Introduzione al Linguaggio Python

Federico Bolelli
federico.bolelli@unimore.it

Cos'è Python?

- Linguaggio di programmazione ad oggetti;
- Linguaggio di alto livello (C++ / Java);
- Linguaggio interpretato;
- Prototipazione veloce;
- Gestione automatica della memoria;
- Sintassi semplice;
- Tipizzazione dinamica;
- Portabilità;

www.python.it

On Platform

- Indipendente dalla piattaforma;
- Interprete scritto in C;
- Disponibile per tutte le piattaforme;
- Open Source
- Versioni disponibili 2.7.x - 3.7.x;

- **Utilizzeremo la versione 3.6**

Materiale Utile

- How to Think Like a Computer Scientist, Allen Downey Jeffrey Elkner Chris Meyers, Green Tea Press
  http://www.greenteapress.com/thinkpython/thinkCSpy.pdf

- Pensare da informatico, Allen Downey Jeffrey Elkner Chris Meyers, Green Tea Press
  https://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwjp0ey754vfAhWIposKHZCEAVgQFjAAegQICBAC&url=http%3A%2F%2Fwww.python.it%2Fdoc%2FHowtothink%2FHowToThink_ITA.pdf.gz&usg=AOvVaw0HZS7xER--MQ5yMI2a1KIl

- A WhirlWind Tour of Python, Jake VanderPlas, O'REILLY
  https://s3-us-west-2.amazonaws.com/python-notes/a-whirlwind-tour-of-python-2.pdf

Interprete

- Python dispone di un interprete interattivo molto comodo e potente:
  - Avvio: digitare python al prompt di una shell
  - Appare così il prompt >>> pronto a ricevere comandi. Possiamo a questo punto inserire qualsiasi costrutto che verrà interpretato al volo:

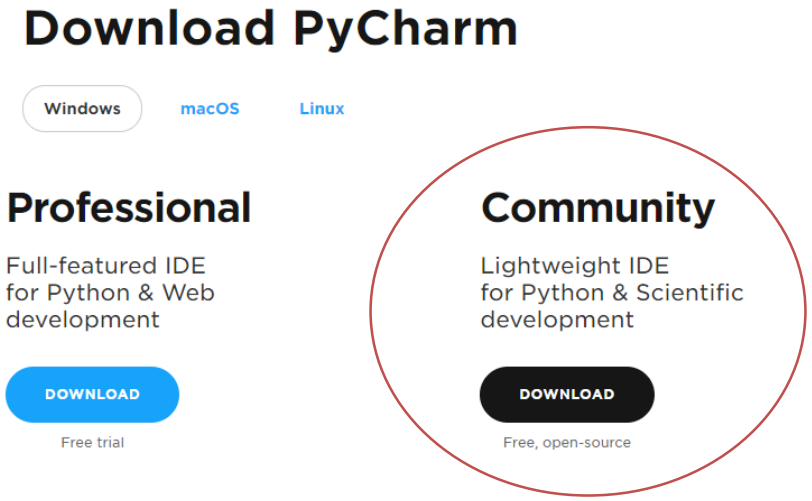
```
>>> 3+5
8
>>> “Hello World!”
Hello World
```

Interprete

- L’interprete è un file denominato:
  - “python” su Unix
  - “python.exe” su Windows
- Se invocato senza argomenti presenta un’interfaccia interattiva;
- Può essere seguito dal nome di file contenente comandi Python. In tal caso il file verrà interpretato ed eseguito.
- I file sorgente Python sono file di testo, generalmente con estensione “.py”

PyCharm

- **PyCharm** è un IDE (ambiente di sviluppo) per Python;
- La versione Community (gratuita) del software si può scaricare a questo link: www.jetbrains.com/pycharm/download



PyCharm Portable

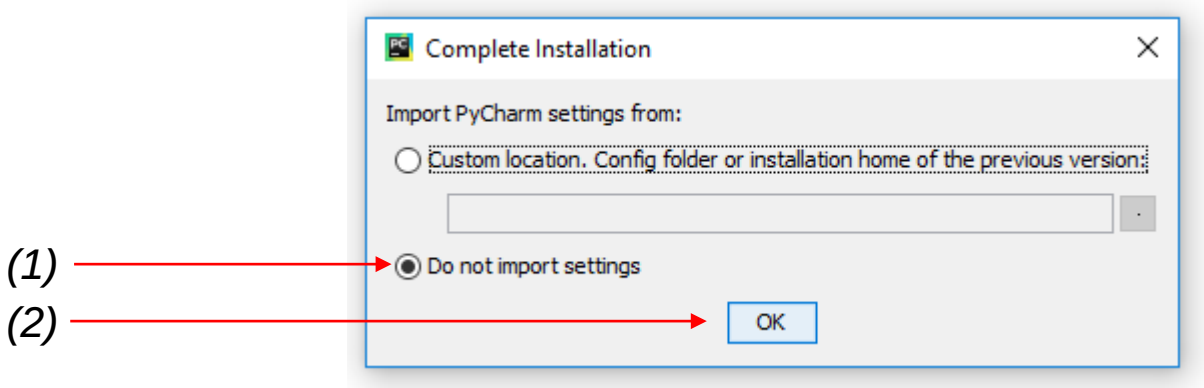
- Se non volete/potete installare programmi sul vostro PC è disponibile una versione portabile a questo link:

  https://drive.google.com/file/d/1RHRdpVg3nRsDT5CItP0dBSxRQBeQV9-P

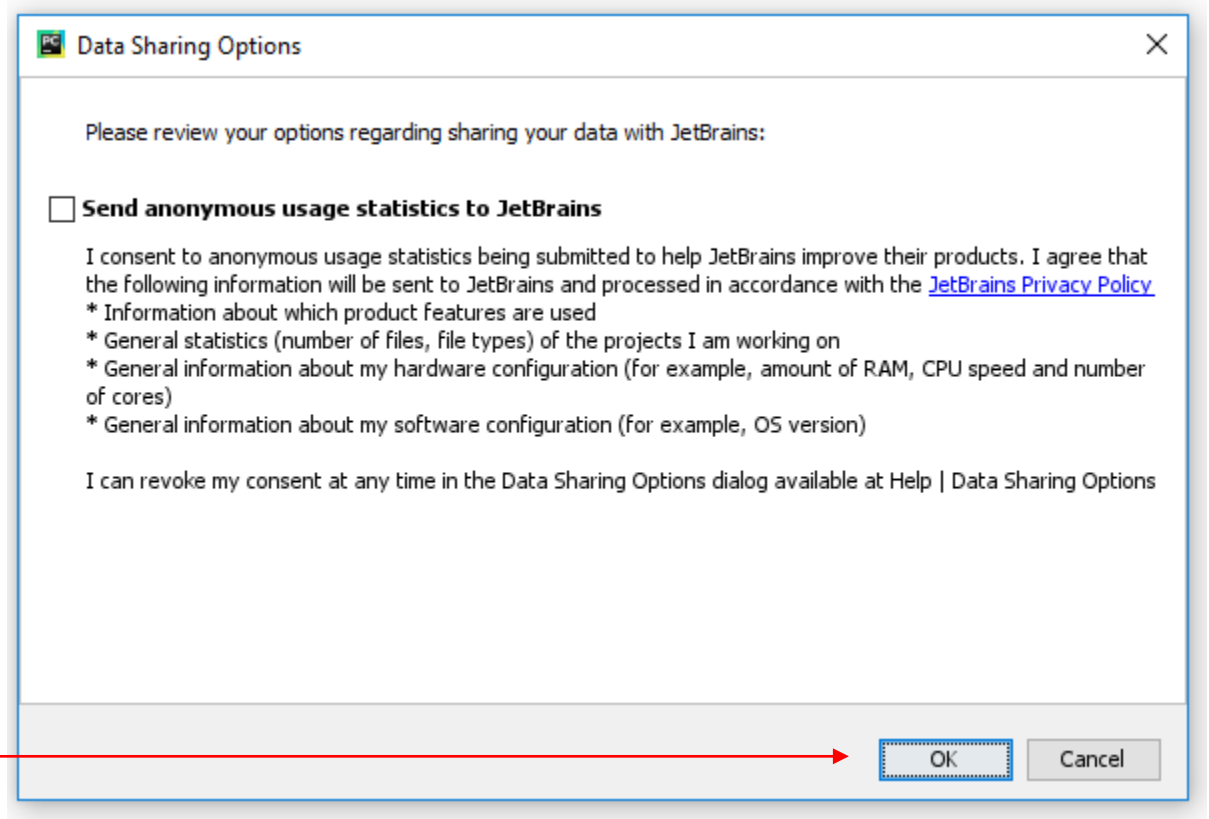
- Dopo aver scaricato la cartella *pycharm.zip* occorre scompattarla. Al suo interno trovate due sottocartelle:
  - Miniconda3-4.5.1-Windows-x86: contiene una versione *portable* dell'interprete Python;
  - PyCharmPortable: contiene la versione *portable* di PyCharm

