

Linguistica Computazionale

Laboratorio in Python - I



Alessio Miaschi

ItaliaNLP Lab, Istituto di Linguistica Computazionale (CNR-ILC), Pisa

alessio.miaschi@ilc.cnr.it

<https://alemmaschi.github.io/>

<http://www.italianlp.it/alessio-miaschi/>

Materiali



Slides e esercizi:

Link:

https://github.com/alemiaschi/Linguistica_Computazionale_24-25

Riferimenti:

- [Learning Python](#) (Mark Lutz, 5th Edition, O'Reilly Media)
- [Natural Language Processing with Python \(Analyzing Text with the Natural Language Toolkit\)](#) (Bird, S. et al, O'Reilly Media)
- [Python ABC](#) (Guida online)

Introduzione

Introduzione

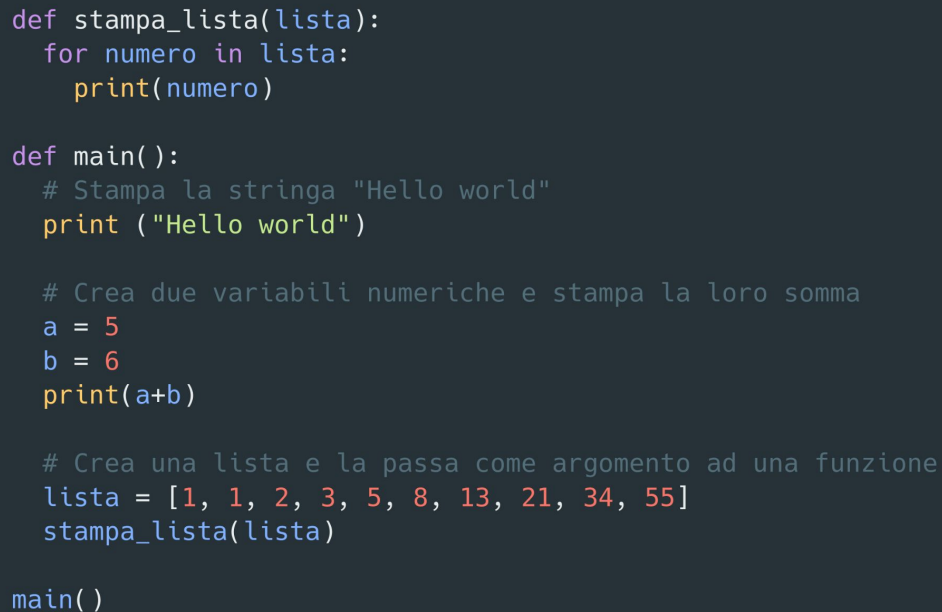
- **Python** (<https://www.python.org/>) è un linguaggio di programmazione sviluppato agli inizi degli anni '90
- È un linguaggio:
 - Interpretato
 - Di alto livello
 - Orientato agli oggetti
 - Estensibile/Integrabile



Applicazioni

- Web development
 - Framework come *Django* e *Flask*
- Data Analysis
 - *Numpy*, *Pandas*
 - Data Visualization: *Matplotlib*, *Seaborn*
- Internet of Things
 - Raspberry Pi
- Web Scraping
 - Es. <https://scrapy.org/>
- Machine Learning
 - *Scikit-learn*, *NLTK*, *Tensorflow*, *Pytorch*
- Game Development
 - *PyGame*

Un programma in python



```
def stampa_lista(lista):  
    for numero in lista:  
        print(numero)  
  
def main():  
    # Stampa la stringa "Hello world"  
    print ("Hello world")  
  
    # Crea due variabili numeriche e stampa la loro somma  
    a = 5  
    b = 6  
    print(a+b)  
  
    # Crea una lista e la passa come argomento ad una funzione  
    lista = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]  
    stampa_lista(lista)  
  
main()
```

Creare ed eseguire un programma python

- Esistono 2 metodi principali per creare ed eseguire un programma in python:
 - scrivendo il programma direttamente nella shell linux, dopo aver eseguito il comando “**python**” (avvio dell’interprete python)

