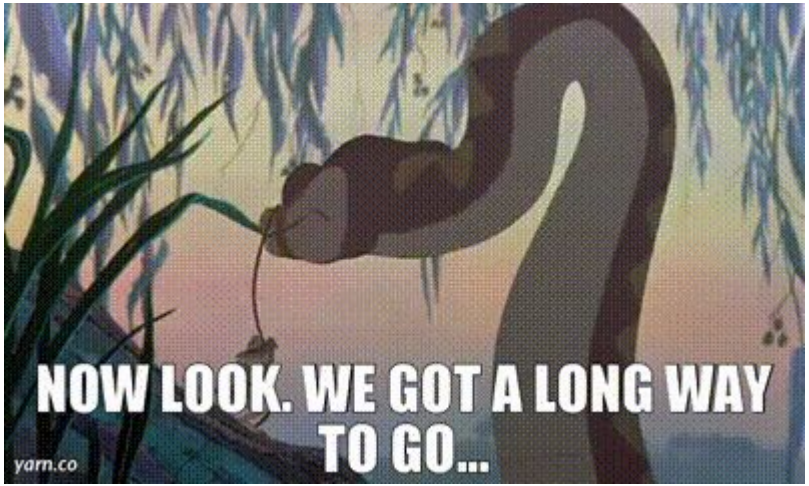
EPICODE

Python
GIORNO 1 - Introduzione, variabili, tipi di dato

Agenda:

- ❑ Algoritmi
- ❑ Python
- ❑ Variabili
- ❑ Data types base e operatori
- ❑ Prime funzioni built-in
- ❑ Collections



Algoritmi

Un algoritmo è una sequenza precisa (comprensibile) di passi elementari che consentono di realizzare un compito, ovvero risolvere un problema.

Ad esempio:

- Istruzioni di montaggio di un mobile
- Prelevamento di denaro da un terminale Bancomat
- Calcolo del massimo comune divisore di due

L'informatica è lo studio sistematico degli algoritmi che descrivono e trasformano l'informazione: la loro teoria, analisi, progetto, efficienza, realizzazione e applicazione.

Algoritmi

Se gli algoritmi servono a risolvere i problemi, da cosa sono composti i problemi?

- Dati: l'istanza del problema che si deve affrontare (input)
- Ciò che si deve determinare: gli elementi incogniti la cui determinazione fornisce il risultato (output)
- La condizione che deve essere soddisfatta dal risultato

Per esempio:

Dati *n* numeri (Dati-Input)

Determinare quel numero *x* (Risultato-Output)

Tale che *x* sia il massimo fra tutti numeri (condizione)

Algoritmi

Dato un problema, un algoritmo è un procedimento risolutivo (almeno uno), cioè un insieme di regole od operazioni (azioni, istruzioni) che eseguite ordinatamente permettono di calcolare i risultati (output) del problema a partire dai dati d'ingresso (input).

Algoritmi

Gli algoritmi devono rispettare tre condizioni fondamentali:

- **Non ambiguo**: le regole devono essere univocamente interpretabili dall'esecutore dell'algoritmo.
- **Eseguibile**: L'esecutore deve essere in grado, con le risorse a sua disposizione, di eseguire ogni singola azione.
- **Finito**: L'esecuzione dell'algoritmo deve terminare in un tempo finito.

Algoritmi

Un algoritmo nella vita quotidiana: preparare la pasta

1. Metti l'acqua (input)
2. Accendi il fuoco
3. Aspetta
4. Se l'acqua non bolle torna a 3
5. Butta la pasta
6. Aspetta un po'
7. Assaggia
8. Se è cruda torna a 6
9. Scola la pasta (output)

Algoritmi

Un altro esempio: il massimo tra *N* numeri

1. Leggi il primo numero
2. Porre il massimo uguale al primo numero
3. Fino a quando i numeri non sono terminati
    1. Leggi un numero
    2. Il numero è maggiore del massimo? Se si scambia il numero con il massimo
4. Visualizza il massimo

Algoritmi

Un algoritmo eseguito da un calcolatore prende il nome di programma e viene espresso in un opportuno linguaggio di programmazione. Ovvero un programma è una rappresentazione dell'algoritmo in un linguaggio comprensibile dal calcolatore.