Come Configurare PyCharm Portable

- Doppio click sul l'eseguibile "PyCharmPortable.exe":

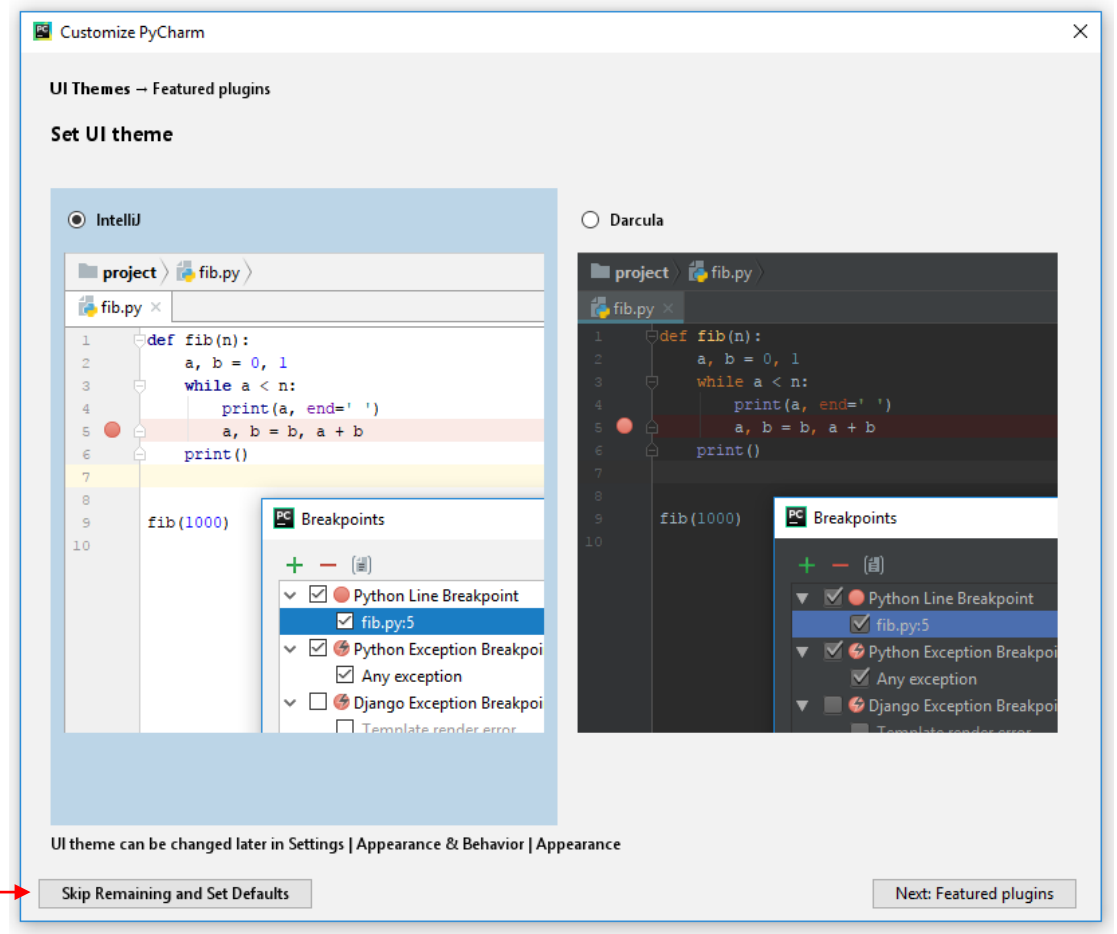


Come Configurare PyCharm Portable



Come Configurare PyCharm Portable

- Scegliete il tema che preferite (*IntelliJ* o *Darcula*) e cliccate su "*Skip Remaining and Set Defaults*"

