

Introduzione a Python

Davide Sanvito, Carmelo Cascone, Daniele Moro

Perchè Python?

- **Facile da imparare!**
- **Enfasi sulla leggibilità**
 - Quasi come scrivere una serie di istruzioni in lingua inglese
- **Linguaggio di alto livello**
 - Orientato ad oggetti, al contrario del C (linguaggio di basso livello)
- **Linguaggio interpretato**
 - Non c'è bisogno di compilatore per eseguire un programma Python
- **Multi-piattaforma (Windows, Mac, Linux, etc.)**
 - Lo stesso programma funziona (quasi sempre) su sistemi operativi differenti
- **Perchè Python3?**
 - Python2.7 è sempre meno usato e non verrà più supportato dal 2020

Come eseguire Python

- **Modalità non-interattiva**

- Crea un file di testo con il codice Python e salvalo con estensione `.py` (es. `my-program.py`)
- Esegui da riga di comando `python3 my-program.py`

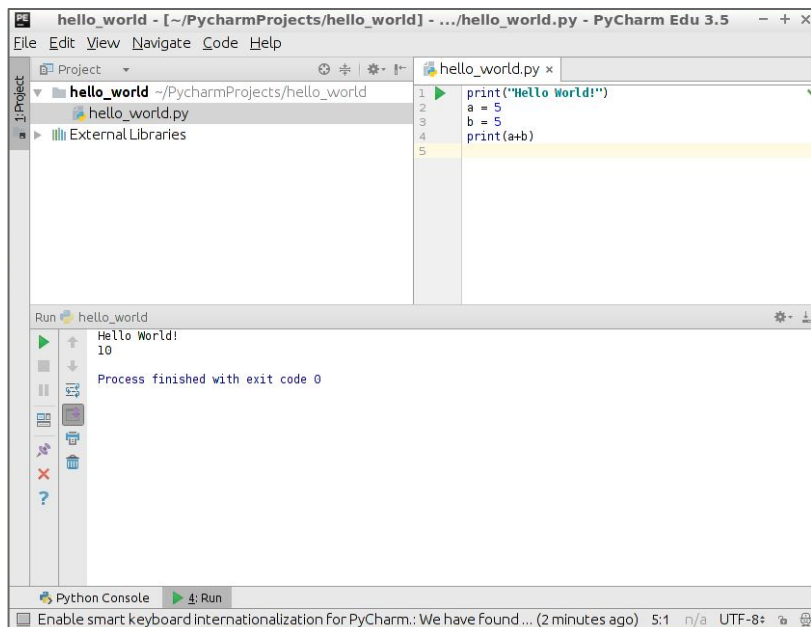
- **Modalità interattiva**

- Esegui solo il comando `python3` (N.B.: ricordarsi di usare `python3` e non semplicemente `python`)
- Scrivi istruzioni Python riga per riga ed osserva il risultato

```
Python 3.4.3+ (default, Oct 14 2015, 16:03:50)
[GCC 5.2.1 20151010] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

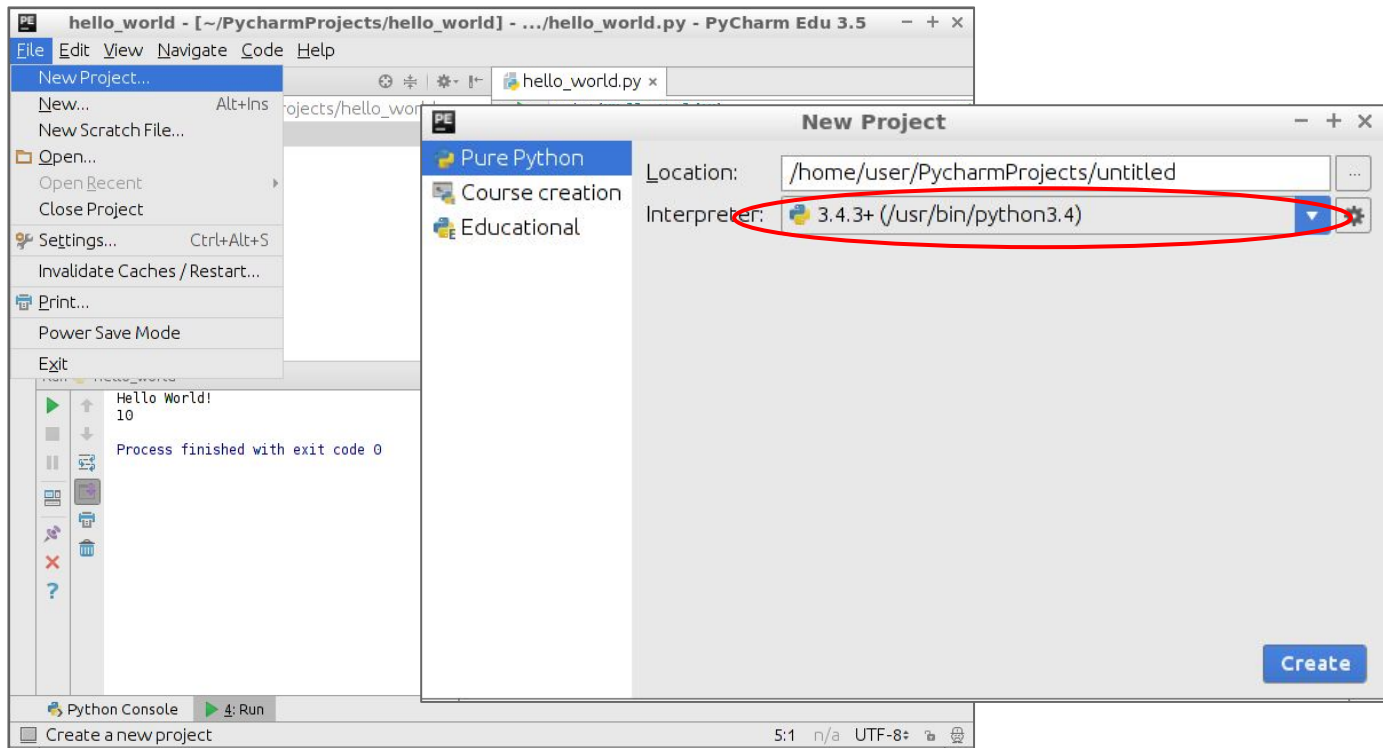
IDE di riferimento: PyCharm

- **Integrated Development Environment (IDE)**
 - Software che aiuta i programmatori nello sviluppo del codice sorgente di un programma
