot, NON E' lo stesso pyplot di matplotlib !!
# Gli autori di bqplot hanno adottato lo stesso nome e convenzioni per permettervi
# di riusare facilmente esempi che già conoscete di matplotlib

from bqplot import pyplot as plt
import numpy as np
from ipywidgets import interactive

x = np.linspace(0, 2 * np.pi, 50)  # mette nel vettore x 50 punti tra 0 e 2 pi greco␣
↪(inclusi)
fig = plt.figure()                 # genera la figure

# plottiamo un coseno
# plot ritorna una lista di linee Line2D, ma all'interno in questo caso ne contiene␣
↪una sola che estraiamo:
lines = plt.plot(x, np.cos(x))
plt.title('Grafico in bqplot')
plt.show()

def aggiorna(k = 1.0):
    lines.y = np.cos(k * x)
slider_frequenza = interactive(aggiorna);

display(slider_frequenza)
```

```
VBox(children=(Figure(axes=[Axis(scale=LinearScale()), Axis(orientation='vertical',␣
↪scale=LinearScale())], fig…
```

```
interactive(children=(FloatSlider(value=1.0, description='k', max=3.0, min=-1.0),␣
↪Output()), _dom_classes=('wi…
```

bqplot - layout

Nel paragrafo precedente, abbiamo visto un primo esempio funzionante che ci permette di variare un parametri e mostrare gli aggiornamenti nel grafico. Come prossimo passo potremmo voler posizionare lo slider a sinistra del plot. Potremmo farlo usando i layout.

Il bello di Bqplot è che è ben integrato con gli ipywidgets. Per esempio, possiamo provare ad inserire il grafico del coseno di bqplot dentro un `HBox`, mettendo a sinistra lo slider usato in precedenza per variare la frequenza:

[41]:
```
from bqplot import pyplot as plt
import numpy as np
from ipywidgets import interactive
from ipywidgets import HBox, Box

x = np.linspace(0, 2 * np.pi, 50)  # mette nel vettore x 50 punti tra 0 e 2 pi greco␣
↪(inclusi)
fig = plt.figure()                 # genera la figure

# plottiamo un coseno
# plot ritorna una lista di linee Line2D, ma all'interno in questo caso ne contiene␣
↪una sola che estraiamo:
lines = plt.plot(x, np.cos(x))
plt.title('HBox con slider e bqplot')

def aggiorna(k = 1.0):
    lines.y = np.cos(k * x)

# otteniamo la variabile widget con interactive (che a differenza di 'interact'␣
↪ritorna un widget !)
slider_frequenza = interactive(aggiorna);

# Creiamo l'HBox passando in una lista prima la variabile dello slider, e poi la␣
↪`fig` di bqplot
HBox([slider_frequenza, fig])
```

```
HBox(children=(interactive(children=(FloatSlider(value=1.0, description='k', max=3.0,␣
↪min=-1.0), Output()), _d…
```

ESERCIZIO: Nell'esempio qua sopra, lo slider appare forse un po' troppo in alto. Se guardi la documentazione degli HBox[410] vedrai che gli `HBox` vengono definiti come dei `Box` aventi proprietà prese dal modello FlexiBox:

```
def HBox(*pargs, **kwargs):
    """Displays multiple widgets horizontally using the flexible box model."""
    box = Box(*pargs, **kwargs)
    box.layout.display = 'flex'
    box.layout.align_items = 'stretch'
    return box
```

Cercando un po' nella documentazione sull'allineamento oggetti[411] in FlexiBox, riuscireste a capire quale parametro Flexbox modificare affinchè lo slider risulti a metà altezza rispetto al grafico (nota che ci sono sia `align-items` e `align-content`)? Fai dei tentativi modificando la funzione `MiaHBox` qua sotto:

[410] https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20Styling.html#The-VBox-and-HBox-helpers
[411] https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20Styling.html#align-items

[42]:
```
# ESERCIZIO: guarda sotto la funzione 'MiaHBox'

from bqplot import pyplot as plt
import numpy as np
from ipywidgets import interactive
from ipywidgets import HBox, Box

x = np.linspace(0, 2 * np.pi, 50)  # mette nel vettore x 50 punti tra 0 e 2 pi greco␣
↪(inclusi)
fig = plt.figure()                 # genera la figure

# plottiamo un coseno
# plot ritorna una lista di linee Line2D, ma all'interno in questo caso ne contiene␣
↪una sola che estraiamo:
lines = plt.plot(x, np.cos(x))
plt.title('HBox con slider e bqplot')

def aggiorna(k = 1.0):
    lines.y = np.cos(k * x)

# otteniamo la variabile widget con interactive (che a differenza di 'interact'␣
↪ritorna un widget !)
slider_frequenza = interactive(aggiorna);

# Ripreso da documentazione di HBox

# ESERCIZIO: MODIFICA QUESTA FUNZIONE

def MiaHBox(*pargs, **kwargs):
    """Displays multiple widgets horizontally using the flexible box model."""
    box = Box(*pargs, **kwargs)
    box.layout.display = 'flex'
    box.layout.align_items = 'stretch'
    return box

MiaHBox([slider_frequenza, fig])
```

```
Box(children=(interactive(children=(FloatSlider(value=1.0, description='k', max=3.0,␣
↪min=-1.0), Output()), _do…
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[43]:
```
# SOLUZIONE

from bqplot import pyplot as plt
import numpy as np
from ipywidgets import interactive
from ipywidgets import HBox, Box

x = np.linspace(0, 2 * np.pi, 50)  # mette nel vettore x 50 punti tra 0 e 2 pi greco␣
↪(inclusi)
fig = plt.figure()                 # genera la figure

# plottiamo un coseno
# plot ritorna una lista di linee Line2D, ma all'interno in questo caso ne contiene␣
↪una sola che estraiamo:
```

(continues on next page)

(continued from previous page)

```
lines = plt.plot(x, np.cos(x))
plt.title('HBox con slider e bqplot')

def aggiorna(k = 1.0):
    lines.y = np.cos(k * x)

# otteniamo la variabile widget con interactive (che a differenza di 'interact'␣
↪ritorna un widget !)
slider_frequenza = interactive(aggiorna);

# Ripreso da documentazione di HBox

def MiaHBox(*pargs, **kwargs):
    """Displays multiple widgets horizontally using the flexible box model."""
    box = Box(*pargs, **kwargs)
    box.layout.display = 'flex'
    box.layout.align_items = 'center'
    return box

MiaHBox([slider_frequenza, fig])
```

```
Box(children=(interactive(children=(FloatSlider(value=1.0, description='k', max=3.0,␣
↪min=-1.0), Output()), _do…
```

[43]:

```
Box(children=(interactive(children=(FloatSlider(value=1.0, description='k', max=3.0,␣
↪min=-1.0), Output()), _do…
```

bqplot - esempi avanzati

Nella cartella esempi-bqplot che sta nella zip degli esercizi abbiamo inserito diversi esempi di bqplot, ti invitiamo a guardarli. Tanto per dare un'idea, ne mettiamo qui alcuni di rilevanti (scorri in fondo alle relative pagine per vedere i grafici visualizzati - a volte potrebbe anche essere necessario rieseguire il tutto con `Kernel->Restart and Run All`):

Esempi base in esempi-bqplot/Basic Plotting:

Pyplot.ipynb

- Basic Plotting.ipynb (usa lo stile proprio di bqplot per creare i grafici)

Esempi di interazione esempi-bqplot/Interactions:

- esempi-bqplot/Interactions/Interaction Layer.ipynb per vedere come permettere all'utente di selezionare un range in un grafico, o delle barre da un istogramma

Esempi avanzati in esempi-bqplot/Applications:

- Esempio visualizzazione reti Mobile Patent Suits
- Wealth of Nations - esempio evoluzione dati nel tempo

6.1.11 HTML

Il codice HTML è il codice con cui sono scritte tutte le pagine web, e gli ipywidgets permettono di creare un widget a partire da codice HTML. Qui per esempio lo usiamo per creare un titolo, ma in genere per fare interfacce di moderata complessità non è indispensabile conoscerlo. Se vuoi saperne di più, prova a seguire il tutorial web 1 di coderdojotrento[412]

[44]:
```
from ipywidgets import HTML
import ipywidgets as widgets

HTML('<h1 style="color:orange">Il mio titolo</h1> <br/>')
```

```
HTML(value='<h1 style="color:orange">Il mio titolo</h1> <br/>')
```

6.1.12 Mappe

E' possibile controllare da Python delle mappe geografiche visualizzate in Jupyter con la libreria ipyleaflet e OpenStreetMap, la mappa libera del mondo realizzata da volontari.

Vedi tutorial nel foglio gui-maps.ipynb[413]

6.1.13 Chatbot

Mostriamo un esempio di come costruire una semplice interfaccia stile chatbot con ipywidgets, che continua a proporre widget di input all'utente e relativo output.

Per i grafici, importeremo bqplot plt invece invece di matplotlib, perchè è più pensato per interagire con ipywidgets !

NOTA: Una volta effettuata una selezione, apparirà sotto un nuovo input, ma l'input precedente sarà ancora modificabile. Come esercizio, potresti provare a rendere l'input precedente non più modificabile una volta che è stato scelto un valore.

[45]:
```
# import matplotlib.pyplot as plt  # usiamo bqplot
from ipywidgets import IntSlider, Label, VBox, HTML
import ipywidgets as widgets

# NOTA: importiamo bqplot plt invece di matplotlib, perchè è più pensato per
#       interagire con ipywidgets !

from bqplot import pyplot as plt


def aggiorna(change):
    if change.new !="Select":
        #print(change)
        #print(change.new)
        scelta=change.new

        labels = ['oggi', 'domani', 'dopodomani']
        ys = [2,5,1]

        fig = plt.figure()

        xticks = labels
```

(continues on next page)

[412] http://coderdojotrento.it/web1
[413] https://it.softpython.org/gui/gui-maps-sol.html

(continued from previous page)

```
        p1 = plt.bar(xticks, ys, width=0.3 )

        #p2 = plt.bar(xticks,y, color=['b','g','r'], width=0.3, align="center",␣
↪bottom=p1)

        plt.title(scelta)

        vbox.children = vbox.children + (fig,crea_widget())


def crea_widget():
    dropdown = widgets.Dropdown(
        options=['Select', 'Trento', 'Rovereto', 'Bolzano'],
        value='Select',
        description='Città:',
        disabled=False,
    )

    dropdown.observe(aggiorna,  names=['value'])
    return dropdown

vbox = VBox([HTML("<h1>Chatbot Quanto pioverà?</h1>"),crea_widget()])


display(vbox)
```

```
VBox(children=(HTML(value='<h1>Chatbot Quanto pioverà?</h1>'), Dropdown(description=
↪'Città:', options=('Select…
```

6.1.14 Webapp

Finora abbiamo visualizzato i widget dentro Jupyter, ma ti starai chiedendo se puoi mostrarli come fossero in un vero e proprio sito.

Per ottenere un risultato simile ad una webapp, possiamo usare Voilà[414]

Installazione voila:

- se hai Anaconda, apri Anaconda Prompt: per istruzioni su come trovarlo (o se non hai idea di cosa sia!), prima di proseguire leggi sezione interprete Python nell'introduzione[415]

ATTENZIONE: la `a` finale di `voila` **non** è accentata

```
conda install -c conda-forge voila
```

- se hai Linux/Mac, scrivi:

```
python3 -m pip install --user voila
```

[414] https://github.com/QuantStack/voila
[415] https://it.softpython.org/intro/intro-sol.html#L'interprete-Python

Esempio my-webapp

Una webapp può semplicemente essere un singolo foglio Jupyter, anche quando prevediamo più pagine per il sito. In genere, si può fare un widget contenitore che chiameremo `mia_app` e quando si vuole simulare il combiamento di una pagina, si sostituisce un pannello all'interno di `mia_app`.

Abbiamo creato un esempio di webapp nel file mia-webapp.ipynb che è fornito nella stessa cartella di questi esercizi.

ATTENZIONE: mia-webapp.ipynb per funzionare ha bisogno che *installi anche bqplot*

Se guardi nel file my-webapp.ipynb](my-webapp.ipynb) troverai una variabile `my_app` che è un widget `VBox`:

```
mia_app = VBox( children=[titolo,     # supponiamo che il titolo sia sempre visibile␣
↪in tutto il sito
                          pagina1,    # al momento la prima 'pagina' è il widget tab
                          credits])   # supponiamo che il titolo sia sempre visibile␣
↪in tutto il sito
```

Sono definite anche due variabili `pagina1` e `pagina2` che sono widget che rappresentano i pannelli centrali. Quando vuoi sostituire un pannello, puoi chiamare la funzione `cambia_pagina`, passando la pagina desiderata:

```
cambia_pagina(pagina2)
```

Eseguiamo la webapp con voila

Voila dovrebbe permetterti di vedere la webapp senza gli ingombranti menu di Jupyter. Vediamo quindi cosa succede quando provi ad eseguire il notebook *my-webapp.ipynb* come se effettivamente fosse una webapp.

Una volta installato `voila`, da dentro il prompt dei comandi, (che è una finestra nera dove puoi immettere comandi testuali per il sistema operativo), raggiungi la cartella dove è contenuto questo foglio di esercizi, cioè `gui`

(per vedere in che cartella sei, scrivi `dir`, per entrare in una cartella che si chiama CARTELLA, scrivi `cd CARTELLA`)

Una volta nella cartella `interactive`, scrivi

```
voila my-webapp.ipynb --VoilaConfiguration.file_whitelist="['.*']"
```

il `--VoilaConfiguration.file_whitelist="['.*']"` serve a dire a voila che vogliamo che tutti i file presenti nella cartella servita siano accessibili, altrimenti per es. le immagini non saranno trovate. Questo va bene quando sviluppi, ma se ha in intenzione di pubblicare realmente il sito leggi la documentazione[416]

Nel prompt, dovrebbero apparire le scritte simili, e si dovrebbe anche aprire un browser internet con visualizzato il notebook come webapp:

```
[Voila] Using /tmp to store connection files
[Voila] Storing connection files in /tmp/voila__4pcw5dx.
[Voila] Serving static files from /home/da/Da/bin/anaconda3/lib/python3.7/site-
↪packages/voila/static.
[Voila] Voila is running at:
http://localhost:8866/
[Voila] Kernel started: 40fff204-d83a-4062-892c-daffaba3f9bd
```

Per spegnere il server, premi `Control-C`. Lo spegne in malo modo ma va bene lo stesso:

[416] https://voila.readthedocs.io/en/latest/customize.html#serving-static-files

```
^C[Voila] Stopping...
[Voila] Kernel shutdown: 40fff204-d83a-4062-892c-daffaba3f9bd
```

Layout per webapp

Per disporre i widget abbiamo fornito alcuni esempi in *my-webapp.ipynb*, altri più avanzati li puoi trovare in questo tutorial (in inglese)[417] e nella documentazione di Jupyter[418]

6.1.15 Collegare i widget

Guardando la sezione precedente sugli eventi, avrai notato che a volte modificando uno slider, se un'altro slider è collegato a volte potrebbe 'ballare' un po'. Magari non è piacevolissimo, ma per i vostri progetti dovrebbero essere sufficienti. Per completezza, menzioniamo comunque che per evitare i tremolii si possono usare i cosiddetti `traitlets` . Qua riportiamo solo un esempio veloce, per approfondire vedere la documentazione[419]:

[46]:
```
import traitlets
slider_intero_del = widgets.IntSlider(description="slider delayed", continuous_
↪update=False)
testo_intero_del = widgets.IntText(description="testo delayed", continuous_
↪update=False)

traitlets.link((slider_intero_del, 'value'), (testo_intero_del, 'value'))
widgets.VBox([slider_intero_del, testo_intero_del])
```

```
VBox(children=(IntSlider(value=0, continuous_update=False, description='slider delayed
↪'), IntText(value=0, des…
```

[47]:
```
import traitlets
slider_intero_con = widgets.IntSlider(description="slider con", continuous_
↪update=True)
testo_intero_con = widgets.IntText(description="testo con", continuous_update=True)

traitlets.link((slider_intero_con, 'value'), (testo_intero_con, 'value'))
widgets.VBox([slider_intero_con, testo_intero_con])
```

```
VBox(children=(IntSlider(value=0, description='slider con'), IntText(value=0,␣
↪continuous_update=True, descript…
```

6.2 Mappe interattive

6.2.1 Introduzione

Vediamo come controllare da Python delle mappe visualizzate in Jupyter con la libreria ipyleaflet[420] e OpenStreetMap[421], la mappa libera del mondo realizzata da volontari.

[417] https://blog.jupyter.org/introducing-templates-for-jupyter-widget-layouts-f72bcb35a662?gi=77d340a2f2fb
[418] https://ipywidgets.readthedocs.io/en/latest/examples/Layout%20Templates.html
[419] http://ipywidgets.readthedocs.io/en/latest/examples/Widget%20Events.html?highlight=traitlets#Traitlet-events
[420] https://ipyleaflet.readthedocs.io/
[421] https://www.openstreetmap.org

ATTENZIONE: Ciò che segue è solo una bozza MOLTO IN-PROGRESS !!!!!

