Lezione 1: Introduzione a Python

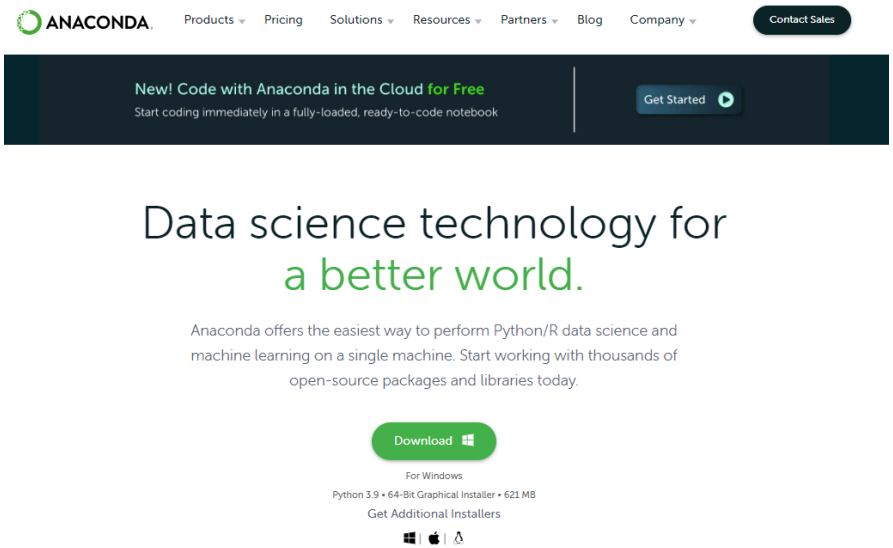
Davide Evangelista e Dario Lanzoni
dario.lanzoni3@studio.unibo.it