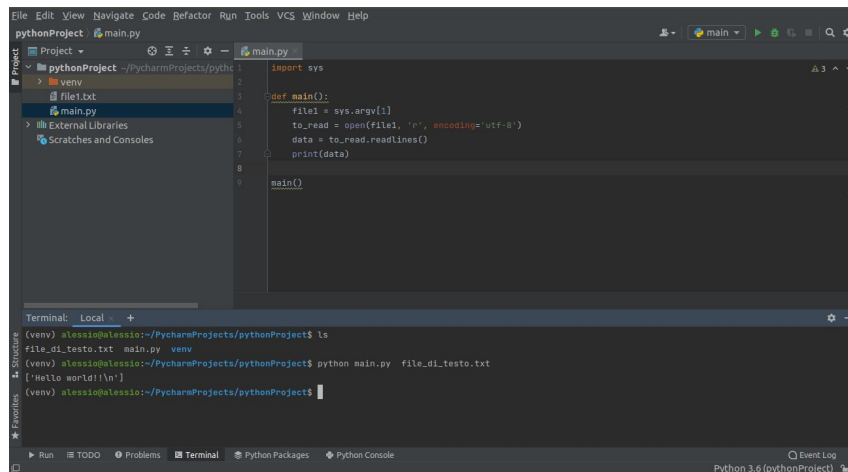
```
alessio@alessio:~$ python
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 5+3
8
>>>
```

- creando (per es. con un editor di testo) un file con estensione **.py** contenente il programma e passandolo come parametro al comando “**python**” nella shell linux

```
alessio@alessio:~$ python programma.py
8
alessio@alessio:~$
```

Creare ed eseguire un programma python

- Ad oggi, sono molto utilizzati anche gli **IDE** (*Integrated Development Environment*): ambienti di sviluppo integrati che supportano i programmatori nello sviluppo e nel debugging di codice
- PyCharm: <https://www.jetbrains.com/pycharm/>



The screenshot displays the PyCharm IDE interface. The main editor window shows a Python script named `main.py` with the following code:

```
import sys

def main():
    file1 = sys.argv[1]
    to_read = open(file1, 'r', encoding='utf-8')
    data = to_read.readlines()
    print(data)

main()
```

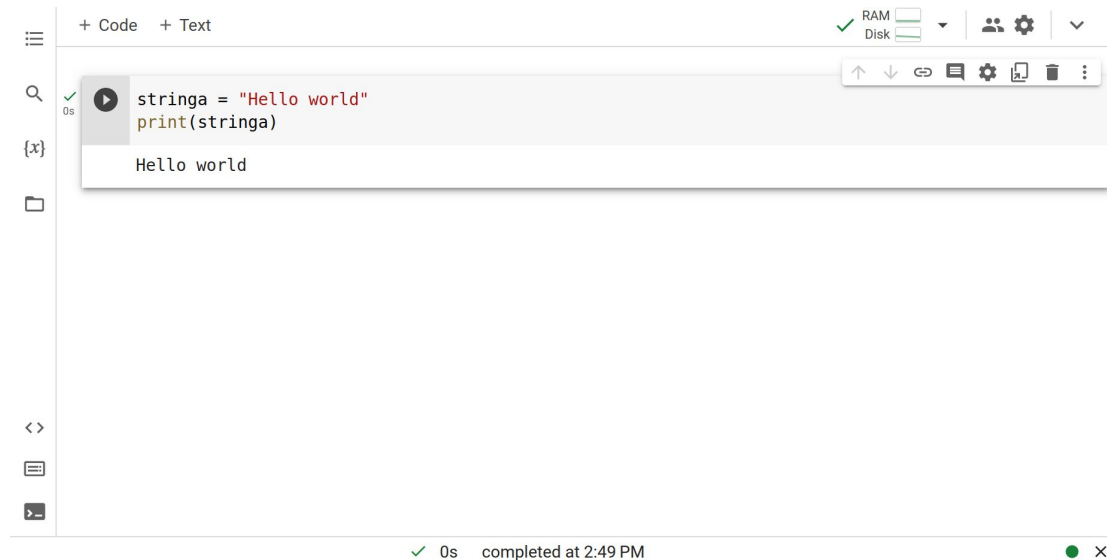
The left sidebar shows the project structure with folders for `venv`, `file1.txt`, and `main.py`. The bottom terminal window shows the execution of the script:

```
(venv) alessio@alessio:~/PycharmProjects/pythonProject$ ls
file_di_testo.txt  main.py  venv
(venv) alessio@alessio:~/PycharmProjects/pythonProject$ python main.py file_di_testo.txt
['Hello world!\n']
(venv) alessio@alessio:~/PycharmProjects/pythonProject$
```

The status bar at the bottom indicates the Python version as `Python 3.6 (pythonProject)`.