Prerequisiti

Per proseguire è necessario prima aver letto il tutorial sulle interfacce utente[422] in Jupyter (che parla degli ipywidgets)

Riferimenti

- per una panoramica sul webmapping, la geolocalizzazione e OpenStreetMap, puoi vedere il tutorial Integrazione[423] dove si mappano gli agritur del Trentino.
- per una rapida guida sull'HTML, vedere tutorial CoderDojoTrento web 1[424]

Installazione ipyleaflet

Anaconda:

Apri Anaconda Prompt (per istruzioni su come trovarlo o se non hai idea di cosa sia, prima di proseguire leggi sezione interprete Python nell'introduzione[425]) ed esegui:

`conda install -c conda-forge ipyleaflet`

Installare `ipyleaflet` con `conda` abiliterà automaticamente l'estensione per te in Jupyter

Linux/Mac:

- installa ipywidgets (`--user` installa nella propria home):

```
python3 -m pip install --user  ipyleaflet
```

- abilita l'estensione così:

```
jupyter nbextension enable --py ipyleaflet
```

Proviamo OpenStreetMap

Per prima cosa prova a navigare OpenStreetMap:

https://www.openstreetmap.org/#map=12/46.0849/11.1461

Spostati con la mappa, cambia lo zoom e nota cosa appare nella barra in alto dell'indirizzo del browser: dopo il `#` troverete le coordinate (latitudine e longitudine) e livello di zoom separati da una `/` :

[422] https://it.softpython.org/gui/gui-sol.html
[423] https://it.softpython.org/integration/integration-sol.html
[424] https://www.coderdojotrento.it/web1
[425] https://it.softpython.org/intro/intro-sol.html#L'interprete-Python

Mettiamo un palloncino

Possiamo visualizzare una mappa in Jupyter sfruttando le coordinate trovate in OpenStreetMap. Inoltre, possiamo inserire dei palloncini (detti *marker*, vedi documentazione[426]), associando ad essi una descrizione formattata in linguaggio HTML - per una breve guida sull'HTML, vedere il tutorial CoderDojoTrento web 1[427]

Nota che nella descrizione possiamo anche aggiungere immagini. In questo caso l'immagine d'esempio (immagini/disi-unitn-it-logo.jpeg) risiede nella sottocartella `immagini` di questo foglio, ma volendo si potrebbero anche linkare foto da un sito qualunque usando il loro indirizzo per esteso, per esempio potresti provare a mettere questo indirizzo del logo CC BY che sta sul sito di softpython: https://it.softpython.org/_images/cc-by.png

[1]:
```
from ipywidgets import HTML, Layout, VBox, Button, Label, Image

from ipyleaflet import Map, Marker, Popup

# definiamo il centro della mappa su Trento (latitudine, longitudine)
centro_mappa = (46.0849,11.1461)

# per il livello di zoom giusto, puoi usare OpenStreetMap
mappa = Map(center=centro_mappa, zoom=12, close_popup_on_click=False)

# adesso andiamo a creare dei widget da mettere nel popup dei palloncini

# cominciamo con un bottone
bottone = Button(description="Cliccami")

def bottone_cliccato(b):
    b.description="Mi hai cliccato !"

bottone.on_click(bottone_cliccato)

# per mostrare del testo formattato che contenga anche immagini, si può usare il␣
↪linguaggio HTML
# per una breve guida sull'HTML, vedere il tutorial  https://www.coderdojotrento.it/
↪web1

# creiamo un widget HTML
# i tre doppi apici """ indicano che iniziamo una cosiddetta multistringa, cioè una␣
↪stringa su più righe
html = HTML("""
Descrizione in <b>linguaggio HTML</b>,<br>
<a target="_blank" href="https://www.coderdojotrento.it/web1">vedi tutorial</a>. <br>
Questa è un immagine: <br/>
<img src="immagini/disi-unitn-it-logo.jpeg">
""")

# creaimo il pannello del popup come una VBox che contiene i due widget definiti␣
↪precedentemente:

pannello_popup = VBox([bottone, html])


# il marcatore sarà un palloncino sul DISI, il Dipartimento di Informatica a Povo,␣
↪Trento
marcatore = Marker(location=(46.06700,11.14985))
```

(continues on next page)

[426] https://ipyleaflet.readthedocs.io/en/latest/api_reference/marker.html
[427] https://www.coderdojotrento.it/web1

(continued from previous page)

```
# associamo al marcatore un popup nella forma di un widget html
marcatore.popup = pannello_popup

# aggiungiamo il marcatore alla mappa
mappa.add_layer(marcatore)

# creiamo un widget che contenga titolo (in html) e la mappa
webapp = VBox([HTML("<h1>CLICCA SUI PALLONCINI</h1>"), mappa])

# infine forziamo Jupyter a mostrare il tutto:
display(webapp)
```

```
VBox(children=(HTML(value='<h1>CLICCA SUI PALLONCINI</h1>'), Map(basemap={'url':
↪'https://{s}.tile.openstreetm…
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[1]:
```
# SOLUZIONE
```

[1]:

[ ]:

6.3 Esempio webapp

Esempio di webapp mostrato per esecuzione con Voila

Per capire il funzionamento, vedere tutorial Applicazioni interattive - Sezione Webapp[428]

Nota: Le celle di testo di Jupyter come questa vengono mostrate anche da Voila.

[1]:
```
import ipywidgets as widgets
from ipywidgets import Button, HBox, VBox, Tab, IntSlider, Label, HTML, AppLayout,␣
↪Layout

# !!!!  IMPORTANTE !!!!
# Il 'pyplot' che vedete qui sotto, che viene importato con il nome di 'plt'
# proviene dalla libreria di bqplot, NON E' lo stesso pyplot di matplotlib !!
# Gli autori di bqplot hanno adottato lo stesso nome e convenzioni per permettervi
# di riusare facilmente esempi che già conoscete di matplotlib

from bqplot import pyplot as plt

x = [2,4,6]
fig = plt.figure()          # genera la figure

lines = plt.plot(x, [15,3,20])
plt.title('Grafico in bqplot')
```

(continues on next page)

[428] https://it.softpython.org/gui/gui-sol.html#Webapp

(continued from previous page)

```
slider1 = IntSlider()

bottone_vai_pag2 = Button(description="VAI PAGINA 2")


slider2 = IntSlider()
hbox2 = HBox([Button(description='clicca qui'), Button(description='cliccami!')])


tab1 = HBox(children=[fig, VBox([slider1, bottone_vai_pag2])])
tab2 = VBox(children=[slider2,
                      hbox2])

# al momento la prima 'pagina' è il widget Tab
pagina1 = widgets.Tab(children=[tab1, tab2],  layout=Layout(min_height='350px'))

pagina1.set_title(0, 'TAB COL PLOT')
pagina1.set_title(1, 'ALTRA TAB')

bottone_vai_pag1 = Button(description="VAI PAGINA 1")

pagina2 = HBox([
                Label("Questa è la seconda pagina"),
                bottone_vai_pag1
               ],
               layout=Layout(min_height='350px') )


# Il codice HTML è il codice con cui sono scritte le pagine web, qui lo
# usiamo per creare il titolo come esempio ma non è indispensabile conoscerlo
# Se vuoi saperne di più, prova a seguire questo tutorial:    http://coderdojotrento.
↪it/web1
titolo = HTML('<h1 style="color:orange">Webapp Incredibile</h1> <br/>')

# testo comune in fondo alla pagina
credits = Label("Credits: Interfacce Incredibili SRL")

# la struttura della nostra webapp è un pila VBox di elementi.
my_app = VBox( children=[titolo,    # supponiamo che il titolo sia sempre visibile in␣
↪tutto il sito
                         pagina1,   # al momento la prima 'pagina' è il widget tab
                         credits])  # supponiamo che il titolo sia sempre visibile␣
↪in tutto il sito

# questa funzione permette di cambiare la parte centrale della webapp passando un␣
↪nuovo widget
def cambia_pagina(nuova_pagina):

    # le parentesi tonde in questo contesto creano una tupla,
    # cioè una sequenza immutabile di elementi):
    my_app.children = (my_app.children[0],    # il widget del titolo precedente
                       nuova_pagina,           # widget che rappresenta la nuova␣
↪pagina
                       my_app.children[2])    # il widget dei credits precedente

def bottone_vai_pag2_cliccato(b):
```

(continues on next page)

(continued from previous page)

```
    cambia_pagina(pagina2)

bottone_vai_pag2.on_click(bottone_vai_pag2_cliccato)


def bottone_vai_pag1_cliccato(b):
    cambia_pagina(pagina1)

bottone_vai_pag1.on_click(bottone_vai_pag1_cliccato)

display(my_app)
```

```
VBox(children=(HTML(value='<h1 style="color:orange">Webapp Incredibile</h1> <br/>'),␣
↪Tab(children=(HBox(childr…
```

[ ]:

```

```

6.4 Database

6.4.1 Scarica zip esercizi

Naviga file online[429]

6.4.2 Introduzione

In questo tutorial affronteremo il tema database in Python:

- Uso SQLStudio con connessione a SQLite
- semplici query SQL da Python
- esempi con libreria Pandas

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
database
    database.ipynb
    database-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `database.ipynb`

[429] https://github.com/DavidLeoni/softpython-it/tree/master/database

- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

6.4.3 Guardiamo il database

Proveremo ad accedere via SQLiteStudio e Python al database Chinhook.

Il modello dati di Chinook rappresenta un uno store online di canzoni, e include tabelle per artisti (`Artist`), album (`Album`), tracce (`Track`), fatture (`Invoice`) e clienti (`Customer`):

I dati provengono da varie sorgenti:

- I dati relativi alle canzoni sono stati creati usando dati reali dalla libreria iTunes
- Le informazioni sui clienti sono state create manualmente usando nomi fittizi

- Gli indirizzi sono georeferenziabili su Google Maps, e altri dati ben formattatati (telefono, fax, email, etc.)
- Le informazioni sulle vendite sono auto-generate usando dati casuali per lungo un periodo di 4 anni

6.4.4 Connessione in SQLStudio

Scarica[430] e prova a lanciare SQLite Studio (non serve nemmeno l'installazione). Se ti dà problemi, in alternativa prova SQLite browser[431]:

Una volta scaricato e szippato SQLStudio, eseguilo e poi:

1. Dal menu in alto, clicca `Database->Add Database` e connettilo al database `chinook.sqlite`:

2. Clicca su `Test connection` per verificare che la connessione funzioni, poi premi `OK`.

Cominciamo a guardare una tabella semplice come `Album`.

ESERCIZIO: Prima di procedere, in SQLiteStudio, nel menu a sinistra sotto il nodo `Tables` fai doppio click sulla tabella `Album` e poi nel pannello principale a destra seleziona la tab `Data`.

Adesso nel pannello principale a destra seleziona la tab `Data`:

[430] https://sqlitestudio.pl/index.rvt?act=download
[431] http://sqlitebrowser.org/

Vediamo che ci sono 3 colonne, due con numeri `AlbumId` e `ArtistId` e una di stringhe, chiamata `Title`

NOTA: I nomi delle colonne in SQL possono essere arbitrariamente scelte da chi crea i database. Quindi non è strettamente necessario che i nomi delle colonne numeriche terminino con `Id`.

6.4.5 Connessione in Python

Proviamo adesso a recuperare gli stessi dati della tabella `Album` in Python. SQLite è talmente popolare che la libreria per accederlo viene fornita direttamente con Python, quindi non ci servirà installare niente di particolare e possiamo tuffarci subito nel codice:

[1]:
```
import sqlite3

conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)
```

L'operazione qua sopra crea un oggetto connessione e lo assegna alla variabile `conn`. A cosa ci connettiamo? Ad un database indicato dalla uri `file:chinook.sqlite?mode=rw`. Ma cos'è una URI? E' una stringa che denota una locazione da qualche parte, potrebbe essere un database accessibile come servizio via internet, o un file sul nostro disco: nel nostro caso vogliamo indicare un database che abbiamo su disco, perciò useremo il protocollo `file:`

SQLite andrà quindi a cercarsi su disco il file `chinook.sqlite`, nella stessa cartella dove stiamo eseguendo Jupyter. Se il file fosse in qualche sottodirectory, potremmo scrivere per es. `qualche/cartella/chinook.sqlite`

NOTA 1: ci stiamo connettendo al database in formato binario `.sqlite` , NON al file di testo `.sql` !

NOTA 2: stiamo specificando che lo vogliamo aprire in modalità `mode=rw`, cioè di lettura + scrittura (Read Write). SE il database non esiste, questa funzione lancerà un errore.

NOTA 3: se volessimo creare un nuovo database, dovremmo usare la modalità lettura + scrittura + creazione (Read Write Creation), specificando come parametro `mode=rwc` (notare la `c` in più)

NOTA 4: in tanti sistemi di database (SQLite incluso), di default quando ci si connette ad un database su disco non esistente, ne creano uno. Questo è causa di tantissime imprecazioni, perchè se si sbaglia a scrivere il nome del database non saranno segnalati errori e ci si ritroverà connessi ad un database vuoto, chiedendosi che fine abbiano fatto i dati. E ci si troverà anche il disco pieno di file di database con nomi sbagliati!

Tramite l'oggetto connessione `conn` possiamo creare un cosiddetto cursore, che ci consentintirà di eseguire query verso il database. Usare una connessione per fare query equivale a chiedere una risorsa del sistema a Python. Le regole di buona educazione ci dicono che quando chiediamo in prestito qualcosa, dopo averlo usato lo si restituisce. La 'restituzione' equivarrebbe in Python a *chiudere* la risorsa aperta. Ma mentre usiamo la risorsa si potrebbe verificare un errore, che

potrebbe impedirci di chiudere la risorsa correttamente. Per indicare a Pyhton che vogliamo che la risorsa venga chiusa automaticamente in caso di errore, usiamo il comando `with` come abbiamo fatto per i file:

[2]:
```
import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

with conn:                          # col blocco with ci cauteliamo da errori imprevisti
    cursore = conn.cursor()         # otteniamo il cursore
    cursore.execute("SELECT * FROM Album LIMIT 5")       # eseguiamo una query in␣
↪linguaggio SQL al database
                                                         # notare che execute di per␣
↪sè non ritorna

    for riga in cursore.fetchall():       # cursore fetchall() genera una sequenza di␣
↪righe di risultato della query.
                                          # in sequenza, le righe una alla volta␣
↪vengono assegnate'oggetto `riga`
        print(riga)                       # stampiamo la riga ottenuta
```

```
(1, 'For Those About To Rock We Salute You', 1)
(2, 'Balls to the Wall', 2)
(3, 'Restless and Wild', 2)
(4, 'Let There Be Rock', 1)
(5, 'Big Ones', 3)
```

Finalmente abbiamo ottenuto la lista delle prime 5 righe dal database per la tabella `Album`.

ESERCIZIO: prova a scrivere qua sotto le istruzioni per stampare direttamente tutto risultato di `cursore.fetchall()`.

- Qual'è il tipo di oggetto che ottieni?
- Inoltre, qual'è il tipo delle singole righe (nota che sono rappresentate in parentesi tonde)?

[3]:
```
# scrivi il codice
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[4]:
```
# SOLUZIONE
# E' una lista Python. Le singole righe sono delle tuple (cioè sequenze immutabili)
import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

with conn:
    cursore = conn.cursor()
    cursore.execute("SELECT * FROM Album LIMIT 5")
    print(cursore.fetchall())
    print(type(cursore.fetchall()))
```

```
[(1, 'For Those About To Rock We Salute You', 1), (2, 'Balls to the Wall', 2), (3,
↪'Restless and Wild', 2), (4, 'Let There Be Rock', 1), (5, 'Big Ones', 3)]
<class 'list'>
```

[4]:

```
[(1, 'For Those About To Rock We Salute You', 1), (2, 'Balls to the Wall', 2), (3,
↪'Restless and Wild', 2), (4, 'Let There Be Rock', 1), (5, 'Big Ones', 3)]
<class 'list'>
```

6.4.6 Performance

I database sono pensati apposta per gestire grandi quantità dati che risiedono su hard-disk. Vediamo brevemente i vari tipi di memoria disponibili nel computer, e come vengono usati dai database:

Memoria	Velocità (in lentezza rispetto a RAM)	Quantità	Note
RAM	1x	4-16 gigabyte	si cancella allo spegnimento del computer
Disco SSD	2x-10x	centinaia di gigabyte	persistente, ma troppe scritture la rovinano
hard disk	100x	centinaia di gigabyte, terabyte	persistente, può sopportare numerose scritture

Se facciamo delle query complesse che potenzialmente vanno a elaborare parecchi dati, non sempre questi possono stare tutti in RAM. Pensiamo come esempio di chiedere al db di calcolare la media delle vendite di tutte le canzoni (supponi di avere terabyte di canzoni). Fortunatamente, spesso il database si arrangia da solo a creare un piano per ottimizzare l'uso delle risorse. Nel caso della media delle canzoni vendute, potrebbe per esempio eseguire autonomamente tutte queste operazioni:

1. caricare dall'hard-disk alla RAM 4 gigabyte di canzoni
2. calcolare media vendite di queste canzoni sul blocco corrente in RAM
3. scaricare la RAM
4. caricare dall'hard-disk alla RAM altri 4 gigabyte di canzoni
5. calcolare media vendite del secondo blocco canzoni in RAM, e fare media con la media risultata dal primo blocco
6. scaricare la RAM
7. etc ….

Nello scenario ideale possiamo scrivere query SQL complesse e sperare che il database se la cavi rapidamente a darci direttamente i risultati che ci servono in Python, salvandoci parecchio di lavoro. Purtroppo a volte ciò non è possibile, ci accorgiamo che il database ci mette una vita e bisogna ottimizzare a mano la query SQL, oppure il modo in cui carichiamo e rielaboriamo i dati in Python. Per ragioni di spazio in questo tutorial tratteremo solo l'ultimo caso, in modo molto semplice.

Prendere i dati un po' alla volta

Nei primi comandi Python sopra abbiamo visto come prelevare un po' di righe dal DB usando l'opzione SQL `LIMIT`, e come caricare tutte queste righe in un colpo solo in una lista Python con `fetchall`. E se volessimo stampare a video *tutte* le righe di una tabella da 1 terabyte, come faremmo? Sicuramente, se provassimo a caricarle tutte in una lista, Python finirebbe con saturare la memoria RAM. In alternativa al `fetchall`, possiamo usare il comando `fetchmany`, che prende un po' di righe alla volta:

[5]:
```
import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

with conn:
```

(continues on next page)

(continued from previous page)

```
    cursore = conn.cursor()
    cursore.execute("SELECT * FROM Album")
    while True: # fintanto che True è vero, cioè il ciclo apparentemente non termina␣
↪mai ...
        righe = cursore.fetchmany(5)    # prende 5 righe
        if len(righe) > 0:              # se abbiamo delle righe, le stampa
            for riga in righe:
                print(riga)
        else:                           # altrimenti interrompe forzatamente il ciclo␣
↪while
            break
```

```
(1, 'For Those About To Rock We Salute You', 1)
(2, 'Balls to the Wall', 2)
(3, 'Restless and Wild', 2)
(4, 'Let There Be Rock', 1)
(5, 'Big Ones', 3)
(6, 'Jagged Little Pill', 4)
(7, 'Facelift', 5)
(8, 'Warner 25 Anos', 6)
(9, 'Plays Metallica By Four Cellos', 7)
(10, 'Audioslave', 8)
(11, 'Out Of Exile', 8)
(12, 'BackBeat Soundtrack', 9)
(13, 'The Best Of Billy Cobham', 10)
(14, 'Alcohol Fueled Brewtality Live! [Disc 1]', 11)
(15, 'Alcohol Fueled Brewtality Live! [Disc 2]', 11)
(16, 'Black Sabbath', 12)
(17, 'Black Sabbath Vol. 4 (Remaster)', 12)
(18, 'Body Count', 13)
(19, 'Chemical Wedding', 14)
(20, 'The Best Of Buddy Guy - The Millenium Collection', 15)
(21, 'Prenda Minha', 16)
(22, 'Sozinho Remix Ao Vivo', 16)
(23, 'Minha Historia', 17)
(24, 'Afrociberdelia', 18)
(25, 'Da Lama Ao Caos', 18)
(26, 'Acústico MTV [Live]', 19)
(27, 'Cidade Negra - Hits', 19)
(28, 'Na Pista', 20)
(29, 'Axé Bahia 2001', 21)
(30, 'BBC Sessions [Disc 1] [Live]', 22)
(31, 'Bongo Fury', 23)
(32, 'Carnaval 2001', 21)
(33, 'Chill: Brazil (Disc 1)', 24)
(34, 'Chill: Brazil (Disc 2)', 6)
(35, 'Garage Inc. (Disc 1)', 50)
(36, 'Greatest Hits II', 51)
(37, 'Greatest Kiss', 52)
(38, 'Heart of the Night', 53)
(39, 'International Superhits', 54)
(40, 'Into The Light', 55)
(41, 'Meus Momentos', 56)
(42, 'Minha História', 57)
(43, 'MK III The Final Concerts [Disc 1]', 58)
(44, 'Physical Graffiti [Disc 1]', 22)
```

(continues on next page)

(continued from previous page)

```
(45, 'Sambas De Enredo 2001', 21)
(46, 'Supernatural', 59)
(47, 'The Best of Ed Motta', 37)
(48, 'The Essential Miles Davis [Disc 1]', 68)
(49, 'The Essential Miles Davis [Disc 2]', 68)
(50, 'The Final Concerts (Disc 2)', 58)
(51, "Up An' Atom", 69)
(52, 'Vinícius De Moraes - Sem Limite', 70)
(53, 'Vozes do MPB', 21)
(54, 'Chronicle, Vol. 1', 76)
(55, 'Chronicle, Vol. 2', 76)
(56, 'Cássia Eller - Coleção Sem Limite [Disc 2]', 77)
(57, 'Cássia Eller - Sem Limite [Disc 1]', 77)
(58, 'Come Taste The Band', 58)
(59, 'Deep Purple In Rock', 58)
(60, 'Fireball', 58)
(61, "Knocking at Your Back Door: The Best Of Deep Purple in the 80's", 58)
(62, 'Machine Head', 58)
(63, 'Purpendicular', 58)
(64, 'Slaves And Masters', 58)
(65, 'Stormbringer', 58)
(66, 'The Battle Rages On', 58)
(67, "Vault: Def Leppard's Greatest Hits", 78)
(68, 'Outbreak', 79)
(69, 'Djavan Ao Vivo - Vol. 02', 80)
(70, 'Djavan Ao Vivo - Vol. 1', 80)
(71, 'Elis Regina-Minha História', 41)
(72, 'The Cream Of Clapton', 81)
(73, 'Unplugged', 81)
(74, 'Album Of The Year', 82)
(75, 'Angel Dust', 82)
(76, 'King For A Day Fool For A Lifetime', 82)
(77, 'The Real Thing', 82)
(78, 'Deixa Entrar', 83)
(79, 'In Your Honor [Disc 1]', 84)
(80, 'In Your Honor [Disc 2]', 84)
(81, 'One By One', 84)
(82, 'The Colour And The Shape', 84)
(83, 'My Way: The Best Of Frank Sinatra [Disc 1]', 85)
(84, 'Roda De Funk', 86)
(85, 'As Canções de Eu Tu Eles', 27)
(86, 'Quanta Gente Veio Ver (Live)', 27)
(87, 'Quanta Gente Veio ver--Bônus De Carnaval', 27)
(88, 'Faceless', 87)
(89, 'American Idiot', 54)
(90, 'Appetite for Destruction', 88)
(91, 'Use Your Illusion I', 88)
(92, 'Use Your Illusion II', 88)
(93, 'Blue Moods', 89)
(94, 'A Matter of Life and Death', 90)
(95, 'A Real Dead One', 90)
(96, 'A Real Live One', 90)
(97, 'Brave New World', 90)
(98, 'Dance Of Death', 90)
(99, 'Fear Of The Dark', 90)
(100, 'Iron Maiden', 90)
(101, 'Killers', 90)
```

