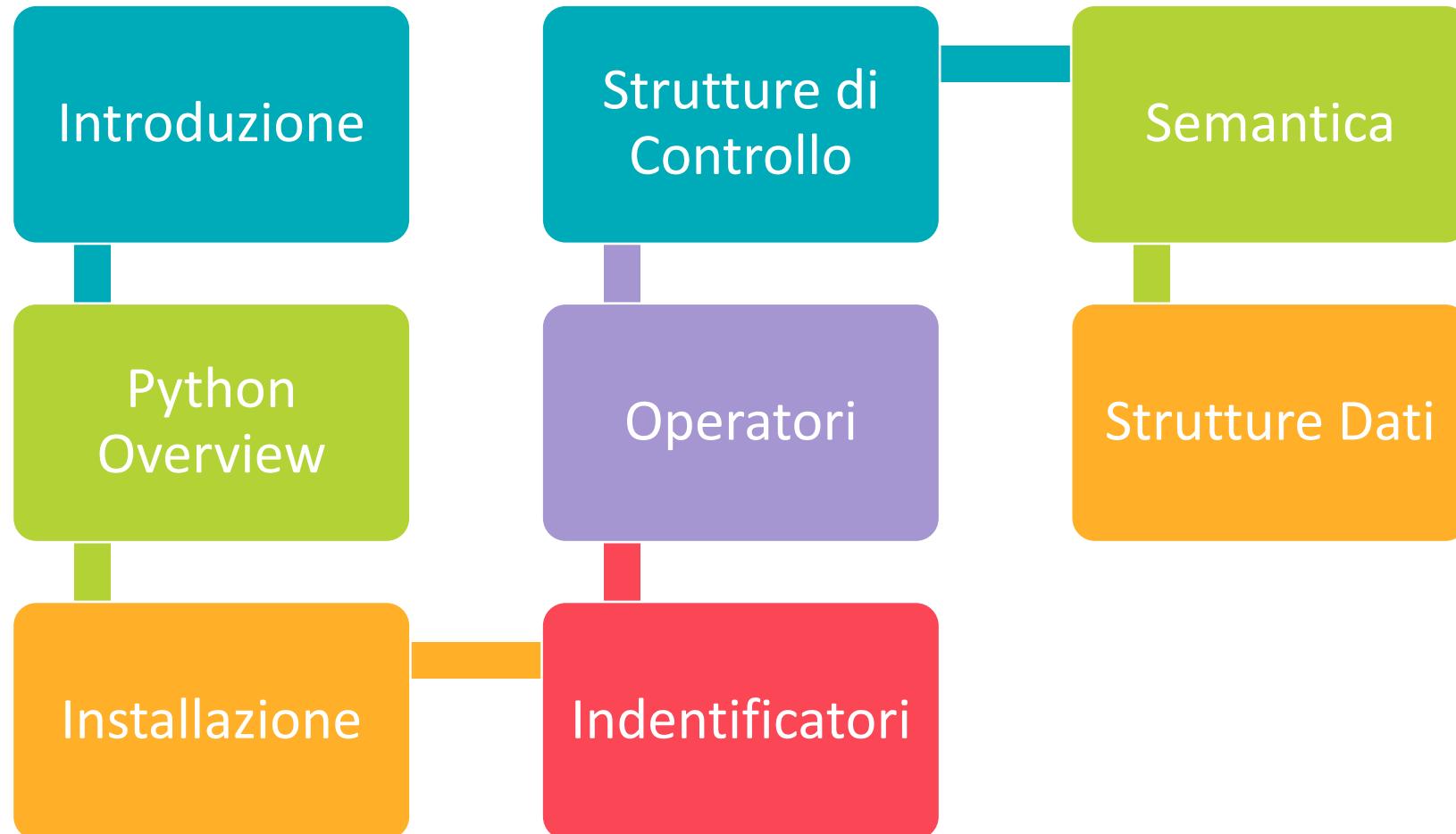


Fondamenti di Data Science e Machine Learning

Introduction to Python

Prof. Giuseppe Polese, a.a. 2024-25

Outline



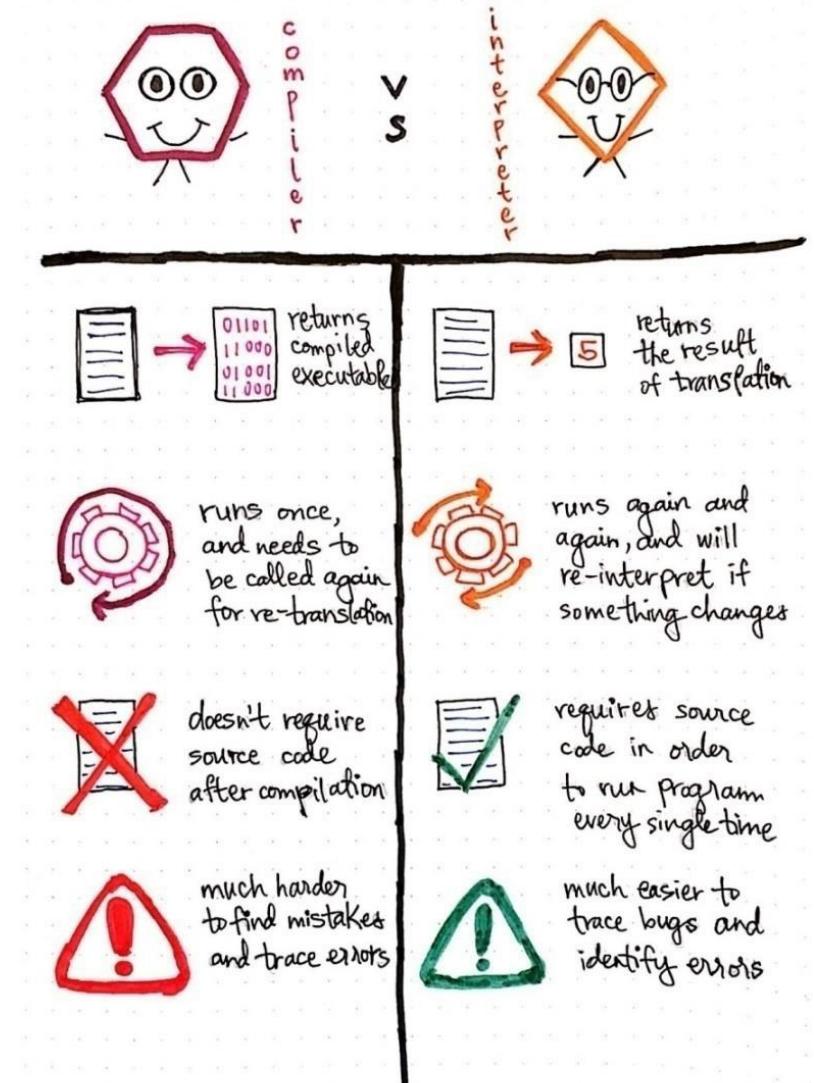
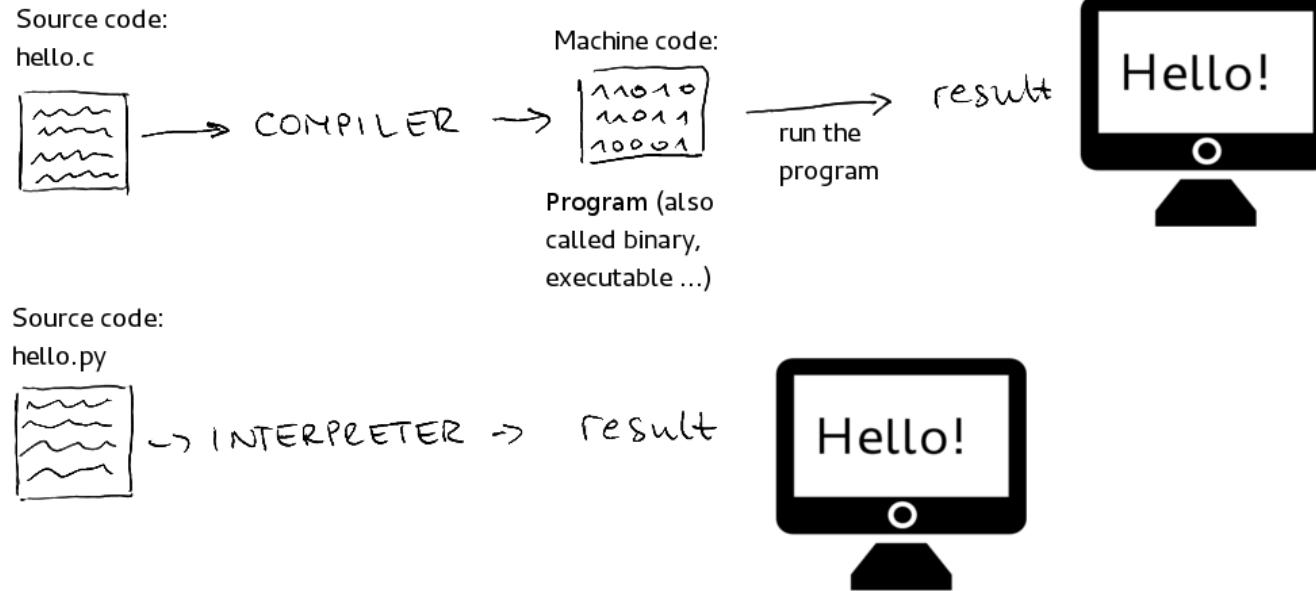
Introduzione