Gli informatici si servono dei linguaggi di programmazione per tradurre gli algoritmi in programmi, attraverso opportune rappresentazioni delle informazioni note (dati in ingresso) e di quelle da ottenere

Python

Python è un linguaggio di programmazione ad alto livello, interpretato e multiparadigma, noto per la sua sintassi chiara e leggibile, che favorisce la produttività degli sviluppatori in una vasta gamma di applicazioni, dallo sviluppo web all'analisi dei dati e all'intelligenza artificiale.

Python

- **Installare direttamente Python**

Servirà un software per poterlo utilizzare (*i.e.*, un IDE — Integrated Development Environment) oppure dovremo utilizzare direttamente il Terminale e un editor di testo (*e.g.*, Notepad su Windows, TextEdit su Mac).

- **Installare Anaconda**

Anaconda (che è una *distribuzione*) installerà sia Python che tutta una serie di software utili per poterci lavorare agevolmente.

In questo modulo utilizzeremo proprio Anaconda, e in particolare l'IDE JupyterLab in esso contenuto.

Python

Per installare direttamente Python:

- Andiamo su https://python.org e clicchiamo su Downloads
- Subito ci chiederà di scaricare l'ultima versione di Python per il nostro sistema operativo (Windows, MacOS, Linux, eccetera)
- Avviamo l'installer e spuntiamo le opzioni "Install launcher for all users (recommended)" e "Add Python 3.10 to PATH" quindi premiamo su Install Now
- Seguiamo le istruzioni a schermo, senza togliere la spunta a nessuna opzione quando compariranno.

Python

Per installare Anaconda:

- Andiamo su https://anaconda.org e clicchiamo su Download
- Selezioniamo il nostro sistema operativo (Windows, MacOS, Linux, eccetera) e la nostra architettura (*e.g.*, su Mac può essere Intel o M1/M2)
- Avviamo l'installer e spuntiamo le opzioni "Install launcher for all users (recommended)" e "Add Python 3.10 to PATH" quindi premiamo su Install Now
- Seguiamo le istruzioni a schermo, senza togliere la spunta a nessuna opzione quando compariranno.

Python

Per verificare l'installazione:

Apriamo il Terminale / Prompt dei Comandi / PowerShell (a seconda del sistema operativo) ed eseguiamo il seguente comando:

```
> python
```

In output su vedremo qualcosa di simile a questo:

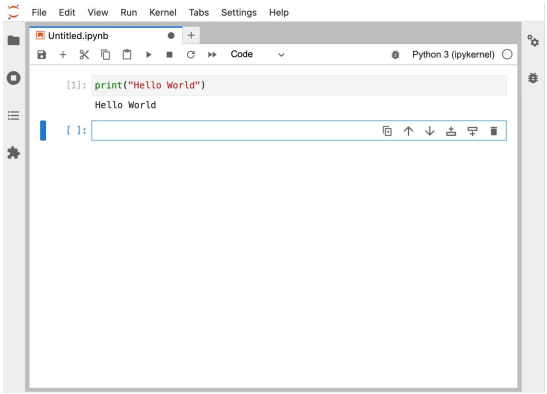
```
Python 3.10.6 (v3.10.3:1a79785e3e, Feb 19 2021, 09:06:10)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more   information.
>>>
```

Python

Utilizzare Python

Un IDE — Integrated Development Environment — è un ambiente di sviluppo avanzato per lo sviluppo di applicazioni in un linguaggio di programmazione. Per lo sviluppo di codice Python utilizzeremo l'IDE JupyterLab, contenuto in Anaconda.

JupyterLab utilizza il browser per funzionare, e quindi ne sfrutta le capacità di visualizzazione.



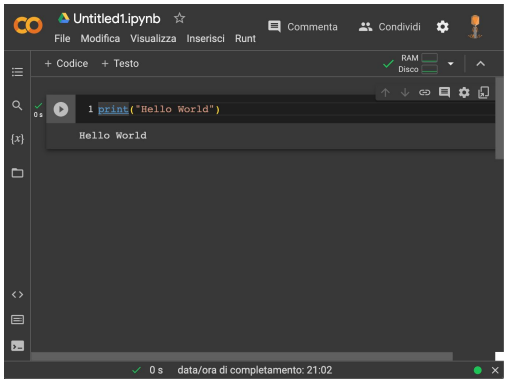
Python

Utilizzare Python

In alternativa si può utilizzare Colaboratory tramite Google Drive.