(continues on next page)

(continued from previous page)

```
(102, 'Live After Death', 90)
(103, 'Live At Donington 1992 (Disc 1)', 90)
(104, 'Live At Donington 1992 (Disc 2)', 90)
(105, 'No Prayer For The Dying', 90)
(106, 'Piece Of Mind', 90)
(107, 'Powerslave', 90)
(108, 'Rock In Rio [CD1]', 90)
(109, 'Rock In Rio [CD2]', 90)
(110, 'Seventh Son of a Seventh Son', 90)
(111, 'Somewhere in Time', 90)
(112, 'The Number of The Beast', 90)
(113, 'The X Factor', 90)
(114, 'Virtual XI', 90)
(115, 'Sex Machine', 91)
(116, 'Emergency On Planet Earth', 92)
(117, 'Synkronized', 92)
(118, 'The Return Of The Space Cowboy', 92)
(119, 'Get Born', 93)
(120, 'Are You Experienced?', 94)
(121, 'Surfing with the Alien (Remastered)', 95)
(122, 'Jorge Ben Jor 25 Anos', 46)
(123, 'Jota Quest-1995', 96)
(124, 'Cafezinho', 97)
(125, 'Living After Midnight', 98)
(126, 'Unplugged [Live]', 52)
(127, 'BBC Sessions [Disc 2] [Live]', 22)
(128, 'Coda', 22)
(129, 'Houses Of The Holy', 22)
(130, 'In Through The Out Door', 22)
(131, 'IV', 22)
(132, 'Led Zeppelin I', 22)
(133, 'Led Zeppelin II', 22)
(134, 'Led Zeppelin III', 22)
(135, 'Physical Graffiti [Disc 2]', 22)
(136, 'Presence', 22)
(137, 'The Song Remains The Same (Disc 1)', 22)
(138, 'The Song Remains The Same (Disc 2)', 22)
(139, 'A TempestadeTempestade Ou O Livro Dos Dias', 99)
(140, 'Mais Do Mesmo', 99)
(141, 'Greatest Hits', 100)
(142, 'Lulu Santos - RCA 100 Anos De Música - Álbum 01', 101)
(143, 'Lulu Santos - RCA 100 Anos De Música - Álbum 02', 101)
(144, 'Misplaced Childhood', 102)
(145, 'Barulhinho Bom', 103)
(146, 'Seek And Shall Find: More Of The Best (1963-1981)', 104)
(147, 'The Best Of Men At Work', 105)
(148, 'Black Album', 50)
(149, 'Garage Inc. (Disc 2)', 50)
(150, "Kill 'Em All", 50)
(151, 'Load', 50)
(152, 'Master Of Puppets', 50)
(153, 'ReLoad', 50)
(154, 'Ride The Lightning', 50)
(155, 'St. Anger', 50)
(156, '…And Justice For All', 50)
(157, 'Miles Ahead', 68)
(158, 'Milton Nascimento Ao Vivo', 42)
```

(continues on next page)

(continued from previous page)

```
(159, 'Minas', 42)
(160, 'Ace Of Spades', 106)
(161, 'Demorou…', 108)
(162, 'Motley Crue Greatest Hits', 109)
(163, 'From The Muddy Banks Of The Wishkah [Live]', 110)
(164, 'Nevermind', 110)
(165, 'Compositores', 111)
(166, 'Olodum', 112)
(167, 'Acústico MTV', 113)
(168, 'Arquivo II', 113)
(169, 'Arquivo Os Paralamas Do Sucesso', 113)
(170, 'Bark at the Moon (Remastered)', 114)
(171, 'Blizzard of Ozz', 114)
(172, 'Diary of a Madman (Remastered)', 114)
(173, 'No More Tears (Remastered)', 114)
(174, 'Tribute', 114)
(175, 'Walking Into Clarksdale', 115)
(176, 'Original Soundtracks 1', 116)
(177, 'The Beast Live', 117)
(178, 'Live On Two Legs [Live]', 118)
(179, 'Pearl Jam', 118)
(180, 'Riot Act', 118)
(181, 'Ten', 118)
(182, 'Vs.', 118)
(183, 'Dark Side Of The Moon', 120)
(184, 'Os Cães Ladram Mas A Caravana Não Pára', 121)
(185, 'Greatest Hits I', 51)
(186, 'News Of The World', 51)
(187, 'Out Of Time', 122)
(188, 'Green', 124)
(189, 'New Adventures In Hi-Fi', 124)
(190, 'The Best Of R.E.M.: The IRS Years', 124)
(191, 'Cesta Básica', 125)
(192, 'Raul Seixas', 126)
(193, 'Blood Sugar Sex Magik', 127)
(194, 'By The Way', 127)
(195, 'Californication', 127)
(196, 'Retrospective I (1974-1980)', 128)
(197, 'Santana - As Years Go By', 59)
(198, 'Santana Live', 59)
(199, 'Maquinarama', 130)
(200, 'O Samba Poconé', 130)
(201, 'Judas 0: B-Sides and Rarities', 131)
(202, 'Rotten Apples: Greatest Hits', 131)
(203, 'A-Sides', 132)
(204, 'Morning Dance', 53)
(205, 'In Step', 133)
(206, 'Core', 134)
(207, 'Mezmerize', 135)
(208, '[1997] Black Light Syndrome', 136)
(209, 'Live [Disc 1]', 137)
(210, 'Live [Disc 2]', 137)
(211, 'The Singles', 138)
(212, 'Beyond Good And Evil', 139)
(213, 'Pure Cult: The Best Of The Cult (For Rockers, Ravers, Lovers & Sinners) [UK]',␣
↪139)
(214, 'The Doors', 140)
```

(continues on next page)

(continued from previous page)

```
(215, 'The Police Greatest Hits', 141)
(216, 'Hot Rocks, 1964-1971 (Disc 1)', 142)
(217, 'No Security', 142)
(218, 'Voodoo Lounge', 142)
(219, 'Tangents', 143)
(220, 'Transmission', 143)
(221, 'My Generation - The Very Best Of The Who', 144)
(222, 'Serie Sem Limite (Disc 1)', 145)
(223, 'Serie Sem Limite (Disc 2)', 145)
(224, 'Acústico', 146)
(225, 'Volume Dois', 146)
(226, 'Battlestar Galactica: The Story So Far', 147)
(227, 'Battlestar Galactica, Season 3', 147)
(228, 'Heroes, Season 1', 148)
(229, 'Lost, Season 3', 149)
(230, 'Lost, Season 1', 149)
(231, 'Lost, Season 2', 149)
(232, 'Achtung Baby', 150)
(233, "All That You Can't Leave Behind", 150)
(234, 'B-Sides 1980-1990', 150)
(235, 'How To Dismantle An Atomic Bomb', 150)
(236, 'Pop', 150)
(237, 'Rattle And Hum', 150)
(238, 'The Best Of 1980-1990', 150)
(239, 'War', 150)
(240, 'Zooropa', 150)
(241, 'UB40 The Best Of - Volume Two [UK]', 151)
(242, 'Diver Down', 152)
(243, 'The Best Of Van Halen, Vol. I', 152)
(244, 'Van Halen', 152)
(245, 'Van Halen III', 152)
(246, 'Contraband', 153)
(247, 'Vinicius De Moraes', 72)
(248, 'Ao Vivo [IMPORT]', 155)
(249, 'The Office, Season 1', 156)
(250, 'The Office, Season 2', 156)
(251, 'The Office, Season 3', 156)
(252, 'Un-Led-Ed', 157)
(253, 'Battlestar Galactica (Classic), Season 1', 158)
(254, 'Aquaman', 159)
(255, 'Instant Karma: The Amnesty International Campaign to Save Darfur', 150)
(256, 'Speak of the Devil', 114)
(257, '20th Century Masters - The Millennium Collection: The Best of Scorpions', 179)
(258, 'House of Pain', 180)
(259, 'Radio Brasil (O Som da Jovem Vanguarda) - Seleccao de Henrique Amaro', 36)
(260, 'Cake: B-Sides and Rarities', 196)
(261, 'LOST, Season 4', 149)
(262, 'Quiet Songs', 197)
(263, 'Muso Ko', 198)
(264, 'Realize', 199)
(265, 'Every Kind of Light', 200)
(266, 'Duos II', 201)
(267, 'Worlds', 202)
(268, 'The Best of Beethoven', 203)
(269, 'Temple of the Dog', 204)
(270, 'Carry On', 205)
(271, 'Revelations', 8)
```

(continues on next page)

(continued from previous page)

```
(272, 'Adorate Deum: Gregorian Chant from the Proper of the Mass', 206)
(273, 'Allegri: Miserere', 207)
(274, 'Pachelbel: Canon & Gigue', 208)
(275, 'Vivaldi: The Four Seasons', 209)
(276, 'Bach: Violin Concertos', 210)
(277, 'Bach: Goldberg Variations', 211)
(278, 'Bach: The Cello Suites', 212)
(279, 'Handel: The Messiah (Highlights)', 213)
(280, 'The World of Classical Favourites', 214)
(281, 'Sir Neville Marriner: A Celebration', 215)
(282, 'Mozart: Wind Concertos', 216)
(283, 'Haydn: Symphonies 99 - 104', 217)
(284, 'Beethoven: Symhonies Nos. 5 & 6', 218)
(285, 'A Soprano Inspired', 219)
(286, 'Great Opera Choruses', 220)
(287, 'Wagner: Favourite Overtures', 221)
(288, 'Fauré: Requiem, Ravel: Pavane & Others', 222)
(289, 'Tchaikovsky: The Nutcracker', 223)
(290, 'The Last Night of the Proms', 224)
(291, 'Puccini: Madama Butterfly - Highlights', 225)
(292, 'Holst: The Planets, Op. 32 & Vaughan Williams: Fantasies', 226)
(293, "Pavarotti's Opera Made Easy", 227)
(294, "Great Performances - Barber's Adagio and Other Romantic Favorites for Strings",
↪ 228)
(295, 'Carmina Burana', 229)
(296, 'A Copland Celebration, Vol. I', 230)
(297, 'Bach: Toccata & Fugue in D Minor', 231)
(298, 'Prokofiev: Symphony No.1', 232)
(299, 'Scheherazade', 233)
(300, 'Bach: The Brandenburg Concertos', 234)
(301, 'Chopin: Piano Concertos Nos. 1 & 2', 235)
(302, 'Mascagni: Cavalleria Rusticana', 236)
(303, 'Sibelius: Finlandia', 237)
(304, 'Beethoven Piano Sonatas: Moonlight & Pastorale', 238)
(305, 'Great Recordings of the Century - Mahler: Das Lied von der Erde', 240)
(306, 'Elgar: Cello Concerto & Vaughan Williams: Fantasias', 241)
(307, 'Adams, John: The Chairman Dances', 242)
(308, "Tchaikovsky: 1812 Festival Overture, Op.49, Capriccio Italien & Beethoven:␣
↪Wellington's Victory", 243)
(309, 'Palestrina: Missa Papae Marcelli & Allegri: Miserere', 244)
(310, 'Prokofiev: Romeo & Juliet', 245)
(311, 'Strauss: Waltzes', 226)
(312, 'Berlioz: Symphonie Fantastique', 245)
(313, 'Bizet: Carmen Highlights', 246)
(314, 'English Renaissance', 247)
(315, 'Handel: Music for the Royal Fireworks (Original Version 1749)', 208)
(316, 'Grieg: Peer Gynt Suites & Sibelius: Pelléas et Mélisande', 248)
(317, 'Mozart Gala: Famous Arias', 249)
(318, 'SCRIABIN: Vers la flamme', 250)
(319, 'Armada: Music from the Courts of England and Spain', 251)
(320, 'Mozart: Symphonies Nos. 40 & 41', 248)
(321, 'Back to Black', 252)
(322, 'Frank', 252)
(323, 'Carried to Dust (Bonus Track Version)', 253)
(324, "Beethoven: Symphony No. 6 'Pastoral' Etc.", 254)
(325, 'Bartok: Violin & Viola Concertos', 255)
(326, "Mendelssohn: A Midsummer Night's Dream", 256)
```

(continues on next page)

(continued from previous page)

```
(327, 'Bach: Orchestral Suites Nos. 1 - 4', 257)
(328, 'Charpentier: Divertissements, Airs & Concerts', 258)
(329, 'South American Getaway', 259)
(330, 'Górecki: Symphony No. 3', 260)
(331, 'Purcell: The Fairy Queen', 261)
(332, 'The Ultimate Relexation Album', 262)
(333, 'Purcell: Music for the Queen Mary', 263)
(334, 'Weill: The Seven Deadly Sins', 264)
(335, 'J.S. Bach: Chaconne, Suite in E Minor, Partita in E Major & Prelude, Fugue and␣
↪Allegro', 265)
(336, 'Prokofiev: Symphony No.5 & Stravinksy: Le Sacre Du Printemps', 248)
(337, 'Szymanowski: Piano Works, Vol. 1', 266)
(338, 'Nielsen: The Six Symphonies', 267)
(339, "Great Recordings of the Century: Paganini's 24 Caprices", 268)
(340, "Liszt - 12 Études D'Execution Transcendante", 269)
(341, 'Great Recordings of the Century - Shubert: Schwanengesang, 4 Lieder', 270)
(342, 'Locatelli: Concertos for Violin, Strings and Continuo, Vol. 3', 271)
(343, 'Respighi:Pines of Rome', 226)
(344, "Schubert: The Late String Quartets & String Quintet (3 CD's)", 272)
(345, "Monteverdi: L'Orfeo", 273)
(346, 'Mozart: Chamber Music', 274)
(347, 'Koyaanisqatsi (Soundtrack from the Motion Picture)', 275)
```

6.4.7 Passare parametri alla query

E se volessimo passare agevolmente dei parametri alla query, come per esempio il numero dei risultati da ottenere? Per fare ciò possiamo usare dei cosiddetti *placeholder*, cioè dei caratteri punto di domanda `?` che segnano dove vorremmo mettere le variabili. In questo caso sistituiremo il `5` con un punto di domanda, e passeremo il `5` in una lista di parametri a parte:

[6]:
```
import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)


with conn:                          # col blocco with ci cauteliamo da errori imprevisti
    cursore = conn.cursor()         # otteniamo il cursore
    cursore.execute("SELECT * FROM Album LIMIT ?", [5]) # eseguiamo una query in␣
↪linguaggio SQL al database
                                                        # notare che execute di per␣
↪sè non ritorna

    for riga in cursore.fetchall():       # cursore fetchall() genera una sequenza di␣
↪righe di risultato della query.
                                          # in sequenza, le righe una alla volta␣
↪vengono assegnate'oggetto `riga`
        print(riga)                       # stampiamo la riga ottenuta
```

```
(1, 'For Those About To Rock We Salute You', 1)
(2, 'Balls to the Wall', 2)
(3, 'Restless and Wild', 2)
(4, 'Let There Be Rock', 1)
(5, 'Big Ones', 3)
```

Si possono anche aggiungere più punti di domanda, basta per ognuno passare il corrispondente parametro nella lista:

[7]:
```
import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)


with conn:                          # col blocco with ci cauteliamo da errori imprevisti
    cursore = conn.cursor()         # otteniamo il cursore
    cursore.execute("SELECT * FROM Album WHERE AlbumId < ? AND ArtistId < ?", [30,5])

    for riga in cursore.fetchall():       # cursore fetchall() genera una sequenza di␣
↪righe di risultato della query.
                                          # in sequenza, le righe una alla volta␣
↪vengono assegnate'oggetto `riga`
        print(riga)                       # stampiamo la riga ottenuta
```

```
(1, 'For Those About To Rock We Salute You', 1)
(2, 'Balls to the Wall', 2)
(3, 'Restless and Wild', 2)
(4, 'Let There Be Rock', 1)
(5, 'Big Ones', 3)
(6, 'Jagged Little Pill', 4)
```

6.4.8 Funzione Esegui Query

Per agevolare le prossime operazioni, ci definiamo una funzione `esegui` che esegue le query che desideriamo e ritorna la lista delle righe ottenute:

IMPORTANTE: Fai `Ctrl+Invio` nella cella seguente così Python in seguito riconoscerà la funzione:

[8]:
```
def esegui(conn, query, params=()):
    """
    Esegue una query usando la connessione conn, e ritorna la lista di risultati␣
↪ottenuti.

    In params, possiamo mettere una lista di parametri
    con i parametri per la nostra query.
    """
    with conn:
        cur = conn.cursor()
        cur.execute(query, params)
        return cur.fetchall()
```

Facciamo una prova:

[9]:
```
import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

esegui(conn, "SELECT * FROM Album LIMIT 5")
```

[9]:
```
[(1, 'For Those About To Rock We Salute You', 1),
 (2, 'Balls to the Wall', 2),
 (3, 'Restless and Wild', 2),
 (4, 'Let There Be Rock', 1),
 (5, 'Big Ones', 3)]
```

Meglio ancora, per maggiore chiarezza possiamo scrivere la query usando una stringa su più linee con le triple doppie virgolette all'inizio e alla fine:

[10]:
```
import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)


esegui(conn, """
SELECT *
FROM Album
LIMIT 5
""")
```

[10]:
```
[(1, 'For Those About To Rock We Salute You', 1),
 (2, 'Balls to the Wall', 2),
 (3, 'Restless and Wild', 2),
 (4, 'Let There Be Rock', 1),
 (5, 'Big Ones', 3)]
```

Proviamo a passare dei parametri:

[11]:
```
esegui(conn, """
SELECT *
FROM Album
WHERE AlbumId < ? AND ArtistId < ?
""", (30, 5))
```

[11]:
```
[(1, 'For Those About To Rock We Salute You', 1),
 (2, 'Balls to the Wall', 2),
 (3, 'Restless and Wild', 2),
 (4, 'Let There Be Rock', 1),
 (5, 'Big Ones', 3),
 (6, 'Jagged Little Pill', 4)]
```

ESERCIZIO: In SQLStudio, creare una query che seleziona gli album con id compreso tra 3 e 5 inclusi:

1. apri il query editor con `Alt+E`
2. immetti la query
3. eseguila premendo F9

ESERCIZIO: chiama `esegui` per eseguire la stessa query, usando i parametri

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[12]:
```
# scrivi qui il comando

esegui(conn,
"""
SELECT * FROM Album
WHERE AlbumId >= ? AND AlbumId <= ?
""", (3, 5))
```

[12]:
```
[(3, 'Restless and Wild', 2), (4, 'Let There Be Rock', 1), (5, 'Big Ones', 3)]
```

[12]:
```
# scrivi qui il comando
```

```
[12]: [(3, 'Restless and Wild', 2), (4, 'Let There Be Rock', 1), (5, 'Big Ones', 3)]
```

Struttura della tabella

ESERCIZIO: Guarda meglio la tab `Structure` di `Album`:

DDL

Confronta quanto sopra con la tab `DDL` (Data Definition Language), che contiene le istruzioni SQL per creare la tabella nel database:

Una caratteristica dei database è la possibilità di dichiarare vincoli sui dati inseriti. Per esempio, qua notiamo che

- la tabella ha una cosiddetta *chiave primaria* (`PRIMARY KEY`): in essa asserisce che non possono esistere due righe con lo stesso `AlbumId`
- la tabella definisce la colonna `ArtistId` come *chiave esterna* (`FOREIGN KEY`), asserendo che ai valori in quella colonna deve sempre corrispondere un id esistente nella colonna `ArtistId` **della tabella** `Artist`. Quindi non

ci si potrà mai riferire ad un artista non esistente

ESERCIZIO: Vai alla tab `Data` e prova a cambiare un `ArtistId` mettendo un numero inesistente (tipo 1000). Apparentemente il database non si lamenterà, ma solo perchè al momento non abbiamo ancora registrato il cambiamento su disco, cioè non abbiamo operato una operazione di *commit*. I commit ci permettono di eseguire più operazioni in modo atomico, nel senso che o tutti i cambiamenti fatti vengono registrati con successo, o non viene fatta nessuna modifica. Prova ad eseguire un commit premendo il bottoncino verde con la spunta (o premere Ctrl-Return). Che succede? Per riparare al danno fatto, esegui *rollback* col bottoncino rosso con la x (o premi Ctrl-Backspace).