Python

- **Python** è un linguaggio di programmazione dinamico
 - Open Source
 - Interpretato
 - Di alto livello
 - General-purpose
 - Object-oriented
- La sua struttura di codifica consente ai programmatore di articolarsi concetti di calcolo in meno righe di codice
 - Più semplice di C++ e Java



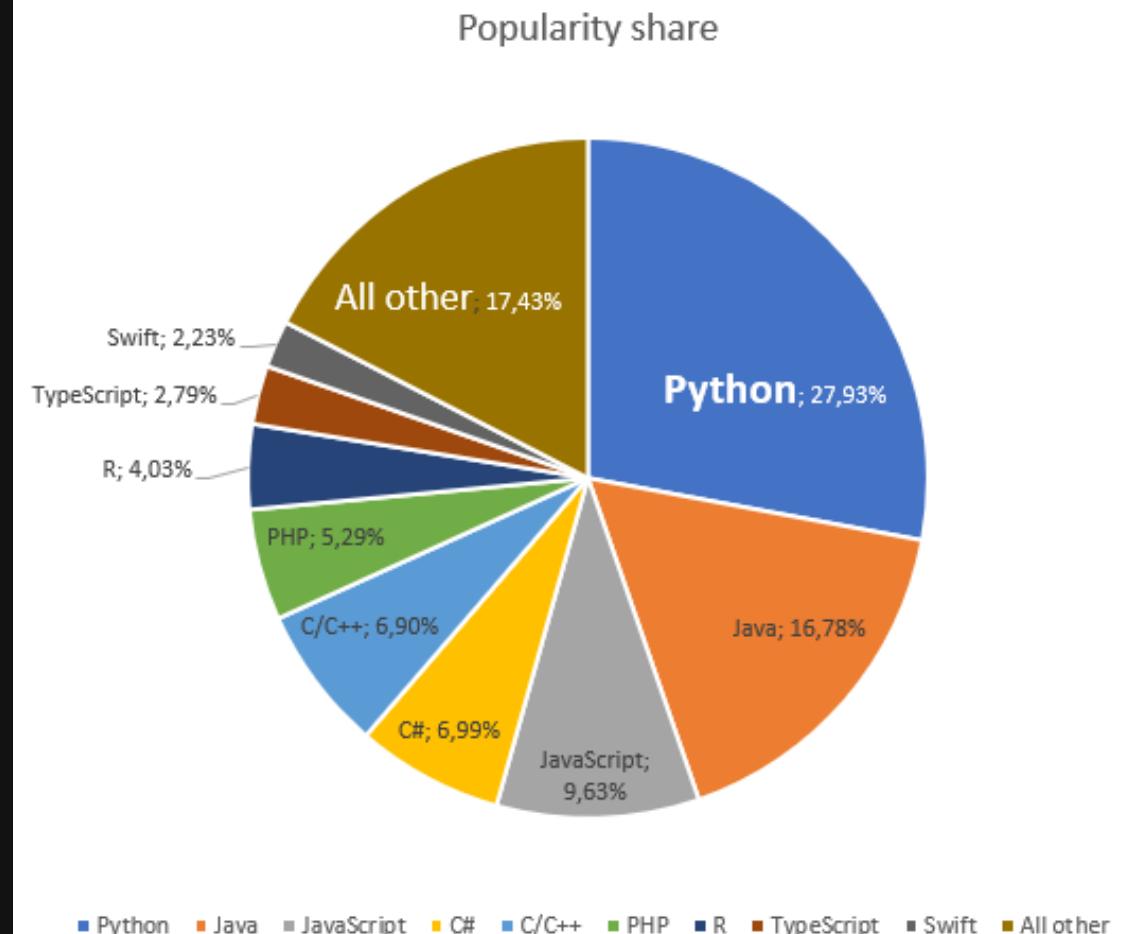
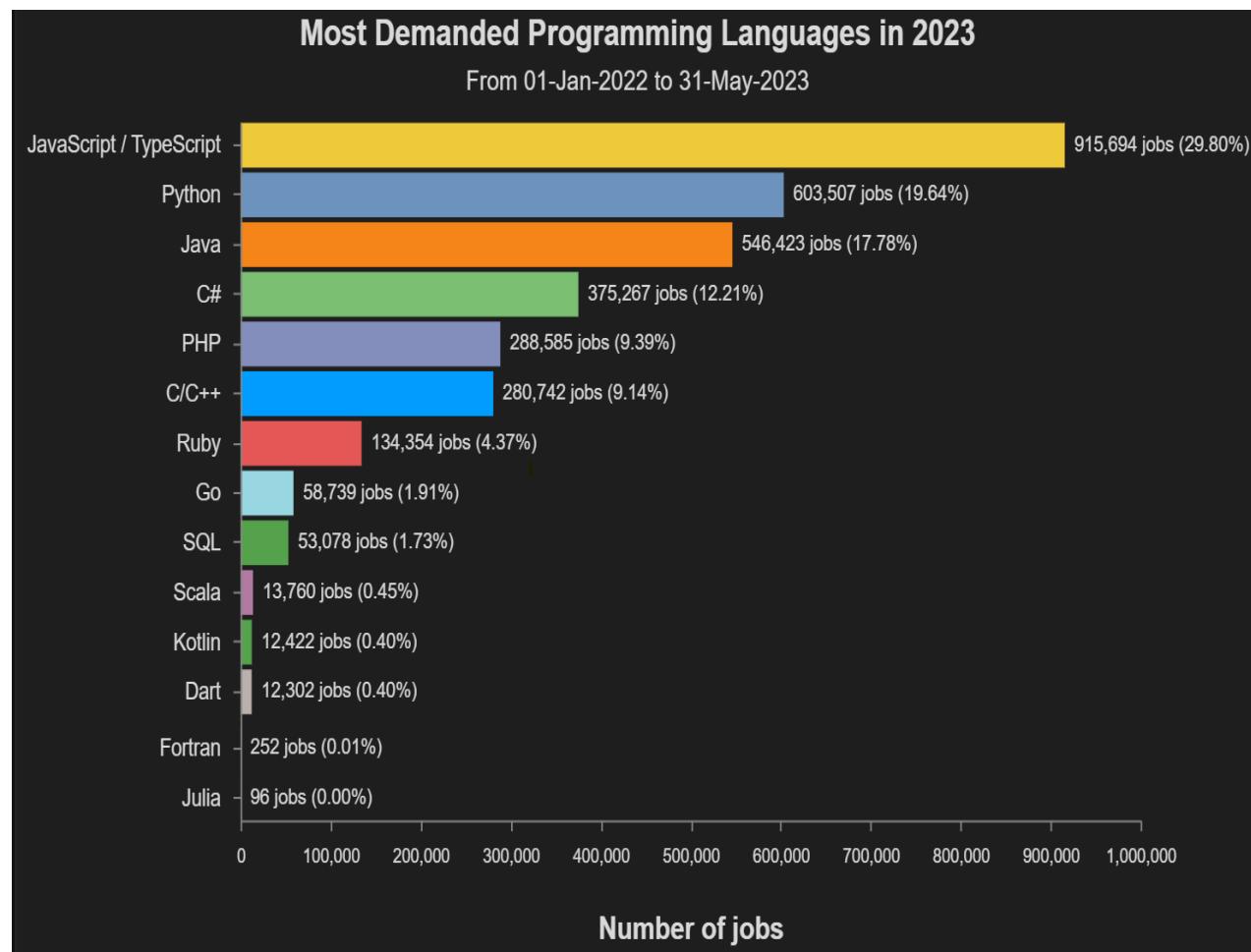
Python è interpretato



- **Cython** è un **compilatore statico ottimizzante** sia per il linguaggio di programmazione Python che per il linguaggio di programmazione Cython