 - Auto-completamento, segnalazione di errori, esecuzione, debugging
- **Noi utilizziamo PyCharm (gratuito)**
 - Versione Educational gratuita
 - <https://www.jetbrains.com/pycharm-edu/>
 - Compatibile con Windows, Mac e Linux

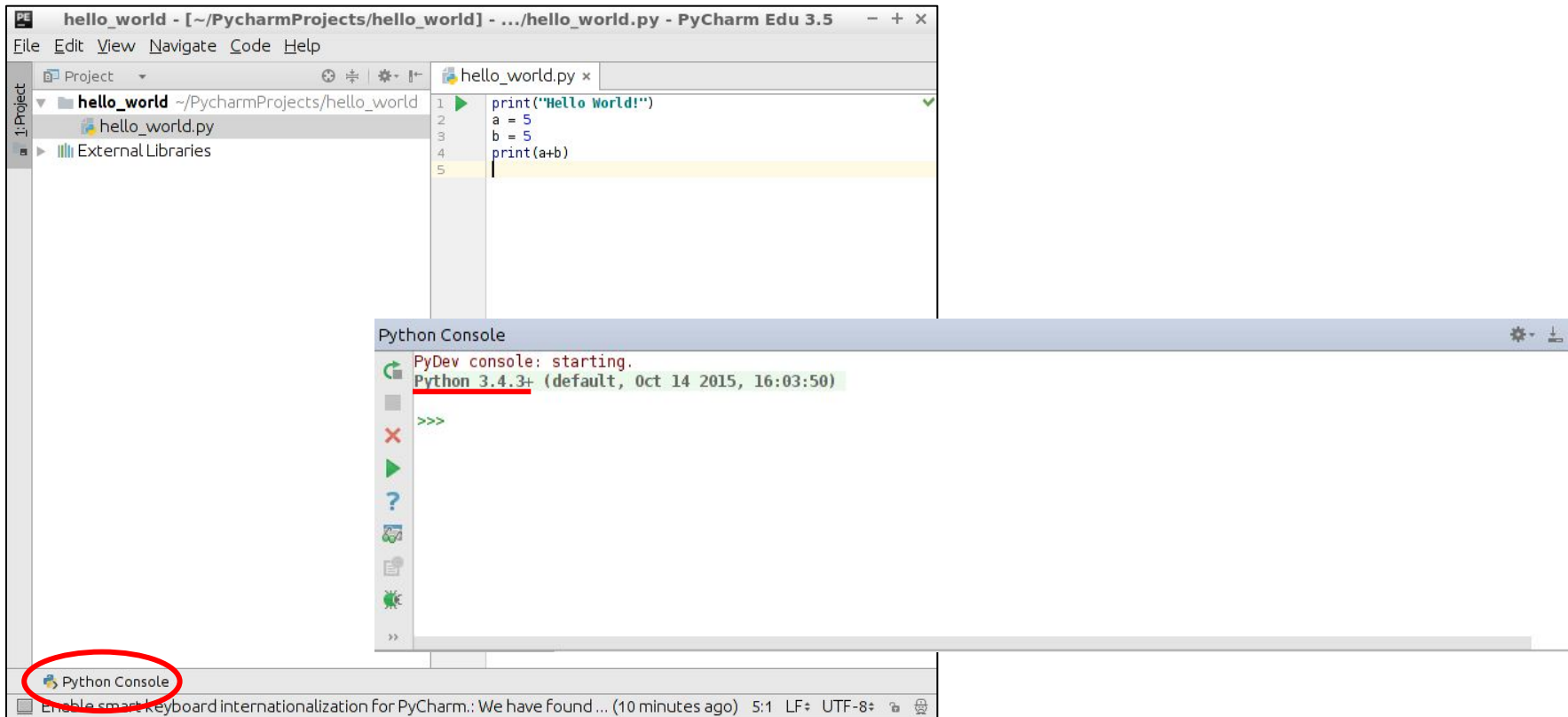


PyCharm: Create un nuovo progetto

- Eseguite PyCharm Edu (selezionare come interprete **python3.4**)



PyCharm: Console Python interattiva



Commenti

- **Porzioni di testo ignorate durante l'esecuzione**
 - Servono a descrivere meglio il codice a chi lo legge
- **In Python cominciano sempre con il carattere #**
 - ...e proseguono fino alla fine della riga
 - Attenzione! Il carattere # può essere utilizzato anche all'interno di stringhe