Creare ed eseguire un programma python

- Alternative: ambienti online collaborativi, notebooks per sviluppare in python direttamente dal browser
- Google Colab: <https://colab.research.google.com/>




Cosa useremo

- Python 3.6+ → <https://www.python.org/downloads/>
 - !! Per utenti Windows: assicurarsi di spuntare la casella **Add Python 3.x to PATH** al momento dell'installazione !!
- PyCharm (Community version) → <https://www.jetbrains.com/pycharm/download/>
 - Alternativa a PyCharm: **Google Colab** (<https://colab.research.google.com/>)
- Editor di testo: *NotePad++*, *Geany*, *Sublime Text*, *Atom*

Le variabili

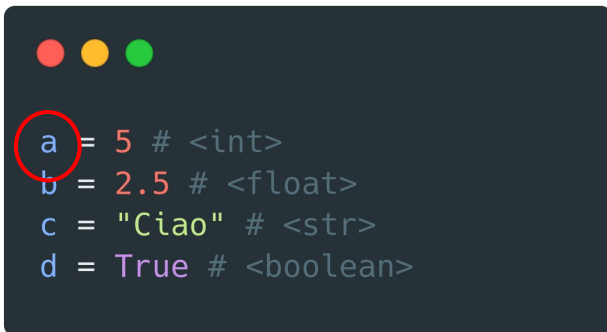
Le variabili



```
a = 5 # <int>
b = 2.5 # <float>
c = "Ciao" # <str>
d = True # <boolean>
```

- Operazione di **assegnamento**
- Ogni variabile ha un nome
- Il nome è composto da:
 - lettere
 - underscore
 - cifre (non può iniziare con una cifra!)

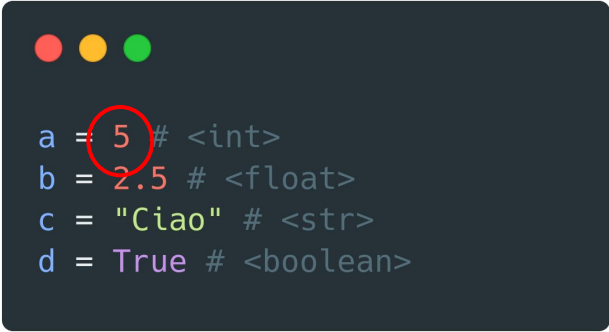
Le variabili



```
a = 5 # <int>
b = 2.5 # <float>
c = "Ciao" # <str>
d = True # <boolean>
```

- Operazione di **assegnamento**
- Ogni variabile ha un nome
- Il nome è composto da:
 - lettere
 - underscore
 - cifre (non può iniziare con una cifra!)

Le variabili



```
a = 5 # <int>  
b = 2.5 # <float>  
c = "Ciao" # <str>  
d = True # <boolean>
```

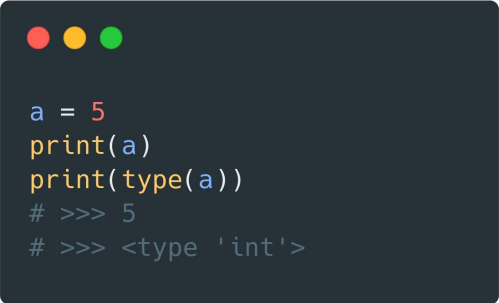
A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains four lines of Python code. The first line is `a = 5 # <int>`, where the number `5` is circled in red. The second line is `b = 2.5 # <float>`. The third line is `c = "Ciao" # <str>`. The fourth line is `d = True # <boolean>`.

