



**University
of Ferrara**

Python

Introduzione

Storia del Linguaggio

- La prima versione di Python chiamata ABC, fu definita nei primi anni '80 presso il National Research Institute for Mathematics and Computer Science di Amsterdam.
- Qualche anno dopo Guido van Rossum, uno degli sviluppatori di ABC, lo migliorò dando origine ad un nuovo linguaggio: **Python**.
 - Il nome deriva dal Monty Python's Flying Circus di cui Van Rossum era fan.



Storia del Linguaggio

- Nel 2000 fu sviluppata la versione 2.0.
- Nel 2008 la versione 3.0, detta anche Python 3000 o Python 3k, introdusse molti cambiamenti e migliorie, l'utilizzo di UNICODE e molte funzioni furono completamente ridefinite .
 - Python 3 non è compatibile con Python 2!
 - In questo corso prenderemo in esame **Python 3**.



Installazione di Python

- Home page: <https://www.python.org/>
- E' possibile scaricare la versione per Windows e Mac ed il tarball per Linux.
- Per i Sistemi Linux solitamente Python è già installato o può esserlo facilmente utilizzando il gestore pacchetti (package manager):
 - `sudo apt install python3`
 - `sudo yum install python3`
 - `sudo pacman -S python`
 - ...

Installazione di Python

- In alternativa, Anaconda <https://www.anaconda.com/>
- Distribuzione di python che semplifica l'installazione di pacchetti, ad es:
- **`conda install numpy`**

IDE

- Ci sono molti IDE che possono essere utilizzati:
 - Pydev, plug-in per Eclipse IDE
 - PyCharm (<https://www.jetbrains.com/pycharm/>), in due versioni una gratuita ed una a pagamento. Questo è uno degli IDE più utilizzati e completi
 - Python plug-in per Visual Studio Code
 - ...
- Online Jupiter notebooks:
 - <https://colab.research.google.com/notebooks/welcome.ipynb#recent=true>, ci si può collegare con il proprio account google oppure con l'account UniFE

Tutorial e riferimenti

- Tutorial:
 - <https://docs.python.org/3/tutorial/index.html> (Inglese)
 - <https://www.html.it/guide/guida-python/> (Italiano)
 - ...
- Libri:
 - Think Python di Allen B. Downey
 - <https://github.com/AllenDowney/ThinkPython> (codice sorgente, Inglese)
 - <https://github.com/AllenDowney/ThinkPythonItalian> (codice sorgente e PDF, Italiano)
 - Molte di queste slide sono riprese da questo libro e dal Tutorial di Python.

Run Python

- Due modi per avviare l'**interprete**:
 - **Interactive mode**: dal Prompt eseguire il comando **python** senza alcun argomento. Verrà visualizzata una nuova linea: **>>>**

```
Python 3.7.3 (default, Mar 26 2019, 21:43:19)
[GCC 8.2.1 20181127] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- A questo punto potrai eseguire i tuoi comandi