```
# this is the first comment
SPAM = 1                    # and this is the second comment
                             # ... and now a third!
STRING = "# This is not a comment."
```

Numeri

- In modalità interpretativa, Python stampa il risultato delle operazioni
- Proviamo ad usarlo come una calcolatrice
- Gli operatori più comuni sono **+**, **-**, ***** e **/**
- Si possono utilizzare le parentesi per raggruppare le operazioni

```
>>> 2+2
4
>>> (50-5*6)/4
5.0
>>> 7//3    # integer division returns the floor
2
>>> 2**3    # exponentiation
8
```


Variabili

- E' possibile assegnare un valore a una variabile
- Python utilizza la “**tipizzazione dinamica**”
 - Non serve specificare il tipo di variabile (es. in C *int*, *long*, *char*, etc.)
- NB: non confondere l'assegnazione `=` con il confronto `==`

```
>>> a = 20
>>> b = 5*9
>>> a * b
900
```

Stringhe

- Sono strutture dati che contengono **dati testuali**
- Sono racchiuse tra **apici (')** o **doppie virgolette (")**
- Per includere virgolette nella stringa si utilizza il **carattere di escape **
- La funzione **print()** mostra a schermo la stringa

```
>>> print('spam eggs')
spam eggs
>>> print('doesn\'t')
doesn't
>>> print("doesn't")
doesn't
>>> print('"Yes," he said.')
"Yes," he said.
>>> print("\\"Yes,\" he said.")
"Yes," he said.
>>> print('"Isn\'t," she said.')
"Isn't," she said.
```

Liste (1)

- Le liste sono delle **sequenze di variabili modificabili**
- Possono essere **indicizzate, ri-ordinate, concatenate**, etc...
- Sono racchiuse fra **parentesi quadre**

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
>>> a[0]
'spam'
>>> a[1:3]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> a.append(9.87)
>>> a
['spam', 'eggs', 100, 1234, 9.87]
```

Liste (2)

- Anche le stringhe sono delle **liste di caratteri**

```
>>> word = 'Help' + 'A'
>>> word
'HelpA'

>>> word[4]
'A'
>>> word[0:2]
'He'
>>> word[:2]      # The first index defaults to zero
'He'
>>> word[2:]      # The last index defaults to the end of the string
'lpA'

>>> word[-1]      # The last character
'A'
>>> word[:-2]     # Everything except the last two characters
'Hel'
```

Strutture di controllo - if/else

- Definiscono la **logica di esecuzione del programma** (se... / altrimenti...)
- Si basano sulla valutazione di una **condizione “booleana”** (vero o falso)
- Gli operatori di confronto più comuni sono `<`, `>`, `==`, `!=`

```
>>> x = 5
>>> if x < 0:
...     print('negative')
... elif x == 0:
...     print('zero')
... else:
...     print('positive')
...
positive
```

Indentazione

- Python utilizza l'indentazione per **raggruppare porzioni di codice**
 - A differenza del C dove il codice viene raggruppato tra parentesi graffe
- Non ci sono regole sul numero di TAB o spazi
- L'unica regola è quella di **indentare in maniera uniforme**
 - Utilizzare solo TAB o spazi in numero uguale per la stessa porzione di codice

```
>>> if True:
...     print('x') #leading space is a TAB
...     print('y') #leading space is four SPACES
File "<stdin>", line 3
    print('y') #leading space is four SPACES
        ^
IndentationError: unindent does not match any outer indentation level
```

Strutture di controllo - for

- Permette di eseguire la stessa porzione di codice su più elementi di una lista

```
>>> # Measure some strings:
... a = ['cat', 'window', 'defenestrate']
>>> for x in a:
...     print(x, len(x))
...
cat 3
window 6
defenestrate 12
```

Funzioni

- Si definiscono con il costrutto `def`
- Possono ritornare anche liste o oggetti

```
>>> def potenza(base, esponente):  
...     return base**esponente  
...  
>>> potenza(2,3)  
8  
>>> potenza(2,4)  
16  
>>> list(range(4)) # Standard Python functions  
[0, 1, 2, 3]
```


Moduli

- File che raggruppano definizioni di funzioni e istruzioni Python
 - Chiamati anche “librerie”
- Python fornisce una libreria di moduli standard (`os`, `math`, `socket`, etc...)
 - <https://docs.python.org/3.4/library/>
- Possono essere importati in altri file e ri-utilizzati (costrutto `import`)
 - Possibilità di importare solo alcune funzioni di un modulo (costrutto `from ... import ...`)

```
>>> import os
>>> os.getcwd()
'C:\\Users\\Davide\\PycharmProjects\\shell'
>>> from math import factorial
>>> factorial(5)
120
```

Programmazione a oggetti

- In Python e altri linguaggi tipo Java gli oggetti sono delle variabili “speciali”
- Possono contenere altre variabili o funzioni
- Vengono creati a partire da una classe (definita a parte)

```
class rettangolo:  
  
    def __init__(self, l1, l2):  
        self.l1 = l1  
        self.l2 = l2  
  
    def area(self):  
        return self.l1*self.l2
```

```
>>> rett1 = rettangolo(2,5)  
>>> rett1.area()  
10  
>>> rett1 = rettangolo(4,4)  
>>> rett1.area()  
16
```