Università degli Studi di Bologna

Area Medica

Insegnamento: INFORMATICA

Lezione: 03bis

Python

Docente: Ciro Polizzi - e-mail: ciro.polizzi@unibo.it

A.A 2022 - 2023

Linguaggio di programmazione

- La sintassi di un linguaggio di programmazione è una raccolta di regole per specificare la struttura o la forma del codice.

- La semantica si riferisce all'interpretazione del codice o al significato associato dei simboli, dei caratteri o di qualsiasi parte di un programma.

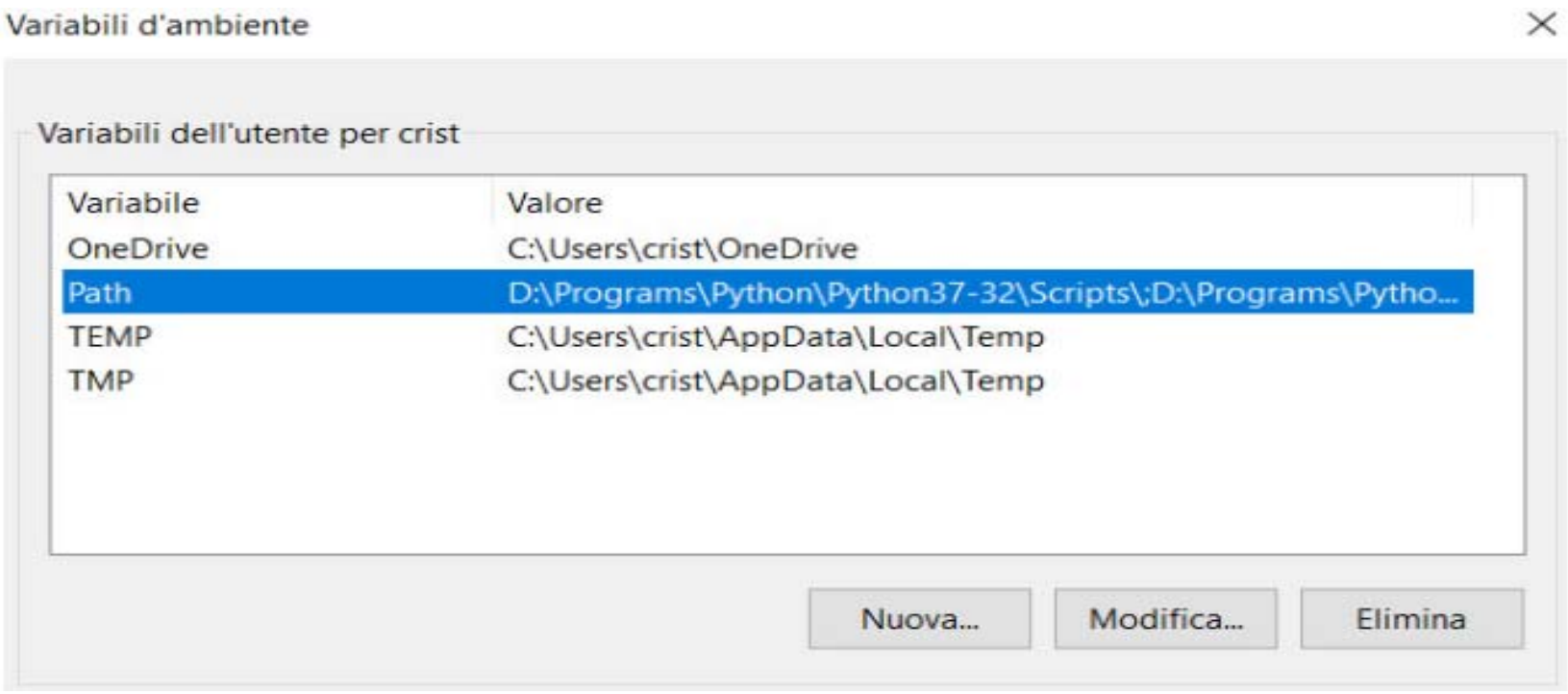
Installare Python

Scaricare il file di installazione: https://www.python.org/downloads/

Durante l'installazione occorre selezionare "Add Python X.X to Path

Se non non è stato fatto occorre manualmente inserire tra le varabili utente il Path di Python.