Query ai metadati

Una cosa interessante e a volte utile di molti database SQL è che spesso i metadati sul tipo di tabelle nel database sono salvate essi stessi come tabelle, quindi è possibile eseguire query SQL su questi metadati. Per esempio, con SQLite si può eseguire una query del genere. Non la spieghiamo nel dettaglio, limitandoci a mostrare qualche esempio di utilizzo:

[13]:
```
def query_schema(conn, tabella):
    """ Ritorna una stringa con le istruzioni SQL per creare la tabella (senza i␣
↪dati)  """
    return esegui(conn, """
    SELECT sql FROM sqlite_master
    WHERE name = ?
    """, (tabella,))[0][0]
```

[14]:
```
import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)


print(query_schema(conn, 'Album'))
```

```
CREATE TABLE [Album]
(
    [AlbumId] INTEGER  NOT NULL,
    [Title] NVARCHAR(160)  NOT NULL,
    [ArtistId] INTEGER  NOT NULL,
    CONSTRAINT [PK_Album] PRIMARY KEY  ([AlbumId]),
    FOREIGN KEY ([ArtistId]) REFERENCES [Artist] ([ArtistId])
                ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

6.4.9 JOIN

Nella tabella `Album` per gli artisti vediamo solo dei numeri. Come possiamo fare una query per vedere anche i nomi degli artisti? Useremo il comando SQL `JOIN` :

ESERCIZIO: Per capire cosa succede, esegui le query in SQLStudio

[15]:
```
esegui(conn, """
SELECT *
FROM Album JOIN Artist
WHERE Album.ArtistId = Artist.ArtistId
LIMIT 5
""")
```

[15]:
```
[(1, 'For Those About To Rock We Salute You', 1, 1, 'AC/DC'),
 (2, 'Balls to the Wall', 2, 2, 'Accept'),
```

(continues on next page)

(continued from previous page)

```
 (3, 'Restless and Wild', 2, 2, 'Accept'),
 (4, 'Let There Be Rock', 1, 1, 'AC/DC'),
 (5, 'Big Ones', 3, 3, 'Aerosmith')]
```

Invece del JOIN possiamo usare una virgola:

```
[16]: esegui(conn, """
      SELECT * FROM Album, Artist
      WHERE Album.ArtistId = Artist.ArtistId
      LIMIT 5
      """)
```

```
[16]: [(1, 'For Those About To Rock We Salute You', 1, 1, 'AC/DC'),
       (2, 'Balls to the Wall', 2, 2, 'Accept'),
       (3, 'Restless and Wild', 2, 2, 'Accept'),
       (4, 'Let There Be Rock', 1, 1, 'AC/DC'),
       (5, 'Big Ones', 3, 3, 'Aerosmith')]
```

Meglio ancora, in questo caso visto che abbiamo lo stesso nome di colonna in entrambe le tabelle, possiamo usare la clausola `USING` che ha anche il beneficio di eliminare la colonna duplicata

NOTA: Per ragioni oscure, in SQLiteStudio la colonna `ArtistId` appare comunque duplicata con nome `ArtistiId:1`

```
[17]: esegui(conn, """
      SELECT *
      FROM Album, Artist USING(ArtistId)
      LIMIT 5
      """)
```

```
[17]: [(1, 'For Those About To Rock We Salute You', 1, 'AC/DC'),
       (2, 'Balls to the Wall', 2, 'Accept'),
       (3, 'Restless and Wild', 2, 'Accept'),
       (4, 'Let There Be Rock', 1, 'AC/DC'),
       (5, 'Big Ones', 3, 'Aerosmith')]
```

Infine possiamo filtrare solo le colonne che ci interessano, `Title` di album e `Name` di `Artisti`. Per chiarezza, possiamo identificare le tabelle con variabili che assegnamo in FROM (qua usiamo i nomi `ALB` e `ART` ma potrebbero essere qualsiasi):

```
[18]: esegui(conn, """
      SELECT ALB.Title, ART.Name
      FROM Album ALB, Artist ART USING(ArtistId)
      LIMIT 5
      """)
```

```
[18]: [('For Those About To Rock We Salute You', 'AC/DC'),
       ('Balls to the Wall', 'Accept'),
       ('Restless and Wild', 'Accept'),
       ('Let There Be Rock', 'AC/DC'),
       ('Big Ones', 'Aerosmith')]
```

6.4.10 Tabella Track

Passiamo adesso ad una tabella più complessa, come `Track`, che contiene un po' di canzoni ascoltate dagli utenti di iTunes:

[19]:
```
esegui(conn, "SELECT * FROM Track LIMIT 5")
```

[19]:
```
[(1,
  'For Those About To Rock (We Salute You)',
  1,
  1,
  1,
  'Angus Young, Malcolm Young, Brian Johnson',
  343719,
  11170334,
  0.99),
 (2, 'Balls to the Wall', 2, 2, 1, None, 342562, 5510424, 0.99),
 (3,
  'Fast As a Shark',
  3,
  2,
  1,
  'F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman',
  230619,
  3990994,
  0.99),
 (4,
  'Restless and Wild',
  3,
  2,
  1,
  'F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman',
  252051,
  4331779,
  0.99),
 (5,
  'Princess of the Dawn',
  3,
  2,
  1,
  'Deaffy & R.A. Smith-Diesel',
  375418,
  6290521,
  0.99)]
```

[20]:
```
query_schema(conn, "Track")
```

[20]:
```
'CREATE TABLE [Track]\n(\n    [TrackId] INTEGER  NOT NULL,\n    [Name] NVARCHAR(200) ␣
↪NOT NULL,\n    [AlbumId] INTEGER,\n    [MediaTypeId] INTEGER  NOT NULL,\n    ␣
↪[GenreId] INTEGER,\n    [Composer] NVARCHAR(220),\n    [Milliseconds] INTEGER  NOT␣
↪NULL,\n    [Bytes] INTEGER,\n    [UnitPrice] NUMERIC(10,2)  NOT NULL,\n    ␣
↪CONSTRAINT [PK_Track] PRIMARY KEY  ([TrackId]),\n    FOREIGN KEY ([AlbumId])␣
↪REFERENCES [Album] ([AlbumId]) \n\t\tON DELETE NO ACTION ON UPDATE NO ACTION,\n    ␣
↪FOREIGN KEY ([GenreId]) REFERENCES [Genre] ([GenreId]) \n\t\tON DELETE NO ACTION ON␣
↪UPDATE NO ACTION,\n    FOREIGN KEY ([MediaTypeId]) REFERENCES [MediaType]␣
↪([MediaTypeId]) \n\t\tON DELETE NO ACTION ON UPDATE NO ACTION\n)'
```

[21]:
```
print(query_schema(conn, "Track"))
```

```
CREATE TABLE [Track]
(
    [TrackId] INTEGER  NOT NULL,
    [Name] NVARCHAR(200)  NOT NULL,
    [AlbumId] INTEGER,
    [MediaTypeId] INTEGER  NOT NULL,
    [GenreId] INTEGER,
    [Composer] NVARCHAR(220),
    [Milliseconds] INTEGER  NOT NULL,
    [Bytes] INTEGER,
    [UnitPrice] NUMERIC(10,2)  NOT NULL,
    CONSTRAINT [PK_Track] PRIMARY KEY  ([TrackId]),
    FOREIGN KEY ([AlbumId]) REFERENCES [Album] ([AlbumId])
                ON DELETE NO ACTION ON UPDATE NO ACTION,
    FOREIGN KEY ([GenreId]) REFERENCES [Genre] ([GenreId])
                ON DELETE NO ACTION ON UPDATE NO ACTION,
    FOREIGN KEY ([MediaTypeId]) REFERENCES [MediaType] ([MediaTypeId])
                ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

[22]:
```
esegui(conn, """
SELECT Name, Composer
FROM Track
LIMIT 5
""")
```

[22]:
```
[('For Those About To Rock (We Salute You)',
  'Angus Young, Malcolm Young, Brian Johnson'),
 ('Balls to the Wall', None),
 ('Fast As a Shark', 'F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman'),
 ('Restless and Wild',
  'F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman'),
 ('Princess of the Dawn', 'Deaffy & R.A. Smith-Diesel')]
```

[23]:
```
esegui(conn, """
SELECT Name, Composer
FROM Track
LIMIT 5
""")[0]
```

[23]:
```
('For Those About To Rock (We Salute You)',
 'Angus Young, Malcolm Young, Brian Johnson')
```

Per la seconda riga:

[24]:
```
esegui(conn, """
SELECT Name, Composer
FROM Track
LIMIT 5
""")[1]
```

[24]:
```
('Balls to the Wall', None)
```

Notiamo che in questo caso manca il compositore. Ma sorge una domanda: nella tabella originale SQL, come era segnato il fatto che il compositore mancasse?

ESERCIZIO: Usando SQLiteStudio, nel menu a sinistra fai doppio click sulla tabella `Track` e poi seleziona la tab `Data` a destra. Scorri le righe finchè non trovi la casella per la colonna `Composer`.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

Notiamo che in SQL le caselle vuote si indicano con `NULL`. Dato che `NULL` non è un tipo Python, durante la conversione da oggetti SQL a oggetti Python `NULL` viene convertito nell'oggetto Python `None`.

Proviamo a selezionare dei valori numerici alla nostra query, come per esempio i `Milliseconds`

[25]:
```
esegui(conn, """
SELECT Name, Milliseconds
FROM Track
LIMIT 5
""")
```

[25]:
```
[('For Those About To Rock (We Salute You)', 343719),
 ('Balls to the Wall', 342562),
 ('Fast As a Shark', 230619),
 ('Restless and Wild', 252051),
 ('Princess of the Dawn', 375418)]
```

[26]:
```
esegui(conn, """
SELECT Name, Milliseconds
FROM Track
LIMIT 5
""")[0]
```

[26]:
```
('For Those About To Rock (We Salute You)', 343719)
```

[27]:
```
esegui(conn, """
SELECT Name, Milliseconds
FROM Track
LIMIT 5
""")[0][0]
```

[27]:
```
'For Those About To Rock (We Salute You)'
```

[28]:
```
esegui(conn, """
SELECT Name, Milliseconds
FROM Track
LIMIT 5
""")[0][1]
```

[28]:
```
343719
```

[29]:
```
esegui(conn, """
SELECT Name, Milliseconds
FROM Track
ORDER BY Milliseconds DESC
LIMIT 5
""")
```

[29]:
```
[('Occupation / Precipice', 5286953),
 ('Through a Looking Glass', 5088838),
 ('Greetings from Earth, Pt. 1', 2960293),
 ('The Man With Nine Lives', 2956998),
 ('Battlestar Galactica, Pt. 2', 2956081)]
```

ESERCIZIO: Prova ad usare `ASC` invece di `DESC`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[30]:
```
# scrivi qui la query

esegui(conn, """
SELECT Name, Composer, Milliseconds
FROM Track
ORDER BY Milliseconds ASC
LIMIT 5
""")
```

[30]:
```
[('É Uma Partida De Futebol', 'Samuel Rosa', 1071),
 ('Now Sports', None, 4884),
 ('A Statistic', None, 6373),
 ('Oprah', None, 6635),
 ('Commercial 1', 'L. Muggerud', 7941)]
```

[30]:
```
# scrivi qui la query
```

[30]:
```
[('É Uma Partida De Futebol', 'Samuel Rosa', 1071),
 ('Now Sports', None, 4884),
 ('A Statistic', None, 6373),
 ('Oprah', None, 6635),
 ('Commercial 1', 'L. Muggerud', 7941)]
```

6.4.11 GROUP BY and COUNT

COUNT

Per contare quante righe ci sono in una tabella, possiamo usare la parola chiave `COUNT(*)` nella `SELECT`. Per esempio, per vedere quante tracce ci sono, possiamo fare così:

[31]:
```
esegui(conn, """
SELECT COUNT(*)
FROM Track
""")
```

[31]:
```
[(3503,)]
```

DOMANDA: è molto meglio fare così, invece che prelevare tutte le righe in Python e contare con `len`. Perchè?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra risposta" data-jupman-hide="Nascondi">Mostra risposta</a>

RISPOSTA:

Perchè facendo i conteggi direttamente in SQL, il database si arrangerà a fare i conti e spedirà a Python solo un numero invece che una quantità di righe che potenzialmente potrebbe essere molto ingente e intasare la memoria.

GROUP BY E COUNT

Ogni Track ha associato un `MediaTypeId`. Potremmo chiederci per ogni media type quante track ci sono.

- Per conteggiare, dovremo usare la parola chiave `COUNT(*) AS Conteggio` nella `SELECT`
- per raggruppare `GROUP BY` dopo la linea del `FROM`
- Per ordinare i conteggi in modo decrescente, useremo anche `ORDER BY Conteggio DESC`

Nota: il `COUNT(*)` conteggierà in questo caso quanti elementi ci sono nei gruppi, non in tutta la tabella:

[32]:
```
esegui(conn, """
SELECT T.MediaTypeId, COUNT(*) AS Conteggio
FROM Track T
GROUP BY T.MediaTypeId
ORDER BY Conteggio DESC
""")
```

[32]:
```
[(1, 3034), (2, 237), (3, 214), (5, 11), (4, 7)]
```

ESERCIZIO: I `MediaTypeId` non sono molto descrittivi. Scrivi qua sotto una query per ottenere coppie con nome del MediaType e rispettivo conteggio. Prova anche ad eseguire la query in SQLStudio:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[33]:
```
# scrivi qui


esegui(conn, """
SELECT MT.Name, COUNT(*) AS Conteggio
FROM Track T, MediaType MT USING (MediaTypeId)
GROUP BY MT.MediaTypeId
ORDER BY Conteggio DESC
""")
```

[33]:
```
[('MPEG audio file', 3034),
 ('Protected AAC audio file', 237),
 ('Protected MPEG-4 video file', 214),
 ('AAC audio file', 11),
 ('Purchased AAC audio file', 7)]
```

[33]:
```
# scrivi qui
```

[33]:
```
[('MPEG audio file', 3034),
 ('Protected AAC audio file', 237),
 ('Protected MPEG-4 video file', 214),
 ('AAC audio file', 11),
 ('Purchased AAC audio file', 7)]
```

ESERCIZIO: Scrivi qua sotto una query per creare una tabella di due colonne, nella prima ci saranno i nomi dei generi musicali e nella seconda quante track di quel genere ci sono nel database.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[34]:
```
# scrivi qui


esegui(conn, """
SELECT G.Name, COUNT(*) AS Conteggio
FROM Track T, Genre G USING (GenreId)
GROUP BY G.GenreId
ORDER BY Conteggio DESC
""")
```

[34]:
```
[('Rock', 1297),
 ('Latin', 579),
 ('Metal', 374),
 ('Alternative & Punk', 332),
 ('Jazz', 130),
 ('TV Shows', 93),
 ('Blues', 81),
 ('Classical', 74),
 ('Drama', 64),
 ('R&B/Soul', 61),
 ('Reggae', 58),
 ('Pop', 48),
 ('Soundtrack', 43),
 ('Alternative', 40),
 ('Hip Hop/Rap', 35),
 ('Electronica/Dance', 30),
 ('Heavy Metal', 28),
 ('World', 28),
 ('Sci Fi & Fantasy', 26),
 ('Easy Listening', 24),
 ('Comedy', 17),
 ('Bossa Nova', 15),
 ('Science Fiction', 13),
 ('Rock And Roll', 12),
 ('Opera', 1)]
```

[34]:
```
# scrivi qui
```

[34]:
```
[('Rock', 1297),
 ('Latin', 579),
 ('Metal', 374),
 ('Alternative & Punk', 332),
 ('Jazz', 130),
 ('TV Shows', 93),
 ('Blues', 81),
 ('Classical', 74),
```

(continues on next page)

(continued from previous page)

```
 ('Drama', 64),
 ('R&B/Soul', 61),
 ('Reggae', 58),
 ('Pop', 48),
 ('Soundtrack', 43),
 ('Alternative', 40),
 ('Hip Hop/Rap', 35),
 ('Electronica/Dance', 30),
 ('Heavy Metal', 28),
 ('World', 28),
 ('Sci Fi & Fantasy', 26),
 ('Easy Listening', 24),
 ('Comedy', 17),
 ('Bossa Nova', 15),
 ('Science Fiction', 13),
 ('Rock And Roll', 12),
 ('Opera', 1)]
```

ESERCIZIO: Ora prova a trovare per ogni genere la lunghezza media in millisecondi usando invece di `COUNT(*)` la funzione `AVG(Track.Milliseconds)`:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Mostra soluzione" data-jupman-hide="Nascondi">Mostra soluzione</a>

[35]:
```
# scrivi qui:


esegui(conn, """
SELECT G.Name, AVG(T.Milliseconds) AS Lunghezza
FROM Track T, Genre G USING (GenreId)
GROUP BY G.GenreId
ORDER BY Lunghezza DESC
""")
```

[35]:
```
[('Sci Fi & Fantasy', 2911783.0384615385),
 ('Science Fiction', 2625549.076923077),
 ('Drama', 2575283.78125),
 ('TV Shows', 2145041.0215053763),
 ('Comedy', 1585263.705882353),
 ('Metal', 309749.4438502674),
 ('Electronica/Dance', 302985.8),
 ('Heavy Metal', 297452.9285714286),
 ('Classical', 293867.5675675676),
 ('Jazz', 291755.3769230769),
 ('Rock', 283910.0431765613),
 ('Blues', 270359.77777777775),
 ('Alternative', 264058.525),
 ('Reggae', 247177.75862068965),
 ('Soundtrack', 244370.88372093023),
 ('Alternative & Punk', 234353.84939759035),
 ('Latin', 232859.26252158894),
 ('Pop', 229034.10416666666),
 ('World', 224923.82142857142),
 ('R&B/Soul', 220066.8524590164),
 ('Bossa Nova', 219590.0),
```

(continues on next page)

(continued from previous page)

```
 ('Easy Listening', 189164.20833333334),
 ('Hip Hop/Rap', 178176.2857142857),
 ('Opera', 174813.0),
 ('Rock And Roll', 134643.5)]
```

[35]:
```
# scrivi qui:
```

[35]:
```
[('Sci Fi & Fantasy', 2911783.0384615385),
 ('Science Fiction', 2625549.076923077),
 ('Drama', 2575283.78125),
 ('TV Shows', 2145041.0215053763),
 ('Comedy', 1585263.705882353),
 ('Metal', 309749.4438502674),
 ('Electronica/Dance', 302985.8),
 ('Heavy Metal', 297452.9285714286),
 ('Classical', 293867.5675675676),
 ('Jazz', 291755.3769230769),
 ('Rock', 283910.0431765613),
 ('Blues', 270359.77777777775),
 ('Alternative', 264058.525),
 ('Reggae', 247177.75862068965),
 ('Soundtrack', 244370.88372093023),
 ('Alternative & Punk', 234353.84939759035),
 ('Latin', 232859.26252158894),
 ('Pop', 229034.10416666666),
 ('World', 224923.82142857142),
 ('R&B/Soul', 220066.8524590164),
 ('Bossa Nova', 219590.0),
 ('Easy Listening', 189164.20833333334),
 ('Hip Hop/Rap', 178176.2857142857),
 ('Opera', 174813.0),
 ('Rock And Roll', 134643.5)]
```

6.4.12 Pandas

Finora abbiamo usato metodi base di Python, ma ovviamente processare il tutto in pandas è più comodo.