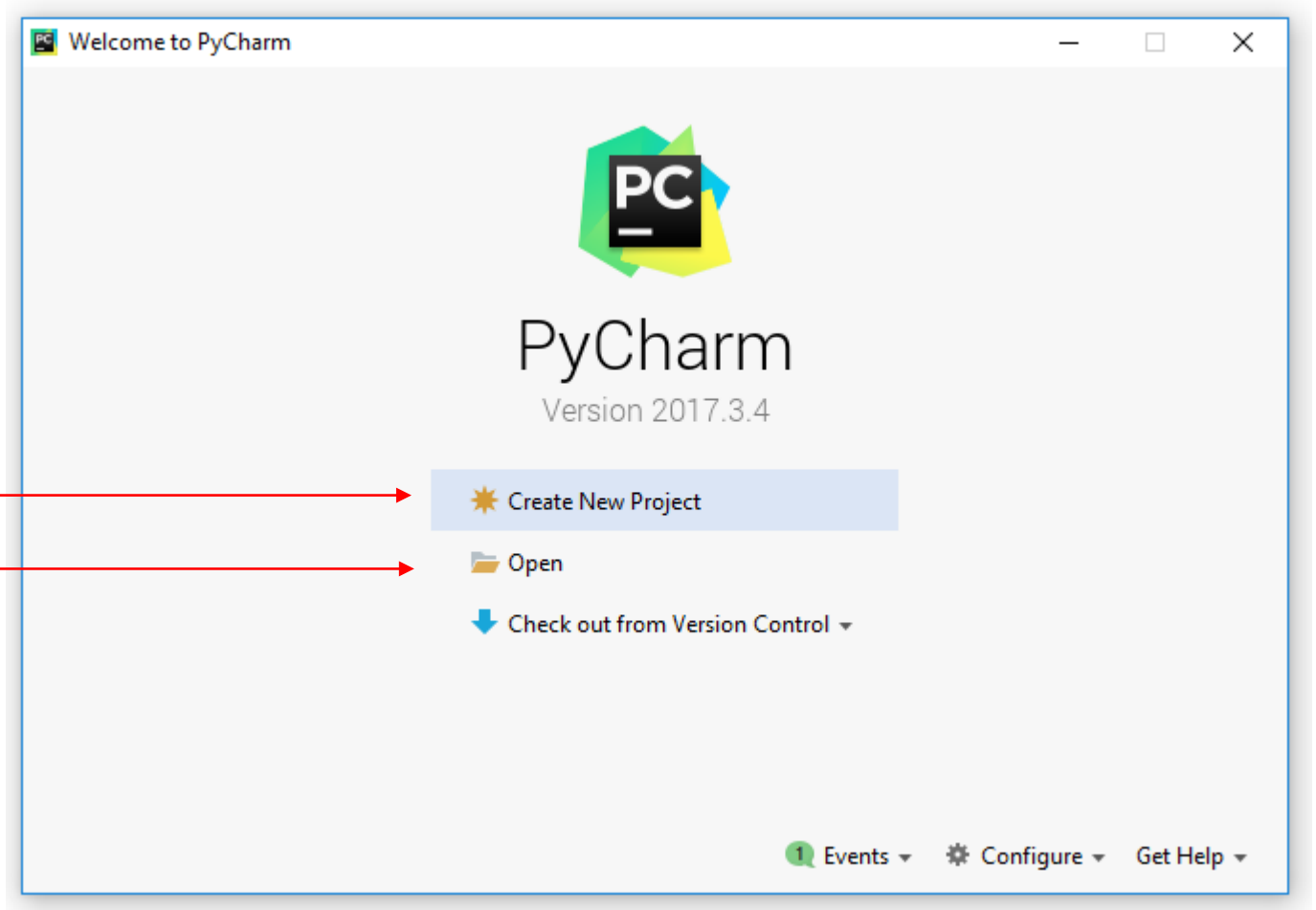


Creazione di un Progetto PyCharm

- Create quindi un nuovo progetto o apritene uno esistente:

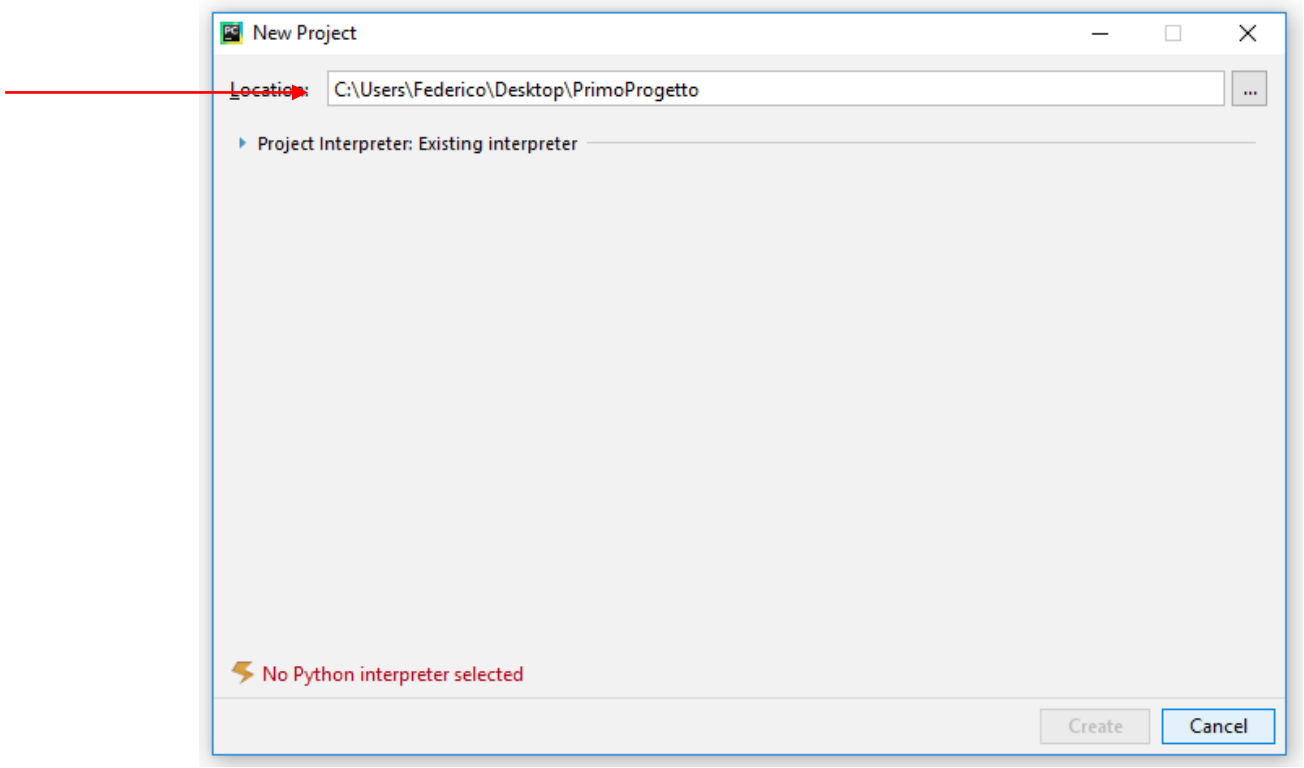
Crea Nuovo

Apri Esistente



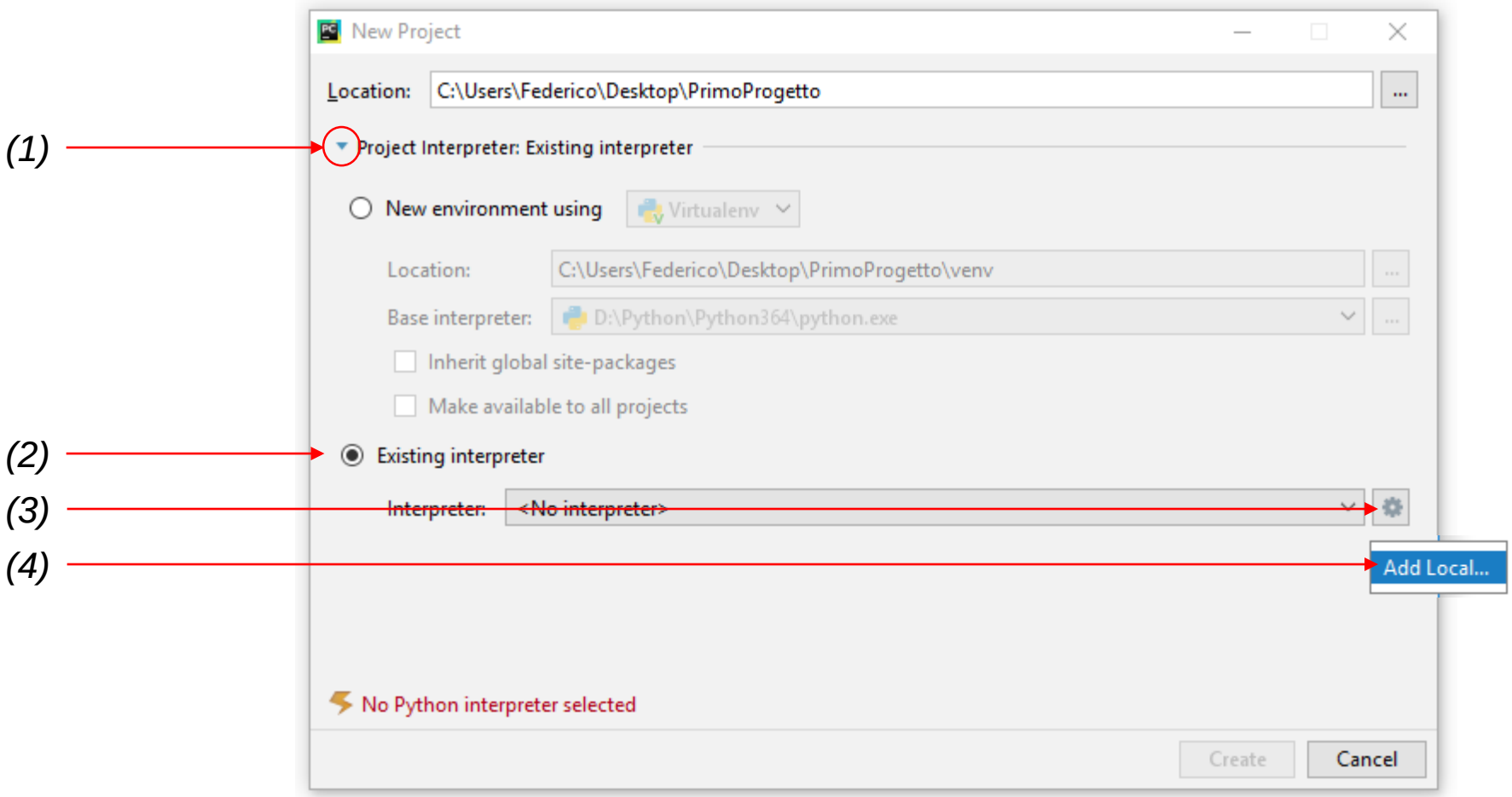
Creazione di un Progetto PyCharm

- Se state creando un nuovo progetto dovete specificare il percorso in cui volete crearlo. Consiglio: create una cartella sul *Desktop* e selezionate quella come *Location* del nuovo progetto.



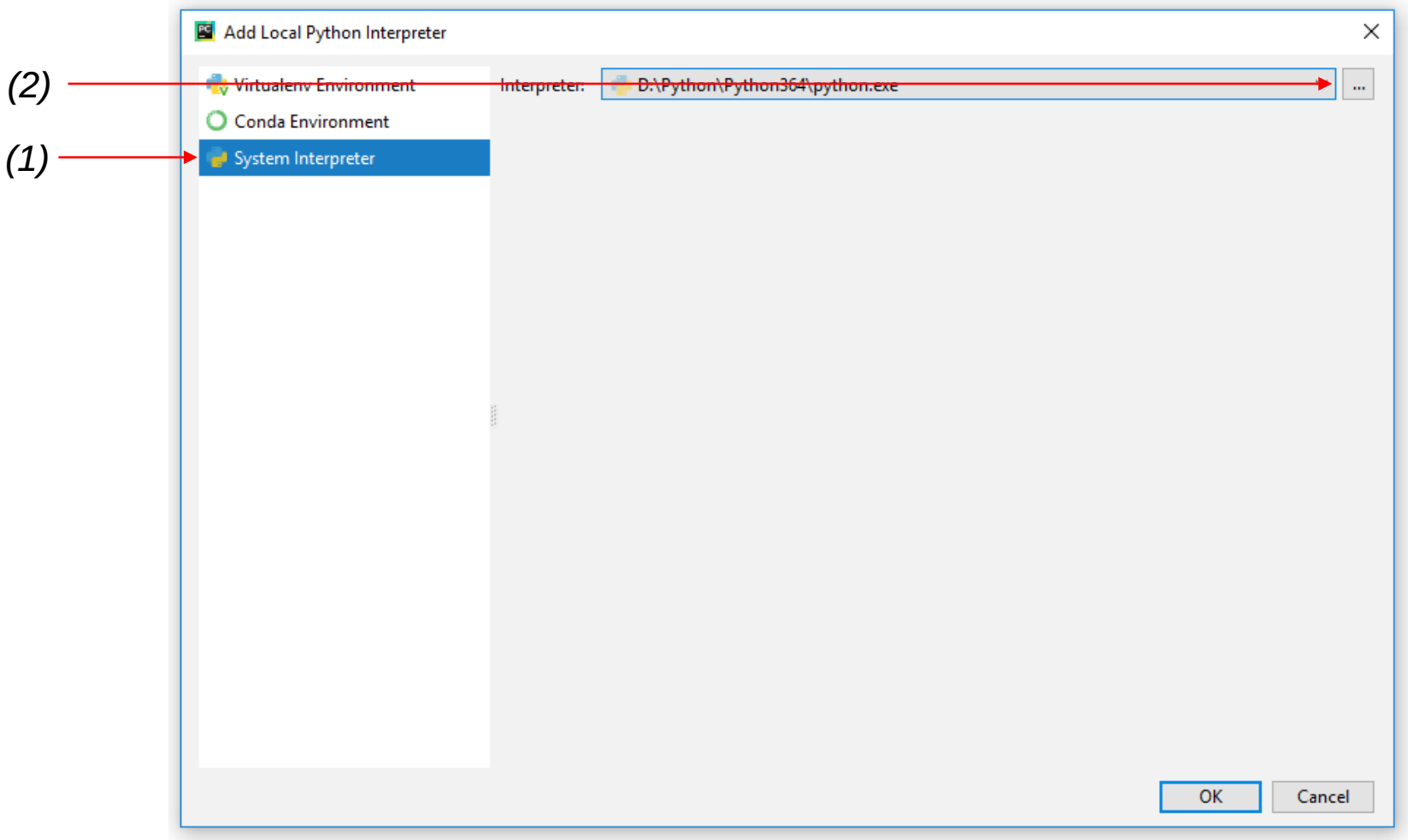
Creazione di un Progetto PyCharm

- Espandete il menu a tendina *Project Interpreter*, *selezionate Existing interpreter*, quindi cliccate su ⚙ e poi su *Add Local*:



Creazione di un Progetto PyCharm

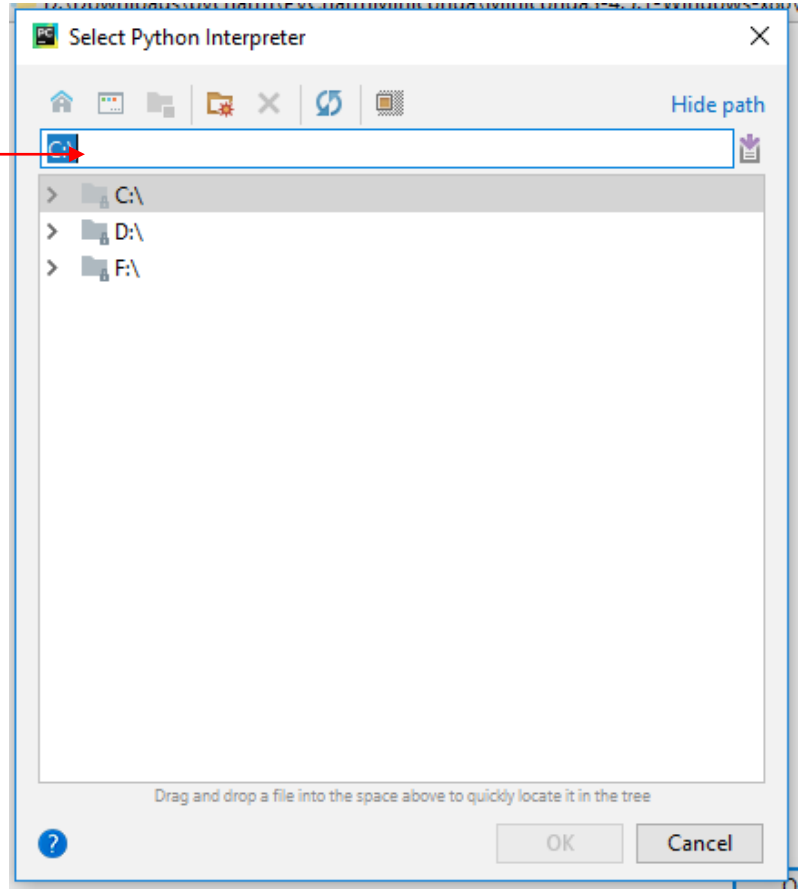
- Cliccate su *System Interpreter* e quindi su ...:



Creazione di un Progetto PyCharm

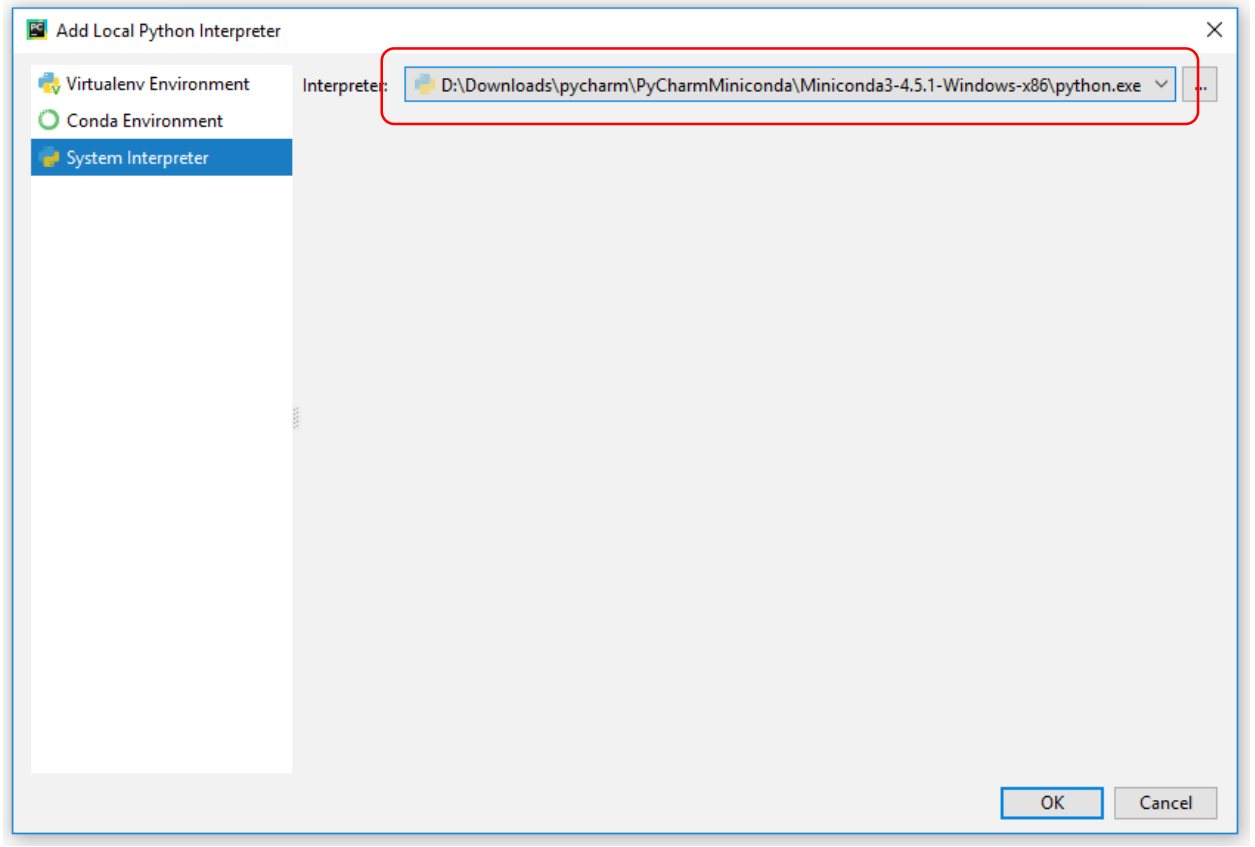
- A questo punto occorre specificare il percorso dell'*interpreter*. Vi ricordate la cartella *Miniconda3-4.5.1-Windows-x86?* Al suo interno troverete un file *python.exe*, quello è il percorso da specificare

Cercate l'eseguibile *python.exe* e scrivete il suo percorso qui, quindi cliccate su *ok*



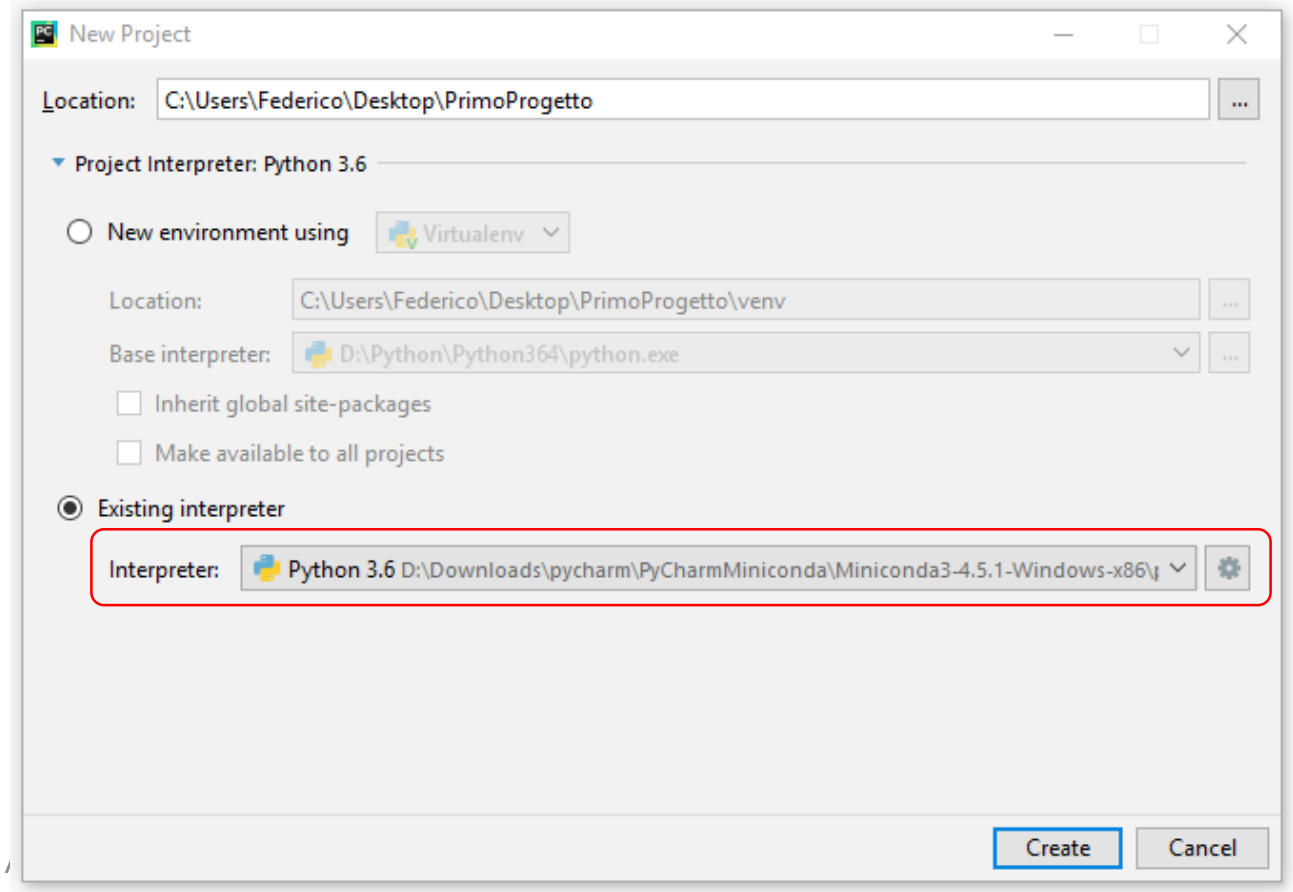
Creazione di un Progetto PyCharm

- Dovreste trovarvi nella situazione illustrata sotto. Il percorso specificato al passo precedente dovrebbe comparire nell'apposito riquadro. Cliccate quindi su *ok* e procedete.