Installare Python



pannello di controllo -> sistema-> impostazioni di sistema avanzate -> variabile d'ambiente

Python

Python è un linguaggio di programmazione "*interpretato*" ma è possibile anche compilare il codice sorgente Python e trasformarlo in bytecode.

Python

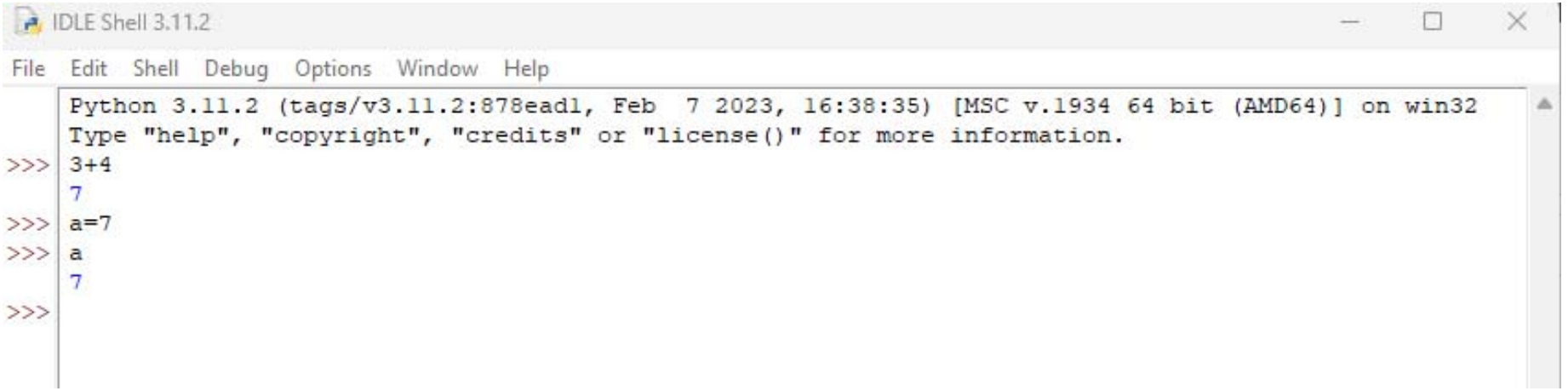
- **Ha una sintassi semplice.**
- **Implementa già molte strutture**
- **È una programmazione orientata agli oggetti.**
- **Riusabilita'**
- **Ha tante funzioni disponibili.**
- **Permette una facile integrazione con altri linguaggi.**

Python

Interfacce grafiche

Python si può usare in modo interattivo o mettere istruzioni pyhton in file ed eseguirlo.

Idle è un'interfaccia grafica che rende disponibile una shell per eseguire programmi o singoli comandi, ed un editor per scrivere file con programmi Python.



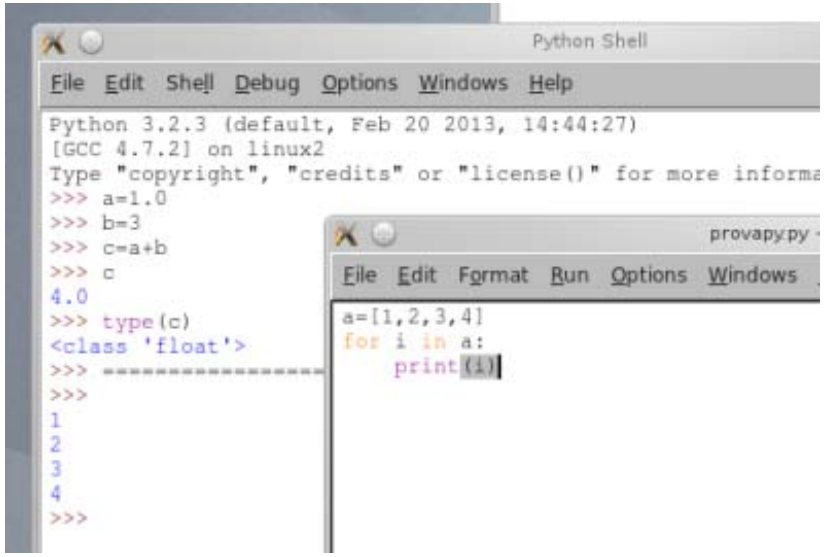
Un file con un programma Python: file.py si esegue con:

>>> Python file.py

Python

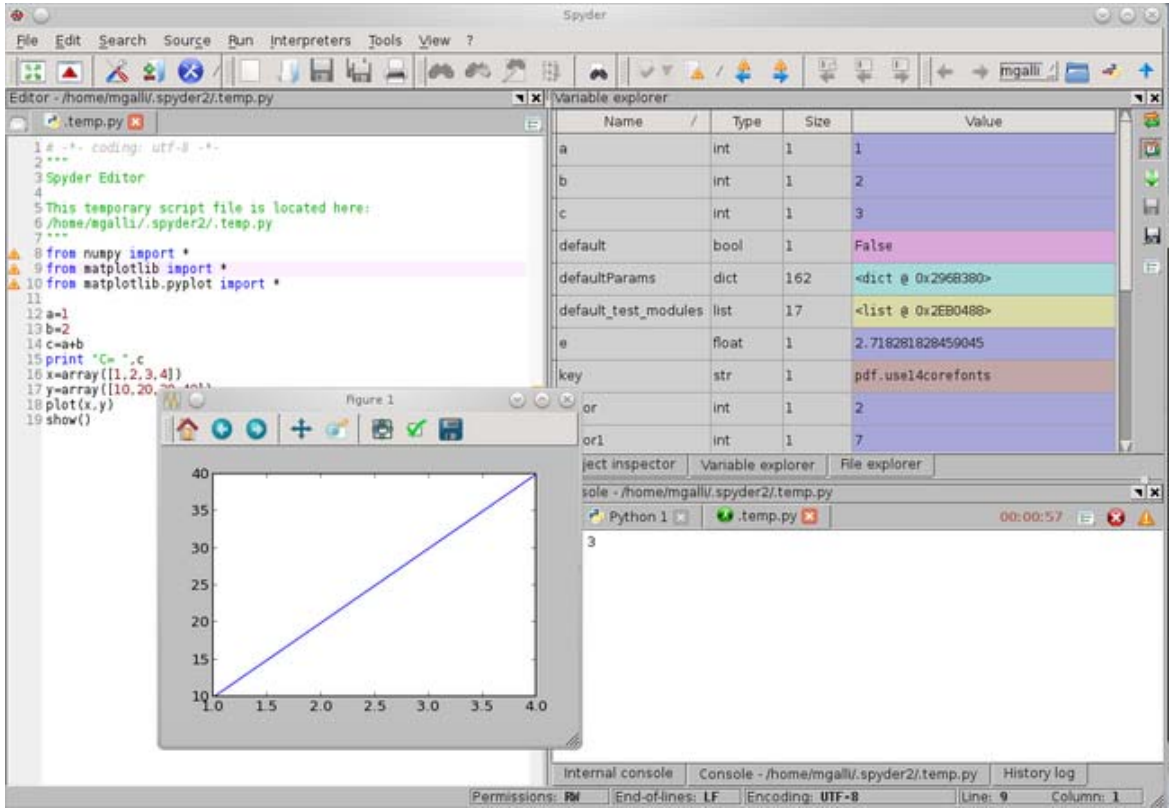
Interfacce grafiche

"Idle" é il tools grafico più "user friendly" ma ne esistono tanti altri ad esempio:"spyder". Idle é un'interfaccia grafica che rende disponibile una shell (interfaccia utente) per eseguire programmi o singoli comandi, ed un editor per scrivere file con programmi Python.



Python

Spyder: è un sistema di sviluppo grafico (vedi spyder-ide.org) per l'analisi di dati scientifici, con uso di moduli come munpy, matplotlib o scipy. La figura mostra l'uso di Spyder per un semplice grafico.