- Operazione di **assegnamento**
- Ogni variabile ha un nome
- Il nome è composto da:
 - lettere
 - cifre (non può iniziare con una cifra!)
 - lettere/cifre + underscore

Tipi di variabili

- Tutte le variabili hanno un **tipo** che ne identifica le caratteristiche:
 - *<int>*: numero intero
 - 1, 2, 0, 100, -4
 - *<float>*: numero in virgola mobile
 - 1.5, 2.3, -0.75, 9.999997
 - *<str>*: stringa di testo
 - "Ciao", "Hello world", "15"
 - *<bool>*: variabile booleana
 - True, False

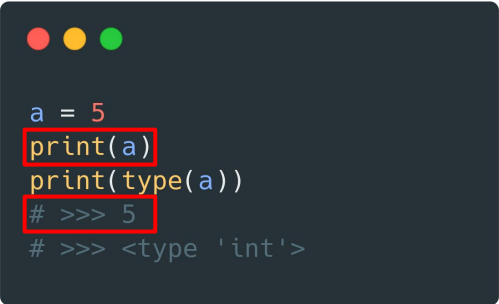
Le variabili



```
a = 5
print(a)
print(type(a))
# >>> 5
# >>> <type 'int'>
```

- Posso accedere al contenuto di una variabile usando l'istruzione **print(<Nome della variabile>)**
- Posso accedere al tipo di una variabile usando l'istruzione **print(type(<Nome della variabile>))**

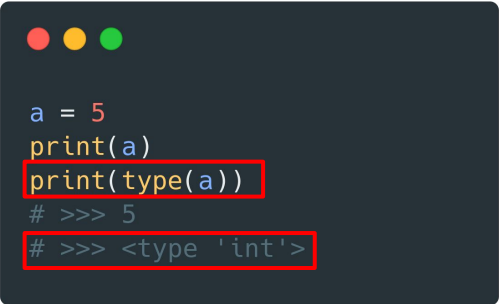
Le variabili

A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. It contains four lines of Python code. The first line is 'a = 5'. The second line is 'print(a)', with 'a' highlighted by a red box. The third line is 'print(type(a))'. The fourth line is '# >>> 5', with '>>>' and '5' highlighted by a red box. The output of the fourth line is '# >>> <type 'int'>'.

```
a = 5
print(a)
print(type(a))
# >>> 5
# >>> <type 'int'>
```

- Posso accedere al contenuto di una variabile usando l'istruzione **print(<Nome della variabile>)**
- Posso accedere al tipo di una variabile usando l'istruzione **print(type(<Nome della variabile>))**

Le variabili

A dark-themed terminal window with three colored window control buttons (red, yellow, green) at the top left. It contains Python code: `a = 5`, `print(a)`, and `print(type(a))`. Below the code, there are two lines of shell output: `# >>> 5` and `# >>> <type 'int'>`. The lines `print(type(a))` and `# >>> <type 'int'>` are highlighted with red rectangular boxes.

```
a = 5
print(a)
print(type(a))
# >>> 5
# >>> <type 'int'>
```

- Posso accedere al contenuto di una variabile usando l'istruzione **`print(<Nome della variabile>)`**
- Posso accedere al tipo di una variabile usando l'istruzione **`print(type(<Nome della variabile>))`**

Operazioni di base

```
a = 10
print(a+1)
# >>> 11

b = 5
print(a-b)
# >>> 5

a = a+1
print(a+b)
# >>> 16

c = 5/2
print(c)
# >>> 2.5
```

Simbolo	Operazione	Esempio	Risultato
+	Addizione	4+3	7
-	Sottrazione	4-3	1
/	Divisione	7/2	3.5
%	Mod (Resto)	7%2	1
*	Moltiplicazione	4*3	12
//	Divisione intera	7//2	3
**	Potenza	7**2	49

Operazioni di base



```
a = "Ciao"  
b = "come va?"  
print(a+b)  
# >>> Ciaocome va?  
  
c = a + " " + b  
print(c)  
# >>> "Ciao come va?"
```

Operazioni di base

- Non è possibile effettuare operazioni tra variabili di tipo diverso:

```
>>> a = 5
>>> b = "ciao"
>>> print(a+b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

- Le variabili devono avere necessariamente lo stesso tipo:

```
>>> a = "5"
>>> b = "ciao"
>>> print(a+b)
5ciao
```

Operazioni di base - Casting

- Il **casting** è l'operazione che restituisce il valore di un'espressione convertito in un altro tipo
- In python è possibile convertire il tipo di una determinata variabile applicando una delle seguenti funzioni:
 - `int()`: converte in tipo `<int>`
 - `float()`: converte in tipo `<float>`
 - `str()`: converte in tipo `<str>`