```
Python 3.7.3 (default, Mar 26 2019, 21:43:19)
[GCC 8.2.1 20181127] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 1
2
>>>
```

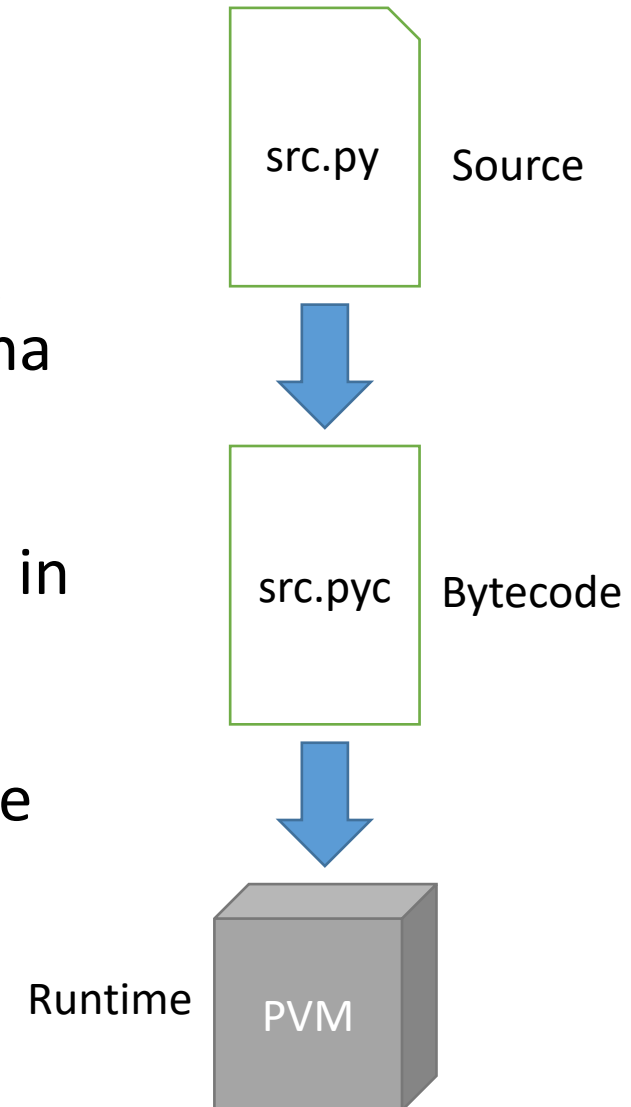

Run Python

- Due modi per avviare l'**interprete**:
 - **Script mode**: è possibile eseguire il comando **python** seguito dal nome del file che si vuole eseguire come argomento.
 - L'estensione per il file Python è **.py**

```
# python myscript.py  
Hello world!  
#
```

Eseguire uno script Python

- Ogni volta che viene invocato il comando **python**, il codice scritto viene scansionato alla ricerca di token, ognuno dei quali viene analizzato in una struttura logica ad albero che rappresenta il programma.
- Questa struttura viene poi trasformata in un codice binario, **bytecode** (file con estensione **.pyc** o **.pyo**).
- Per poter eseguire questo codice, viene utilizzato uno speciale interprete noto come **Python Virtual Machine** (PVM).



Hello World!

- Usando l'interprete interattivo