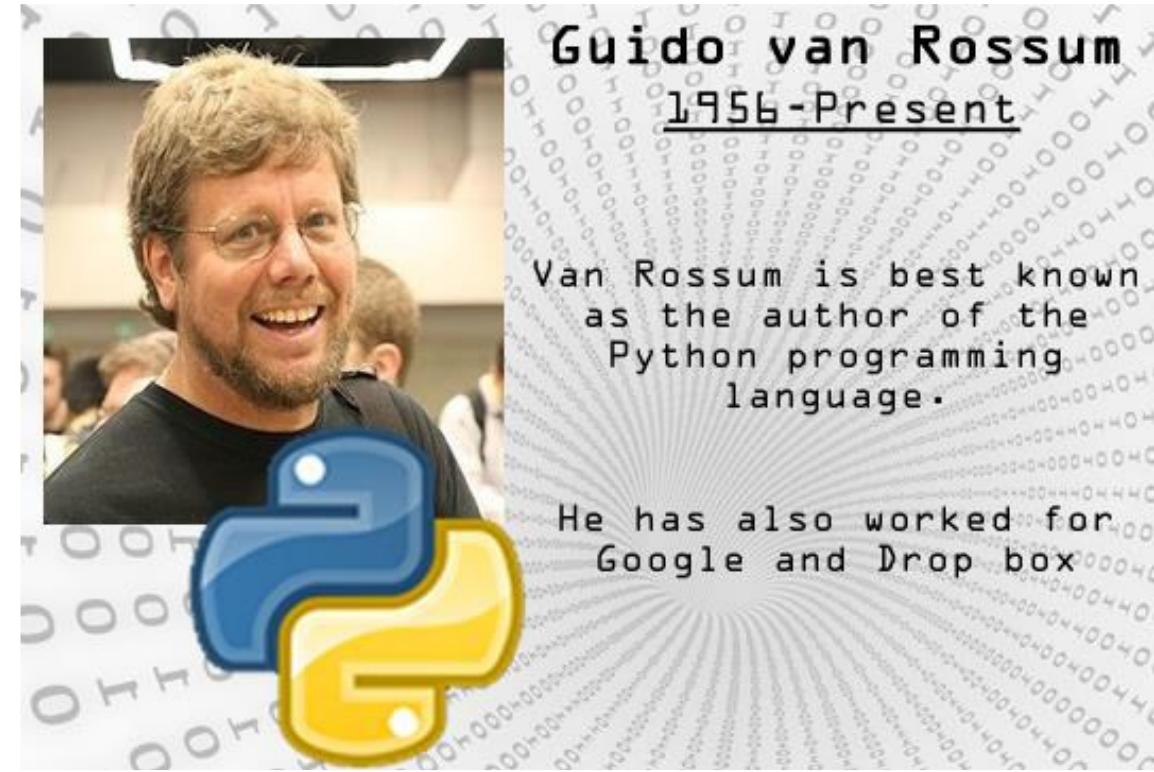
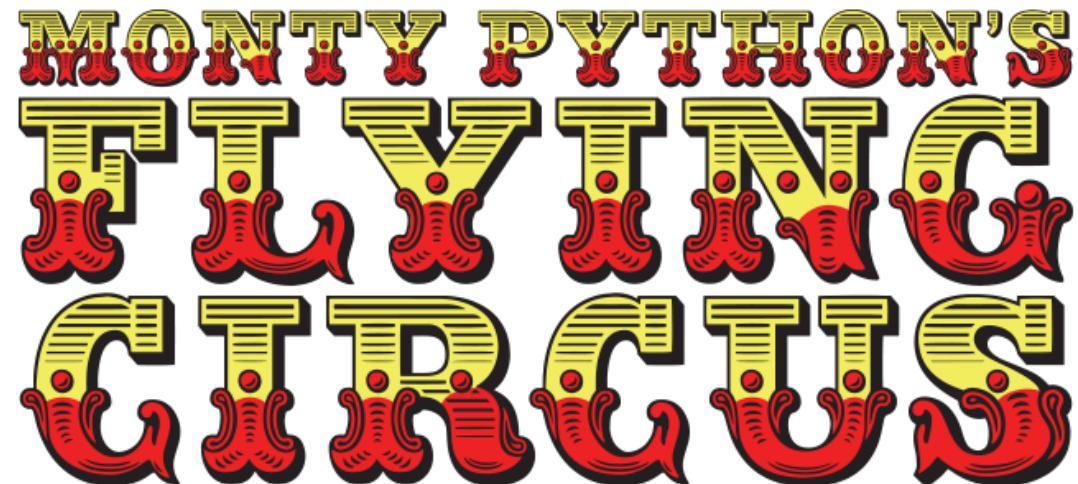
Python: la stella nascente

- **Python** è nato ufficialmente il 20 febbraio 1991, con il numero di versione 0.9.0
 - Ha intrapreso un percorso di crescita straordinario



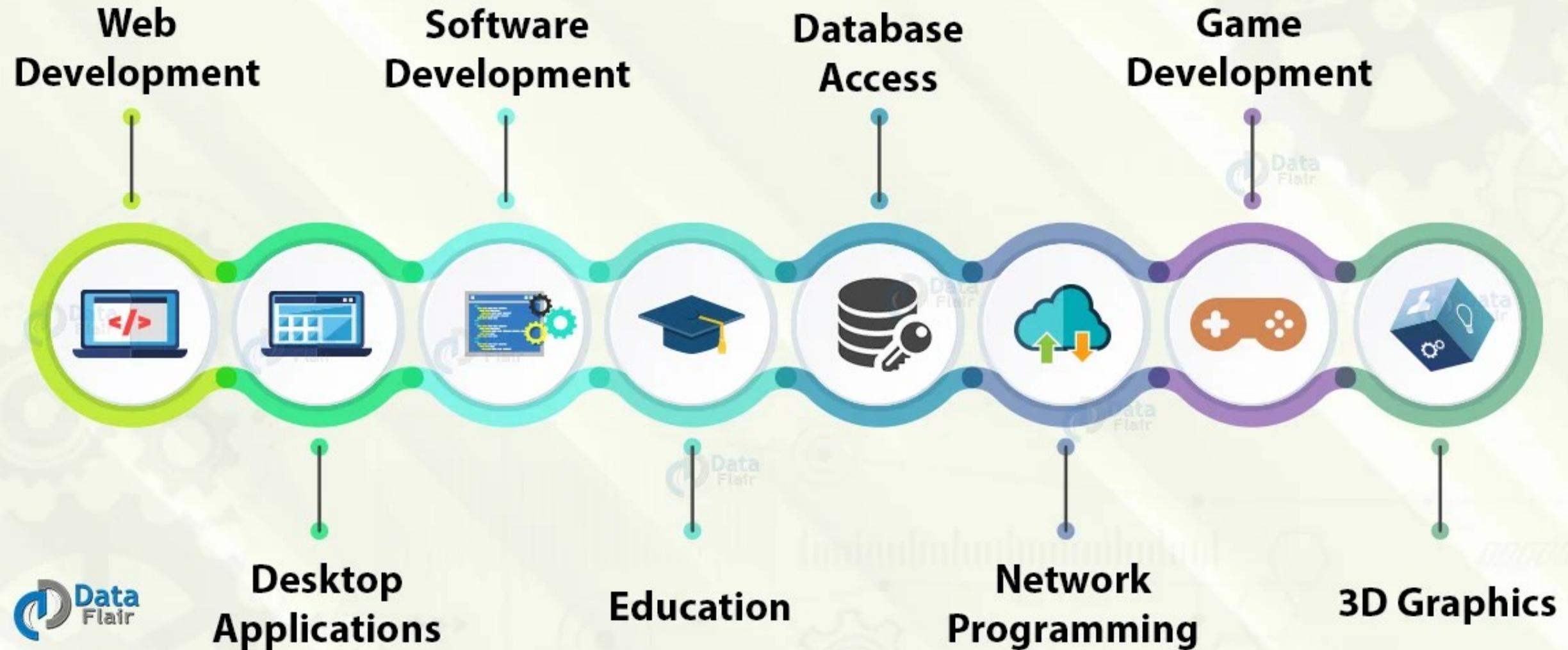
Storia del nome

Guido van Rossum, il creatore del linguaggio Python, ha chiamato il linguaggio in onore dello show della BBC «**Monty Python's Flying Circus**»



**Non gli piacevano
particolarmente i serpenti!!**

Python Applications

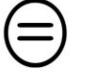


Versioni di Python (1)

- **3.x** è l'ultima versione di Python
 - Pro: Funzionalità consistenti ed ottimizzate
 - Contro: Non tutti i moduli di terze parti lo supportano
- **2.x** è la versione più usata
 - Molti framework funzionano ancora su questa versione



Versioni di Python (2)

PYTHON 2.X	PYTHON 3.X
← LEGACY	FUTURE →
It is still entrenched in the software at certain companies	It will take over Python 2 by the end of 2019
 LIBRARY	 LIBRARY
Many older libraries built for Python 2 are not forwards compatible	Many of today's developers are creating libraries strictly for use with Python 3 
0100 0001 ASCII	UNICODE
Strings are stored as ASCII by default	Text Strings are Unicode by default 
 7/2=3	7/2=3.5 
It rounds your calculation down to the nearest whole number	This expression will result in the expected result
 print "WELCOME TO GEEKSFORGEEKS"	print("WELCOME TO GEEKSFORGEEKS") 
It rounds your calculation down to the nearest whole number	This expression will result in the expected result

- Ci sono molte **differenze** tra le varie versioni:
 - Sintassi
 - Operazioni Matematiche
 - Unicode
 - ...