Per maggiori informazioni su Pandas, guarda il relativo tutorial[432]

[36]:
```
import pandas

df = pandas.read_sql_query("SELECT Name, Composer, Milliseconds from Track", conn)
```

[37]:
```
df
```

[37]:
```
                                                Name  \
0                For Those About To Rock (We Salute You)
1                                       Balls to the Wall
2                                         Fast As a Shark
3                                       Restless and Wild
```

(continues on next page)

[432] https://it.softpython.org/pandas/pandas-sol.html

(continued from previous page)

```
4                                  Princess of the Dawn
5                                 Put The Finger On You
6                                       Let's Get It Up
7                                      Inject The Venom
8                                            Snowballed
9                                            Evil Walks
10                                              C.O.D.
11                                   Breaking The Rules
12                             Night Of The Long Knives
13                                           Spellbound
14                                              Go Down
15                                          Dog Eat Dog
16                                    Let There Be Rock
17                                       Bad Boy Boogie
18                                        Problem Child
19                                             Overdose
20                          Hell Ain't A Bad Place To Be
21                                    Whole Lotta Rosie
22                                        Walk On Water
23                                  Love In An Elevator
24                                             Rag Doll
25                                        What It Takes
26                             Dude (Looks Like A Lady)
27                                    Janie's Got A Gun
28                                              Cryin'
29                                             Amazing
…                                                  …
3473                                        October Song
3474                                  What Is It About Men
3475                                        Help Yourself
3476                                   Amy Amy Amy (Outro)
3477                                            Slowness
3478                           Prometheus Overture, Op. 43
3479                   Sonata for Solo Violin: IV: Presto
3480  A Midsummer Night's Dream, Op.61 Incidental Mu…
3481    Suite No. 3 in D, BWV 1068: III. Gavotte I & II
3482  Concert pour 4 Parties de V**les, H. 545: I. P…
3483                                        Adios nonino
3484  Symphony No. 3 Op. 36 for Orchestra and Sopran…
3485                                      Act IV, Symphony
3486  3 Gymnopédies: No.1 - Lent Et Grave, No.3 - Le…
3487  Music for the Funeral of Queen Mary: VI. "Thou…
3488                 Symphony No. 2: III. Allegro vivace
3489           Partita in E Major, BWV 1006A: I. Prelude
3490          Le Sacre Du Printemps: I.iv. Spring Rounds
3491                                        Sing Joyfully
3492                             Metopes, Op. 29: Calypso
3493  Symphony No. 2, Op. 16 -  "The Four Temperamen…
3494  24 Caprices, Op. 1, No. 24, for Solo Violin, i…
3495    Étude 1, In C Major - Preludio (Presto) - Liszt
3496                                       Erlkonig, D.328
3497  Concerto for Violin, Strings and Continuo in G…
3498  Pini Di Roma (Pinien Von Rom) \ I Pini Della V…
3499  String Quartet No. 12 in C Minor, D. 703 "Quar…
3500                L'orfeo, Act 3, Sinfonia (Orchestra)
3501  Quintet for Horn, Violin, 2 Violas, and Cello …
3502                                        Koyaanisqatsi
```

(continues on next page)

(continued from previous page)

```
                                            Composer  Milliseconds
0            Angus Young, Malcolm Young, Brian Johnson        343719
1                                                 None        342562
2     F. Baltes, S. Kaufman, U. Dirkscneider & W. Ho…        230619
3     F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D…        252051
4                           Deaffy & R.A. Smith-Diesel        375418
5            Angus Young, Malcolm Young, Brian Johnson        205662
6            Angus Young, Malcolm Young, Brian Johnson        233926
7            Angus Young, Malcolm Young, Brian Johnson        210834
8            Angus Young, Malcolm Young, Brian Johnson        203102
9            Angus Young, Malcolm Young, Brian Johnson        263497
10           Angus Young, Malcolm Young, Brian Johnson        199836
11           Angus Young, Malcolm Young, Brian Johnson        263288
12           Angus Young, Malcolm Young, Brian Johnson        205688
13           Angus Young, Malcolm Young, Brian Johnson        270863
14                                               AC/DC        331180
15                                               AC/DC        215196
16                                               AC/DC        366654
17                                               AC/DC        267728
18                                               AC/DC        325041
19                                               AC/DC        369319
20                                               AC/DC        254380
21                                               AC/DC        323761
22     Steven Tyler, Joe Perry, Jack Blades, Tommy Shaw        295680
23                             Steven Tyler, Joe Perry        321828
24    Steven Tyler, Joe Perry, Jim Vallance, Holly K…        264698
25               Steven Tyler, Joe Perry, Desmond Child        310622
26               Steven Tyler, Joe Perry, Desmond Child        264855
27                           Steven Tyler, Tom Hamilton        330736
28               Steven Tyler, Joe Perry, Taylor Rhodes        309263
29                            Steven Tyler, Richie Supa        356519
…                                                   …           …
3473                         Matt Rowe & Stefan Skarbek        204846
3474  Delroy "Chris" Cooper, Donovan Jackson, Earl C…        209573
3475         Freddy James, Jimmy hogarth & Larry Stock        300884
3476  Astor Campbell, Delroy "Chris" Cooper, Donovan…        663426
3477                                              None        215386
3478                              Ludwig van Beethoven        339567
3479                                       Béla Bartók        299350
3480                                              None        387826
3481                            Johann Sebastian Bach        225933
3482                          Marc-Antoine Charpentier        110266
3483                                   Astor Piazzolla        289388
3484                                    Henryk Górecki        567494
3485                                     Henry Purcell        364296
3486                                        Erik Satie        385506
3487                                     Henry Purcell        142081
3488                                        Kurt Weill        376510
3489                            Johann Sebastian Bach        285673
3490                                  Igor Stravinsky        234746
3491                                     William Byrd        133768
3492                                Karol Szymanowski        333669
3493                                     Carl Nielsen        286998
3494                                 Niccolò Paganini        265541
3495                                              None         51780
3496                                              None        261849
```

(continues on next page)

(continued from previous page)

```
3497                          Pietro Antonio Locatelli        493573
3498                                              None        286741
3499                                     Franz Schubert        139200
3500                                  Claudio Monteverdi         66639
3501                            Wolfgang Amadeus Mozart        221331
3502                                       Philip Glass        206005

[3503 rows x 3 columns]
```

ATTENZIONE a quanti dati carichi !

Pandas è molto comodo, ma come già detto nella nel relativo tutorial[433] Pandas carica tutto in memoria RAM che tipicamente sono dai 4 ai 16 giga. Se hai grandi database potresti avere dei problemi per cui valgono i metodi e considerazioni fatte nella sezione *Performance*

ESERCIZIO: Millisecondi e bytes occupati dovrebbero ragionevolmente essere linearmente dipendenti. Dimostralo con Pandas.

[38]:
```
# scrivi qui
```

[ ]:
```
```

6.5 Web development

6.5.1 Scarica zip esercizi

Naviga file online[434]

6.5.2 Introduzione

In questi esempi cercheremo di costruirci un piccolo server web, con librerie standard di Python.

ATTENZIONE: Tutorial sperimentale e incompleto!

Questo tutorial crea un sito web DA ZERO: ci sono modi più semplici di realizzare una webapp direttamente con Jupyter, per farlo segui prima il tutorial Interfacce utente[435]

[433] http://it.softpython.org/pandas/pandas-sol.html
[434] https://github.com/DavidLeoni/softpython-it/tree/master/web
[435] https://it.softpython.org/gui/gui-sol.html

Che fare

- scompatta lo zip in una cartella, dovresti ottenere qualcosa del genere:

```
web
    _static
        esempio.txt
    web.ipynb
    web-sol.ipynb
    jupman.py
```

ATTENZIONE: Per essere visualizzato correttamente, il file del notebook DEVE essere nella cartella szippata.

- apri il Jupyter Notebook da quella cartella. Due cose dovrebbero aprirsi, prima una console e poi un browser. Il browser dovrebbe mostrare una lista di file: naviga la lista e apri il notebook `web.ipynb`
- Prosegui leggendo il file degli esercizi, ogni tanto al suo interno troverai delle scritte **ESERCIZIO**, che ti chiederanno di scrivere dei comandi Python nelle celle successive.

Scorciatoie da tastiera:

- Per eseguire il codice Python dentro una cella di Jupyter, premi `Control+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E selezionare la cella seguente, premi `Shift+Invio`
- Per eseguire il codice Python dentro una cella di Jupyter E creare una nuova cella subito dopo, premi `Alt+Invio`
- Se per caso il Notebook sembra inchiodato, prova a selezionare `Kernel -> Restart`

6.5.3 Struttura del sito

Si suppone che tutti i file html, css, js etc vadano nella directory `_static`

In `web_server.py` trovi il codice del server.

6.5.4 Facciamo partire il server

Apri Anaconda Prompt: per istruzioni su come trovarlo (o se non hai idea di cosa sia!), prima di proseguire leggi sezione interprete Python nell'introduzione[436]

Una volta nella console del Prompt dei comandi (che è una finestra nera dove puoi immettere comandi testuali per il sistema operativo), raggiungi la cartella `web`

(per vedere in che cartella sei, scrivi `dir`, per entrare in una cartella che si chiama CARTELLA, scrivi `cd CARTELLA`)

Una volta nella cartella `web`, scrivi

```
python web_server.py
```

(Se non parte metti `3` dopo `python` prova `python3 web_server.py`

Dovrebbero apparire le scritte

```
starting server...
running server...
```

[436] https://it.softpython.org/intro/intro-sol.html#L'interprete-Python

Per spegnere il server, premi `Control-C`. Lo spegne in malo modo ma va bene lo stesso:

```
CTraceback (most recent call last):
  File "web_server.py", line 99, in <module>
    run()
  File "web_server.py", line 97, in run
    httpd.serve_forever()
  File "/usr/lib/python3.5/socketserver.py", line 232, in serve_forever
    ready = selector.select(poll_interval)
  File "/usr/lib/python3.5/selectors.py", line 376, in select
    fd_event_list = self._poll.poll(timeout)
KeyboardInterrupt
```

6.5.5 Proviamo il server

Contenuti dinamici

Assicurati che il server sia in funzione. Una volta che è partito, nel browser prova ad andare all'indirizzo

`http://localhost:8081/din/saluta`

Nel browser dovrebbe apparire la scritta `Hello world`

Questo è un esempio di contenuto generato *dinamicamente*. In questo esempio, accederemo a tutti i contenuti dinamici con indirizzi che contengono `/dir/`

Contenuti statici

Proviamo ad accedere al file `_static/esempio.txt`

Nel browser, vai all'indirizzo http://localhost:8081/esempio.txt

Il server dovrebbe cercare il file `_static/esempio.txt`, leggerlo e mandarlo al browser. Nel browser dovrebbe apparire il contenuto del file d'esempio, cioè:

```
Abracadabra
```

Struttura del codice

Nel file `web_server.py`, ci sono due funzioni, `get_dinamico(self)` e `get_statico(self)`. Probabilmente dovrai lavorare solo su `get_dinamico(self)`

6.5.6 Come parsare il path per estrarre parametri

Abbiamo visto come generare contenuti dinamici in base al path passato nel browser. Però se vogliamo implementare una ricerca, come `/din/ricerca` dovremo passare dei parametri come il testo e eventualmente il valore di altri filtri. Come facciamo? Lascio qua sotto un'idea di come fare il cosiddetto *parsing dei parametri*.

```
# se self.path =  '/saluta?param1=hello&param2=mondo'

# questo codice va all'inizio della do_GET

from urllib.parse import urlparse
```

(continues on next page)

(continued from previous page)

```
query = urlparse(self.path).query
query_components = dict(qc.split("=") for qc in query.split("&"))
# param1 = query_components["param1"]
# query_components = { "param1" : "Hello",
                       "param2" : "mondo" }
# print(param1)  # Hello
```

[2]:
```
# scrivi qui
```

[ ]:
```

```

CHAPTER

SEVEN

D - PROGETTI

7.1 Come fare un progetto

7.1.1 Introduzione

Questa guida spiega come creare un progetto completo in Jupyter, in particolare

- come scrivere un report col formato Markdown
- come assicurarsi che il progetto prodotto sia eseguibile su computer di altri persone
- ambienti virtuali
- installazione librerie

7.1.2 Challenge

Questo sito prevede delle challenge. Per affrontarle al meglio, dovrete aver dimostrato di aver fatto almeno alcune tra le seguenti attività:

- Analisi
- Pulizia
- Integrazione
- Arricchimento
  - georeferenziazione
  - semantico
- Ricerca
  - Base
  - Avanzata
  - Calcolo similarità
  - Ranking
- Predizione
- Presentazione
  - demo per pubblico non tecnico

VEDI DESCRIZIONE COMPLETA ATTIVITA' NEL TEMPLATE DEL PROGETTO

7.1.3 Che fare

1 - Scarica lo zip con il template (naviga files online[437])

Una volta szippato il template, troverai una cartella chiamata `NAME-SURNAME-ID`, dentro contiene questi file:

```
NAME-SURNAME-ID
    img
        example.png
    project.ipynb
    markdown.ipynb
    demo.ipynb
    requirements.txt
```

2 - Rinomina la cartella con i tuoi dati

3 - lancia Jupyter dalla cartella appena rinominata

4 - edita il file `project.ipynb` , **seguendo attentamente le indicazioni** *nei requisiti tecnici che seguono*

7.1.4 Requisiti tecnici

Dimensione dati

Per questioni di praticità, ti conviene usare file di dati con dimensione fino a 50 mb. Se sono più grandi, tagliali.

Testo Markdown

Ti conviene scrivere report in Jupyter stesso, usando la sintassi Markdown descritta nel notebook *markdown.ipynb* che trovi nello zip del progetto. Quindi tipicamente non serve scrivere altri documenti Word o Latex in parallelo ! L'unica eccezione sono i fogli di calcolo, vedere *sezione tabelle in Markdown*

Codice Python

NOTA: non spaventarti se i requisiti sembrano troppo formali: li ho scritti per limitare i problemi tecnici, se poi insorgono comunque pazienza! L'idea è anche farvi capire cosa succede quando si riesegue codice su computer diversi dal proprio.

Tutto il codice nel notebook del progetto dovrebbe poter essere rieseguibile senza errori su computer altrui. A tal fine:

- Il progetto deve essere fatto in Python **3**
- la versione esatta di tutte le librerie utilizzate deve essere scritta nel file `requirements.txt` (vedere sotto)

[437] https://github.com/DavidLeoni/softpython-it/tree/master/project

File `requirements.txt`

Tipicamente ogni progetto necessita di alcune librerie per funzionare. Invece di installarle una ad una con `pip`, si può comodamente elencarle in un file chiamato `requirements.txt`, e poi dire a `pip` di installarle tutte in un colpo solo.

Per capire come compilare il file , guardate il `requirements.txt` di esempio fornito nello zip, in cui sono indicate tutte le librerie usate per i vari tutorial finora, assieme alla versione utilizzata. Se avete bisogno di usare altre librerie e / o cambiare le versioni, editate pure il file a piacere.

Versioni dei pacchetti

Per capire che versione avete installata di un dato pacchetto (i.e. `pandas`), eseguite questo comando:

```
python3 -m pip show pandas
```

Se avete installato il pacchetto, dovreste ottenere un output simile a questo:

```
Name: pandas
Version: 0.23.4
Summary: Powerful data structures for data analysis, time series, and statistics
Home-page: http://pandas.pydata.org
Author: None
Author-email: None
License: BSD
Location: /home/da/.local/lib/python3.5/site-packages
Requires: pytz, python-dateutil, numpy
Required-by: traittypes, pollster, geopandas, fuzzymatcher, bqplot
```

Chi volesse usare il progetto ed essere sicuro che il tutto funzioni, dovrebbe:

- creare sul suo computer un ambiente virtuale nella cartella del progetto e attivarlo. Gli ambienti virtuali servono a isolare le librerie usate da un progetto dal resto del sistema - ai fini del corso non è necessario che tu li crei, ma se vuoi usare python in vari progetti consiglio caldamente di leggere *Ambienti virtuali*
- installere tutte le librerie specificate in `requirements.txt` con il comando :

  ```
  python3 -m pip install -r requirements.txt
  ```

- aprire il file `project.ipynb` in Jupyter e tenterà di rieseguire il codice con il comando `Kernel->Restart & run All`.

Se tutto è predisposto correttamente, in teoria tutte le celle nel notebook si dovrebbero eseguire senza problemi di sorta.

Interfacce grafiche

Se volete implementare dei widget grafici in Jupyter, tenete conto che sviluppare interfacce grafiche per il web è un mestiere tedioso - ottenere i risultati desiderati è spesso un lungo processo costituito da innumerevoli prove e altrettanti fallimenti. Inoltre, i widget di Jupyter al 2018 sono ancora da considerarsi software sperimentale, che spesso non è documentato a sufficienza. Vi conviene non puntare ad avere componenti perfettamente allineati e immagini ottimamente dimensionate, piuttosto programmate in modo corretto le reazioni dei vari componenti.

Fare attenzione a

- mischiare unità di misura diverse
- celle vuote
- sostituzioni regex sbagliate
- usare scale giuste nel grafico (magari serve la logaritmica ?)
- problemi di encoding
  - se salvate file in windows e chi utilizza il progetto ha Linux / Mac, ci potrebbero essere sorprese ! Accertatevi di avere usato encoding uniforme ovunque, sia in lettura che in scrittura.
- nomi di file
  - Windows non distingue tra nomi di file con caratteri maiuscoli e minuscoli
  - esempio: se referenziate in Jupyter un file come `ciao.jpg` e il file su disco si chiama `Ciao.JPG`, il file verrà letto in Windows, ma non in Linux/Mac. Per evitare problemi, mettete sempre i nomi dei file in minuscolo, estensione inclusa come `ciao.jpg`
  - evitate nomi di file con spazi, usate invece il trattino –
  - Attenti a usare link con percorsi assoluti a file fuori dalla cartella che mi consegnerete (per es. `C:\Users\Paolo\excel.xls`)

VIETATO IL DISORDINE!

DETESTO quando mi si consegnano cartelle a caso, perchè poi devo andare a scartabellarmi i file in cerca di roba da eseguire sperando di interpretare le vostre intenzioni. Quindi prima di consegnare a chichessia, date un occhio che sia tutto in ordine e possibilmente fate provare lo zip al componente del vostro gruppo che *meno* ne sa di informatica. Se riesce ad installare lui/lei senza suggerimenti, allora forse ci riuscirò anch'io.