```
>>> print('Hello World!')  
Hello World!
```

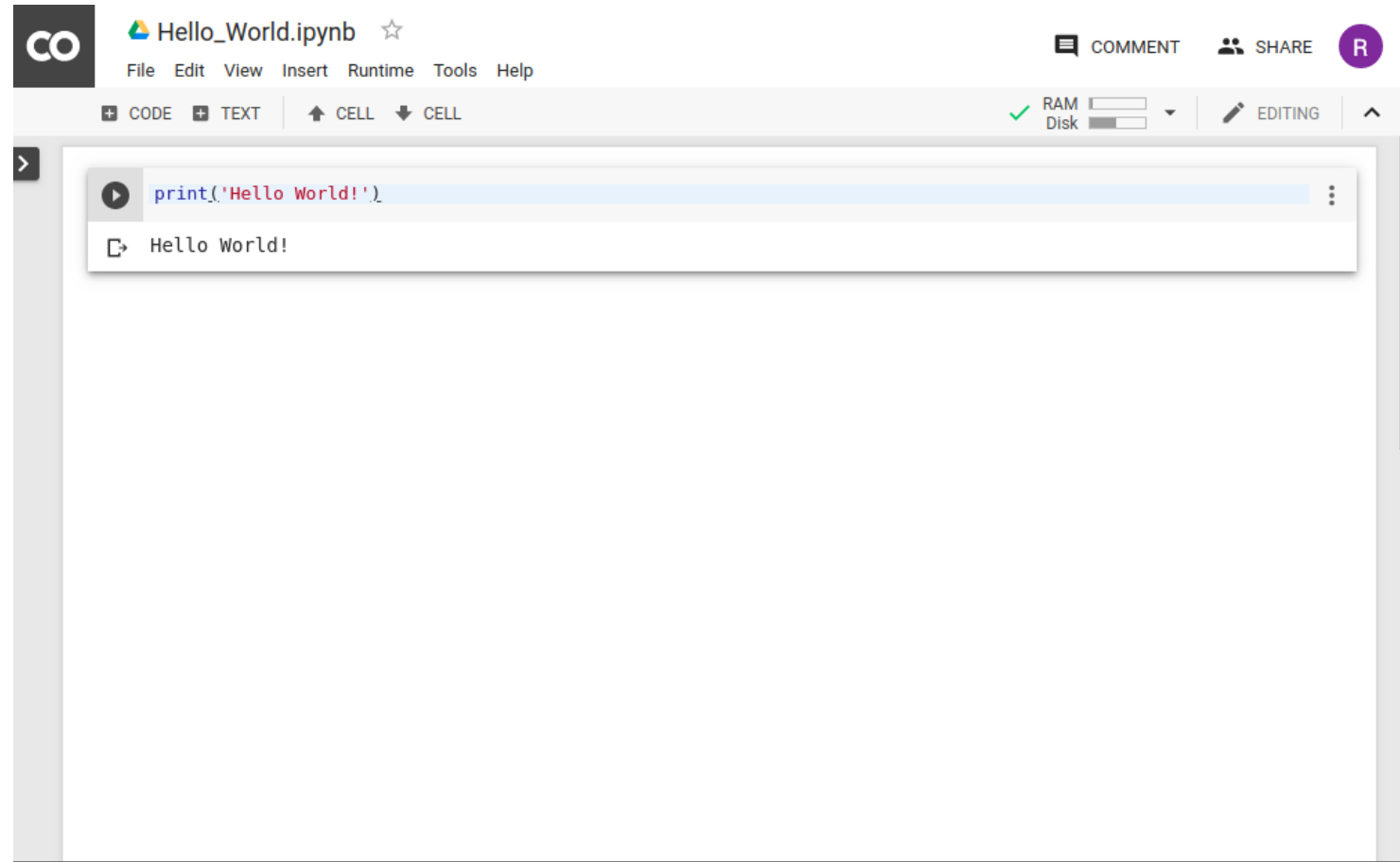
- Usando un file script

script.py

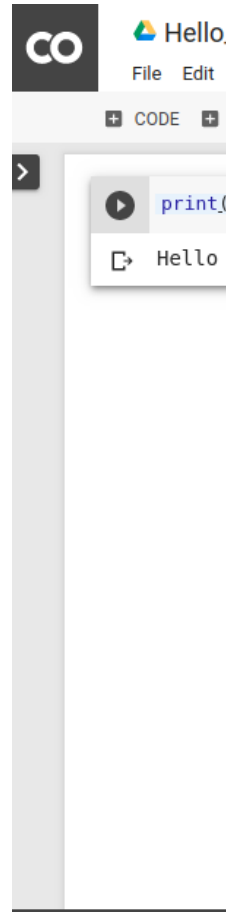
```
print('Hello World!')
```

```
# python script.py  
Hello World!
```

Hello World!



Hello World!



Puoi creare un nuovo Colab notebook cliccando sul bottone **+** in **Drive**, e selezionando **other > Colaboratory**.

Verrà così creato un notebook con una **code cell**, una cella all'interno della quale è possibile scrivere il codice da eseguire.

Per eseguire il codice è sufficiente fare click sul bottone play a sinistra della cella stessa.

Se all'interno dello stesso notebook si trovano più code cells, l'esecuzione di ciascuna di queste terrà in considerazione il codice all'interno delle celle precedenti

Qui è possibile trovare un esempio:

https://colab.research.google.com/drive/16RiNuLY23he7wGE_e7-nuHjh1B8wXOJr

Variabili

- I nomi delle variabili possono contenere sia lettere che numeri .
 - Devono, però, iniziare sempre con una lettera.
- Non ci sono limiti nel numero di caratteri e parole che si possono utilizzare:
 - Per separare le parole all'interno di un nome si può usare l'underscore “_” → usare una lettera maiuscola non è sbagliato ma va contro la convenzione di denominazione e dovrebbe essere evitata.

Variabili

- Ci sono 31 parole chiavi che non possono essere utilizzate per nominare una variabile :

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	in	raise	continue	
finally	is	return	nonlocal	
def	for	lambda	try	

Basic Input/Output

- Abbiamo già visto l'istruzione per scrivere sullo standard output → la funzione `print`