Installazione ed Esecuzione

Installazione e Esecuzione (1)

- Per **Windows**

- Visitare il sito <https://www.python.org/downloads/> e scarica **l'ultima** versione
- Durante l'installazione assicurati di selezionare l'opzione
 - «**Aggiungi Python [version] alla PATH**»
- In alternativa, definire manualmente una variabile d'ambiente:
 - **PATH** -> **Aggiungi** -> **C:\[user-dir]\[python-dir]**
- Eseguire python su **Windows**
 - **Start** -> **Esegui**
 - Nella finestra di dialogo, digitare **cmd**, ed **invio**
 - Digitare **python** e assicurarsi che non ci siano **errori**

Installazione e Esecuzione (2)

- Per **Mac OS X**

- Utilizzare la Homebrew
 - **Brew install python3**
 - Eseguire python su **Mac OS X**
 - Apri il terminale:
 - Combinazione tasti: [**Command + Space**]
 - Nella finestra di dialogo, digitare **Terminale**, ed **invio**
 - Esegui **python3** e assicurarsi che non ci siano **errori**

Installazione e Esecuzione (3)

- Per **GNU/Linux**
 - Utilizzare il gestore dei pacchetti per installare python3, ad esempio:
 - ***sudo apt-get update && sudo apt-get install python3***
 - Eseguire python su **GNU/Linux**
 - Apri il terminale:
 - Esegui **python3** e assicurarsi che non ci siano **errori**

PyCharm

- **PyCharm** è un **IDE** dedicato a Python e Django.

- Esso fornisce un'ampia gamma di strumenti

essenziali per gli sviluppatori

- Consente di creare un ambiente conveniente per

l'uso del linguaggio



- **PyCharm** può essere scaricato dal seguente collegamento:
 - <https://www.jetbrains.com/pycharm/download/>

Benvenuti in PyCharm

Welcome to PyCharm



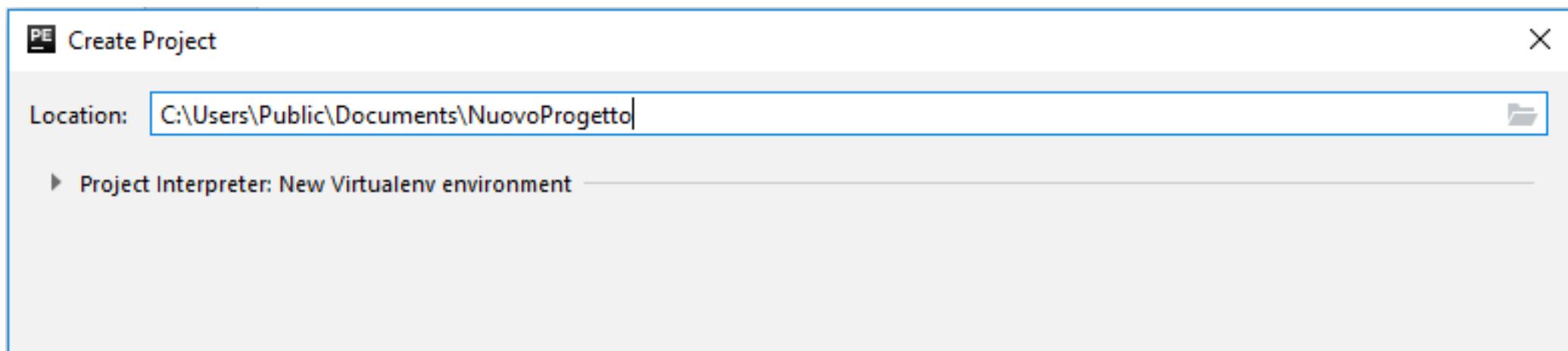
PyCharm
Version 2018.2

- + Create New Project
- 📁 Open
- ⬆ Check out from Version Control ▾

⚙ Configure ▾ Get Help ▾

Creare un progetto

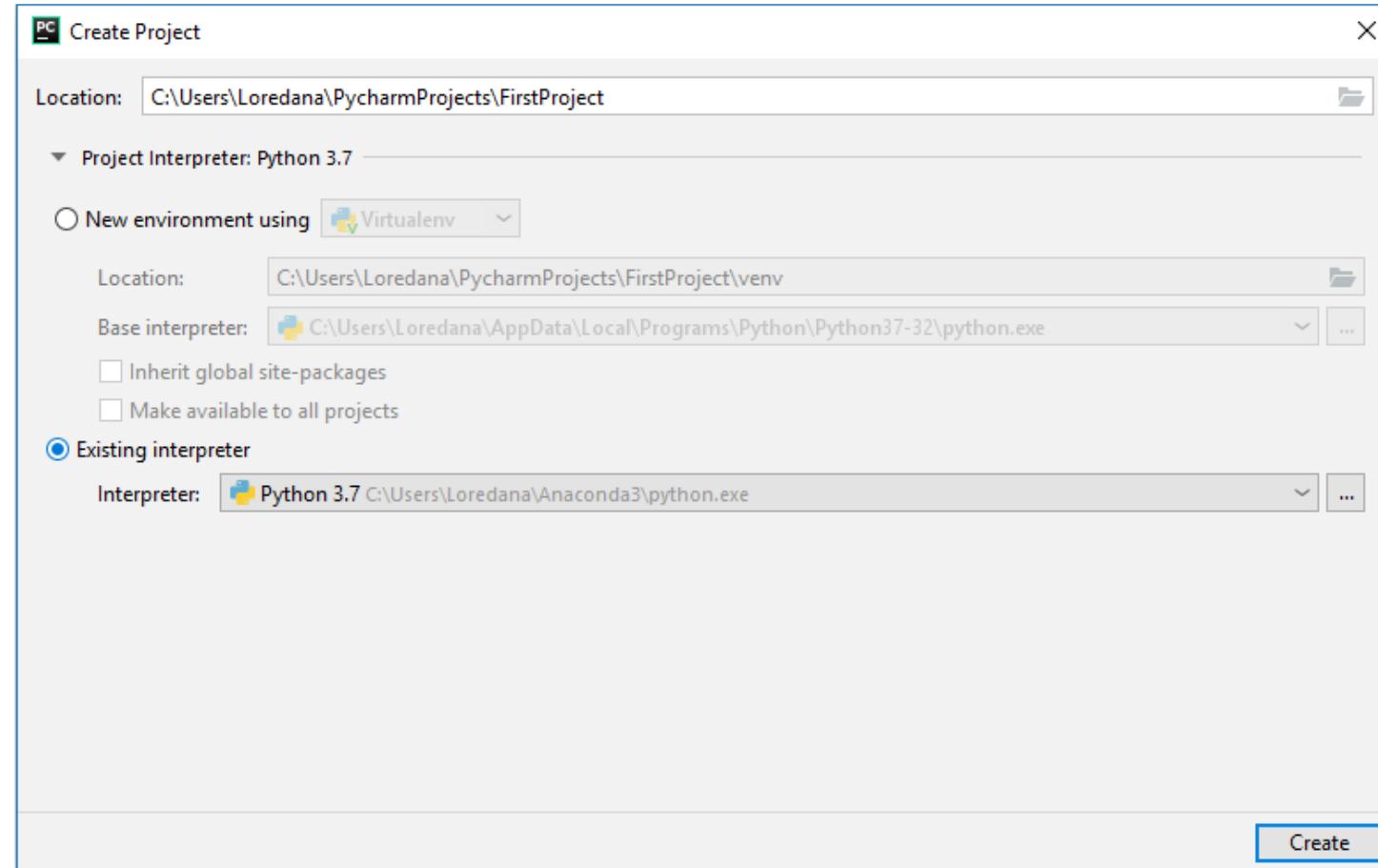
- Creare un **progetto** in **PyCharm**:
 - Seleziona «**Crea nuovo Progetto**», oppure
 - **File -> Nuovo Progetto**
 - Definisci il percorso in cui tutti i tuoi file verranno salvati
 - Clicca su «**Crea**»