```
a = 10
b = float(a)
print(b)
# >>> 10.0

num = 18
testo = str(num)
print(testo)
# >>> 18
print(type(testo))
# >>> <type 'str'>
```

Condizioni

```
a = 5
b = 2
print(a > b)
# >>> True
print(a < b)
# >>> False
a = a*2
print(a == 10)
# >>> True
print(a != 10)
# >>> False
```

- Python può valutare anche espressioni logiche, tramite l'utilizzo degli operatori:
 - **>** (*maggiore di*), **<** (*minore di*),
 - **>=** (*maggiore uguale di*), **<=** (*minore uguale di*)
 - **==** (*uguale a*), **!=** (*diverso da*)

Condizioni

```
a = 5
b = 2
print(a > 2 and b < 10)
# >>> True
print(b == 2 or b > 10)
# >>> True
print(not a == 5)
# >>> False

c = not (a >= b and not (b == 3 or b < a))
print(c)
# >>> True
```


- È possibile creare espressioni logiche più complesse usando gli operatori logici:
 - **and**
 - **or**
 - **not**

Condizioni

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Dati in input

- La funzione **input()** viene utilizzata per consentire all'utente di immettere i dati direttamente da tastiera
- **input()** accetta un singolo argomento, che viene accettato una volta premuto il tasto *Invio*



```
nome = input("Inserisci il tuo nome: ")  
# >>> Inserisci il tuo nome: Alessio  
print(nome)  
# >>> Alessio
```

Esercizi

- Chiedere all'utente di inserire due numeri reali (tipo *float*) **x** e **y** e stampare il valore di $(\mathbf{x+y})/(\mathbf{x-y})$
- Chiedere all'utente di inserire un numero intero e verificare se tale numero è pari
- Scrivere un codice in python che calcola il numero delle ore corrispondenti all'età di una persona
- Calcolare il prezzo di un prodotto applicando uno sconto del 25%

Liste, tuple e dizionari

Liste

- Una **lista** è una **sequenza** di elementi
- Permette di raggruppare più variabili (dello stesso tipo) all'interno di un'unica struttura



```
citta = ["Roma", "Milano", "Napoli", "Pisa"] # Lista di stringhe
print(type(citta))
print(citta)
# >>> <class 'list'>
# >>> ['Roma', 'Milano', 'Napoli', 'Pisa']
```

```
numeri = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # Lista di numeri interi
print(numeri)
# >>> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
lista = [] # Lista vuota
```

Liste

- Per selezionare un singolo elemento di una lista, dobbiamo accedere all'**indice** associato a quell'elemento nella lista

```
citta = ["Roma", "Milano", "Napoli", "Pisa"]  
print(citta[2])  
# >>> 'Napoli'
```

- Gli indici di una lista sono numerati a partire da **zero**:

"Roma"	"Milano"	"Napoli"	"Pisa"
0	1	2	3

Liste

- Python rileverà un errore nel caso volessimo accedere ad un elemento non presente all'interno della nostra lista

```
>>> citta = ["Roma", "Milano", "Napoli", "Pisa"]
>>> print(citta[4])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Liste

- È possibile accedere anche a più elementi di una lista (sottoliste) attraverso lo *slicing*

```
citta = ["Roma", "Milano", "Napoli", "Pisa"]
nuova_lista = citta[1:3]
print(nuova_lista)
# >>> ['Milano', 'Napoli']

print(citta[1:])
# >>> ['Milano', 'Napoli', 'Pisa']

print(citta[:3])
# >>> ['Roma', 'Milano', 'Napoli']
```


Liste

- È possibile accedere anche a più elementi di una lista (sottoliste) attraverso lo *slicing*

```
citta = ["Roma", "Milano", "Napoli", "Pisa"]
nuova_lista = citta[1:3]
print(nuova_lista)
# >>> ['Milano', 'Napoli']

print(citta[1:])
# >>> ['Milano', 'Napoli', 'Pisa']

print(citta[:3])
# >>> ['Roma', 'Milano', 'Napoli']
```