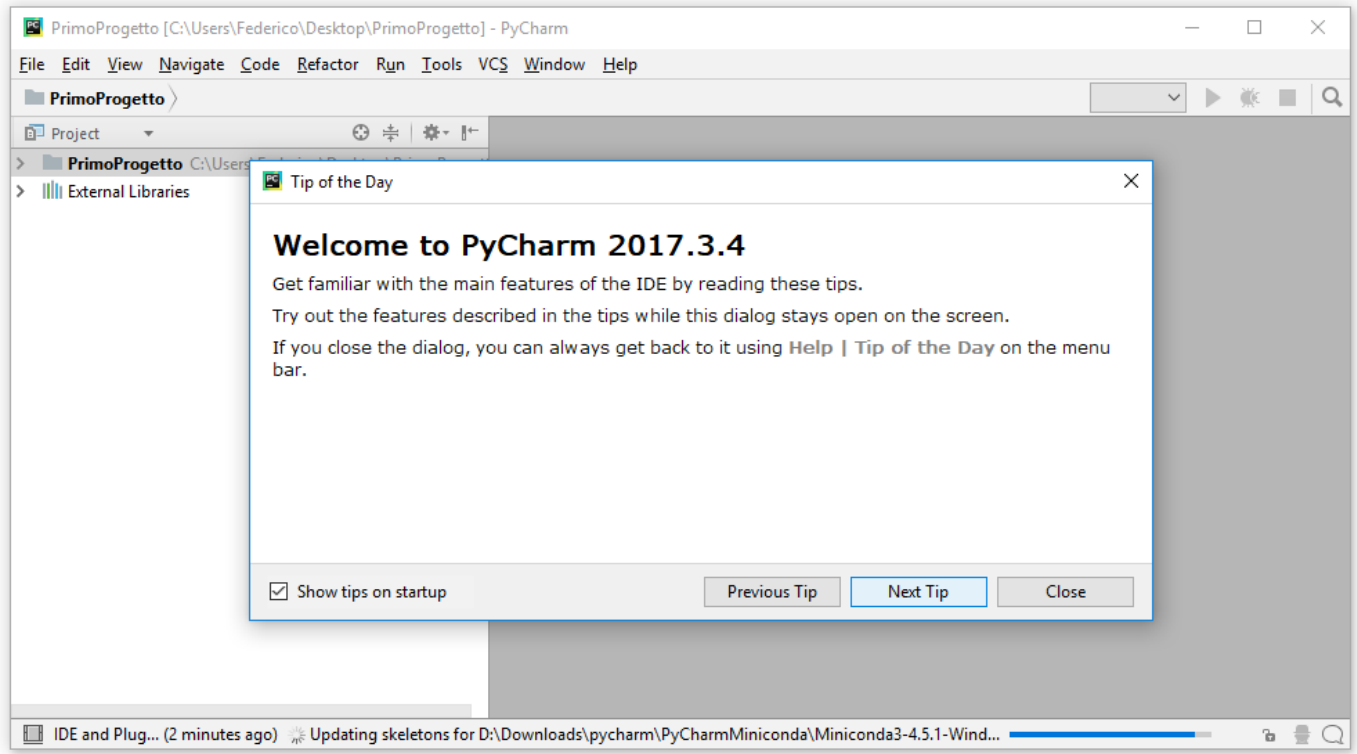
Creazione di un Progetto PyCharm

- Se tutte le operazioni sono state eseguite correttamente dovreste visualizzare la finestra riportata di seguito. Il percorso nel riquadro dovrebbe essere simile a quello illustrato. Cliccate quindi su *Create.*



Creazione di un Progetto PyCharm

- Il progetto è stato creato impostando correttamente l'*interpreter.* Cliccate su *Close* per chiedere lo finestra dei suggerimenti e iniziate ad usare PyCharm:



Un Tour Veloce della Sintassi Python

- "#" identifica un commento:

```
# Questo è un commento
x = 2  # Un commento può anche seguire uno statement del linguaggio
```

- Mancanza di punti e virgola ";":

```
x = 2
y = 5
```

- Per mandare a capo uno statement posso usa "\" o "()" :

```
x = (3 + 4
     + 2)
y = 7 + 8 \
    + 2
```

Un Tour Veloce della Sintassi Python

- Operatore di accesso a moduli / metodi “.”:

```
my_list = []
my_list.append(8)
```

- A differenza della maggior parte degli altri linguaggi di programmazione Python non usa le parentesi graffe “{ }” per identificare blocchi di codice. Tutto si base su “:” e “Indentazione”:

```
if a == 6:
    # L’indentazione identifica un blocco di codice
    x = 2
    y = 5
z = 8
```

Un Tour Veloce della Sintassi Python

- Gli spazi bianchi all'interno di una linea non hanno significato:

```
x=1+2
x = 1 + 2
x    =      1       +       2
```

- Le parentesi tonde "()" possono essere usate per raggruppare operazioni o effettuare chiamate a funzione:

```
2 * (3 + 4)

my_list = [4, 2, 3, 1]
my_list.sort()
```

Un Tour Veloce della Sintassi Python

- La funzione “print()” serve per visualizzare a video un qualsiasi oggetto Python:

```
x = 3 + 2
y = "ciao"

print(x)  # Visualizza 5 a video
print(y)  # Visualizza ciao a video
```

- Attenzione! Nella versione 2.x di Python “print” era uno statement del linguaggio e non una funzione.

La Semantica di Python: Variabili e Oggetti

- Per assegnare una valore ad una variabile si usa “=” :

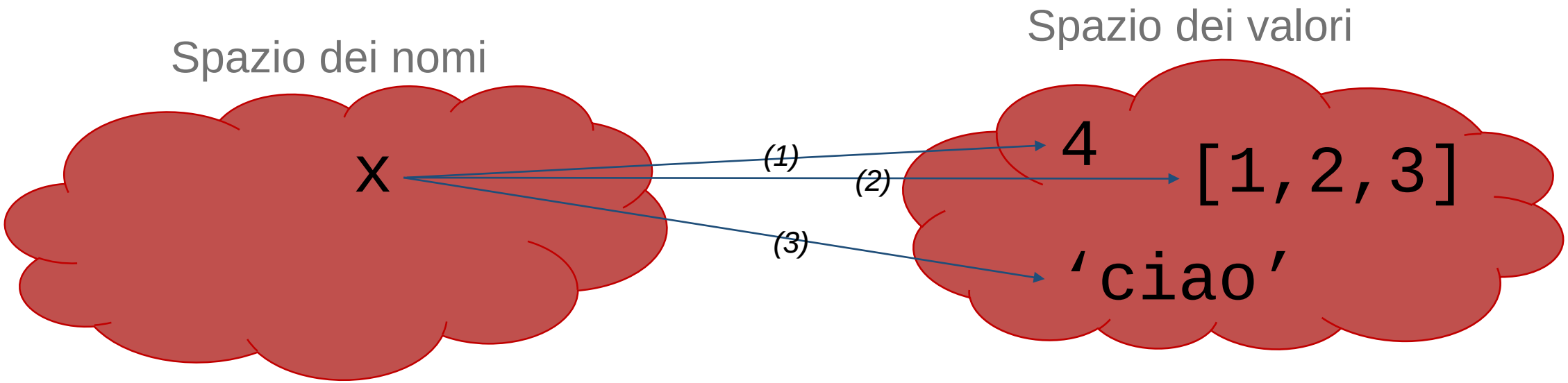
```
x = 4  # Corretto
4 = x  # Sbagliato
```

- In molti linguaggi di programmazione come il C e il C++ le variabili vengono viste come “contenitori di memoria”:

```
// Codice C
int x = 4
```

- In Python le variabili possono essere pensate come “puntatori”.