```
>>> print(1)
1
>>> print('hello')
hello
>>> s = 'hi'
>>> a = 3
>>> b = 5
>>> print(a,b,a+b,s)
3 5 8 hi
```

- Se lavoriamo con l'`interprete interattivo` è sufficiente scrivere il nome della variabile

```
>>> a = 3
>>> a
3
```


Basic Input/Output

- Per consentire all'utente di scrivere dei dati utilizzando lo standard input è possibile usare la funzione `input`

```
>>> in = input()
This is my first input
>>> in
'This is my first input'
```

- Possiamo passare alla funzione `input` una stringa da visualizzare in attesa dell'input dall'utente

```
>>> mess = input('Write your message\n')
Write your message
Hi!
>>> mess
'Hi!'
```

Tipi di dato in Python

- In Python tutto è un **oggetto**.
- Nei linguaggi simili al C, le variabili sono associate a specifiche aree di memoria la cui dimensione dipende dal loro tipo .
 - E' necessario specificare il tipo quando si dichiara una variabile.
- In Python gli **oggetti** hanno un **tipo** specifico (numeri, stringhe, liste, ecc.), mentre le **variabili** sono solo **etichette**, **referimenti** ad un dato oggetto.
 - Le variabili non hanno tipo.

Tipi di dato in Python

- Consideriamo il seguente codice dove alla variabile **x** vengono associati differenti valori.

```
x = 10
```

```
x = 7
```

```
x = "Python"
```

```
x = [1,2,3]
```

Tipi di dato in Python

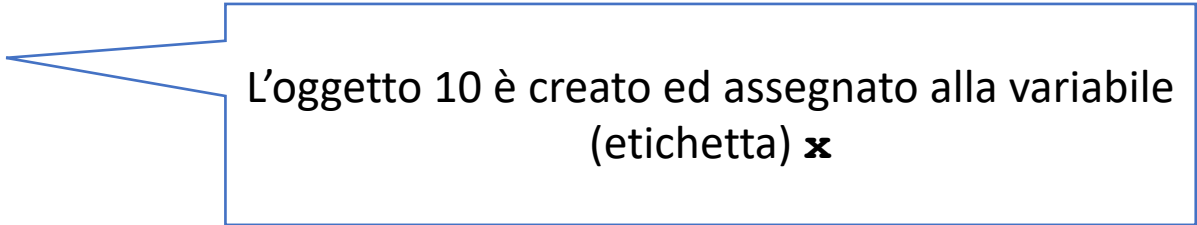
- Consideriamo il seguente codice dove alla variabile **x** vengono associate differenti valori.

```
x = 10
```

```
x = 7
```

```
x = "Python"
```

```
x = [1,2,3]
```



L'oggetto 10 è creato ed assegnato alla variabile (etichetta) **x**

Tipi di dato in Python

- Consideriamo il seguente codice dove alla variabile **x** vengono associate differenti valori.

x = 10

x = 7

x = "Python"

x = [1, 2, 3]

Poi, l'oggetto 7 è creato ed assegnato alla variabile (etichetta) **x**. Ora l'oggetto 10 non è più referenziato da **x**, l'unico riferimento a 10 adesso si riferisce a 7. Il Garbage Collector rimuoverà questo oggetto dalla memoria.

Tipi di dato in Python

- Consideriamo il seguente codice dove alla variabile **x** vengono associate differenti valori.