7.2 Progetto SoftPython - Template

ATTENZIONE: QUESTO E' SOLO UN TEMPLATE, LEGGI ANCHE TUTTA LA PAGINA COME FARE UN PROGETTO[438]

METTERE:

TITOLO

NOME GRUPPO

NOMI / MATRICOLA PARTECIPANTI

DATA

[438] https://it.softpython.org/project-howto.html

7.2.1 Introduzione

Descrivere

- gli obbiettivi che ci si è posti
- risultati attesi (i.e. file CSV con dati integrati da mostrare su Umap)
- Descrivere brevemente quali attività sono state svolte. Mettere solo una **breve sintesi**, più avanti nel report vi sarà chiesto di descriverle nel dettaglio. Esempi:

```
Analisi
Pulizia
Integrazione
Arricchimento
    georeferenziazione
    semantico
Ricerca
    Base
    Avanzata
    Calcolo similarità
    Ranking
Predizione
Presentazione
    slides
    demo
```

- librerie Python utilizzate per il progetto

7.2.2 Sorgenti dati

Descrivere le sorgenti dati (almeno due), mettendo:

- nome origine e possibilmente URL
- le licenze
- encoding
- dimensione in Kb (i file dovrebbero essere max 50 megabyte, se più grandi cercate un modo per ridurli e se avete problemi chiedete come fare)
- schemi dei dati
  - CSV: intestazione colonne, tipi delle colonne (stringa, numero, data …)
  - JSON: spiegare sommariamente le scheletro del JSON (se volete strafare scrivete un JSON schema[439], ma non l'abbiamo visto a lezione)
  - XML: spiegare sommariamente le scheletro dell'XML (possibilmente guardate se l'XML già fornisce un XML schema[440], ma non l'abbiamo fatto a lezione, se avete dubbi chiedete)
  - SQL: mettere schema del DB (molti browser di database possono generare diagrammi per gli schemi)
  - per altri formati: descrivere a parole
- qualche dato di esempio

[439] http://json-schema.org
[440] https://en.wikipedia.org/wiki/XML_schema

- Se i dati sono da ottenere tramite web API[441], chiedersi se è possibile ottenere gratuitamente tutti i dati desiderati entro i limiti d'uso dell'API

IMPORTANTE: ricordarsi di includere nella cartella del progetto una copia dei dati! Questa è fondamentale ai fini della riproducibilità del notebook, e vale in particolare per i dati ottenuti da Web API e pagine HTML, che possono variare nel tempo (i.e. dati meteo).

7.2.3 Analisi

ATTIVITA' OBBLIGATORIA

Cercare di capire bene cosa c'è dentro il dataset. Cercare problemi - se si può in Python o altrimenti con LibreOffice / Excel. Per esempio, in una colonna 'Categoria', quanti valori distinti ci sono? Sono sempre scritti allo stesso modo? Se c'è una colonna con gli indirizzi, gli indirizzi sono sempre formattati nello stesso modo?

- mostrare raggruppamenti, grafici frequenze. Esempi:
  - motivare con numeri perchè si sono scelte certe colonne piuttosto che altre
  - per le colonne usate, quante celle vuote c'erano in proporzione a quelle piene, con grafico percentuali a torta
  - qual'era la distribuzione dei valori in una data colonna?
  - quante righe sono state mantenute e quante eliminate?
- correlazioni tra valori (potreste usare librerie come Pandas)
- mostrare punti su mappa con UMap, notare raggruppamenti
- ricavare qualche modello dai dati, con plot su grafico (es: relazione lineare pressione / temperatura)
- mettere qualche widget per variare parametri del modello

7.2.4 Pulizia

Elencare eventuali problemi da correggere, e come li si è risolti. Esempi:

- dati mal formattati
- interpretazione dati mancanti, come li si è convertiti
  - i valori assenti sono da assumersi uguali a 0, stringa vuota, `None`, lista vuota ? …
- Mappe con punti in posti assurdi

7.2.5 Integrazione

Quando si integrano due dataset con contenuti simili, ma che possono avere formati e campi diversi (es. file CSV agritur e file XML delle strutture alberghiere)

Per un esempio di integrazione, vedere tutorial integrazione[442]

- Si sono uniti dei dataset ? Come?
- Su quali colonne è stata effettuata la join?
- Che problemi sono stati sorti?
- Come sono stati risolti?

[441] https://en.wikipedia.org/wiki/Web_API
[442] https://it.softpython.org/integration/integration-sol.html

7.2.6 Arricchimento

Si arricchiscono i dati quando si precisa meglio il loro significato. Per esempio, si specifica la posizione geografica aggiungendo le colonne latitudine e longitudine, ottenute passando l'indirizzo testuale a servizi di geocoding. Vedere tutorial integrazione[443]. Altro esempio: si determina il significato preciso delle parole nel testo usando disambiguazione semantica come Dandelion.

- Sono stati usati dei metodi, librerie Python o servizi via WEBAPI per precisare il significato dei dati?
- Sono state trovate coordinate geografiche ? Come?
- Sono state estratte entità / concetti dal testo ? Come ?

7.2.7 Ricerca

L' *information retrieval* è l'attività che svolgono i sistemi informatici quando devono recuperare *rapidamente* delle informazioni *rilevanti* . Vedere tutorial information retrieval[444]

Si è implementato un sistema di ricerca? Come? Che fattori sono stati considerati ?

Ricerca base

Possiamo categorizzare la ricerca come 'base' se permette

- matching di parole esatto, es cerco "pavimentazione" e se c'è "pavimentazioni" con la "i" alla fine o "manto stradale" non la trova
- ricerca di valori tra due limiti. Es cerco tutti i bandi pubblicati entro due date. I valori da cercare sono già facilmente disponibili in celle separate in formato machine readable e non serve estrarli con tecniche particolari
- inserimento query di ricerca come parametro di funzioni python
- visualizzazione risultati come semplici print testuali

Ricerca avanzata

Possiamo considerare una ricerca 'avanzata' se richiede una elaborazione preventiva dei dati per una maggiore precisione. Per es. si vuole ricercare per importo complessivo di un bando, ma il valore dell'importo è nella descrizione e va preventivamente estratto in una colonna a parte. Vedere regex su softpython[445]

Miglioramento dell'interfaccia grafica (vedere tutorial applicazioni interattive[446]):

- inserimento della ricerca in una casella di ricerca widget jupyter
- possibilità di impostare qualche parametro della ricerca con dei widget (es checkbox per includere / escludere agritur nella ricerca)
- visualizzazione della ricerca in widget

[443] https://it.softpython.org/integration/integration-sol.html
[444] https://it.softpython.org/information-retrieval/information-retrieval-sol.html
[445] https://it.softpython.org/search/regex-sol.html
[446] https://it.softpython.org/gui/gui-sol.html

Ricerca semantica

La ricerca 'semantica' prende in considerazione il significato delle parole, vedi teoria information retrieval[447]

- matching di parole non esatto, es cerco "pavimentazione" e se c'è "pavimentazioni" con la "i" alla fine o "manto stradale" trova una corrispondenza
- ricerca per similarità: Sono stati adottati metodi per calcolare la similarità tra gli elementi da cercare? Quali sono le performance?

Ranking

- I risultati della ricerca vengono presentati in un ordine casuale o preciso?
- Sono ordinati per *rilevanza* ?
- Quali fattori si usano per calcolarla?
- Che formula si è usata per creare il ranking ?

7.2.8 Predizione

- Si è implementato un sistema predittivo (es. date n - 1 colonne riesco a prevedere il valore dell'ennesima con una precisione del X %)?
- Come è stata calcolata la formula per predire la colonna mancante?

7.2.9 Presentazione

Slides

Sono state create delle slides? Dove si trovano (se online si *deve* consegnare una copia pdf nello zip) ?

Demo

NOTA: questa parte è per la *demo*, che è distinta dalla sezione *Analisi*. Se prima hai usato dei widget per mostrare i parametri dei modelli, quei widget non vanno riportati qui

- Se il tuo progetto prevede una parte di demo in Jupyter, includi nel foglio separato demo.ipynb i widget funzionanti
- Se invece hai fatto il progetto come sistema esterno a Jupyter, **spiega bene qui come eseguirlo**, includendo degli screenshot del sistema.

Condividere codice: Se hai codice da condividere tra più fogli, invece del copia-e-incolla considera creare file di codice sorgente Python con estensione `.py` come `esempio.py`. In entrambi i fogli poi potresti fare

```
from esempio import *
```

[447] https://it.softpython.org/information-retrieval/information-retrieval-sol.html#Prendiamo-le-distanze

7.2.10 Problematiche riscontrate

Descrivere i problemi maggiori riscontrati nel progetto

7.2.11 Conclusioni

Descrivere:

- se si sono raggiunti gli obbiettivi posti
- eventualmente cosa si potrebbe aggiungere di interessante

[ ]:

```
```

7.3 Markdown

Per formattare il testo, Jupyter mette a disposizione un linguaggio chiamato Markdown. Perchè dovresti imparare Markdown? E' semplice, molto popolare ed è probabile ritrovarlo in molti posti (blog, sistemi di documentazione tecnica, etc).

Qua riportiamo solo informazioni essenziali, per altre puoi consultare la Guida di Jupyter (inglese)[448]

7.3.1 Celle Markdown

Per dire a Jupyter che una cella è codice Markdown e non Python, dal menu seleziona `Cell->Cell type->Markdown`. Una shortcut veloce è premere `Esc` seguito poi da il tasto `m`

7.3.2 Paragrafi

Per suddividere paragrafi basta inserire una riga vuota:

Per esempio, scrivendo così:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor␣
↪incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud␣
↪exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu␣
↪fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in␣
↪culpa qui officia deserunt mollit anim id est laborum.
```

Si ottiene questo:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

[448] http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html

7.3.3 Titoli

I titoli (*headers*) si scrivono anteponendo dei cancelletti al titolo stesso:

```
# Titolo pagina

E' importante.

## Primo capitolo

Ma perchè?

### Primo paragrafo

Che ne so.

### Secondo paragrafo

E spiegalo !


## Secondo capitolo

Taci e scrivi.
```

Titolo pagina

E' importante.

7.3.4 Primo capitolo

Ma perchè?

Primo paragrafo

Che ne so.

Secondo paragrafo

E spiegalo !

7.3.5 Secondo capitolo

Taci e scrivi.

7.3.6 Link

Un link si scrive mettendo il testo visibile dall'utente tra parentesi quadre e l'indirizzo vero e proprio tra parentesi tonde:

```
[Questo è il testo del link](http://www.google.com)
```

Il risultato sarà il seguente:

Questo è il testo del link[449]

7.3.7 Liste

Le liste si scrivono anteponendo ad ogni elemento un asterisco:

Per esempio, scrivendo questo:

```
* Elemento 1
* Elemento 2
    * Sotto elemento
    * Altro sotto elemento
* Elemento 3
```

Verrà visualizzato così:

- Elemento 1
- Elemento 2
  - Sotto elemento
  - Altro sotto elemento
- Elemento 3

Potete anche usare liste numerate anteponendo agli elementi un numero e un punto.

Per esempio, questo:

```
1. Elemento 1
2. Elemento 2
    1. Sotto elemento
    2. Altro sotto elemento
3. Elemento 3
```

Verrà visualizzato così:

1. Elemento 1
2. Elemento 2
   1. Sotto elemento
   2. Altro sotto elemento
3. Elemento 3

Se la lista è lunga e dovete editarla spesso, invece di numeri espliciti conviene mettere sempre `1.` e Markdown automaticamente calcolerà i numeri corretti:

[449] http://www.google.com

```
1. Elemento 1
1. Elemento 2
    1. Sotto elemento
    1. Altro sotto elemento
1. Elemento 3
```

verrà visualizzato così:

1. Elemento 1
2. Elemento 2
    1. Sotto elemento
    2. Altro sotto elemento
3. Elemento 3

7.3.8 Immagini

Purtroppo Jupyter non supporta il copia e incolla di immagini, puoi solo inserire dei link alle immagini stesse da mettere nella stessa cartella del notebook oppure in una sottocartella.

Una volta che la cella viene eseguita, se il percorso al file è corretto apparirà l'immagine. Per indicare il percorso, scrivi punto esclamativo, parentesi quadre aperta-chiusa, e poi tra parentesi tonde il percorso del file (possibilmente usando lo slash `'/'`), per es se l'immagine `notebook_icon.png` sta nella sotto-cartella `img`, scrivi così:

```
![](img/example.png)
```

Eseguendo la cella, apparirà l'immagine:

Nota che puoi usare qualunque formato di immagine (`jpg`, `png`, `bmp`, `svg`, per gli altri prova a vedere se vengono visualizzati).

7.3.9 Variabili e nomi tecnici

Per visualizzare in evidenza variabili e nomi tecnici, come `x`, `faiQualcosa`, `percorso-di-file`, puoi includere il nome tra due cosiddetti backtick `

NOTA: il backtick ` NON è l'apice che usiamo di solito: `'`

Per scrivere questo strano apice rovesciato, guarda qua, se non va fai copia e incolla !

- Windows:
    - se hai il tastierino numerico: tenere premuto `Alt Gr`, scrivere `96` sul tastierino numerico, rilasciare `Alt Gr`
    - se non ce l'hai: prova a premere tasto windows + \ (in alto a sinistra)

- Mac: `alt+9`
- Linux: `Alt-Gr` + carattere apice normale `'`

7.3.10 Codice JSON / XML / Python

Se in una cella Markdown vuoi visualizzare testo posizionato esattamente come lo scrivi, racchiudilo in un blocco delimitato da file di tre tre backtick ``` :

```

```
testo posizionato come
    voglio
 io
```

```

Risultato:

```
testo posizionato come
    voglio
 io
```

Il codice python / json / xml e altri possono essere formattati automaticamente da Jupyter. Basta scriveterlo in blocchi da tre backtick come prima e in più specificare il linguaggio subito dopo i primi tre backtick, per esempio un json scritto così:

```json

```
{
   "a" : ["b"],
   "c" : {
           "d":5
         }

}
```

```

Risulterà formattato in questo modo:

```
{
   "a" : ["b"],
   "c" : {
           "d":5
         }

}
```

7.3.11 Formule matematiche

E' possibile scrivere formule in Latex[450] (ma non è trattato in questo libro) mettendole tra segni di dollaro `$`. Per esempio `$\sum$` verrà visualizzato così:

$\sum$

Con due dollari ai lati `$$\sum$$` la formula viene centrata:

$$\sum$$

7.3.12 Tabelle

Scrivere tabelle piccole in Markdown è ancora fattibile:

Per esempio, scrivere questo:

```
io |sono | una tabella
---|-----|------------
4|ciao|3
2|hello world|7
```

risulta visualizzato così :

io	sono	una tabella
4	ciao	3
2	hello world	7

ma per tabelle grandi Markdown è terribile. Quindi si è più che giustificati a usare alternative, per esempio allegare fogli Excel. Si può anche prendere screenshot delle tabelle e includerli come immagini.

```
[ ]:
```

7.4 Idee per progetti

Ultimo aggiornamento: 31 Agosto 2018

7.4.1 Introduzione

In questa pagina sono raccolte idee per progetti che potresti fare. Sono solo spunti, quindi sentitivi liberi di modificarli a piacere. Riteniamo che imparare Python possa essere molto più interessante se si ha in mente un qualche esempio di analisi da replicare in Jupyter, anche in misura semplificata solo per provare.

[450] http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf#chapter.5

7.4.2 Dataset da Kaggle

Area di interesse: tutte !

Kaggle è il sito di riferimento per data scientists, ricco di dataset e spunti interessanti ben categorizzati (per scaricare i dataset occorre registrarsi). Dategli un'occhiata perchè merita:

https://www.kaggle.com/datasets

Alcuni aree presenti tra le 1000:

- linguistica[451]
- sociologia[452]
- finanza[453]
- fisica[454]
- biologia[455]
- ingegneria meccanica[456]
- ingegneria civile[457]

Si può semplicemente usare il sito come fonte dati, oppure seguire le sfide proposte: prendendo per esempio le recensioni, si potrebbe scrivere un programmino rudimentale che cerca di capire se una recensione Amazon è positiva guardando se contiene le parole 'bello/a', 'fantastico/a', e poi confrontare i risultati con il numero di stelle che l'utente ha assegnato all'articolo per vedere quanto concordano.

Qua sotto riportiamo come prendere da Kaggle per esempio il CSV dei video trending di YouTube[458] che contengono cose tipo i like, la descrizione, il numero di commenti etc :

[451] https://www.kaggle.com/datasets?sortBy=hotness&group=public&page=1&pageSize=20&size=all&filetype=all&license=all&tagids=11208
[452] https://www.kaggle.com/datasets?sortBy=hotness&group=public&page=1&pageSize=20&size=all&filetype=all&license=all&tagids=11215
[453] https://www.kaggle.com/datasets?sortBy=hotness&group=public&page=1&pageSize=20&size=all&filetype=all&license=all&tagids=11108
[454] https://www.kaggle.com/datasets?sortBy=relevance&group=public&search=physics&page=1&pageSize=20&size=all&filetype=all&license=all
[455] https://www.kaggle.com/datasets?sortBy=hotness&group=public&page=1&pageSize=20&size=all&filetype=all&license=all&tagids=7103
[456] https://www.kaggle.com/datasets?sortBy=hotness&group=public&page=1&pageSize=20&size=all&filetype=all&license=all&tagids=12308
[457] https://www.kaggle.com/datasets?sortBy=relevance&group=public&search=civil+&page=1&pageSize=20&size=all&filetype=all&license=all&tagids=12304
[458] https://www.kaggle.com/datasnaek/youtube-new

7.4.3 Analisi dati socio-economici GapMinder

Area di interesse: sociologia, economia

Il progetto GapMinder[459] mira a ridurre l'ignoranza globale proponendo strumenti online di facile utilizzo per analizzare dati statistici su indicatori quali per esempio il reddito pro-capite per nazione, la diffusione delle malattie, i flussi migratori, etc. Si potrebbero quindi creare in Jupyter delle interfacce che riproducano in qualche modo i grafici di GapMinder.

[459] https://www.gapminder.org/tools/#_chart-type=bubbles

- Come dati, si potrebbero scaricare manualmente due-tre dataset da questa lista[460] e mostrarli in un grafico creato con matplotlib (vedere *tutorial SoftPython visualizzazione dati*)
- aggiungere un widget selettore (vedere *tutorial interfacce grafiche SoftPython*) per filtrare i paesi da mostrare nel grafico, sullo stile dell'esempio *Wealth of Nations* di Bqplot
- aggiungere uno slider per permettere all'utente di variare l'anno della visualizzazione
- supportare altri tipi di visualizzazioni, per esempio mappe[461]
- aggiungere un selettore per poter scegliere il tipo di grafico da visualizzare

Altre fonti dati

- dati.trentino.it[462]
- dati.gov.it[463]

7.4.4 Analisi quotazioni di borsa

Area di interesse: economia

Visualizzare in Jupyter grafici sull'andamento di valute e serie storiche macroeconomiche presi da

- Alpha Vantage(https://www.alphavantage.co/): per le serie storiche dei prezzi dei titoli finanziari
- FRED (https://fred.stlouisfed.org/): per le serie storiche economiche (macroeconomiche) USA

[460] https://www.gapminder.org/data/
[461] https://www.gapminder.org/tools/#_chart-type=map
[462] http://dati.trentino.it
[463] https://www.dati.gov.it

Per entrambi i siti è richiesta una key per scaricare i dati. Queste sono ottenibli gratuitamente dopo essersi registrati

L'utilizzo dei dati provenienti da Alpha Vantage è libero, viene solo limitata la frequenza delle richieste (quindi mettete delle attese programmate tra uno scaricamento e l'altro). Invece l'utilizzo dei dati provenienti da FRED è soggetto a dei specifici termini d'uso (https://research.stlouisfed.org/fred_terms.html) I dati scaricati sono di volta in volta decisi dall'utente che deve specificare le sigle del titolo (ES: AAPL (Apple), MSFT (Microsoft), GE (General Electric), ^GSPC (S&P 500)) oppure della variabile economica (ES: BAA (Moody's Seasoned Baa Corporate Bond Yield), GDPCA (Real Gross Domestic Product), CPIAUCSL (Consumer Price Index for All Urban Consumers: All Items)) che vuole analizzare. Le dimensioni dei file scaricati sono variabili.

I dati scaricati (ultimi 100 dati giornalieri) da Alpha Vantage sono in formato csv con al seguente struttura (intestazione ed esempio riga):

```
timestamp, open, high, low, close, adjusted_close, volume, dividend_amount, split_
↪coefficient

2018-02-02, 166.0000, 166.8000, 160.1000, 160.5000, 160.5000, 85437085, 0.0000, 1.0000
```

Ad eccezione di 'timestamp' che indica la data dei dati di riga, i restanti valori sono numeri che rappresentano ad esempio il prezzo di apertura, chiusura, chiusura aggiustato, volumi scambiati, ecc.

I dati scaricati (l'intera serie storica dei dati annuali, è possibile modificare eventualmente la frequenza modificando opportuamente l'url) da FRED sono in formato xml con al seguente struttura:

```
<observations realtime_start="2018-02-06" realtime_end="2018-02-06" observation_start=
↪"1600-01-01" observation_end="9999-12-31" units="lin" output_type="1" file_type="xml
↪" order_by="observation_date" sort_order="asc" count="88" offset="0" limit="100000">
<observation realtime_start="2018-02-06" realtime_end="2018-02-06" date="1929-01-01"␣
↪value="1066.782"/>
```

(continues on next page)

(continued from previous page)

```
<observation realtime_start="2018-02-06" realtime_end="2018-02-06" date="1930-01-01"␣
↪value="976.305"/>
```

7.4.5 Analisi quotazioni criptovalute

Area di interesse: economia

Visualizzare in Jupyter grafici sull'andamento di criptovalute presi da coinmarketcap.com[464]

- Esempio formato dati ottenibili:
  - https://api.coinmarketcap.com/v1/ticker/
- Per altre API, vedere: https://coinmarketcap.com/api/
- Creare in Jupyter un configuratore di widget (vedere *tutorial interfacce grafiche SoftPython*), sull'esempio del *Cryptocurrency Price Ticker Widget*[465]. L'output del programma dovrebbero essere oggetti widget Jupyter (non HTML).
- Addizionalmente, il configuratore potrebbe anche generare del codice HTML come questo (per informazioni sull'HTML, vedere *tutorial estrazione dati SoftPython*):

```
<script type="text/javascript"
        src="https://files.coinmarketcap.com/static/widget/currency.js">
</script>
<div class="coinmarketcap-currency-widget"
     data-currencyid="1"
     data-base="USD"
     data-secondary=""
     data-ticker="true"
     data-rank="true"
     data-marketcap="true"
     data-volume="true"
     data-stats="USD"
     data-statsticker="false">
 </div>
```

Il codice HTML può essere creato come oggetti BeautifulSoup o anche come semplice concatenazione di stringhe

Notate che se in una cella di Jupyter si scrive all inizio `%%HTML` e sotto si incolla il codice HTML come quello sopra, dovrebbe apparire qualcosa come questo:

[1]:
```
%%HTML

<script type="text/javascript"
        src="https://files.coinmarketcap.com/static/widget/currency.js">
</script>
<div class="coinmarketcap-currency-widget"
     data-currencyid="1"
     data-base="USD"
     data-secondary=""
     data-ticker="true"
     data-rank="true"
     data-marketcap="true"
```

(continues on next page)

[464] http://coinmarketcap.com
[465] https://coinmarketcap.com/widget/

(continued from previous page)

```
    data-volume="true"
    data-stats="USD"
    data-statsticker="false">
</div>
```

```
<IPython.core.display.HTML object>
```

7.4.6 Frequenza parole articoli Wikipedia

Area di interesse: lettere e filosofia, storia

Riportare in Jupyter la frequenza di parole da articoli presi da Wikipedia. A breve verranno aggiunti dettagli!

7.4.7 Filosofi influenti

Area di interesse: Lettere e Filosofia, Storia

Tutti conosciamo Wikipedia, l'enciclopedia online creata da volontari di tutto il mondo. In Wikipedia a volte alcune informazioni sono semistrutturate, come per esempio gli infobox. Vediamone uno per il filosofo Parmenide. Dall'infobox della versione inglese si può notare che è presente il campo 'influenced', che ci dice chi ha influenzato Parmenide:

Parmenides

Bust of Parmenides discovered at Velia, thought to have been partially modeled on a Metrodorus bust.[1]

Born	c. 515 BC[2] Elea, Magna Graecia
Era	Pre-Socratic philosophy
Region	Western philosophy
School	Eleatic school
Main interests	Metaphysics (ontology)
Notable ideas	"Thought and being are the same"[3] The truth–appearance distinction
Influences	[show]
Influenced	[hide]

Zeno of Elea, Melissus of Samos, Socrates, Plato, Aristotle, Spinoza, Nietzsche, Heidegger, Emanuele Severino

Sarebbe interessante estrarre queste informazioni per fare per esempio grafici che mostrino i legami di chi ha influenzato chi nel corso della storia, tenendo naturalmente presente il fatto che Wikipedia non è sempre completa e al 100% affidabile. Questo è un esempio:

Cose che si potrebbero fare:

- mostrare grafo dei filosofi
- creare un widget selettore, in qui si seleziona un pensatore e vengono mostrati in una lista i pensatori che ha influenzato
- mostrare chi non ha influenzato nessuno (secondo Wikipedia !). Se un personaggio è un filosofo famoso dovrà pur aver influenzato qualcuno quindi potremmo creare uno strumento che mostri a potenziali contributori di Wikipedia pagine di filosofi che necessitano ulteriori informazioni.
- altri dettagli verranno aggiunti a breve !

Formato file filosofi:

descrizione (in inglese)[466]

Il dataset è stato ottenuto con questa query SPARQL su dbpedia[467] . Per ottenere un file processabile in Python con gli strumenti che affronteremo durante il corso, si può richiedere un file JSON con query sparql su dbpedia, formato json[468]

[466] http://brendangriffen.com/blog/gow-influential-thinkers

[467] http://dbpedia.org/snorql/?query=SELECT+*%0D%0AWHERE+%7B%0D%0A%3Fp+a+%3Chttp%3A%2F%2Fdbpedia.org%2Fontology%2FPhilosopher%3E+.%0D%0A%3Fp+%3Chttp%3A%2F%2Fdbpedia.org%2Fontology%2Finfluenced%3E+%3Finfluenced.%0D%0A%7D

[468] http://dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&query=PREFIX+owl%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F2002%2F07%2Fowl%23%3E%0D%0APREFIX+xsd%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F2001%2FXMLSchema%23%3E%0D%0APREFIX+rdfs%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F2000%2F01%2Frdf-schema%23%3E%0D%0APREFIX+rdf%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-ns%23%3E%0D%0APREFIX+foaf%3A+%3Chttp%3A%2F%2Fxmlns.com%2Ffoaf%2F0.1%2F%3E%0D%0APREFIX+dc%3A+%3Chttp%3A%2F%2Fpurl.org%2Fdc%2Felements%2F1.1%2F%3E%0D%0APREFIX+%3A+%3Chttp%3A%2F%2Fdbpedia.org%2Fresource%2F%3E%0D%0APREFIX+dbpedia2%3A+%3Chttp%3A%2F%2Fdbpedia.org%2Fproperty%2F%3E%0D%0APREFIX+dbpedia%3A+%3Chttp%3A%2F%2Fdbpedia.org%2F%3E%0D%0APREFIX+skos%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F2004%2F02%2Fskos%2Fcore%23%3E%0D%0ASELECT+*%0D%0AWHERE+%7B%0D%0A%3Fp+a+%3Chttp%3A%2F%2Fdbpedia.org%2Fontology%2FPhilosopher%3E+.%0D%0A%3Fp+%3Chttp%3A%2F%2Fdbpedia.org%2Fontology%2Finfluenced%3E+%3Finfluenced.%0D%0A%7D&output=json

Riportiamo qua un estratto del file:

```
{
  "head": {
            "link": [],
            "vars": ["p", "influenced"]
          },

  "results": {
                "distinct": false,
                "ordered": true,
                "bindings": [

                                {
                                  "p": {
                                           "type": "uri",
                                           "value": "http://dbpedia.org/resource/
↪Parmenides"
                                        },
                                  "influenced": {
                                                    "type": "uri",
                                                    "value": "http://dbpedia.org/
↪resource/Socrates"
                                                 }
                                },

                                {
                                  "p": {
                                           "type": "uri",
                                           "value": "http://dbpedia.org/resource/
↪Socrates"
                                        },
                                  "influenced": {
                                                    "type": "uri",
                                                    "value": "http://dbpedia.org/
↪resource/Aristotle"
                                                 }
                                },

                                ETC... TAGLIATA ......................
}
```

Possiamo ignorare

```
{
  "head": {
            "link": [],
            "vars": ["p", "influenced"]
          },

  "results": {
                "distinct": false,
                "ordered": true,
```

e concentrarci sulla parte che ci interessa, che è quella sotto `results["bindings"]` e vediamo essere distinta da una sequenza di oggetti contenenti ciascuno un riferimento ad un filosofo e un riferimento ad un'altro filosofo che è stato influenzato dal primo. In questo caso è riportato che Parmenide ha influenzato Socrate:

```
{
  "p": {
          "type": "uri",
          "value": "http://dbpedia.org/resource/Parmenides"
      },
  "influenced": {
                  "type": "uri",
                  "value": "http://dbpedia.org/resource/Socrates"
               }
},
```

In particolare in questo caso Parmenide è identificato dall' indirizzo http://dbpedia.org/resource/Parmenides (provate a cliccarci sopra per vedere cosa DBpedia contiene su Parmenide) e il personaggio Socrate viene esplicitato dall'indirizzo http://dbpedia.org/resource/Socrates (di nuovo provate a cliccarci sopra)

7.4.8 Personaggi culturalmente importanti

Pantheon[469] è un progetto online per aiutare a comprendere il processo di sviluppo culturale globale. Il progetto consisterebbe nel ricreare in fogli Jupyter alcune delle visualizzazioni presenti sul sito di Pantheon, come per esempio questa:

In particolare, vengono offerti tre dataset[470] estratti da Wikipedia con cui sono state realizzate le visualizzazioni online.

[469] http://pantheon.media.mit.edu
[470] http://pantheon.media.mit.edu/about/datasets

- visualizzazione scatterplot[471] che compara domini per rispondere a domande tipo "Quali luoghi di nascita hanno prodotto personalità nei campi dell'economia e della medicina?".
- altri dettagli verranno aggiunti a breve

7.4.9 Analisi attività parlamento

Area di interesse: Giurisprudenza

Open Parlamento[472], uno strumento di OpenPolis, contiene una quantità di informazioni e statistiche sull'operato dei nostri parlamentari:

Qua troviamo un esempio per il Partito Democratico:

https://parlamento17.openpolis.it/lista-dei-parlamentari-in-carica/camera/nome/asc/filter_group/71

L'obiettivo del progetto sarebbe replicare parte dei contenuti del sito in Jupyter. Per ottenere i dati, si può usare la comoda API fornita da OpenPolis all'indirizzo api3.openpolis.it[473]

Vediamo per esempio come ottenere i gruppi parlamentari:

Notate che nella URL di sopra il `parlamento17` all'inizio indica la diciassettesima legislatura (la corrente nel 2018) e il `71` alla fine, che è il codice assegnato al partito. I codici li potete ottenere con le API di OpenPolis. Per vedere quale codice numerico è assegnato ai vari gruppi parlamentari, guardate qua (il `/17/` indica sempre la 17esima legislatura):

[471] http://pantheon.media.mit.edu/scatterplot/domain_vs_domain/ECONOMIST/MEDICINE/-4000/2000/H15/pantheon
[472] https://parlamento17.openpolis.it
[473] http://api3.openpolis.it

http://api3.openpolis.it/parlamento/17/groups

Se andate col browser a questa URL sopra, dovreste vedere qualcosa di simile - nell'interfaccia web potete comodamente navigare le api e vedere i risultati delle chiamate, che sono sempre in formato JSON:

- Per integrare dati con altre fonti, potreste scaricare delle pagine HTML da siti di news a tema politica, estrarre nomi di partiti politici e visualizzare in interfacce grafiche in Jupyter dei link a corrispondente pagine dei politici in OpenParlamento. Notate che negli articoli i nomi dei partiti potrebbero essere leggermente diversi o sigle (es. 'PD','Lega' invece di 'Lega Nord'), quindi potreste implementare delle funzioni che cercano di identificare il partito.
- Come altri esempi, potreste mostrare i parlamentari del gruppo. Questa API prende i parlamentari del Partito Democratico (`group=71`) per la 17esima legislatura (`/17/`):

http://api3.openpolis.it/parlamento/17/parliamentarians?group=71&page=1

Vengono ritornati solo 25 risultati. per avere i 25 successivi, incrementare il numero assegnato alla `page=1`

Analisi attività parlamento: usare Wikidata per trovare i segretari di partito

Volendo, per trovare altre informazioni come i segretari di partito, potreste usare Wikidata[474].

Provate ad eseguire questa query SPARQL (per eseguirla online cliccate il bottone blue play in basso a sinistra, i risultati appariranno sotto): http://tinyurl.com/y8yel2mv

Non li dà tutti, ma per es. almeno Renzi e Salvini li trovate. Anche se non conoscete il linguaggio SPARQL, se provate a smanettare volendo potreste anche ricavare qualcosa in più, prendendo come esempio attributi da Partito Democratico[475] e Renzi[476]

Il seguente codice Python lo potete ottenere cliccando nel query editor cliccando in basso a destra->Code -> Python

```
#python3 -m pip install sparqlwrapper
#https://rdflib.github.io/sparqlwrapper/

from SPARQLWrapper import SPARQLWrapper, JSON
sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
sparql.setQuery("""SELECT DISTINCT ?political_party ?political_partyLabel ?party_
↪chief_representative ?party_chief_representativeLabel ?Openpolis_ID WHERE {
  SERVICE wikibase:label { bd:serviceParam wikibase:language "[it],it". }
  ?political_party wdt:P31 wd:Q7278.
  ?political_party wdt:P17 wd:Q38.
  OPTIONAL { ?political_party wdt:P210 ?party_chief_representative.
             ?party_chief_representative wdt:P1229 ?Openpolis_ID. }
}
LIMIT 500""")
sparql.setReturnFormat(JSON)
results = sparql.query().convert()

for result in results["results"]["bindings"]:
    print(result)
```

Il codice Python vi stamperà qualcosa del genere:

```
{'political_party': {'value': 'http://www.wikidata.org/entity/Q286140', 'type': 'uri'}
↪, 'political_partyLabel': {'value': 'Sinistra Ecologia Libertà', 'xml:lang': 'it',
↪'type': 'literal'}}
{'political_party': {'value': 'http://www.wikidata.org/entity/Q541679', 'type': 'uri'}
↪, 'political_partyLabel': {'value': 'Democratici di Sinistra', 'xml:lang': 'it',
↪'type': 'literal'}}
{'political_party': {'value': 'http://www.wikidata.org/entity/Q662849', 'type': 'uri'}
↪, 'political_partyLabel': {'value': 'Alleanza Nazionale', 'xml:lang': 'it', 'type':
↪'literal'}}
{'political_party': {'value': 'http://www.wikidata.org/entity/Q764125', 'type': 'uri'}
↪, 'political_partyLabel': {'value': "Partito d'Azione", 'xml:lang': 'it', 'type':
↪'literal'}}
```

In particolare per il Partito Democratico e Renzi, formattandolo bene otterresti questo:

```
{
   'political_party': {
       'value': 'http://www.wikidata.org/entity/Q47729',
       'type': 'uri'
   },
```

(continues on next page)

[474] https://www.wikidata.org
[475] https://www.wikidata.org/wiki/Q47729
[476] https://www.wikidata.org/wiki/Q47563

(continued from previous page)

```
    'political_partyLabel': {
       'value': 'Partito Democratico',
       'xml:lang': 'it',
       'type': 'literal'
    },
    'party_chief_representative': {
        'value': 'http://www.wikidata.org/entity/Q47563',
        'type': 'uri'
    },
    'party_chief_representativeLabel': {
      'value': 'Matteo Renzi',
      'xml:lang': 'it',
      'type': 'literal'
    },
    'Openpolis_ID': {
        'value': '6877',
        'type': 'literal'
    }
}
```

7.4.10 Interfacce e tecnologie per la comunicazione

Area di interesse: Interfacce e tecnologie per la comunicazione

- Visualizzazione e analisi in Jupyter di risultati di esperimenti di HCI ?
- Altri dettagli verranno aggiunti a breve.

7.4.11 Analisi genoma

Area di interesse: Biotecnologia

Altri dettagli verranno aggiunti a breve

7.4.12 Supply Chain

Area di interesse: Ingegneria industriale (DII)

Realizzare qualche interfaccia in Jupyter sul modello di questa: http://arcg.is/1Hm45L

I dati si possono prelevare da Kaggle , cercando dataset taggati supply chain[477]

[477] https://www.kaggle.com/datasets?sortBy=hottest&group=public&page=1&pageSize=20&size=all&filetype=all&license=all&tagids=11131

7.4.13 Tracciamento di oggetti astronomici

Area di interesse: Ingegneria industriale (DII), Ingegneria Civile ambientale (DICAM), Fisica, Matematica indirizzo didattico

Il progetto consiste nel realizzare in un foglio Jupyter un tracciatore di oggetti astronomici come satelliti, pianeti, auto Tesla… Come modello si può guardare l'ISS Tracker[478] che segue la Stazione Spaziale Internazionale:

[478] http://www.isstracker.com/

In Jupyter si potrebbe:

- avere una barra di testo dove si immette l'ora (in alternativa si può usare uno slider)
- da un menu drop-down, poter selezionare un qualche oggetto celeste, tipo ISS
- usando i file del NORAD[479] , mostrarne la posizione dell'oggetto selezionate su mappa terrestre in cella Jupyter
- mostrare la posizione dell'oggetto su mappa celeste rispetto ad un punto d'osservazione, per calcolare la posizione esatta si può usare la libreria PyEphem, per qualche esempio vedere tutorial CoderDojo Trento[480]

Dati AstroPI

Volendosi concentrare sulla Stazione Spaziale Internazionale (ISS), sarebbe interessante riutilizzare dati ambientali interni ed esterni raccolti sulla ISS per gli esperimenti AstroPI proposti da CoderDojo Trento in collaborazione con l'Agenzia Spaziale Europea e la Raspberry Foundation - per una presentazione molto generale vedere comunicato stampa[481].

Per le tempistiche, vedere timeline[482]: correntemente (periodo Febbraio - Maggio 2018) gli esperimenti sono in fase di attuazione sulla ISS, e dati 'freschi' saranno scaricati dalla stazione entro il 10 Maggio 2018 ma sono disponibili anche dati da edizioni passate. In particolare per voi di interessante ci sono i due progetti presentati per la Mission Space Lab:

Progetto Team Lamponi - ISS Orbit

In questo progetto sono raccolte foto della Terra per calcolare la velocità della alla Stazione Spaziale Internazionale (ISS) in base a oggetti rilevati sul suolo, paragonandola alla velocità reale:

[479] https://www.celestrak.com/NORAD/elements/
[480] https://www.coderdojotrento.it/astropi2
[481] https://www.coderdolomiti.it/2018/02/08/astropi-challenge-2017-18/
[482] https://astro-pi.org/missions/space-lab/

- descrizione progetto ISS Orbit[483]
- immagini esempio rilevanti[484] (sono ~161 metri per pixel)

Le serie completa di immagini da edizioni passate la trovate nel progetto del 2015 environ_pi - notare che trattandosi di esperimenti 'alla buona', si vedono i bordi dell'oblò e dei riflessi. In più in alcune immagini sono nere perchè riprese dal lato non illuminato dal Sole :

- descrizione environ_pi[485]
- scarica zip environ_pi[486]

Progetto Trentini DOP - Space Pressure

In quest'altro vengono prese rilevazioni da sensori interni alla stazione, come pressione, temperatura, accelerazione, etc.

- descrizione progetto Space Pressure[487]
- formato dati[488] - NOTA: questi sono rilevati dalla scrivania :-)

Come esempio di dati veri da edizioni passate, puoi guardare il tutorial *Pandas in SoftPython*, in cui usiamo questi dati commentati[489] di AstroPI

In base a quanto sopra, si potrebbe quindi:

- mostrare dentro dei box testuali i parametri ambientali della stazione rilevati nell'istante più prossimo
- in un altro riquadro, mostrare la foto ripresa nell'istante più vicino dalla ISS, oppure anche risultati dello stitching[490] operato da OpenCV
- mostrare velocità teorica in base a stitching e reale, con differenza in percentuale
- se interessa, volendo si potrebbe fare più analisi delle immagini con OpenCV, tipo identificare solo le nuvole o capire se la ISS sta sorvolando il mare / terra / foreste …

7.4.14 Ricerca sequenze numeri interi database OEIS

Area di interesse: Matematica

Un utile strumento a disposizione dei matematici è OEIS[491], l'enciclopedia online delle sequenze intere. Il sito permette di scrivere in input una sequenza di numeri interi come `1,2,3,6,11,23,47,106,235`, e il sito proverà a ritornare delle ricorrenze numeriche che la generano, più altre informazioni come gli articoli scientifici che ne parlano:

[483] https://github.com/CoderDojoTrento/AstroPi_2017-18/tree/master/Team%20Lampone%20-%20ISS%20Orbit/versione_ok
[484] https://github.com/CoderDojoTrento/AstroPi_2017-18/tree/master/Team%20Lampone%20-%20ISS%20Orbit/versione_ok/postprocessing/file_esempio
[485] https://astro-pi.org/principia/science-results/#trees
[486] https://github.com/astro-pi/enviro-pi/archive/master.zip
[487] https://github.com/CoderDojoTrento/AstroPi_2017-18/tree/master/Trentini%20DOP%20-%20Space%20Pressure
[488] https://github.com/CoderDojoTrento/AstroPi_2017-18/blob/master/Trentini%20DOP%20-%20Space%20Pressure/file_esempio/TDOP_2018-02-06_19.43.43.csv
[489] https://www.raspberrypi.org/learning/astro-pi-flight-data-analysis/worksheet/
[490] https://github.com/CoderDojoTrento/AstroPi_2017-18/blob/master/Team%20Lampone%20-%20ISS%20Orbit/versione_ok/postprocessing/EnviroPi_20160223_184131_keypoints.jpg
[491] http://oeis.org

Scrivo qua qualche spunto per il progetto:

- fare dei widget in Jupyter che permettano all'utente di inserire una sequenza numerica, poi
- con la libreria `request` spiegata in tutorial *Integrazione dati SoftPython*, chiamare le API di OEIS e ottenere dati sulla sequenza, in formato testuale nativo (più difficile da parsare) o JSON (più semplice)
- se si ottiene una ricorrenza, fare un plot dei punti precalcolati forniti
- provare a fare un parser (anche con regex, vedere tutorial *Ricerca in SoftPython*) che estrae nomi di autore dalla sezione `references` e li visualizza in un widget (tipo un `VBox` di `Label`):