Liste

- Le liste sono strutture **mutabili** → è possibile modificarne gli elementi



```
citta = ["Roma", "Milano", "Napoli", "Pisa"]  
citta[3] = "Livorno"  
print(citta)  
# >>> ['Roma', 'Milano', 'Napoli', Livorno]
```

Liste

- Le stringhe, in quanto sequenze di caratteri, possono essere considerate in maniera simile alle liste

```
stringa = "Questa è una stringa"
nuova_stringa = stringa[0:6]
print(nuova_stringa)
# >>> 'Questa'

print(stringa[7:])
# >>> 'è una stringa'

print(stringa[:12])
# >>> 'Questa è una'
```

Liste

- Le liste supportano una serie di **funzioni** e **metodi** particolarmente utili

```
numeri = [1, 2, 5, 9, 6, 5, 10]
print(len(numeri))
# >>> 7

print(max(numeri))
# >>> 10

print(min(numeri))
# >>> 1

print(numeri.index(9))
# >>> 3

print(numeri.count(5))
# >>> 2
```

Liste

- Le liste supportano una serie di **funzioni** e **metodi** particolarmente utili

```
numeri = [1, 2, 5, 9, 6, 5, 10]
print(len(numeri))
# >>> 7

print(max(numeri))
# >>> 10

print(min(numeri))
# >>> 1

print(numeri.index(9))
# >>> 3

print(numeri.count(5))
# >>> 2
```

```
numeri = [1, 2, 5, 9, 6, 5, 10]
numeri.append(9)
print(numeri)
# >>> [1, 2, 5, 9, 6, 5, 10, 9]

numeri.insert(1, 30)
print(numeri)
# >>> [1, 30, 2, 5, 9, 6, 5, 10, 9]

numeri.sort()
# >>> [1, 2, 5, 5, 6, 9, 9, 10, 30]
```

Tuple

- Le tuple sono sequenze di elementi, simili alle liste



```
citta = ("Roma", "Pisa", "Napoli", "Bari")
```

```
print(citta[1])
```

```
# >>> 'Pisa'
```

```
print(citta[1:3])
```

```
# >>> ('Pisa', 'Napoli')
```

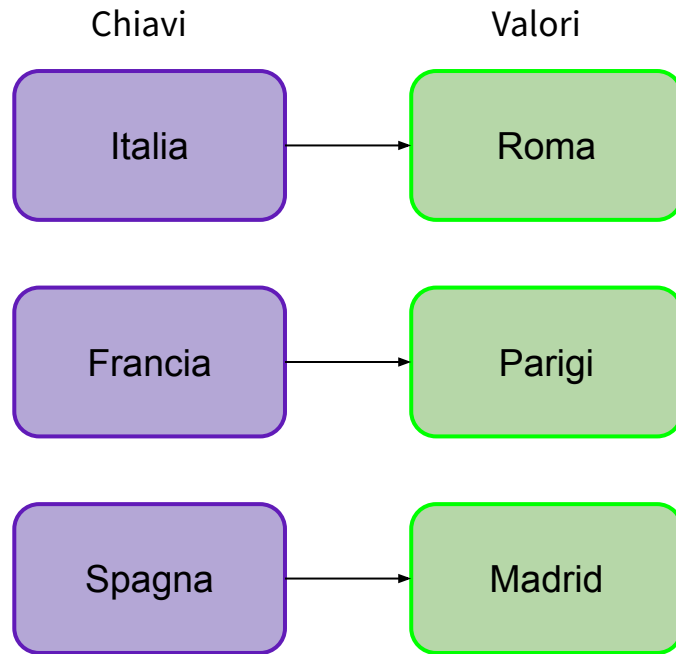
Tuple

- La principale differenza è che sono sequenze **immutabili**: una volta creata una tupla, non sarà più possibile modificarla

```
>>> citta = ("Roma", "Pisa", "Napoli", "Bari")
>>> citta[2] = "Milano"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Dizionari

- I dizionari sono strutture dati che contengono elementi (*items*) formati da coppie $\langle \text{chiave}, \text{valore} \rangle$
- In un dizionario, ogni chiave (univoca) è associata ad un determinato valore (e.g. numero, stringa, lista, ecc)



Dizionari



```
dizionario = {"Italia": "Roma",  
              "Francia": "Parigi",  
              "Spagna": "Madrid"}
```

Dizionari

Chiavi



A diagram illustrating a dictionary in Python. It features a dark blue rounded rectangle with three colored circles (red, yellow, green) in the top-left corner, resembling a window. Inside, the code `dizionario = {"Italia": "Roma", "Francia": "Parigi", "Spagna": "Madrid"}` is displayed. A red arrow points from the word "Chiavi" to a red rounded rectangle that encloses the keys of the dictionary: `"Italia"`, `"Francia"`, and `"Spagna"`.