Scegli un interprete

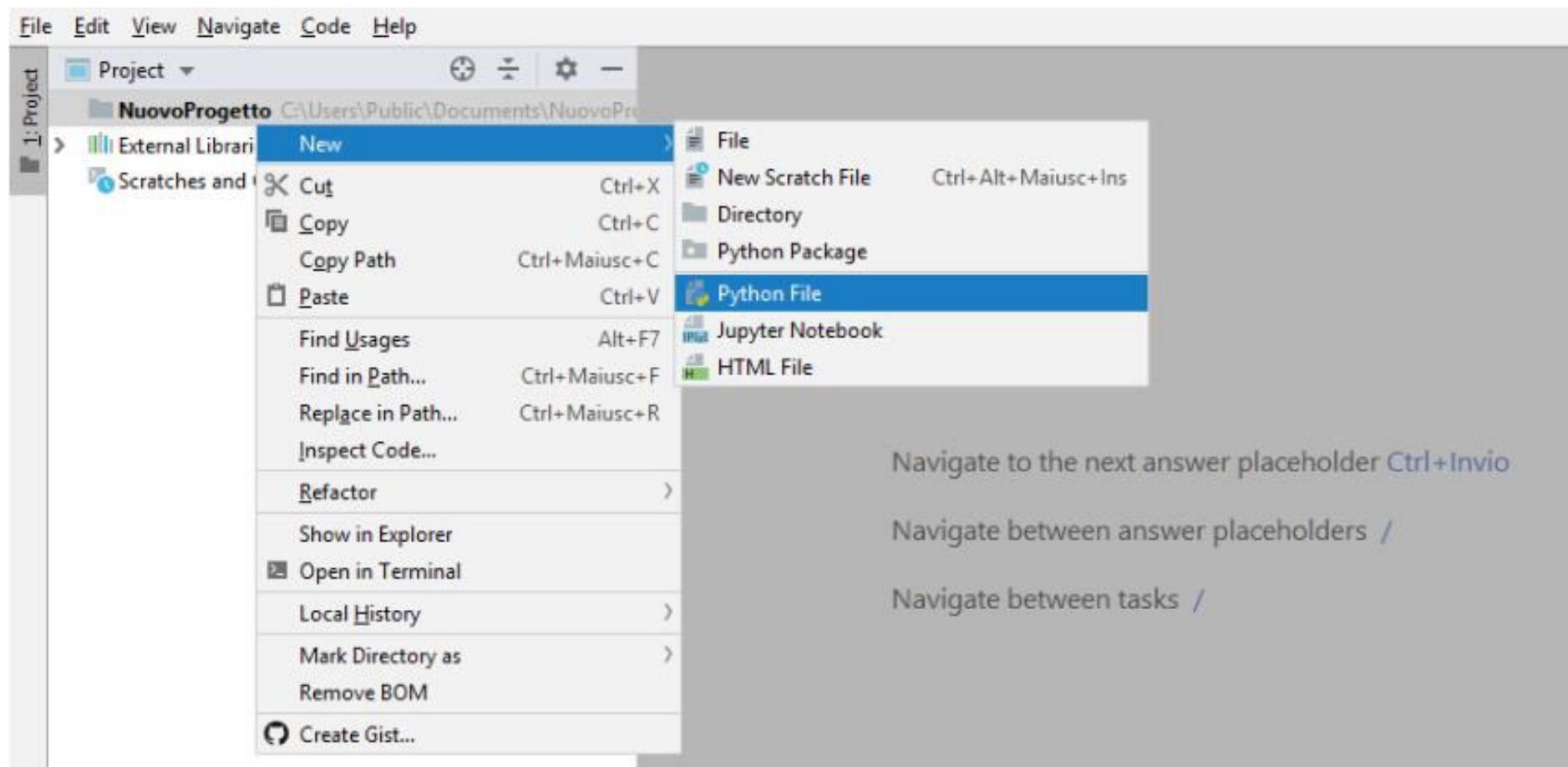
Python è un linguaggio di **script**.

Significa che il codice viene
convertito in codice macchina da
un **interprete** Python.



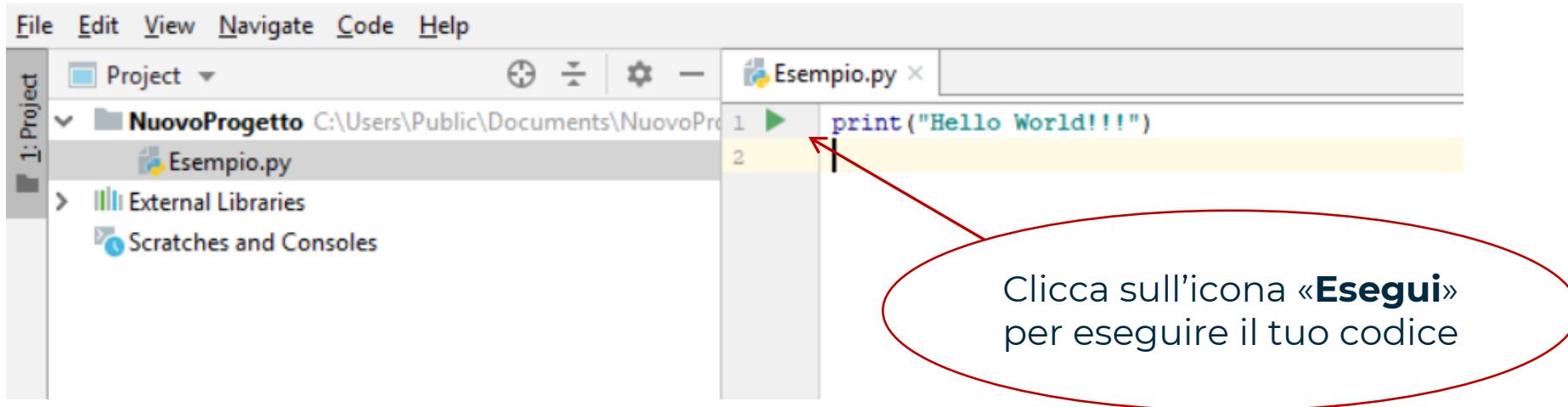
Crea un file python

- Selezionare la **radice** del progetto nella finestra dello strumento **Progetto**
- Definisci il **nome** del file



Modificare il codice sorgente

- Python «Ciao Mondo»:

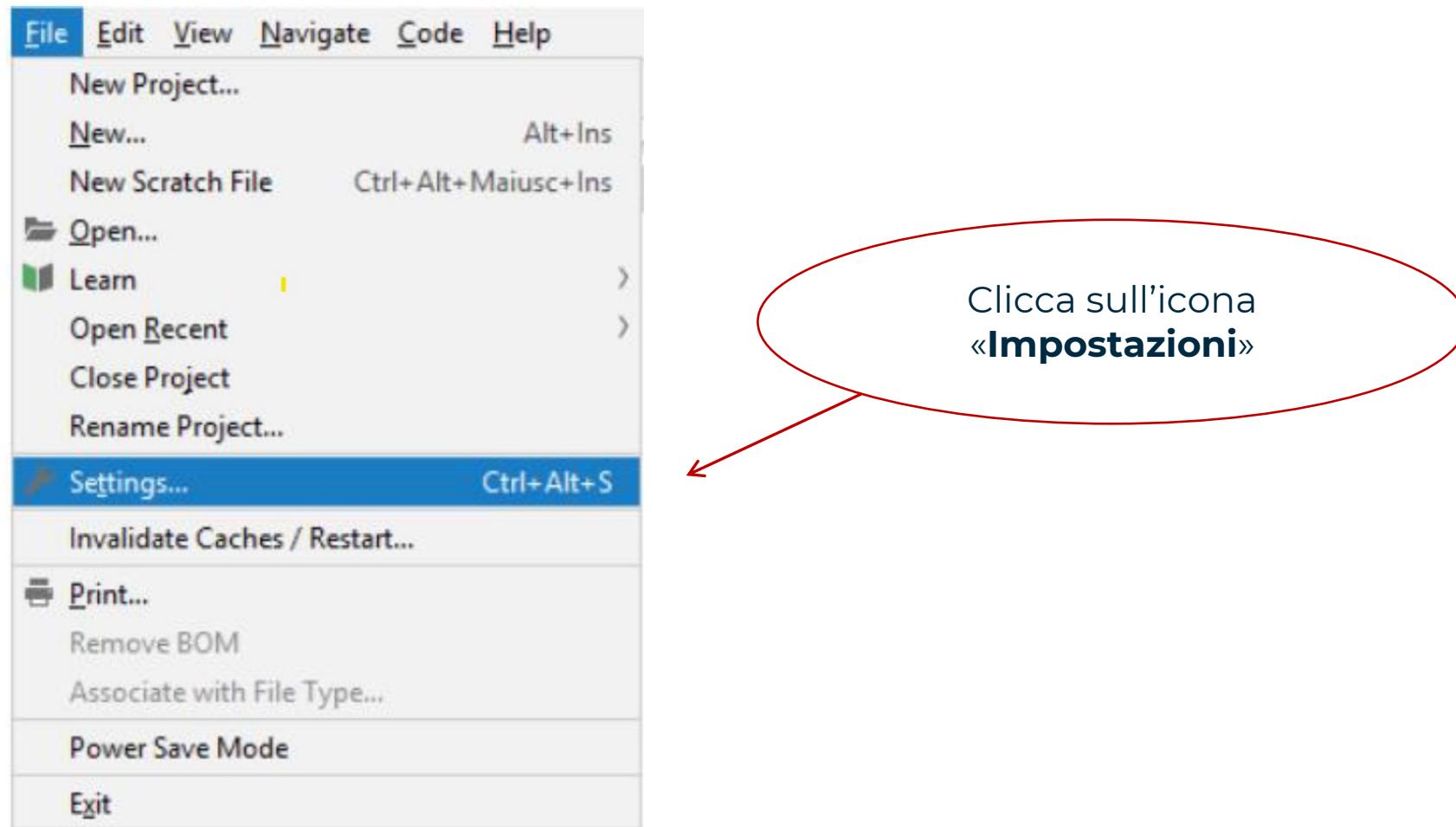


The screenshot shows the 'Run' interface of the IDE. It displays the output of the executed code 'Esempio.py':

```
Hello World!!!
Process finished with exit code 0
```

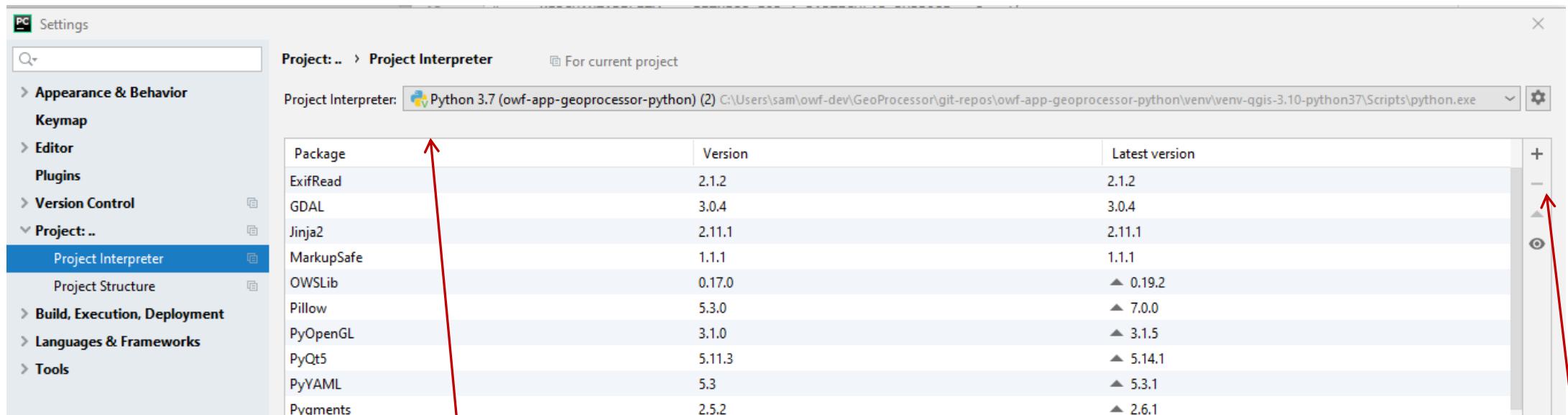
Installazione Librerie (1)

- Installazione delle librerie in **PyCharm**:



Installazione Librerie (2)

- Installazione delle librerie in **PyCharm**:

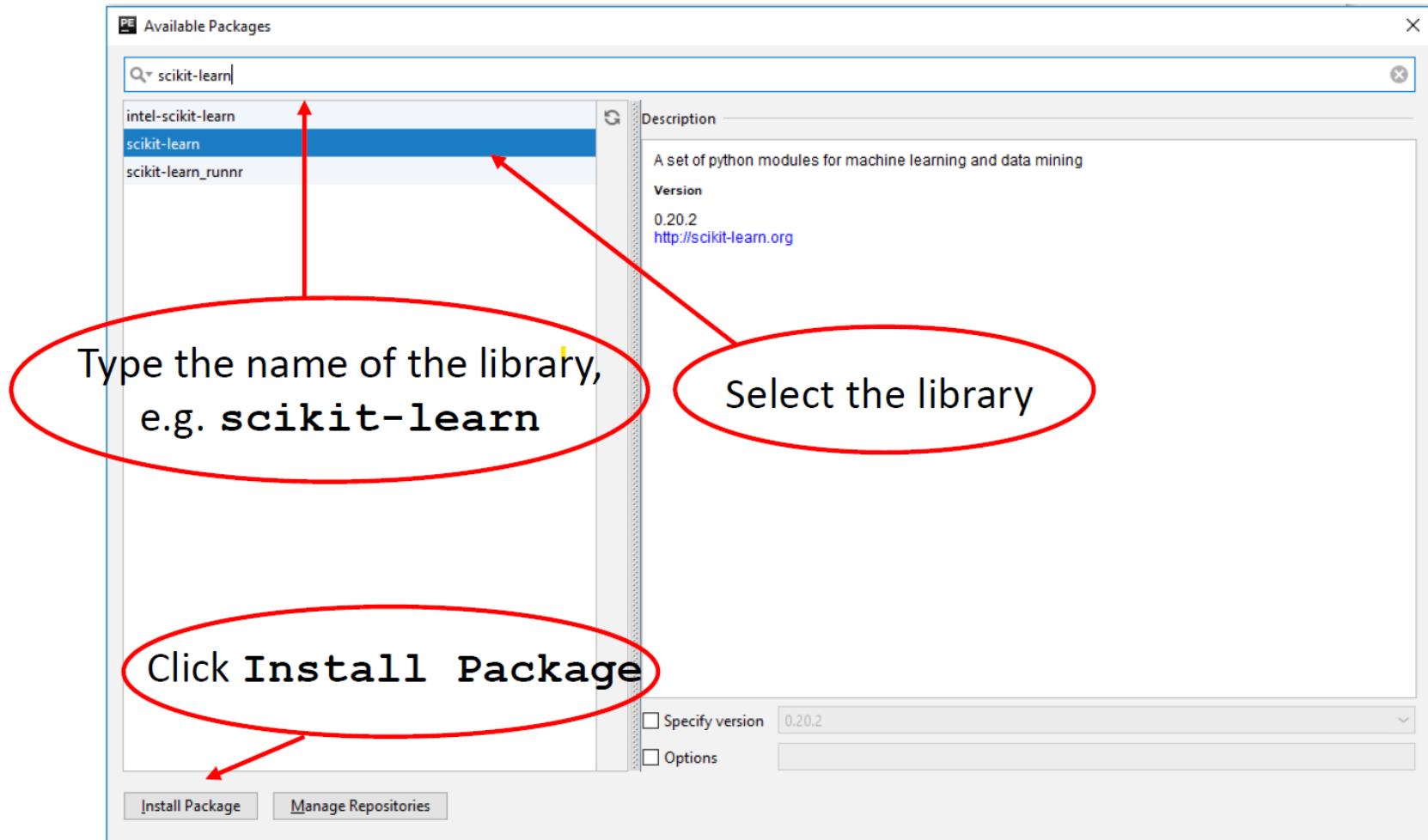


Controlla che **almeno** un
interprete python sia
selezionato

Per aggiungere una
libreria
Clicca sull'icona «+»

Installazione Librerie (3)

- Installazione delle librerie in **PyCharm**:



Python: Le basi

Un esempio di codice

```
x = 34 - 23          # A comment.  
y = "Hello"          # Another one.  
z = 3.45  
if z == 3.45 or y == "Hello":  
    x = x + 1  
    y = y + "World"      # String concat.  
print (x)  
print (y)
```

Identifieri (1)

- Gli identifieri ci aiutano a differenziare un'entità da un'altra
 - Le **entità** Python come **classi**, **funzioni** e **variabili** sono chiamati **identifieri**
- **Nomi degli identifieri**
 - Possono contenere:
 - **Lettere** [a-z/A-Z]
 - **Cifre** [0-9]
 - **Underscore** [_]
 - Non possono iniziare con una cifra

Identifieri (2)

Alcune parole chiave **non possono** essere usate, alcuni esempi

FALSE	class	finally	is	return
none	continue	for	lambda	try
TRUE	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Indentazione