Python

Altre interfacce dedicate alla programmazione con Python, commerciali e gratuite;

- "Ipython" e' una shell Python con diverse estensioni per l'interazione con il sistema operativo;
- "Ipython notebook" attiva un web server locale e permette di utilizzare l'interfacia del browser come shell grafica per l'ambiente Python;
- "PyDev", un addon per il sistema di sviluppo "Eclipse", vedi: pydev.org
- Altre sono "Eric", "*Pycharm*" "*Komodo IDE*" gran parte degli editor per la programmazione supportano il linguaggio.
- *Ambiente integrato: Anaconda - Jupiter lab Jupiter Notebook*

L'ambiente di programmazione IDLE

- Interprete (shell): è la finestra che si apre all'avvio dell'ambiente di programmazione, e consente la valutazione di espressioni e l'esecuzione di istruzioni in modalità interattiva

- Editor di testi (una o più finestre che si possono aprire attraverso la voce di menu File > New window): consente la scrittura in un file di testo di un programma, e successivamente di eseguirlo

Eseguire un programma

I programmi Python sono sequenze d'istruzioni, e per essere eseguiti devono essere memorizzati prima in un file di testo.
Per la scrittura di programmi si consiglia l'uso dell'editor di testi dell'ambiente di programmazione.
I file Python hanno estensione *.py*
Nell'ambiente IDLE L'esecuzione viene avviata selezionando dalla finestra dell'editor la voce di menu *Run* > *Run module*, o in alternativa premendo il **tasto F5**

Tipi di variabili

Che cosa è una variabile?
È un'area di memoria della RAM che viene destinata a contenere un dato e la quantità di memoria allocata è proporzionale proprio al tipo di dato.
Una variabile differisce da una costante (proprio perché durante l'esecuzione del programma essa può essere modificata.
In Python non esiste a differenza di altri linguaggi non è possibile definire una variabile come "costante"

In una **tipizzazione dinamica** inoltre la quantità di memoria assegnata ad una variabile può cambiare dinamicamente anche in funzione del suo contenuto.

Assegnazione variabili e tipizzazione

Nella maggior parte dei linguaggi di programmazione, prima di assegnare un valore ad una variabile occorre definire esplicitamente il tipo (Es: intero, stringa, carattere ..).
Questo modalità di tipizzazione viene chiamata "statica" ed avviene con una dichiarazione esplicita della variabile prima ancora del suo utilizzo (int a, float b, ecc.) o comunque congiuntamente all'*instanziazione* della variabile (int a=2).

In Python le variabili vengono tipizzate dinamicamente ovvero quando si istanzia una variabile è il valore stesso che le viene assegnato a definirne implicitamente il tipo.
Ad esempio: a=2 ed essendo "2" viene sottinteso che la sua tipizzazione è "int" questo è l'equivalente in forma esplicita int a=2
Ad esempio: a=2.5 in forma esplicita equivale a: float a=2.5

Nota Bene: nei linguaggi di programmazione viene usato il PUNTO in luogo della VIRGOLA

Tipi di variabili

I tipi di variabili ammesse in Python sono le seguenti:

int = interi	Es: a=2
float = virgola mobile (decimali)	Es: b=2.5
complex = numeri complessi	Es: z=2+3j
bool = booleani	Es: c=True oppure c=False
string = stringa	Es: d="prova" oppure e='prova'

I Tipi

int	long	float	complex
bool	none	str	bytes
list	tuple	dict	set
range			

Variabile set

city = set(["Milano", "Roma", "Torino"])
Poi assegno tre nomi di capitali europee alla variabile capitali
capitali = set(["Roma", "Parigi", "Londra"])

Operazione	Operatore	Esempio
Unione (OR)	|	city | capitali
Intersezione (AND)	&	city & capitali
Differenza Interferenza asimmetrica	-	city - capitali
Interferenza simmetrica (XOR)	^	city ^ capitali