La Semantica di Python: Variabili e Oggetti




```
(1) x = 4           # x è un intero
(2) x = 'ciao'      # ora x è una stringa
(3) x = [1, 2, 3]   # ora x è una lista
```

La Semantica di Python: Variabili e Oggetti

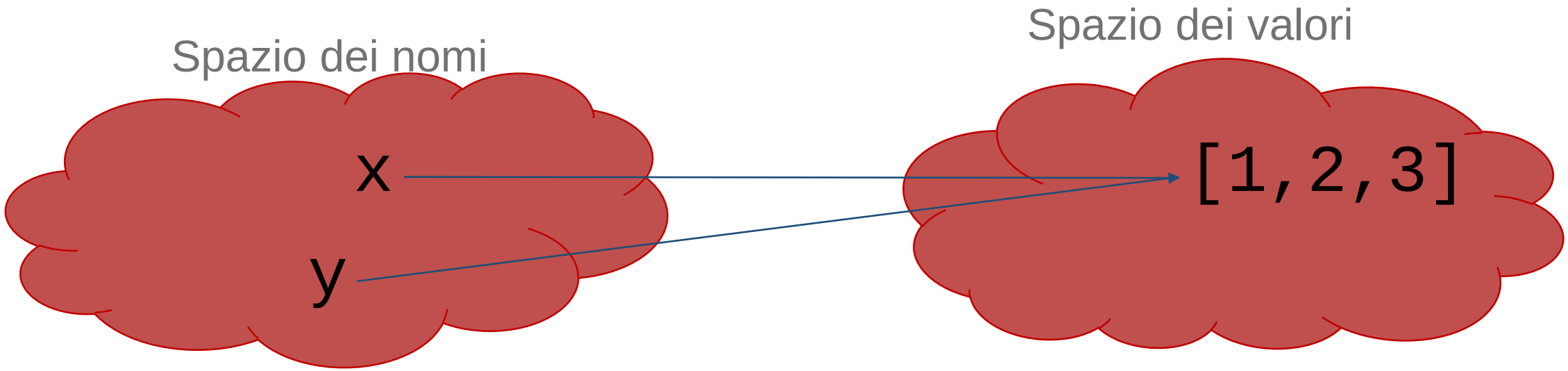
- La tipizzazione dinamica usata dal Python ciò che lo rende estremamente facile da leggere e veloce da scrivere.
- Attenzione però, se due “puntatori a variabile” puntano allo stesso oggetto, la modifica di uno cambierà anche l’altro:

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

print(x)  # Visualizza x, ovvero [1, 2, 3]
print(y)  # Visualizza y, ovvero [1, 2, 3]

x.append(4)  # Aggiungo l'elemento 4 alla lista x

print(x)  # Visualizza x, ovvero [1, 2, 3, 4]
print(y)  # Visualizza y, ovvero [1, 2, 3, 4]
```

La Semantica di Python: Variabili e Oggetti

- Infatti, questo è quello che accade in Python quando eseguiamo il codice:

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



La Semantica di Python: Variabili e Oggetti

- Questa rappresentazione potrebbe complicare le operazioni aritmetiche, quindi Python fa distinzione tra oggetti *mutabili* ed *immutabili*. Numeri, stringhe e tutti gli oggetti semplici sono immutabili, ovvero se ne può cambiare il valore solamente cambiando l'oggetto a cui questi puntano:

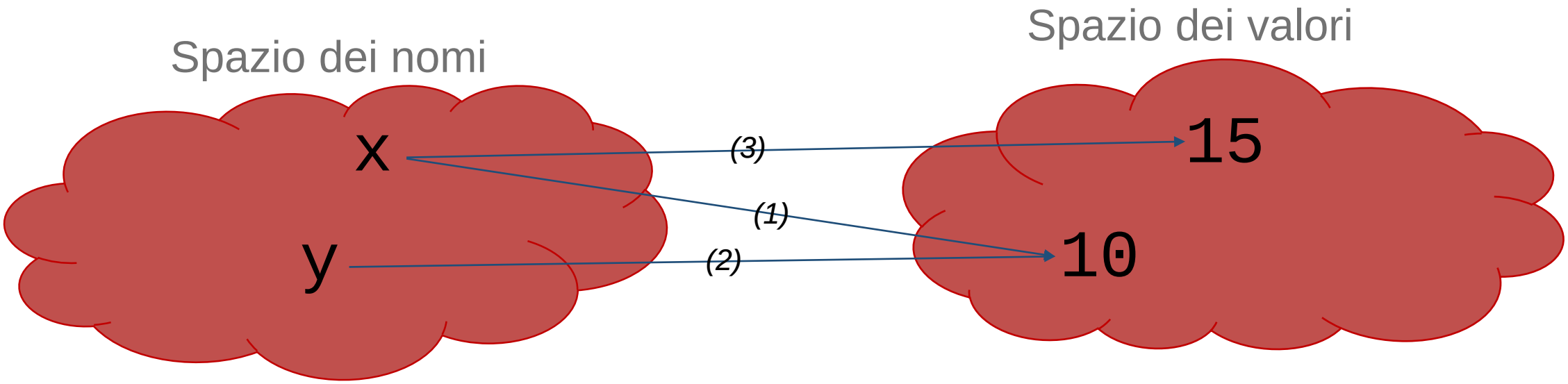
```
x = 10
y = 10
x = x + 5

print("x =", x)
print("y =", y)
```

- Cosa ci aspettiamo venga visualizzato dalle due "print()"?

La Semantica di Python: Variabili e Oggetti

- Risposta:

x = 15
y = 10




```
(1) x = 10
(2) y = 10
(3) x = x + 5
```

La Semantica di Python: Variabili e Oggetti

- Abbiamo visto che le variabili non hanno alcun tipo di informazione ad esse connessa, quindi si potrebbe pensare che il Python sia un linguaggio *type-free*. Non è così!

```
x = 4
type(x) # Restituisce il tipo di x, in questo caso int

x = 'ciao'
type(x) # Restituisce il tipo di x, in questo caso str

x = 3.14159
type(x) # Restituisce il tipo di x, in questo caso float
```

- Tutte le informazioni, compreso il tipo, sono connesse agli oggetti a cui le variabili puntano.

La Semantica di Python: Variabili e Oggetti

- Nei linguaggi di programmazione ad oggetti un oggetto è una entità a cui sono associati metadati (*attributi*) e funzionalità (*metodi*). Sia gli attributi che I metodi sono acceduti con il ".";
- *In Python tutto è un oggetto*, anche i tipi semplici:

```
x = 4.5
print(x.real, "+", x.imag, 'i')

# Output:
# 4.5 + 0.0 i
```

- "real" e "imag" ad esempio sono attributi che caratterizzano tutti i tipi numerici. Essi fornisco la parte reale e la parte immaginaria del numero.

La Semantica di Python: Variabili e Oggetti

- I metodi sono come gli attributi, ad eccezione del fatto che per essere invocati richiedono le parentesi tonde “()”:

```
x = 4.5
x.is_integer()

# Output:
# False

x = 4.0
x.is_integer()