Pro: non ha bisogno di alcuna installazione e utilizza gli hardware di Google.

Contro: non opera sulla nostra macchina locale, quindi eventuali file vanno caricati e ha delle limitazioni di utilizzo nella versione base gratuita.



Variabili

Una variabile è un contenitore di dati

Variabili

Si utilizza l'operatore = per eseguire l'assegnazione di un valore specifico (potremmo dire il "contenuto") in una variabile (appunto il "contenitore")
La variabile specificata va a sinistra dell'operatore tramite un nome; a destra va il valore da inserire, che, vedremo più avanti, può essere anche un'espressione (il cui risultato verrà calcolato **prima** di eseguire l'assegnazione)

```
[ ]   1 a = 5
      2 name = "Mario Rossi"
      3 number = 4.2
      4 a + number
```

Variabili

Variabili

La nomenclatura delle variabili deve rispettare queste regole:

- Il nome di una variabile deve iniziare con una lettera o con il trattino basso (_)
- Il nome di una variabile non può iniziare con un numero
- Il nome di una variabile può contenere solo caratteri alfanumerici e trattini bassi (A-z, 0-9 e _)
- I nomi delle variabili sono case sensitive (`data`, `Data` e `DATA` sono tre variabili diverse)

In generale, è consigliabile utilizzare nomi parlanti e, se la variabile contiene più parole, separarle con un underscore (ad esempio: `il_mio_numero`)

Data types base

Un data type, o tipo di dato, è una classificazione che specifica quale tipo di valore è memorizzato in una variabile o utilizzato in un'espressione all'interno di un linguaggio di programmazione.

I tipi di dati determinano le operazioni che possono essere eseguite su di essi.

Data types base

I data type fondamentali in Python sono:

- Intero (`int`): Numeri interi senza parte decimale, ad esempio `5`, `-10`, `1000`
- Float (`float`): Numeri in virgola mobile con parte decimale, ad esempio `3.14`, `-0.001`, `2.0`
- Stringa (`str`): Sequenza di caratteri racchiusa tra virgolette singole o doppie, ad esempio `"hello"`, `"world"`, `"123"`
- Booleano (`bool`): Tipo di dato che rappresenta i valori di verità `True` (vero) e `False` (falso)
- Dato nullo (`None`): Tipo di dato che rappresenta un dato mancante

Operatori

Operatori per `int` e `float`:

- `+`: Addizione
- `-`: Sottrazione
- `*`: Moltiplicazione
- `/`: Divisione
- `//`: Divisione intera (arrotonda al più grande intero non maggiore del risultato)
- `%` : Resto della divisione intera (modulo)
- `**`: Esponenziazione (elevamento a potenza)

Operatori

Operatori di assegnazione per `int` e `float`:

- `+=`
- `-=`
- `*=`
- `/=`

Utilizzando questi operatori è possibile assegnare direttamente ad una variabile il risultato di un'operazione, ad esempio:

```
[ ]  1 a = 5
     2 a += 1
```

è uguale a

```
[ ]  1 a = 5
     2 a = a + 1
```

Operatori

Operatori per stringhe (`str`):

- `+`: Concatenazione delle stringhe (unisce due stringhe)
- `*`: Ripetizione delle stringhe (ripete una stringa per un determinato numero di volte)

Funzioni built-in base

Una funzione è un blocco di codice riutilizzabile che esegue un'operazione specifica quando chiamato, spesso con parametri che possono essere passati per modificare il comportamento della funzione, e può restituire un valore risultante.