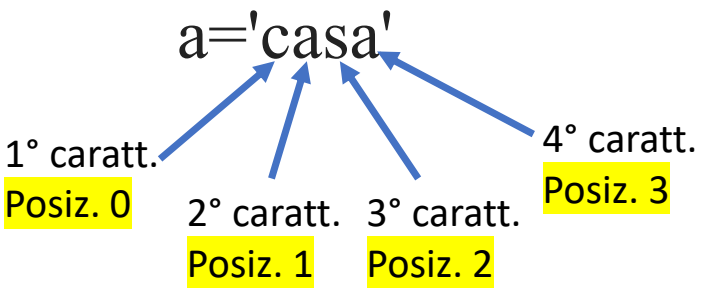
print (city | capitali)
risultato:
{'Torino', 'Milano', 'Parigi', 'Londra', 'Roma'}

Operazioni su stringhe

```
>>> s=‘Genetica’
>>> s[0:2]  # sottostringa con elementi da 0 (incluso) a 2 (escluso)
'Ge'
>>> s[:2]   # dall'inizio all'elemento con indice 2 (escluso)
'Ge'
>>> s[3:5]  # dall'elemento con indice 3 (incluso) a 5 (escluso)
‘ne'
>>> s[4:]   # dall'elemento con indice 4 (incluso) alla fine
‘tica'
>>> s[-2:]  # dall'elemento con indice -2 (incluso) alla fine
‘ca‘
```

stringa

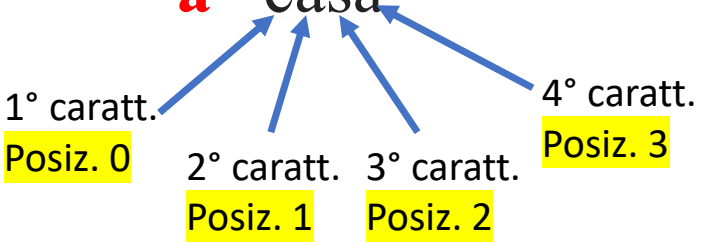
Una **stringa** è una sequenza di caratteri con un ordine prestabilito.
Un **carattere** è un'unità minima d'informazione corrisponde a un grafema (o a un simbolo) della forma scritta di una lingua naturale (a, b, c, 1, 2, >, & ….)
Un **carattere speciale** appartiene ad un sottoinsieme dell'insieme dei caratteri e che in un determinato ambiente (programma), svolge delle "funzioni" particolari.
alcuni sono = ; : > < " & \ [ ]



N:B. La posizione dei caratteri inizia con ZERO

stringa

a='casa'



len(a) --> 4 (numero dei caratteri inclusi gli spazi)
a[0] --> 'c' (visualizza il primo carattere della stringa)
a[1] --> 'a' (visualizza il secondo carattere della stringa)
a[2] --> 's' (visualizza il terzo carattere della stringa)
a[3] --> 'a' (visualizza l'ultimo carattere della stringa)

Es. di concatenazione di stringhe
a='casa' e b=' mia'; a+b --> 'casa mia'

*N.B. Gli elementi di una stringa sono **i caratteri***

Operazioni su stringhe

```
>>> s = ‘Genetica'
>>> ‘G' in s            # controlla se il carattere ‘G' è contenuto nella stringa s
True
>>> 'x' in s            # il carattere 'x' non è in s, quindi ritorna False
False
>>> 'x' not in s        # "not in" esegue l'operazione inversa
True
>>> ‘Ge' in s           # controlla se la sottostringa ‘Ge' è contenuto nella stringa s
True
>>> ‘ge' in s           # il controllo è case-sensitive, quindi ritorna False
False
```

Concatena, len, repeat

```
>>> ‘Ge' + ‘netica'
‘Genetica‘
>>> ‘Ge' * 2
‘GeGe‘
>>> 'Ba' + 'na' * 2
'Banana‘
>>> len(‘Genetica')
8
>>> len(s)
8
```

Metodi: metodi di str

```
>>> s = ‘Genetica'
>>> s.upper()  # il metodo upper ritorna una nuova stringa tutta uppercase
‘GENETICA'
>>> s.lower()  # il metodo lower ritorna una nuova stringa tutta lowercase
‘genetica'
```

Visualizzare l’help delle funzioni

```
>>> len                    # len è una funzione built-in

>>> help(len)              # con help()  breve spiegazione
```