```
"reference": [
    "F. Bergeron, G. Labelle and P. Leroux, Combinatorial Species and Tree-Like␣
↪Structures, Camb. 1998, p. 279.",
    "N. L. Biggs et al., Graph Theory 1736-1936, Oxford, 1976, p. 49.",
    "A. Cayley, On the analytical forms called trees, with application to the theory␣
↪of chemical combinations, Reports British Assoc. Advance. Sci. 45 (1875), 257-305 =␣
↪Math. Papers, Vol. 9, 427-460 (see p. 459).",
    "S. R. Finch, Mathematical Constants, Cambridge, 2003, pp. 295-316.",
    "J. L. Gross and J. Yellen, eds., Handbook of Graph Theory, CRC Press, 2004; p.␣
↪526.",
```

Esempi API OEIS

Esempio (tagliato) di dati già ben parsato in JSON: http://oeis.org/search?fmt=json&q=1,2,3,6,11,23,47,106,235

```
{
    "greeting": "Greetings from The On-Line Encyclopedia of Integer Sequences! http://
↪oeis.org/",
    "query": "1,2,3,6,11,23,47,106,235",
    "count": 1,
    "start": 0,
    "results": [
        {
            "number": 55,
            "id": "M0791 N0299",
            "data": "1,1,1,1,2,3,6,11,23,47,106,235,551,1301,3159,7741,19320,48629,
↪123867,317955,823065,2144505,5623756,14828074,39299897,104636890,279793450,
↪751065460,2023443032,5469566585,14830871802,40330829030,109972410221,300628862480,
↪823779631721,2262366343746,6226306037178",
```

(continues on next page)

(continued from previous page)

```
            "name": "Number of trees with n unlabeled nodes.",
            "comment": [
                "Also, number of unlabeled 2-gonal 2-trees with n 2-gons.",
                "Main diagonal of A054924.",
                "Left border of A157905. - _Gary W. Adamson_, Mar 08 2009",
                "From _Robert Munafo_, Jan 24 2010: (Start)",
                "Also counts classifications of n items that require exactly n-1␣
↪binary partitions; see Munafo link at A005646, also A171871 and A171872.",
                "The 11 trees for n = 7 are illustrated at the Munafo web link.",
```

Esempio (tagliato) di dati in formato interno: http://oeis.org/search?fmt=text&q=1,1,1,5,3,60,487

```
# Greetings from The On-Line Encyclopedia of Integer Sequences! http://oeis.org/

Search: seq:1,1,1,5,3,60,487
Showing 1-1 of 1

%I A007299 M3736
%S A007299 1,1,1,1,5,3,60,487,13710027
%N A007299 Number of Hadamard matrices of order 4n.
%C A007299 More precisely, number of inequivalent Hadamard matrices of order n if two␣
↪matrices are considered equivalent if one can be obtained from the other by␣
↪permuting rows, permuting columns and multiplying rows or columns by -1.
%C A007299 The Hadamard conjecture is that a(n) > 0 for all n >= 0. - _Charles R␣
↪Greathouse IV_, Oct 08 2012
```

Per dettagli sul formato, vedere:

- Internal format[492]
- Explanation of Terms Used in Reply From[493]

Se il parsing diretto non vi entusiasma, potete anche provare ad usare la libreria PyOEIS[494]

[492] http://oeis.org/eishelp1.html
[493] http://oeis.org/eishelp2.html
[494] http://pyoeis.readthedocs.io/en/latest/index.html

7.5 Challenges

7.5.1 Vai alle Challenges 2019

7.5.2 Vai alle Challenges 2018

```
[ ]:
```

CHAPTER

EIGHT

RIFERIMENTI

Citiamo i riferimenti prima divisi per argomento e poi varie risorse approssimativente in ordine di difficoltà crescente (una comparazione molto sommaria e incompleta è listata in questa tabella[495])

8.1 Riferimenti per argomento

8.1.1 Basi

- Pensare in Python - Capitolo 1[496]: Lo scopo del programma
- Pensare in Python - Capitolo 2[497]: Variabili, espressioni ed istruzioni
- Nicola Cassetta - Lezione 1[498]
- Nicola Cassetta - Lezione 2[499]

Basi - booleani

- Pensare in Python, Capitolo 5, Istruzioni condizionali e ricorsione[500] , in particolare Sezioni 5.2 e 5.3, Espressioni booleane[501] Puoi saltare la ricorsione
- Nicola Cassetta, Lezione 7, Le istruzioni condizionali: if, else, elif[502]

8.1.2 Stringhe

- Pensare in Python, Capitolo 8, Stringhe[503]
- Pensare in Python, Capitolo 9, Giochi di parole[504]
- Nicola Cassetta, Lezione 2, Stringhe e dati booleani[505]

[495] https://docs.google.com/spreadsheets/d/1w9WzEVnmPs3de4fHsywMdhcVwuNxy5_ol24AimddbC4/edit#gid=0
[496] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2002.html
[497] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2003.html
[498] http://ncassetta.altervista.org/Tutorial_Python/Lezione_01.html
[499] http://ncassetta.altervista.org/Tutorial_Python/Lezione_02.html
[500] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2006.html
[501] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2006.html#sec59
[502] http://ncassetta.altervista.org/Tutorial_Python/Lezione_04.html
[503] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2009.html
[504] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2010.html
[505] http://ncassetta.altervista.org/Tutorial_Python/Lezione_02.html

- Esercizi Zoppetti 04 Tipi di base - Stringhe[506] fino a G. front_back incluso
- extra per chi vuole fare *text mining:*
  - Leggere prima parte su codifica Unicode del capitolo Stringhe dal libro Immersione in Python[507]
  - Guardarsi la libreria NLTK[508]

8.1.3 Liste

- Pensare in Python, Capitolo 10, Liste[509] Esercizi 10.1, 10.2, 10.3, 10.4, 10.5, 10.6, 10.7
- Nicola Cassetta, Lezione 11, Liste[510] In particolare esercizio 11.1 e 11.2
- Capitolo 10 Pensare in Python — Es 10.1, 10.2, 10.3, 10.4, 10.5, 10.6, 10.7
- Lezione 11 Nicola Cassetta — In particolare Esercizio 11.1 e 11.2
- Esercizi Zoppetti 03 Tipi di base - Liste[511] fino a find_indeces incluso

8.1.4 Tuple

- Pensare in Python, Capitolo 12, Tuple[512]
- Nicola Cassetta, Lezione 12 - tuple e range[513]

8.1.5 Dizionari

- Pensare in Python, Capitolo 11, Dizionari[514]

8.1.6 Controllo di flusso - if

- Nicola Cassetta - Lezione 7 - Le istruzioni condizionali[515]

8.1.7 Controllo di flusso - cicli for

- Nicola Cassetta, Lezione 13, il ciclo for[516]

[506] http://www.ifac.cnr.it/~zoppetti/corsopython/
[507] http://gpiancastelli.altervista.org/dip3-it/stringhe.html
[508] https://www.nltk.org/
[509] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2011.html
[510] http://ncassetta.altervista.org/Tutorial_Python/Lezione_11.html
[511] http://www.ifac.cnr.it/~zoppetti/corsopython/
[512] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2013.html
[513] http://ncassetta.altervista.org/Tutorial_Python/Lezione_12.html
[514] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2012.html
[515] http://ncassetta.altervista.org/Tutorial_Python/Lezione_07.html
[516] http://ncassetta.altervista.org/Tutorial_Python/Lezione_13.html

8.1.8 Controllo di flusso - cicli while

- Pensare in Python, Capitolo 7, Iterazione[517]
- Nicola Cassetta, Lezione 8, L'istruzione while[518]
- Nicola Cassetta, Lezione 9, altre istruzioni per il controllo di flusso[519]

8.1.9 Matrici Numpy

- i tutorial Nicola Zoppetti, parte Numpy[520]
- Python Data Science Handbook, parte Numpy (inglese)[521]

8.1.10 Gestione errori ed eccezioni

Nicola Cassetta - 19: La gestione delle eccezioni[522]

8.1.11 Funzioni

- Pensare in Python, Capitolo 3, Funzioni[523]
- Pensare in Python, Capitolo 6, Funzioni produttive[524] puoi fare tutto saltando la parte 6.5 sulla ricorsione. **NOTA**: nel libro viene usato il termine strano 'funzioni produttive' per quelle funzioni che ritornano un valore, ed il termine ancora più strano 'funzioni vuote' per funzioni che non ritornano nulla ma fanno qualche effetto tipo stampa a video: ignora e dimentica questi termini !
- Nicola Cassetta, Lezione 4, Funzioni[525]

8.1.12 Sequenze

- Pensare in Python - Capitolo 19.2[526]

[517] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2008.html
[518] http://ncassetta.altervista.org/Tutorial_Python/Lezione_08.html
[519] http://ncassetta.altervista.org/Tutorial_Python/Lezione_09.html
[520] http://www.ifac.cnr.it/~zoppetti/corsopython/
[521] https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html
[522] http://ncassetta.altervista.org/Tutorial_Python/Lezione_19.html
[523] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2004.html
[524] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2007.html
[525] http://ncassetta.altervista.org/Tutorial_Python/Lezione_04.html
[526] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2020.html#sec227

8.1.13 List Comprehensions

Pensare in Python - Capitolo 19.2[527]

8.1.14 Insiemi

- Pensare in Python, Capitolo 19.5, Ulteriori strumenti - insiemi[528]

8.2 Libro Passo dopo passo impariamo a programmare con Python

Espone argomenti di base di Python in modo molto dettagliato e contiene diversi esercizi facili.

- PDF[529]

di Aurora Martina, Angelo Raffaele Meo, Clotilde Moro, Mario Scovazzi

E' inteso per ragazzini delle medie, ma non fatevi ingannare dalla grafica fumettosa: il materiale per i più giovani tende spesso ad avere il gran pregio di non lasciare nulla al caso.

- Attenzione: è solo per Python 2 ma noi usiamo il 3, quindi se vedi `print "Ciao"` devi scriverlo `print("ciao")` con le tonde
- il libro usa parecchio il comando input per chiedere dati all'utente, noi non l'abbiamo visto perchè in Jupyter tende a non funzionare bene. Conviene provarlo scrivendo script in Spyder, e inserendo le risposte nella tab dell'interprete
- EXTRA: per divertirtsi, in fondo contiene anche tutorial su come creare giochi con pygame

8.3 Lezioni di Nicola Cassetta

Tutoria passo passo, adatti a principianti

- versione online[530]

8.4 Pensare in Python seconda edizione

NOTA: C'è una nuova versione in inglese che si chiama FOPP ed è parecchio migliore, vedi note qui[531]

Molto discorsivo, passo passo, adatto a principianti

- versione online[532]
- zip offline[533]
- PDF[534]

[527] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2020.html#sec227
[528] https://davidleoni.github.io/ThinkPythonItalian/html/thinkpython2020.html#sec230
[529] http://linuxdidattica.org/polito/manuale-python-V2.pdf
[530] http://ncassetta.altervista.org/Tutorial_Python/index.html
[531] https://en.softpython.org/references.html#Foundations-of-Python-Programming
[532] https://davidleoni.github.io/ThinkPythonItalian/html/index.html
[533] https://davidleoni.github.io/ThinkPythonItalian/thinkpython_italian_html.zip
[534] https://github.com/DavidLeoni/ThinkPythonItalian/raw/master/thinkpython_italian.pdf

orig. Think Python, di Allen B. Downey

Licenza: Creative Commons Attribuzione Non Commerciale 3.0[535], come riportato nella pagina della versione originale in inglese[536]. Tradotto: potete farvelo stampare in copisteria, se vi fanno storie su questioni di copyright mostrategli la licenza.

- Liberamente stampabile: sì
- Libro su carta da comprare in italiano: no
- Libro su carta da comprare in inglese: sì

8.5 W3Resources website

(inglese) Contiene parecchi esercizi facili in su basi Python, fateli:

- Basic1[537], Basic2[538], String[539], List[540], Dictionary[541], Tuple[542], Sets[543], Condition Statements and Loops[544], Functions[545], Lambda[546], CSV Read Write[547]

8.6 Corso Python 3 di Nicola Zoppetti

Contiene parecchi esercizi, fateli !

- versione online[548]

8.7 Guida Introduttiva a Python 3 guida ufficiale

Presenta vari argomenti (ma senza esercizi), traduzione di Maurizio Da Lio

- PDF[549]

[535] http://creativecommons.org/licenses/by-nc/3.0/deed.it
[536] http://greenteapress.com/wp/think-python-2e/
[537] https://www.w3resource.com/python-exercises/
[538] https://www.w3resource.com/python-exercises/basic/
[539] https://www.w3resource.com/python-exercises/string/
[540] https://www.w3resource.com/python-exercises/list/
[541] https://www.w3resource.com/python-exercises/dictionary/
[542] https://www.w3resource.com/python-exercises/tuple/
[543] https://www.w3resource.com/python-exercises/sets/
[544] https://www.w3resource.com/python-exercises/python-conditional-statements-and-loop-exercises.php
[545] https://www.w3resource.com/python-exercises/python-functions-exercises.php
[546] https://www.w3resource.com/python-exercises/lambda/index.php
[547] https://www.w3resource.com/python-exercises/csv/index.php
[548] http://www.ifac.cnr.it/~zoppetti/corsopython/
[549] https://fraccaro.org/python/guida_python_3_ita.pdf

8.8 Immersione in Python 3

(orig: Dive into Python 3) Più pratico, contiene tutorial più mirati (es. trattare file XML)

- versione online[550]
- zip offline[551]
- PDF[552] + Tabella dei contenuti generata

Licenza: Creative Commons Attribuzione Condividi allo stesso modo 3.0[553] come riportato in fondo al sito del libro[554]. Tradotto: potete farvelo stampare in copisteria, se vi fanno storie su questioni di copyright mostrategli la licenza.

- Liberamente stampabile: sì
- Libro su carta da comprare in italiano: no
- Libro su carta da comprare in inglese: sì

8.9 Corso Scientific Programming Master Data Science - Trento

(inglese) Qui si può trovare qualche testo d'esame che dovresti essere in grado di fare - sono tipicamente più difficili di quelli presenti in questo libro, ma offrono anche una panoramica su diversi tematiche con dati reali. Il formato sono fogli Jupyter simili a quello che trovi su SoftPython.

- 17/07/2020: NACE codes[555]
- 16/06/2020: Zoom surveillance[556]
- 07/11/2019: Eventi Comune di Trento[557]
- 31/10/2019: Offerte lavoro EURES[558]
- 26/08/2019: Staff Università of Trento[559]
- 02/07/2019: Botteghe storiche[560]
- 10/06/2019: Case ITEA[561] e Qualità dell'aria[562]
- 13/11/2018: Matrici[563] e numeri di telefono[564]

[550] http://gpiancastelli.altervista.org/dip3-it
[551] http://gpiancastelli.altervista.org/dip3-it/d/diveintopython3-it-html-latest.zip
[552] http://gpiancastelli.altervista.org/dip3-it/d/diveintopython3-it-pdf-latest.zip
[553] http://creativecommons.org/licenses/by-sa/3.0/deed.it
[554] http://gpiancastelli.altervista.org/dip3-it/
[555] https://datasciprolab.readthedocs.io/en/latest/exams/2020-07-17/exam-2020-07-17-solution.html#Part-A---NACE-codes
[556] https://datasciprolab.readthedocs.io/en/latest/exams/2020-06-16/exam-2020-06-16-solution.html#Part-A---Zoom-surveillance
[557] https://datasciprolab.readthedocs.io/en/latest/exams/2019-11-07/exam-2019-11-07-solution.html#Part-A
[558] https://datasciprolab.readthedocs.io/en/latest/exams/2019-10-31/exam-2019-10-31-solution.html#Part-A---offerte-lavoro-EURES
[559] https://datasciprolab.readthedocs.io/en/latest/exams/2019-08-26/exam-2019-08-26-solution.html#Part-A---University-of-Trento-staff
[560] https://datasciprolab.readthedocs.io/en/latest/exams/2019-07-02/exam-2019-07-02-solution.html#A1-Botteghe-storiche
[561] https://datasciprolab.readthedocs.io/en/latest/exams/2019-06-10/exam-2019-06-10-solution.html#A1-ITEA-real-estate
[562] https://datasciprolab.readthedocs.io/en/latest/exams/2019-06-10/exam-2019-06-10-solution.html#A2-Air-quality
[563] https://datasciprolab.readthedocs.io/en/latest/exams/2018-11-13/exam-2018-11-13-solution.html#1.-matrices
[564] https://datasciprolab.readthedocs.io/en/latest/exams/2018-11-13/exam-2018-11-13-solution.html#2.-phones

8.10 Geeks for Geeks

(inglese) Contiene molti esercizi - non ha soluzioni nè assert specifici ma se fate il login e spedite soluzioni, il sistema eseguirà dei test sul server e vidarà un repsonso.

- Esempio Filtra difficoltà per school+basic+easy con topic String[565]

Potete selezionare molti più topic cliccando more>> in Topic Tags:

8.11 Introduction to Scientific Programming with Python

(inglese) Introduzione con focus sul calcolo numerico, potete guardare i primi 7 capitoli fino ai dizionari.

Di Joakim Sundnes.

- PDF[566] per Python (solo teoria)
- Esercizi di accompagnamento (tanti!)[567] – utili, anche se alcuni sono troppo ingegneristici per il taglio di questo libro
- EXTRA: se vuoi approfondire, contiene anche capitoli sulle classi che sono certamente utili.

```
[ ]:
```

[565] https://practice.geeksforgeeks.org/explore/?category%5B%5D=Strings&difficulty%5B%5D=-2&difficulty%5B%5D=-1&difficulty%5B%5D=0&page=1
[566] https://link.springer.com/content/pdf/10.1007%2F978-3-030-50356-7.pdf
[567] https://www.uio.no/studier/emner/matnat/ifi/INF1100/h16/ressurser/INF1100_exercises_5th_ed.pdf