Funzioni built-in base

- Le funzioni trasformano uno o più input in uno o più output mediante un blocco di codice (generalmente non visibile)
- Il nome della funzione esprime in forma sintetica ciò che la funzione "fa", cioè le operazioni effettuate sull'input per ottenere l'output, e in generale il suo scopo
- La funzione è un programma, e quindi contiene del codice che effettua le trasformazioni desiderate
- In generale non ci interesserà sapere qual è il codice, ma solo cosa qual è il suo scopo e come va usata

Funzioni built-in base

- Per "attivare" la funzione (cioè ottenere un output) è necessario usare le parentesi tonde; al loro interno andranno gli input (detti argomenti o parametri), separati da virgole
- Gli input possono essere uno, o più di uno; quasi sempre in numero definito; alcune volte in numero indefinito, e talvolta (quando l'output è implicito nell'obiettivo della funzione) nessun input (ma le parentesi vanno sempre messe anche in questo caso)
- L'output può essere uno, più di uno, o nessuno, o può essere particolare (ad esempio stampa a video, inserimento in un database, invio di una richiesta dati sul web, eccetera)

Funzioni built-in base

Se i tipi di dati determinano le operazioni che possono essere eseguite su di essi, come posso sapere che tipo di dato è contenuto in una variabile?

Attraverso la funzione `type()` avremo come valore di ritorno il datatype di ciò che inseriamo tra le parentesi tonde.

- Input: il valore o la variabile di cui vogliamo conoscere il datatype
- Output: la tipologia del dato passato come input

Proviamo:

```
[ ]  1  a = True
     2  name = "Mario Rossi"
     3
     4  type(5)
     5  type("ciao")
     6  type(4.2)
     7  type(a)
     8  type(name)
```

Funzioni built-in base

Altra funzione base è `print()`, che serve a visualizzare su console il contenuto della variabile specificata come parametro di input

- Input: ciò che si vuole stampare (valore o variabile)
- Output: stampa (in generale a schermo) dell'input

Proviamo:

```
1 a = 5
2 name = "Mario Rossi"
3 print(a)
4 print(name)
5 print(a, name)
```

Funzioni built-in base

Per avere la documentazione di una funzione, possiamo usare la funzione `help()`

- Input: funzione di cui si vuole avere la documentazione (senza le parentesi!)
- Output: la documentazione richiesta

```
1 help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Collections

Le collections sono strutture dati in un linguaggio di programmazione che consentono di organizzare e manipolare gruppi di elementi in modo efficiente.

Le principali collections disponibili in Python sono:

- Liste
- Tuple
- Set
- Dizionari

Liste

Una lista è una struttura dati ordinata e modificabile. Le liste consentono l'accesso agli elementi tramite un indice numerico, e supportano l'inserimento, la rimozione, la concatenazione e l'iterazione.

Per definire una lista utilizziamo le parentesi quadre e separiamo i singoli elementi con una virgola, per esempio:

```
[ ]   1 my_int_list = [1, 2, 3, 4]
      2 my_string_list = ["oh", "yeah", "I", "am", "learning"]
      3 my_mixed_list = ["all", 1, "kind", 3.5, "of_elements", True]
```

Liste

Possiamo inserire in una lista anche delle variabili, altre liste, altre collections: qualsiasi oggetto può essere inserito in una lista. Questa è una caratteristica distintiva di Python che non è condivisa con gli altri linguaggi di programmazione: la sua dinamicità e l'interpretabilità permettono una flessibilità maggiore.