«*dir*» per visualizzare i metodi di un oggetto

```
>>> dir(str)  # restituisce una lista di metodi e attributi di str
[..., 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs',
 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',
 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

>>> str.upper              # upper è un metodo di str
<method 'upper' of 'str' objects>

>>> help(str.upper)              # si può usare help() per vedere una breve spiegazione
Help on method_descriptor:
upper(...)
    S.upper() -> str
    Return a copy of S converted to uppercase.
```

Strumenti per visualizzare l'help di un metodo di un oggetto

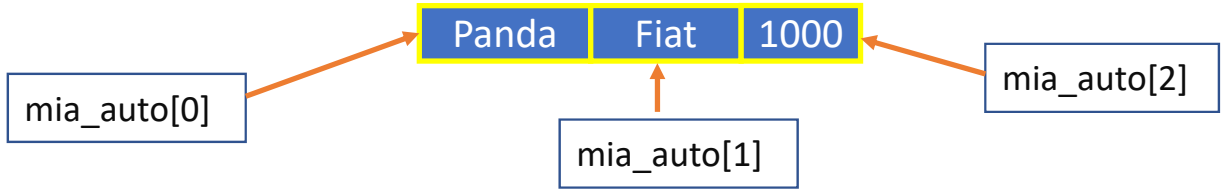
```
>>> help(str.replace)  # visualizza l'help per il metodo str.replace
Help on method_descriptor:
replace(...)
    S.replace(old, new[, count]) -> str
    Return a copy of S with all occurrences of substring
    old replaced by new.  If the optional argument count is
    given, only the first count occurrences are replaced.

>>> 'Python'.replace('thon', 'Py')  # sostituisce 'thon' con 'Py' in 'Python'
'PyPy'
>>> s = 'Python, Python, Python!'
>>> s.replace('thon', 'Py', 2)  # come sopra, ma massimo 2 sostituzioni
'PyPy, PyPy, Python!'
```

liste

Una lista può essere formato da un certo numero di variabili di un qualunque TIPO
Si possono creare vettori contenenti numeri interi, stringhe, float ecc e possono anche essere formati da tipi misti
Come nella filosofia di Python anche il vettore viene instanziato in maniera dinamica e la tipizzazione è automatica.

Es: vogliamo raccogliere in un vettore le caratteristiche di un'auto ad es: modello, marca e cilindrata:
mia_auto=['Panda','Fiat',1000]

Possiamo definire un vettore come un insieme di variabili aggregate che sono poste all'interno di una stessa area di memoria che è indirizzabile tramite il nome stesso del vettore + un indice (che indica la sua posizione all'interno del vettore) . È un'area unica di memoria che racchiuse al suo interno delle variabili tra loro "omogenee, in questo caso per valore semantico) ma assolutamente indipendenti.



tramite un nome del vettore + un solo indice.

liste

A differenza di tuple e stringhe che sono immutabili, le liste possono essere mutate.
È quindi possibile assegnare un nuovo valore agli elementi, rimuovere elementi usando la keyword del, o cambiare gli elementi usando uno dei metodi aggiuntivi delle liste:

liste

Le liste vengono definite elencando tra parentesi quadre ([]) una serie di oggetti separati da virgole (,). È possibile creare una lista vuota usando le parentesi quadre senza nessun elemento all'interno.

```
>>> nums = [0, 1, 2, 3] # nuova lista di 4 elementi
>>> nums
[0, 1, 2, 3]
>>> type(nums) # verifichiamo che il tipo sia "list"
<class 'list'>
>>> empty = [] # nuova lista vuota
>>> empty
[]
>>> one = ['Python'] # nuova lista con un elemento
>>> one
['Python']
```

Funzioni e metodi delle liste

```
>>> letters = ['a', 'b', 'c', 'd', 'e']
>>> letters[0] # le liste supportano indexing
'a'
>>> letters[-1]
'e'
>>> letters[1:4] # slicing
['b', 'c', 'd']
>>> 'a' in letters # gli operatori di contenimento "in" e "not in"
True
>>> letters + ['f', 'g', 'h'] # concatenazione (ritorna una nuova lista)
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
>>> [1, 2, 3] * 3 # ripetizione (ritorna una nuova lista)
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

liste