```
x = 10
```

```
x = 7
```

```
x = "Python"
```

```
x = [1, 2, 3]
```

Dato che le variabili non hanno tipo, sono solo dei riferimenti a oggetti, possono essere assegnati a tipi diversi.

Il tipo della variabile può essere dedotto grazie all'oggetto referenziato, in modo che si possa capire cosa poter fare con la variabile.

Tipi di dato in Python

- Python usa il **duck typing**, che segue la regola “se cammina come una papera e fa il verso della papera, allora dev’essere una papera”
- L'idoneità è determinata verificando la presenza di metodi e proprietà.



Tipi di dato in Python

Tipo	Mutabilità	Descrizione
<code>int</code>	Immutabile	Numeri Interi: 1
<code>bool</code>	Immutabile	Booleani: True, False
<code>float</code>	Immutabile	Numeri a virgola mobile: 2.3
<code>complex</code>	Immutabile	Numeri complessi con parte reale e immaginaria: 1+2.3j
<code>str</code>	Immutabile	Stringa: 'hi'
<code>tuple</code>	Immutabile	Può contenere tipi diversi: (1, 'hi', False)
<code>bytes, bytearray</code>	Immutabile	Sequenza di byte
<code>list</code>	Mutabile	Lista, può contenere tipi differenti: [1, 'hi', False]
<code>set, frozenset</code>	Mutabile	Insieme non ordinato, può contenere tipi differenti: {1, 'hi', False}
<code>dict</code>	Mutabile	Array associativi, mappe: {'key': 1, 3: 'string'}

Mutabilità

- **Immutable:** Un oggetto con un valore fisso. Gli oggetti immutabili includono numeri, stringhe e tuple. Questi oggetti non possono essere modificati. Se si dovesse memorizzare un valore differente, verrà creato un nuovo oggetto.
- **Mutable:** Un oggetto mutabile può cambiare il proprio valore.

Tipi di dato in Python

- E' possibile identificare il tipo di un valore o di una variabile usando la funzione **type**.

```
>>> type('Hello')
<type 'str'>
>>> type(7)
<type 'int'>
>>> type('7')
<type 'int'>
>>> x = 10
>>> type(x)
<type 'int'>
```

Tipi di dato in Python

- E' possibile verificare se un valore o una variabile sia di un determinato tipo usando la funzione **isinstance**

```
>>> isinstance('Hello',str)
True
>>> isinstance(7,str)
False
>>> isinstance(7,int)
True
>>> x = 10
>>> isinstance(x,int)
True
```

Tipi di dato in Python

- Python ha anche un valore speciale detto **None**.
- Questo è il valore che ritornano le funzioni void (funzioni che non hanno la dichiarazione return) oppure hanno la dichiarazione di return, ma senza argomenti.
- E' simile a null degli altri linguaggi, e può essere utilizzata per inizializzare variabili.
- Il suo tipo è **NoneType**

Operatori

- + somma
- - sottrazione
- * moltiplicazione
- / divisione ($50/25=2.0$, $59/60 = 0.9833$)
- // divisione intera ($2//2= 1$, $2.5//2 = 1.0$)
- ** esponenziale
- % modulo ($10\%3 = 1$)
- Python rispetta le seguenti regole di precedenza:
1. parentesi, 2. esponenziale 3. moltiplicazione e divisione, 4. addizione e sottrazione

Operatori Bitwise

- **$x \ll y$** Ritorna x con i bit shiftati verso sinistra di y posizioni (e i nuovi bit inseriti a destra saranno pari a 0).
 - Equivale a moltiplicare x per $2^{**}y$.
- **$x \gg y$** Ritorna x con i bit shiftati a destra di y posizioni.
 - Equivale a fare la divisione intera `//` di x per $2^{**}y$.
- **$x \& y$** fa un "AND bit a bit".
- **$x | y$** fa un "OR bit a bit".
- **$\sim x$** Ritorna il complemento di x , numero che si ottiene cambiando tutti gli 1 in 0 e tutti gli 0 in 1.
- **$x \wedge y$** fa un "XOR bit a bit".

Operatori Booleani

- `==` uguale
- `!=` diverso
- `<` minore
- `<=` minore o uguale
- `>` maggiore
- `>=` maggiore o uguale

Tutti questi operatori
restituiscono un Booleano

- `and`
- `or`
- `not`

In Python `0`, `""` (stringa vuota), **`False`** e **`None`** sono tutti considerati come **`false`**. Tutto il resto viene considerato come **`true`**.

La funzione **`bool`** ritorna il valore booleano del valore di un oggetto.

Operatori con le stringhe, commenti e assegnamenti

- + concatenazione
 - `'Hello' + 'World' = 'Hello World'`
- * ripetizione
 - `'Hi'*3 = 'HiHiHi' = 'Hi' + 'Hi' + 'Hi'`
 - `'Ba' + 'na'*2 = 'Banana'`
- # commento in linea
 - `x = 10 # questo è un assegnamento`
- = assegnamento, può essere multiplo
 - `x = 10 # questo è un assegnamento`
 - `x, y, z = 10, 20, 30 # questo è un assegnamento multiplo`

Commenti su più linee