```
1 my_first_variable = "variabile"
2 my_second_variable = 2
3 my_list = [my_first_variable, my_second_variable, ["una",  "stringa"], 123, True]
4 print(my_list)
```

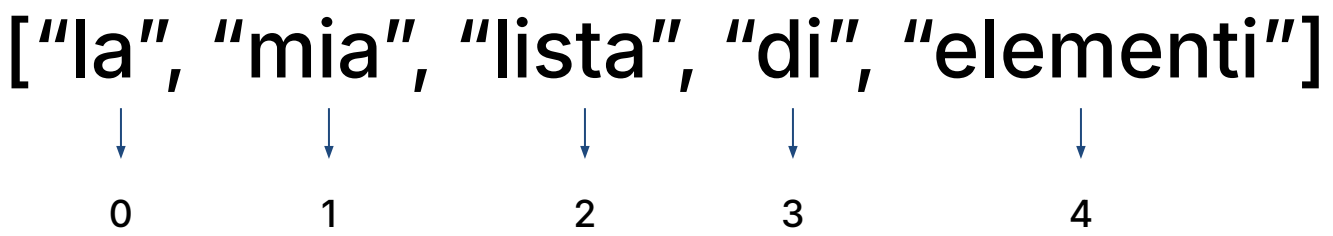
```
['variabile', 2, ['una', 'stringa'], 123, True]
```

Liste — indexing

Le liste in Python sono collections ordinate, questo significa che gli elementi al loro interno mantengono l'ordine con cui li abbiamo definiti.

Questo permette di accedere ai singoli elementi presenti nella lista andando a riferirci al loro index (*indice*),o alla loro posizione.

Gli index sono 0-based: significa che il conteggio parte da 0, non da 1.



Liste — indexing

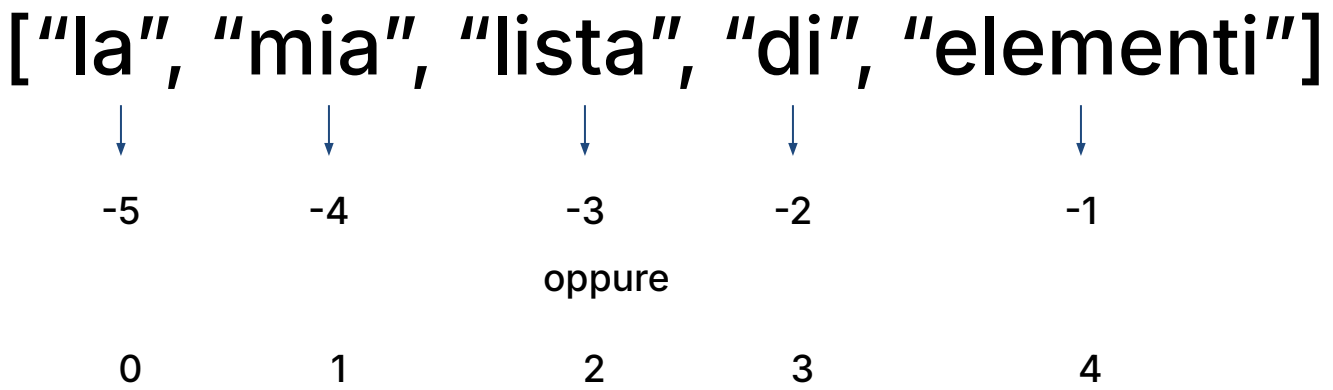
Per accedere ad un singolo elemento inseriamo il suo indice tra parentesi quadre dopo la lista, per esempio:

```
1 my_list = ["la", "mia", "lista", "di", "elementi"]
2 print(my_list[0])
```

```
la
```

Liste — indexing

Possiamo indicizzare una lista anche partendo dal fondo, utilizzando degli indici negativi. In questo caso partiamo da -1 e proseguiamo da destra a sinistra:



Liste — indexing

Essendo le liste mutabili, posso usare gli index anche per cambiare il valore di uno specifico elemento ad una specifica posizione,

per esempio:

```
1 my_list = ["la", "mia", "lista", "di", "elementi"]
2 print(my_list)
3 my_list[2] = "listona"
4 print(my_list)
```

```
['la', 'mia', 'lista', 'di', 'elementi']
['la', 'mia', 'listona', 'di', 'elementi']
```

Liste — slicing

Possiamo accedere ad un subset di elementi tramite lo slicing. Sempre tra parentesi quadre, sempre dopo la lista, possiamo inserire

Liste — slicing

Il valore di start e di stop mi indica quali quale slice, o “fetta”, della lista voglio estrarre,

per esempio:

```
1 my_list = ["una", "lista", "ordinata", "di", "elementi"]
2 print(my_list[1:3])
```

```
['lista', 'ordinata']
```

Il valore di step può non essere specificato e viene settato a 1 di default, ma se lo specifichiamo va a definire ogni quanti elementi estrarre quello all’index desiderato (quanto è lungo il “passo” che facciamo mentre percorriamo la nostra lista),

per esempio:

```
1 my_list = ["mettiamo", "tanti", "elementi", "in", "questa", "lista", "di", "elementi"]
2 print(my_list[1:7:2])
```

```
['tanti', 'in', 'lista']
```

Liste — slicing

Abbiamo notato che lo slicing ha un effetto strano quando utilizziamo l'operatore due punti: sembra escludere l'ultimo elemento. Questo accade perché gli indici che inseriamo in realtà si riferiscono ai tagli tra gli elementi, da cui appunto il termine "slicing" (affettare).

Quando recuperiamo un singolo elemento effettuiamo un **indexing**, e otteniamo **solo l'elemento**; invece quando recuperiamo più elementi usiamo lo **slicing**, e otteniamo **una lista**

Liste — slicing

Oltre a poter indicizzare i singoli elementi tramite indici negativi, possiamo anche usarli per fare slicing,

per esempio:

```
1 my_list = ["la", "mia", "lista", "di", "elementi"]
2 print(my_list[-5:-2])
```

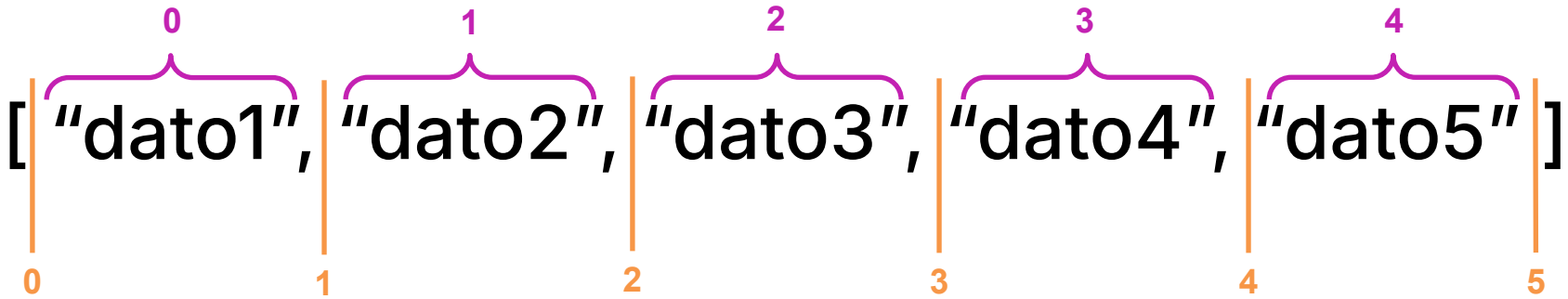
```
['la', 'mia', 'lista']
```

Liste — indexing VS slicing

Ricapitolando:

- Se vogliamo accedere ad un **singolo elemento** usiamo l'indexing, e quindi la sua posizione (l'indice, appunto)
- Se vogliamo accedere ad un **sottoinsieme di elementi** usiamo lo slicing, che numera le posizioni **tra** gli elementi

INDEXING



SLICING

Liste — metodi

Un **metodo** è una **funzione interna ad un oggetto** Python, e serve per manipolare quello stesso oggetto.

Possiamo quindi manipolare le liste in tanti altri modi, utilizzando i loro metodi. I principali sono:

- `list.append(elemento)` : Aggiunge un elemento alla fine della lista.
- `list.extend(iterabile)` : Estende la lista aggiungendo gli elementi dell'iterabile dato.
- `list.insert(indice, elemento)` : Inserisce un elemento in una posizione specifica nella lista.
- `list.remove(elemento)` : Rimuove la prima occorrenza dell'elemento dalla lista.
- `list.index(elemento[, start[, stop]])` : Restituisce l'indice della prima occorrenza dell'elemento nella lista.
- `list.count(elemento)` : Restituisce il numero di occorrenze dell'elemento nella lista.
- `list.sort(key=None, reverse=False)` : Ordina la lista in loco in ordine crescente (o decrescente se `reverse=True`).
- `list.reverse()` : Inverte l'ordine degli elementi nella lista.

Liste — metodi

Per usare i metodi si usa il nome della variabile che contiene l'oggetto, poi l'operatore punto, che è un semplice puntino: `.`, e infine il nome del metodo desiderato. Facciamo alcuni esempi:

```
prezzi = [10, 20, 30]
print(prezzi)
>>> [10, 20, 30]
prezzi.append(150)
print(prezzi)
>>> [10, 20, 30, 150]
i = prezzi.index(30)
print(i)
>>> 2
print( prezzi[i] )
>>> 30
prezzi.remove(20)
print(prezzi)
>>> [10, 30, 150]
```

Liste — metodi

Esempi:

```
print(prezzi)
>>> [10, 30, 150]
altri_prezzi = [5, 200, 9, 10, 275, 1]
prezzi.extend(altri_prezzi)
print(prezzi)
>>> [10, 30, 150, 5, 200, 9, 10, 275, 1]
prezzi.sort()
print(prezzi)
>>> [1, 5, 9, 10, 10, 30, 150, 200, 275]
print( prezzi.count(275) )
>>> 1
print( prezzi.count(10) )
>>> 2
print( prezzi.count(500) )
>>> 0
```

Stringhe

Le stringhe sono delle sequenze ordinate di caratteri.
Sono immutabili, cioè non è possibile modificarne gli elementi.

Le stringhe non sono delle collection, perché contengono solo elementi di un certo tipo. Sono però delle sequenze degli stessi, e quindi possiamo accedere ai singoli elementi tramite **indexing** o **slicing** (in realtà indexing e slicing si possono usare anche su qualunque collection):

```
scuola = "Epicode"
print( scuola[0] )
>>> E
print( scuola[-2] )
>>> d
print( scuola[1:4] )
>>> pic
```

Stringhe

Le stringhe si possono definire tramite apici doppi " o singoli '
Stringhe multiriga possono essere definite mediante tre apici doppi """ o tre singoli '''

```
tipologia_prodotto = "jeans"
print(tipologia_prodotto)
>>> jeans

codice_documento = 'SCP-173'
print(codice_documento)
>>> SCP-173

messaggio = """Manutenzione programmata
Per favore, riavviare il server"""
print(messaggio)
>>> Manutenzione programmata
>>> Per favore, riavviare il server
```

Stringhe — metodi

Anche le stringhe hanno i propri metodi.

I più utilizzati sono:

- `str.upper()`: Restituisce una copia della stringa con tutti i caratteri convertiti in maiuscolo
- `str.lower()`: Restituisce una copia della stringa con tutti i caratteri convertiti in minuscolo
- `str.strip()`: Restituisce una copia della stringa senza spazi vuoti all'inizio e alla fine
- `str.replace(old, new)`: Restituisce una copia della stringa con tutte le occorrenze della sottostringa `old` sostituite con la sottostringa `new`
- `str.split(separator)`: Restituisce una **lista** di sottostringhe divise utilizzando il separatore specificato; se non se ne specifica uno, di default utilizza lo spazio

Stringhe — metodi

Esempi:

```
libro = "Il Signore degli Anelli"
print(libro)
>>> Il Signore degli Anelli
print( libro.upper() )
>>> IL SIGNORE DEGLI ANELLI
print(libro)
>>> Il Signore degli Anelli
```

Le stringhe sono immutabili, quindi per conservare in memoria la modifica dobbiamo utilizzare un'altra variabile, oppure sovrascrivere:

```
libro_maiusc = libro.upper()
print(libro_maiusc)
>>> IL SIGNORE DEGLI ANELLI
```

Stringhe — metodi

```
libro = "Il Signore degli Anelli"
print(libro)
>>> Il Signore degli Anelli

print( libro.lower() )
>>> il signore degli anelli

nuovo_titolo = libro.replace("degli Anelli", "dei Fornelli")
print(nuovo_titolo)
>>> Il Signore dei Fornelli

parole = libro.split(" ")
print(parole)
>>> ["Il", "Signore", "degli", "Anelli"]

print( parole[2] )
>>> degli
```

Funzione built-in `len()`

Per conoscere il numero di elementi di una collection (ma anche delle stringhe) possiamo usare la funzione built-in `len()`

```
titolo = "Il Signore degli Anelli"
numero_caratteri = len(titolo)
print(numero_caratteri)
>>> 23

scuola = 'Epicode'
print( len(scuola) )
>>> 7

introiti = [10.50, 24.99, 9.99, 22.45, 14.99]
print( len(introiti) )
>>> 5
```

Stringhe — metodi

Esistono molte altre funzioni built-in e metodi per stringhe e liste. Possiamo indagare e scoprirle a seconda della necessità del momento utilizzando la documentazione di Python.

Le documentazioni possono sembrare complesse, ma sono la risorsa fondamentale a cui riferirci ogni qual volta abbiamo bisogno di approfondire un concetto, trovare una funzione o anche soltanto quando ci siamo dimenticati la sintassi di un comando specifico.

Cerchiamo di prenderci confidenza fin da subito e saremo inarrestabili.

EPICODE

GRAZIE

Epicode