```
dizionario = {"Italia": "Roma",  
              "Francia": "Parigi",  
              "Spagna": "Madrid"}
```

Dizionari



Dizionari

- Ad ogni chiave possono essere associate diverse tipologie di elementi:
 - <int>, <float>, <str>, liste, dizionari

```
diz = {}

diz1 = {"a": 1, "b": 2, "c": 3, "d": 4}


diz2 = {"Nome": "Alessio",
        "Cognome": "Miaschi"}

diz3 = {"Italia": ["Roma", "Pisa"],
        "Francia": ["Parigi", "Lione"],
        "Germania": ["Berlino", "Monaco", "Amburgo"]}

diz4 = {"Matricola": 40000,
        "Nome": "Paolo",
        "Cognome": "Rossi",
        "Voti": [21, 28, 30, 25, 27, 24]}
```

Dizionari

- È possibile accedere ad ogni valore di un dizionario attraverso la chiave corrispondente:



```
diz1 = {"a": 1, "b": 2, "c": 3, "d": 4}
valore = diz1["b"]
print(valore)
# >>> 2
```

Dizionari

- È possibile accedere ad ogni valore di un dizionario attraverso la chiave corrispondente:



```
diz2 = {"Italia": ["Roma", "Pisa"],  
        "Francia": ["Parigi", "Lione"],  
        "Germania": ["Berlino", "Monaco", "Amburgo"]}  
print(diz2["Francia"])  
# >>> ['Parigi', 'Lione']  
  
print(diz2["Germania"][1])  
# >>> Monaco
```

Dizionari

- Come per le stringhe, python rileverà un errore nel caso volessimo accedere ad una coppia <chiave, valore> non presente all'interno del dizionario

```
>>> diz = {"a": 1, "b": 2, "c": 3, "d": 4}
>>> print(diz["f"])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'f'
```

Dizionari

- È possibile aggiungere nuove coppie *<chiave, valore>*, così come modificare quelle già esistenti

```
diz = {"a": 1, "b": 2, "c": 3, "d": 4}

# Aggiunta nuova coppia <chiave, valore>
diz["e"] = 100
print(diz)
# >>> {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 100}

# Sostituire un valore
diz["b"] = 0
print(diz)
# >>> {'a': 1, 'b': 0, 'c': 3, 'd': 4, 'e': 100}
```


Dizionari

- Come per le liste, i dizionari supportano una serie di **funzioni** e **metodi**:

```
diz = {"a": 1, "b": 2, "c": 3, "d": 4}
```

```
# Dimensione del dizionario
```

```
print(len(diz))
```

```
# >>> 4
```

```
# Ottenere la lista delle chiavi
```

```
print(diz.keys())
```

```
# >>> dict_keys(['a', 'b', 'c', 'd'])
```

```
# Ottenere la lista dei valori
```

```
print(diz.values())
```

```
# >>> dict_values([1, 2, 3, 4])
```

Dizionari

- È possibile verificare se una data chiave è presente all'interno del dizionario



```
diz = {"a": 1, "b": 2, "c": 3, "d": 4}
```

```
print("a" in diz)
```

```
# >>> True
```

```
print("f" in diz)
```

```
# >>> False
```

Esercizi

- Cosa succede se sommo due liste (e.g. `[1, 2, 3]` e `[4, 5, 6]`)?
- Data la lista `a = [1, 2, 3, 4, 5, 6]` cosa si ottiene stampando `a[-1]`?
- Come verificare se un numero è presente fra i valori di un dizionario (e.g. se 2 è presente fra i valori di `diz = {"a": 2, "b": 3}`)?
- Chiedere all'utente di inserire una stringa e crearne una nuova invertendo il primo carattere con l'ultimo