```
#Questo è un lungo commento  
#e si estende su  
#più linee
```

- Un altro modo per fare questo è utilizzare apici triplici, indifferentemente ''' oppure """".

```
'''Anche questo è  
Un esempio perfetto di  
commento su più linee'''
```

Conversioni di Tipo

- **int(a, base)** : converte qualsiasi tipo di dato in un intero. 'base' specifica la base in cui si trova la stringa se il tipo di dati è una stringa, ad esempio **int('1101', 2)**
- **float()** : converte qualsiasi tipo di dato in un numero a virgola mobile
- **str()** : converte un intero in una stringa..
- **ord()** : converte un carattere in un intero.
- **hex()** : converte un intero in una stringa esadecimale.
- **oct()** : converte un intero in un ottale
- **tuple()** : converte in una tupla.
- **set()** : converte in un insieme.
- **list()** : converte un qualsiasi tipo in un tipo lista.
- **dict()** : converte le tuple (chiave, valore) in un dizionario.
- **complex(real, imag)** : converte numeri reali in numeri complessi

Funzioni

- Come in qualsiasi linguaggio di programmazione, le funzioni sono una sequenza di istruzioni eseguibili utilizzando il nome della funzione e che talvolta restituiscono un valore.
- Noi abbiamo già visto 3 funzioni: input, print, e type.
- Le funzioni accettano 0 o più argomenti, che possono essere oggetti di tipo diverso, usando la sintassi: **function (arguments) .**

Funzioni

- Le funzioni accettano 0 o più argomenti, che possono essere oggetti di tipo diverso, usando la sintassi:
function(arguments) .

```
>>> len("Python") # con una stringa
```

```
6
```

```
>>> len([1,2,3]) # con un array
```

```
3
```

```
>>> len({'a':3, 'b':5}) # con un dizionario
```

```
2
```

Metodi

- I metodi sono simili alle funzioni ma sono collegati al tipo dell'oggetto.
- Possono essere richiamati sull'oggetto utilizzando la notazione a punto "."

```
>>> s = "Python"
>>> s.upper() # restituisce "PYTHON"
>>> s.lower() # restituisce "python"
```

Due funzioni utili

- **help(obj)** : questa funzione restituisce una breve descrizione dell'oggetto
- **dir(obj)** : questa funzione restituisce la lista di metodi ed attributi associati a quell'oggetto

Stringhe

- Per dichiarare una stringa è sufficiente assegnare un testo racchiuso tra apici ad una nuova variabile.
- E' possibile usare apici singoli (carattere ') oppure doppi apici (carattere ").
 - E' possibile usare il carattere di `escape`
 - `"This is Riccardo's string" = 'This is Riccardo\'s string'`
 - Le stringhe lunghe possono essere divise su più line utilizzando gli apici tripli
 - E' anche possibile confrontare stringhe tra di loro utilizzando gli operatori relazionali.

Stringhe

- Se non si vuole che i caratteri preceduti da \ vengano considerati come caratteri speciali, si può utilizzare la *raw string* facendo precedere al primo apice una r:

```
>>> print('C:\some\name') # here \n means newline!
```

```
C:\some
```

```
ame
```

```
>>> print(r'C:\some\name') # note the r before the quote
```

```
C:\some\name
```


Stringhe

- E' possibile distribuire i letterali di tipo stringa su più linee. Un modo per farlo è utilizzare i tripli apici: `"""..."""` oppure `'''...'''`
- Un «fine linea» viene automaticamente incluso all'inizio della stringa ma è possibile evitarlo aggiungendo `\` alla fine della linea.

Stringhe

```
print("""\
Usage: thingy [OPTIONS]
    -h                Display this usage message
    -H hostname       Hostname to connect to
""")
```

Produce il seguente output (da notare che non viene inclusa la linea vuota iniziale):

```
Usage: thingy [OPTIONS]
    -h                Display this usage message
    -H hostname       Hostname to connect to
```

Stringhe

- **Indexing:** è possibile utilizzare un indice per indicare un carattere all'interno di una stringa. L'indice dev'essere un intero.

```
>>> s = 'Python'
>>> s[0]    # elemento in pos 0 (primo)
'P'
>>> s[5]    # elemento in pos 5 (sesto)
'n'
>>> s[-1]   # elemento in pos -1 (ultimo)
'n'
>>> s[-4]   # elemento in pos -4 (quart'ultimo)
't'
```

Stringhe

- **Slicing:** ovvero la possibilità di ottenere delle sottostringhe usando l'indexing:

```
>>> s = 'Python'
>>> s[0:2]    # sottostringa dall'elemento 0(incluso) fino al 2 (escluso)
'Py'
>>> s[:2]     # dall'inizio fino a 2 (escluso)
'Py'
>>> s[3:5]    # dall'indice 3 (incluso) fino al 5 (escluso)
'ho'
>>> s[4:]     # dall'indice 4 (incluso) fino alla fine
'on'
>>> s[-2:]    # dall'indice -2 (incluso) fino alla fine
'on'
```

Stringhe

- **Test di inclusione:** utilizzando l'operatore `in` e `not in`

```
>>> s = 'Python'
>>> 'P' in s # controlla se il carat. 'P' sia presente nella stringa s
True
>>> 'x' in s # il caratter 'x' non è in s, ritorna False
False
>>> 'x' not in s # "not" per ottenere l'inverso del test
True
>>> 'Py' in s # controlla se la sottostringa 'Py' sia in s
True
>>> 'py' in s # il test è case-sensitive
False
```

Stringhe

- **Lunghezza di una stringa:** si usa la funzione **len**

```
>>> len("Python")
```

```
6
```

```
>>> s = "string"
```

```
>>> len(s)
```

```
6
```

Stringhe

- L'operatore + può essere usato per concatenare più stringhe

```
"Python" + " is a string of " + str(6) +  
"characters"
```



Ricorda di convertire in stringa, i tipi
che non lo sono già

Stringhe

- Le stringhe possono essere ripetute con *:

```
>>> 3 * 'un' + 'ium'
'ununinium'
```

- Due o più *letterali di stringhe* (es. racchiusi tra apici) posti l'uno accanto all'altro vengono automaticamente concatenati .