- Una delle caratteristiche più uniche di Python
 - L'uso **dell'indentazione** per differenziare i blocchi di codice

```
# Correct Indentation
x = 1
if x == 1:
    print ('x has a value of 1')
else:
    print ('x does NOT have a value of 1')
```

```
# Wrong indentation in the else statement
x = 1
if x == 1:
    print ('x has a value of 1')
else:
    print ('x does NOT have a value of 1')
```

Commenti

- Ci sono due tipologie di commenti:
 - Commenti **a riga singola**
 - Commenti **su più righe**

```
# This is a single line comment
print ("Hello Python World") # Another one

""" This is an example of
a multiline comment """
```

Istruzione su più righe

- Un'istruzione **Python** può essere divisa in più di una riga
 - Può essere fatto in modo implicito o esplicito
 - Il backslash [\] contrassegna esplicitamente la continuazione
 - **È importante rientrare correttamente la riga continua**

```
# Implicit line continuation
x = ('1' + '2' +
      '3' + '4')
# Explicit line continuation
y = '1' + '2' + \
    '11' + '12'
weekdays = ['Monday', 'Tuesday', 'Wednesday',
            'Thursday', 'Friday']
weekend = {'Saturday',
            'Sunday'}
```

Più istruzioni su una riga

- **Python** consente più istruzioni su una singola riga
 - La divisione delle istruzioni deve essere fatta esplicitamente
 - Il punto e virgola [;] può essere utilizzato per dividere le istruzioni
 - **È importante rientrare correttamente la riga continua**

```
""" Code example for multi-statement on a single
line """
x = 'Hello'; print (x)
```

Oggetti e Operatori

Tipi di oggetti (1)

- Tutti i dati in un programma Python sono rappresentati da oggetti o dalle relazioni tra oggetti

Ogni oggetto ha un'identità, un tipo e un valore

Type	Example	Comment
none	none	# singleton null object
boolean	true, false	
integer	-1, 0, 1, sys.maxint	
long	1L, 9787L	
float	3.141592654 inf, float('inf') # infinity -inf # neg infinity nan, float('nan') # not a number	

Tipi di oggetti (2)

Type	Example	Comment
complex	2+8j	# note use of j
string	"word", 'word'	# use single or double quote
tuple	empty=()	# empty tuple
	(1, true, 'ML')	# unalterable list
list	empty=[]	# empty list
	[1, true, 'ML']	# alterable list
set	empty={}	# empty set
	set(1, True, 'ML')	# alterable set
dictionary	empty={}	
	{'1':'A', '2':'B'}	# alterable object
file	f=open('filename', 'rb')	

Tipi di oggetti (3)

- Quando utilizzare un oggetto specifico
 - **[list]**
 - Hai bisogno di una sequenza ordinata
 - Hai bisogno di raccolte omogenee
 - I valori possono essere modificati più avanti nel programma
 - **[tuple]**
 - È necessaria una sequenza ordinata
 - Sono necessarie raccolte omogenee
 - I valori non possono essere modificati successivamente nel programma

Tipi di oggetti (4)

- Quando utilizzare un oggetto specifico
 - **[set]**
 - Non è necessario archiviare duplicati
 - Non sei preoccupato per l'ordine o gli articoli
 - **[dictionary]**
 - È necessario mettere in relazione i valori con le “chiavi”
 - Cercare i valori in modo efficiente utilizzando una chiave

Operatori di Base

Gli **operatori** sono i **simboli speciali** che possono **manipolare** il valore degli operandi

- Il linguaggio **Python** supporta i seguenti operatori
 - Operatori **aritmetici**
 - Operatori di **confronto o relazionali**
 - Operatori di **assegnazione**
 - Operatori **bit a bit**
 - Operatori **logici**
 - Operatori di **appartenenza**
 - Operatori di **identità**

Operatori Aritmetici

Operator	Description	Example
+	Addition	$x + y = 30$
-	Subtraction	$x - y = -10$
*	Multiplication	$x * y = 200$
/	Division	$y / x = 2$
%	Modulus	$y \% x = 2$
**	Exponentiation	$x ** b = 10^{20}$
//	Integer division rounded toward $-\infty$	$-11 // 3 = -4$ $9 // 2 = 4$

Operatori di confronto o relazionali

Operator	Description	Example
<code>==</code>	Evaluates the equality	$(x==y)$ is not true
<code>!=</code>	Evaluates the diversity	$(x \neq y)$ is true
<code><></code>	Evaluates the diversity	$(x <> y)$ is true
<code>></code>	Evaluates the majority	$(x > y)$ is not true
<code><</code>	Evaluates the minority	$(x < y)$ is true
<code>>=</code>	Evaluates the majority or the equality	$(x \geq y)$ is not true
<code><=</code>	Evaluates the minority or the equality	$(x \leq y)$ is true

Operatori di assegnazione

Operator	Description	Example
=	Basic assignment	$z=x+y$
+=	Addition and assignment	$z+=x$ (equivalent to $z=z+x$)
-=	Subtraction and assignment	$z-=x$ (equivalent to $z=z-x$)
=	Multiplication and assignment	$z^=x$ (equivalent to $z=z^*x$)
/=	Division and assignment	$z/=x$ (equivalent to $z=z/x$)
%=	Modulus and assignment	$z\%=x$ (equivalent to $z=z \% x$)
=	Exponentiation and assignment	$z^{}=x$ (equivalent to $z=z^{**}x$)
//=	Integer division rounded toward $-\infty$ and assignment	$z//=x$ (equivalent to $z=z//x$)

Operatori bit a bit e logici

Operator	Description	Example
&	Binary AND	(x&y)
	Binary OR	(x y)
^	Binary XOR	(x^y)
~	Binary Ones Complement	(~x)
<<	Binary Left Shift	x<<2
>>	Binary Right Shift	X>>2

Operator	Description	Example
and	Logical AND	(var1 and var2)
or	Logical OR	(var1 or var2)
not	Logical NOT	not(var1 and var2)

Operatori di appartenenza e di identità

Operator	Description	Example
in	Results TRUE if a value is in the sequence	var1 in var2
not in	Results TRUE if a value is not in the sequence	var1 not in var2

Operator	Description	Example
is	Results TRUE for the same objects	var1 is var2
is not	Results TRUE for different objects	var1 is not var2

```
# Example code
var1 = 1
var2 = 2
var3 = [1,3,5]
print (var1 is not var2)
print (var1 in var3)
print (var2 not in var3)
```

Strutture di Controllo

Strutture di controllo

- Una **struttura di controllo** è la scelta fondamentale o il processo decisionale nella programmazione
- Una struttura di controllo è una **porzione di codice** che analizza i valori delle variabili e **decide una direzione** da seguire in base a una determinata condizione
- In Python esistono principalmente due tipi di strutture di controllo:
 - **Selezione**
 - **Iterazione**

Selezione (1)

- Esistono due versioni del costrutto di selezione:
 - **if**
 - **if... else**

```
# Example code for a simple 'if' statement
var = -1
if var < 0:
    print (var)
    print("the value of var is negative")
```

""" If there is only a single clause then it
may go on the same line as the header statement """
if (var == -1): print("the value of var is negative")

Selezione (2)

```
# Example code for the 'if else' statement
var = 1

if var < 0:
    print("the value of var is negative")
    print (var)
else:
    print("the value of var is positive")
    print (var)
```

Selezione (3)

```
# Example code for nested if else statements
Score = 95

if score >= 99:
    print("A")
elif score >= 75:
    print("B")
elif score >= 60:
    print("C")
elif score >= 35:
    print("D")
else:
    print("F")
```

Iterazione (1)

- Python fornisce due istruzioni di ciclo essenziali:
 - **for**
 - **while**
- **[for]**
 - Permette di eseguire un blocco di codice per un numero specifico di volte o contro una condizione specifica finché non viene soddisfatta

```
# First example of a 'for loop' statement
print("First Example")
for item in [1,2,3,4,5]:
    print('item:', item)
```

Iterazione (2)

```
# Second example of a 'for loop' statement
print("Second Example")
letters = ['A', 'B', 'C']
for letter in letters:
    print('First loop letter:', letter)

# Third Example - Iterating by sequence index
print("Third Example")
for index in range(len(letters)):
    print('First loop letter:', letters[index])

# Fourth Example - Using else statement
print("Fourth Example")
for item in [1,2,3,4,5]:
    print('item:', item)
else:
    print('looping over item complete!')
```

Iterazione (3)

- **[while]**

- L'istruzione while ripete un insieme di codice fino a quando la condizione non risulterà vera

```
# Example code for while loop statement
count = 0
while (count < 3):
    print('The count is:', count)
    count = count + 1

# Example code for a 'while with a else' statement
count = 0
while count < 3:
    print(count, 'is less than 3')
    count = count + 1
else:
    print(count, 'is not less than 3')
```