# Output:
# True
```

La Semantica di Python: Variabili e Oggetti

- Quando dico che tutto in Python è un oggetto intendo proprio tutto. Anche i *metodi* e gli *attributi* di un oggetto sono a loro volta oggetti con il loro tipo

```
x = 4.5
type(x.is_integer)

# Output:
# <class 'builtin_function_or_method'>
```

Gli Operatori Aritmetici

Operator	Name	Description
`a + b`	Addition	Sum of a and b
`a - b`	Subtraction	Difference of a and b
`a * b`	Multiplication	Product of a and b
`a / b`	True division	Quotient of a and b
`a // b`	Floor division	Quotient of a and b, removing fractional parts
`a % b`	Modulus	Remainder after division of a by b
`a ** b`	Exponentiation	a raised to the power of b
`-a`	Negation	The negative of a
`+a`	Unary plus	a unchanged (rarely used)

Gli Operatori Aritmetici

- Gli operatori aritmetici possono essere combinati in maniera intuitiva utilizzando le parentesi tonde “()” per raggruppare le operazioni:

```
# Addizione, Sottrazione, Moltiplicazione
(4 + 8) * (6.5 - 3)

# Output
# 42
```

Gli Operatori Aritmetici

- La divisione intera (*floor division*) non è altro che il risultato della divisione privato della parte decimale:

```
# True division
print(11 / 2)

# Output
# 5.5

#Floor division
print(11 // 2)

# Output
# 5
```

- Attenzione, il comportamento dell'operatore "/" è diverso in Python 2.x

Gli Operatori Aritmetici

- Abbiamo già visto che l'operatore di assegnamento è l' "=".
- L'operatore di assegnamento può essere combinato con gli operatori aritmetici visti in precedenza:

```
a += b  # Si comporta come a = a + b
a -= b  # Si comporta come a = a - b
a *= b  # Si comporta come a = a * b
a /= b  # Si comporta come a = a / b
a //= b  # Si comporta come a = a // b
a **= b  # Si comporta come a = a ** b
```

Gli Operatori di Confronto

Operation	Description
a == b	a equal to b
a != b	a not equal to b
a < b	a less than b
a > b	a greater than b
a <= b	a less than or equal to b
a >= b	a greater than or equal to b

Gli Operatori Booleani

- In Python esistono tre tipi di operatori booleani: *and*, *or* e *not*;
- Gli operatori booleani vengono solitamente utilizzati in combinazione con gli operatori di confronto per verifare condizioni complesse:

```
x = 4
(x < 6) and (x > 2)

# Output: True

(x > 10) or (x % 2 == 0)

# Output: True

not(x < 6)

# Output: False
```

Operatori di Identità e Appartenenza

Operator	Description
`a is b`	True if a and b are identical objects
`a is not b`	True if a and b are not identical objects
`a in b`	True if a is a member of b
`a not in b`	True if a is not a member of b

Tipi di Dato Semplice

Type	Example	Description
`int`	`x = 1`	Integers (i.e., whole numbers)
`float`	`x = 1.0`	Floating-point numbers (i.e., real numbers)
`complex`	`x = 1 + 2j`	Complex numbers (i.e., numbers with a real and imaginary part)
`bool`	`x = True`	Boolean: True/False values
`str`	`x = 'abc'`	String: characters or text
`NoneType`	`x = None`	Special object indicating nulls

Tipi di Dato Strutturati

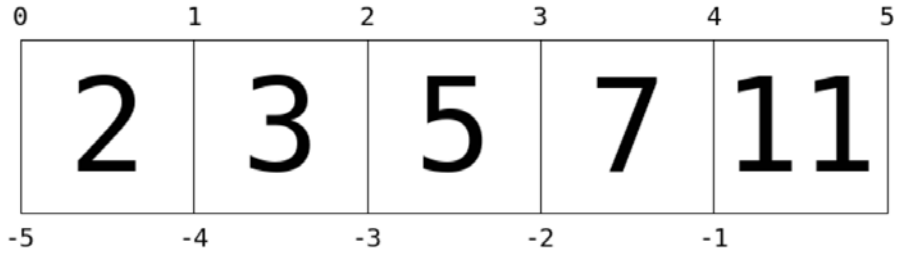
Type Name	Example	Description
`list`	`[1, 2, 3]`	Ordered collection
`dict`	`{'a':1, 'b':2, 'c':3}`	Unordered (key,value) mapping
`set`	`{1, 2, 3}`	Unordered collection of unique values

List Indexing and Slicing

- Python consente l'accesso ai tipi di dato composto mediante l'*indexing* e lo *slicing*.
- L'*indexing* consente di accedere ad un singolo elemento dell'oggetto:

```
list = [2, 3, 5, 7, 11]
print(list[0])  # Stampa 2
print(list[4])  # Stampa 11
print(list[-1]) # Stampa 11
print(list[-2]) # Stampa 7
print(list[5])  # Genera un errore: list index out of range
```

Schema di indicizzazione per la lista [2, 3, 5, 7, 11]



List Indexing and Slicing

- Lo *slicing* permette l'accesso ad elementi multipli:

```
list = [2, 3, 5, 7, 11]
print(list[0:3])   # Stampa la lista [2, 3, 5]
print(list[:3])    # Stampa la lista [2, 3, 5]
print(list[:])     # Stampa la lista [2, 3, 5, 7, 11]
print(list[0:3:2]) # Stampa la lista [2, 5]
```

- Sia *l'indexing* che lo *slicing* possono anche essere usati per settare i valori di dati composti.

Statement Condizionali: *if*, *elif* e *else*

- Consentono al programmatore di eseguire determinati blocchi di codice sulla base di condizioni booleane:

```
x = -15
if x == 0:
    print(x, "è zero")
elif x > 0:
    print(x, "è positivo")
elif x < 0:
    print(x, "è negativo")
else:
    print(x, "è qualcosa che non ho mai visto prima ... ")

# Output: - 15 è negativo
```

Ciclo *for*

- I cicli consentono di eseguire ripetutamente un certo blocco di codice. Se volessi ad esempio stampare ogni elemento di una lista potrei sfruttare il ciclo ***for*** nel seguente modo:

```
for element in [2, 3, 4, 8]:
    print(element, end=' ')

# Output: 2 3 4 8
```

- L'oggetto alla destra della clausola "***in***" deve essere un iteratore, uno degli iteratori più utilizzati in Python è il ***range***:

```
for i in range(10):
    print(i, end=' ')

# Output: 0 1 2 3 4 5 6 7 8 9
```

Ciclo *while*

- Il ciclo ***while*** itera fino a quando una determinata condizione booleana viene raggiunta:

```
i = 0
while i < 10:
    print(i, end=' ')
    i += 1

# Output: 0 1 2 3 4 5 6 7 8 9
```

- L'argomento del ciclo ***while*** viene valutato come condizione booleana; il blocco di codice contenuto nel ciclo viene eseguito fino a quando la condizione booleana risulta vera (True).

Break e *Continue*

- *Break* e *continue* sono due *statement* del linguaggio Python che possono essere usati per controllare/modificare il flusso di esecuzione di un ciclo:
  - *Break* interrompe l'esecuzione di un ciclo;
  - *Continue* salta l'esecuzione del codice che segue lo *statement* all'intero del ciclo e passa all'iterazione successiva.

Definizione e Utilizzo di Funzioni

- Una funzione rappresenta un blocco di codice a cui viene assegnato un nome. Il codice di una funzione può essere invocato usado le parentesi tonde “()”:

```
def RealImag(val):
    return val.real, val.imag

print(RealImag(1.0 + 5j))

# Output: (1.0, 5.0)
```