```
>>> 'Py' 'thon'
'Python'
```


Stringhe

- Questa proprietà è particolarmente utile quando si vuole dividere stringhe particolarmente lunghe:

```
>>> text = ('Put several strings within parentheses '  
...         'to have them joined together.')
```

```
>>> text
```

```
'Put several strings within parentheses to have them joined together.'
```

Stringhe

- **Formato di una stringa:** si usa il metodo **format**

```
>>> x, y = 10, 12.3
>>> s = "x is equal to {}, while y is {}"
>>> s.format(x,y)
"x is equal to 10, while y is 12.3"
```

Le { } rappresentano un **placeholder**
(segnaposto).

Vengono sostituiti dagli argomenti
passati al metodo format nell'ordine.

Stringhe

- Altri metodi interessanti:
 - **find(substring)** : fornisce l'indice del primo carattere di una corrispondenza della sottostringa da sinistra o -1 se tale carattere non esiste.
 - **rfind(substring)** : come il precedente ma da destra
 - **find(substring, i, j)** : come **find()** , ma cerca solo all'interno di **string[i:j]**.
 - **split()** : divide una stringa in una lista di sottostringhe in corrispondenza degli spazi bianchi o di una stringa passata per argomento.

Il costrutto `if`

```
if condizione1:
    # istruzioni
elif condizione2:
    # istruzioni
elif condizione3:
    # istruzioni
else:
    # istruzioni
```

- Gli `elif` e gli `else` sono opzionali.
- Il codice deve essere rientrato. Python non usa le parentesi graffe per racchiudere le istruzioni ma usa il rientro tramite **tabulazione** (o spazi, di solito 4)

Il costrutto `if`

```
if a == 10:
    # istruzioni
elif a < 0:
    # istruzioni
elif a > 10:
    # istruzioni
else:
    # istruzioni
```

- Gli `elif` e gli `else` sono opzionali.
- Il codice deve essere rientrato. Python non usa le parentesi graffe per racchiudere le istruzioni ma usa il rientro tramite **tabulazione** (o spazi, di solito 4)

Comandi

- Un comando per linea con newline come terminatore
- E', però, possibile estendere su più linee un'espressione usando il carattere di continuazione di linea (\). Per esempio:

```
a = 1 + 2 + 3 + \  
4 + 5 + 6 + \  
7 + 8 + 9
```

- Questa è la continuazione di riga esplicita.
- In Python, la continuazione di riga è implicita tra parentesi tonde (), parentesi quadre [] e parentesi graffe { }. Ad esempio, possiamo implementare l'istruzione multilinea scritta sopra come:

```
a = (1 + 2 + 3 +  
4 + 5 + 6 +  
7 + 8 + 9)
```

Comandi

- E' possibile scrivere più comandi per linea separandoli tramite ';'

 a = 1; b = 2; c = 3
- Sintassi per il blocchi: il codice in un blocco deve sempre essere rientrato (spazio o tab)
- Quando si utilizza la riga di comando, un prompt di punti indica l'aspettativa di ulteriori input

```
>>> a=0
>>> if a==1:
...     if a==0:
...         a=1
...
>>>
```

Comandi

- L'indentazione può essere ignorata se si scrive il comando tutto su una linea.
- E', però, buona prassi usare l'indentazione per rendere il codice più leggibile. Per esempio:

```
if True:  
    print('Hello')  
    a = 5
```

oppure

```
if True: print('Hello'); a = 5
```


Esercizio 1

- Definire le seguenti variabili: $n1 = 1$, $n2 = 2$, $s1 = \text{'hi'}$, $s2 = \text{'hello'}$
- Usare il costrutto if per verificare se il tipo assegnato ad $n1$ ed $n2$ è lo stesso . Se così fosse, stampare il valore delle variabili e la loro relazione. Ad esempio « $n1$ è più grande di $n2$ »
- Usare il costrutto if per verificare se $s1$ ed $s2$ sono dello stesso tipo. Se così fosse, verificare se $s1$ è una sottostringa di $s2$ (se è vero stampare “ $s1$ è sottostringa di $s2$ ”), altrimenti verificare se $s2$ è contenuto in $s1$ e stampare “ $s2$ è sottostringa di $s1$ ”. In alternativa stampare che $s1$ ed $s2$ sono tra loro differenti.