Semantica di riferimento (1)

- L'assegnazione manipola i riferimenti
 - **x=y** non crea una copia dell'oggetto a cui fa riferimento **y**
 - **x=y** fa sì che **x** faccia riferimento all'oggetto a cui fa riferimento **y**
- Tipi di dati:
 - **integers** (Numeri Interi)
 - **floats** (Numeri in virgola mobile)
 - **strings** (Stringhe)

```
x = 3          # Creates 3, name x refers to 3
y = x          # Creates name y, refers to 3
y = 4          # Creates ref for 4. Change y
print(x)       # No effects on x, still ref 3
```

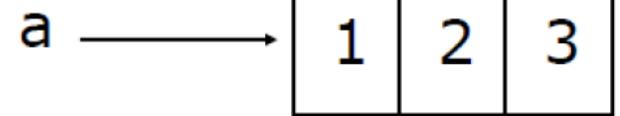
Semantica di riferimento (2)

- Altri tipi di dati:

- **list** (Liste)
- **Dictionary** (Dizionari)

- **Tipi definiti dall'utente**

`a = [1, 2, 3]`

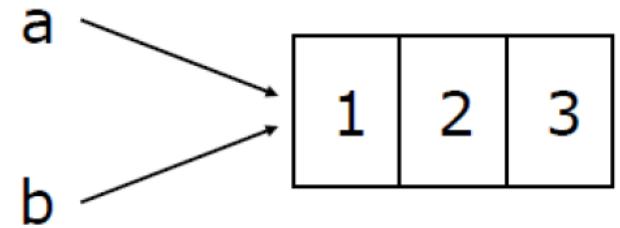


- Con questa tipologia di dati,

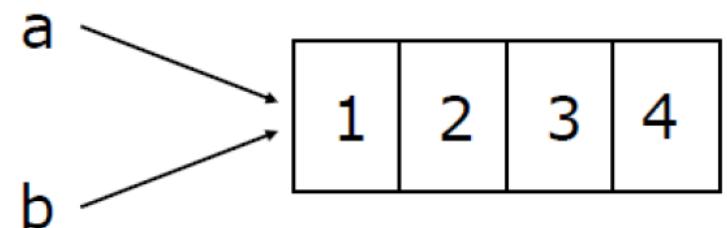
l'assegnazione è differente

- Questi dati sono «**mutabili**»

`b = a`



`a.append(4)`



Liste

- Le **liste** di Python sono il tipo di dati più **flessibile**
 - Possono essere creati scrivendo un elenco di valori separati da virgole tra parentesi quadre
 - Non è necessario che gli elementi nell'elenco siano dello stesso tipo di dati

```
# Create lists
list_1 = ['Statistics', 'Programming', 2016, 2017, 2018]
list_2 = ['a', 'b', 1, 2, 3, 4, 5, 6, 7]

# Accessing values in lists
print("list_1[0]: ", list_1[0])
print("list_2[1:5]: ", list_2[1:5])
```

----- output -----

```
list_1[0]: Statistics
list2_2[1:5]: ['b', 1, 2, 3]
```

Aggiunta e aggiornamento di valori

```
# Adding new value to list
list_1 =['c', 'b', 'a', 3, 2, 1]
print("list_1 values: ", list_1)
list_1.append(2019)
print("list_1 values post append: ", list_1)

----- output -----
list_1 values:  ['c', 'b', 'a', 3, 2, 1]
list_1 values post append:  ['c', 'b', 'a', 3, 2, 1, 2019]
```

```
# Updating existing value of list
print("list_1 values: ", list_1)
print("Index 2 value: ", list_1[2])
list_1[2]= 2015
print("Index 2's new value : ", list_1[2])

----- output -----
```

```
Values of list_1:  ['c', 'b', 'a', 3, 2, 1, 2019]
Index 2 value :  a
Index 2's new value :  2015
```

Rimozione di valori

```
# Deleting list elements
list_1 =['c' , 'b' ,2015 ,3 ,2 ,1 ,2019]
print("list_1 values: ", list_1)
del list_1[5]
print("After deleting value at index 5: ", list_1)

----- output -----
list_1 values:  ['c', 'b', 2015, 3, 2, 1, 2019]
After deleting value at index 5:  ['c', 'b', 2015, 3, 2, 2019]
```

Operazioni di base con le liste (1)

```
# Example code
list_1 =['c' , 'b' ,3,2,1]
print("Length: ", len(list_1))
print("Concatenation: ", [1,2,3] + [4,5,6])
print("Repetition: ", ['Hello'] * 4)
print("Membership: ", 3 in [1,2,3])
print("Iteration: ")
for x in [1,2]: print(x)
```

----- output -----

```
Length: 5
Concatenation: [1, 2, 3, 4, 5, 6]
Repetition : ['Hello', 'Hello', 'Hello', 'Hello']
Membership : True
Iteration :
1
2
```

Operazioni di base con le liste (2)

```
list_1 =['Statistics', 'Programming', 2015, 2017, 2018]
# Negative sign will count from the right
print("slicing: ", list_1[-2])
""" If you don't specify the end explicitly, all
elements from the specified start index will be printed
"""

print("slicing range: ", list_1[1:])
# Comparing elements of lists
print("Compare two lists: ", cmp([1,2,3,4], [1,2,3]))
print("Max of list: ", max([1,2,3,4,5]))
print("Min of list: ", min([1,2,3,4,5]))
```

----- output -----

```
slicing : 2017
slicing range: ['Programming', 2015, 2017, 2018]
Compare two lists: 1
Max of list: 5
Min of list: 1
```

Operazioni di base con le liste (3)

```
# Example code
print("Count number of 1: ", [1,1,2,3,4,5].count(1))
list_1 =[ 'Statistics' , 'Programming' ,2015,2017,2018]
list_2 =[ 'a' , 'b' ,1,2,3,4,5,6,7]
list_1.extend(list_2)
print("Extended: ", list_1)
print("Index for : ", list_1.index("Programming"))
print(list_1)
print("pop last item in list: ", list_1.pop())
print("pop the item with index 2: ", list_1.pop(2))

----- output -----
Count number of 1 in list: 2
Extended : ['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2,
3, 4, 5, 6, 7]
Index for Programming : 1
['Statistics', 'Programming', 2015, 2017, 2018, 'a', 'b', 1, 2, 3, 4, 5, 6, 7]
pop last item in list: 7
pop the item with index 2: 2015
```

Operazioni di base con le liste (4)

Example code

```
list_1 = ['Statistics', 'Programming', 2017,  
         2018, 'a', 'b', 1, 2, 3, 4, 5, 6]
```

```
list_1.remove("b")
```

```
print("removed b from list: ", list_1)
```

```
list_1.reverse()
```

```
print("Reverse: ", list_1)
```

```
list_1 = ['a', 'b', 'c', 1, 2, 3]
```

```
list_1.sort()
```

```
print("Sort ascending: ", list_1)
```

```
list_1.sort(reverse=True)
```

```
print("Sort descending: ", list_1)
```

---- output ----

```
removed b from list: ['Statistics', 'Programming', 2017, 2018, 'a', 1, 2,  
3, 4, 5, 6]
```

```
Reverse: [6, 5, 4, 3, 2, 1, 'a', 2018, 2017, 'Programming', 'Statistics']
```

```
Sort ascending: [1, 2, 3, 'a', 'b', 'c']
```

```
Sort descending: ['c', 'b', 'a', 3, 2, 1]
```

List Comprehension vs ciclo for

- **Separa** le lettere della parola «human» e **aggiungi** le lettere come elementi di una **lista**

```
# For Loop code
h_letters = []
for letter in 'human':
    h_letters.append(letter)
print(h_letters)
```

```
# List Comprehension
h_letters = [ letter for letter in 'human' ]
print(h_letters)
```

----- output -----

```
['h', 'u', 'm', 'a', 'n']
```

List Comprehension

- Sintassi

[expression for item in list]

[expression for item in list]
[letter for letter in 'human']

- **List Comprehension:** identifica l'azione di ricevere una stringa o una tupla da una lista e applicare operazioni su di essa
- Non tutti i cicli possono essere riscritti come List Comprehension

Condizioni con List Comprehension

- List Comprehension può essere usata per applicare istruzioni condizionali al fine di modificare l'elenco esistente

```
# List Comprehension with conditionals
number_list = [ x for x in range(20) if x % 2 == 0]
print(number_list)

# List Comprehension with nested if
num_list = [ y for y in range(100) if y%2==0 if y%5==0]
print(num_list)

# List Comprehension with if...else
obj = ['Even' if i%2==0 else 'Odd' for i in range(10)]
print(obj)
```