```
>>> letters = ['a', 'b', 'c', 'b', 'a']
>>> len(letters) # numero di elementi
5
>>> min(letters) # elemento più piccolo (alfabeticamente nel caso di
stringhe)
'a'
>>> max(letters) # elemento più grande
'c'
>>> letters.index('c') # indice dell'elemento 'c'
2
>>> letters.count('c') # numero di occorrenze di 'c'
1
>>> letters.count('b') # numero di occorrenze di 'b'
2
```

Metodi di lista

lista.append(elem)	: aggiunge elem alla fine della lista;
lista.extend(seq)	:estende la lista aggiungendo alla fine gli elementi di seq;
lista.insert(indice, elem)	:aggiunge elem alla lista in posizione indice;
lista.pop()	: rimuove e restituisce l'ultimo elemento della lista;
lista.remove(elem)	: trova e rimuove elem dalla lista;
lista.sort()	: ordina gli elementi della lista dal più piccolo al più grande;
lista.reverse()	: inverte l'ordine degli elementi della lista;
lista.copy()	: crea e restituisce una copia della lista;
lista.clear()	: rimuove tutti gli elementi della lista;

Metodi di lista

comando	commento	risultato
>>> letters = ['a', 'b', 'c']	# creo una lista di tre elementi	
>>> letters.append('d')	# aggiunge 'd' alla fine	['a', 'b', 'c', 'd]
>>> letters.extend(['e', 'f'])	# aggiunge 'e' e 'f' alla fine	['a', 'b', 'c', 'd', 'e', 'f']
>>> letters.append(['e', 'f'])	# aggiunge la lista come elemento alla fine	['a', 'b', 'c', 'd', 'e', 'f', ['e', 'f']]
>>> letters.pop()	# rimuove e ritorna l'ultimo elemento (la lista)	['e', 'f']
>>> letters.pop()	# rimuove e ritorna l'ultimo elemento ('f')	'f'
>>> letters.pop(0)	# rimuove e ritorna l'elemento in posizione 0 ('a')	'a'
>>> letters.remove('d')	# rimuove l'elemento 'd'	['b', 'c', 'e']
>>> letters.reverse()	# inverte l'ordine "sul posto" e non ritorna niente	['e', 'c', 'b']
>>> letters[1] = 'x'	# sostituisce l'elemento in posizione 1 ('c') con 'x'	['e', 'x', 'b']
>>> del letters[1]	# rimuove l'elemento in posizione 1 ('x')	['e', 'b']
>>> letters.clear()	# rimuove tutti gli elementi rimasti	[]

Tipi di variabili "verifica"

Funzione type() :

int	con a=2	type(a) --> <class 'int'>
float	con b=2.5	type(b) --> <class 'float '>
complex	con z=2+3j	type(z) --> <class 'complex'>
bool	con c=True	type(c) --> <class 'boolean'>
string	con d="prova"	type(c) --> <class 'string'>

Tipi di variabili "casting"

Il casting converte un tipo d variabile in un altro in modo temporaneo
Alcuni esempi di casting:

con a=2	float(a)	2.0	*type(a)*	-->	*<class 'int'>*
con b=2.5	int(b)	2			
con z=2+3j	str(z)	'(2+3j)'			
con c=True	str(c)	'True'			
con d="1234"	int(d)	1234			

Gli operatori di confronto possono restituire solo due valori: **True** oppure **False**

== uguale Es: a==b restituisce False

!= diverso Es: a!=b restituisce True

> maggiore Es: a>b restituisce False

< minore Es: a<b restituisce True

>= maggiore o uguale Es: a>=b restituisce False

<= minore o uguale Es: a<=b restituisce True

Operatori Booleani

- **and** – Ritorna *True* solo se entrambi gli operatori sono veri, altrimenti ritorna *False*
- **or** – Ritorna *True* se almeno uno dei due operatori è vero, altrimenti ritorna *False*.
- **not** – Ritorna *True* se l'operando è falso, altrimenti ritorna *False*.

- a=5 b=7 a==5 and b==7 -->True a==7 and b==7 -->False