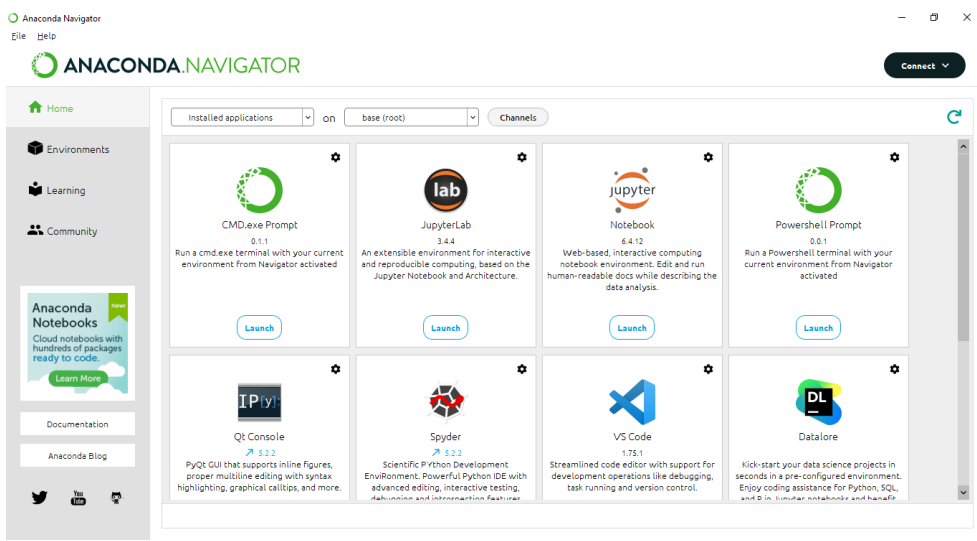
Università di Bologna

27 Febbraio 2025

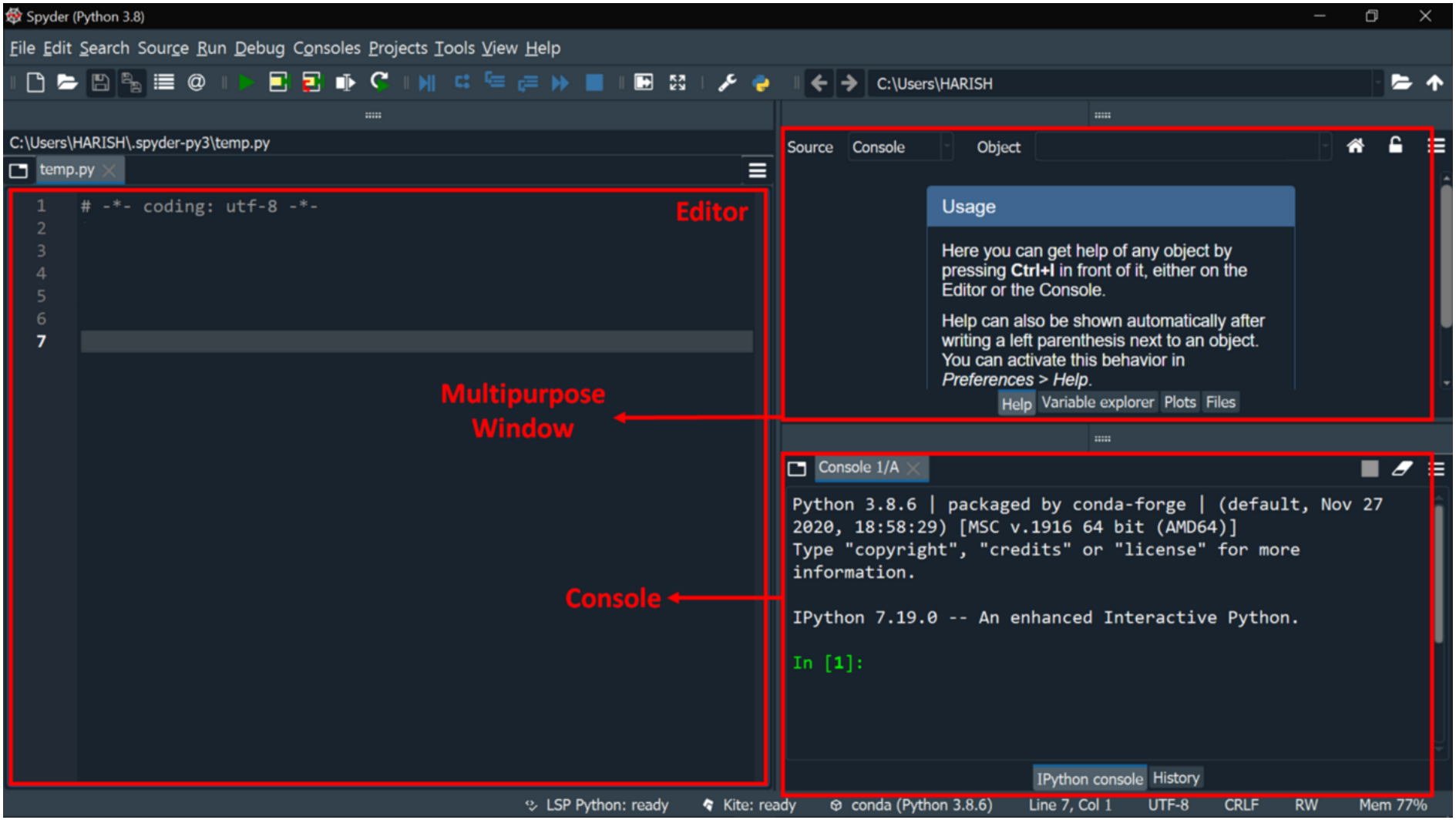
Installazione

1. Scaricare ed installare **Anaconda**
2. Aprire **Spyder** da Anaconda Navigator (o da qualche scorciatoia)





Interfaccia Spyder



Python

- Linguaggio interpretato ad alto livello.
- Applicazioni in numerosi settori: sviluppo web, scripting, calcolo scientifico, business analytics e intelligenza artificiale.
- Tipicamente richiede tempi di esecuzione più lunghi dei linguaggi compilati come C++, ma è molto più leggebile e snello come codice. (debugging più facile!!)
- Linguaggio più diffuso e richiesto nel mercato del lavoro.
- Usato da organizzazioni quali Google, NASA, Instagram e Netflix.
- ChatGPT!

Funzione help e working directory

- La maggior parte delle azioni si effettuano richiamando funzioni: definite dall'utente o presenti in librerie built-in/importate. `print('Hello world!')`
- Per importare una libreria si utilizza il comando `import nome_libreria`.
- Per conoscere la sintassi di una funzione si utilizza il comando `help(nome_funzione)`.
- È opportuno specificare in quale cartella cercare/salvare i file. Importando `os` la funzione `os.getwd()` restituisce il percorso dell'attuale directory di lavoro che è possibile cambiare attraverso `os.chdir()`.

Variabili

- Le variabili possono essere chiamate in qualunque modo, purché il primo carattere del nome della variabile sia una lettera.
- Due variabili `pippo` e `Pippo` sono considerate diverse.
- I valori delle variabili vengono assegnati tramite = (da non confondere con l'operatore logico ==).
- Le variabili assegnate vengono salvate nel workspace, nell'angolo in alto a destra di Spyder, oppure utilizzando il comando `%who` o `%whos`.
- Per cancellare una variabile dal workspace, si può usare il comando `del`.

Sintassi operatori

Operatori aritmetici

+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
**	Elevamento a potenza
%	Resto della divisione
//	Divisione intera

Operatori relazionali

==	Uguale a
!=	Diverso da
>	Maggiore
>=	Maggiore o uguale
<	Minore
<=	Minore o uguale

Operatori logici

not	negazione
and	congiunzione AND
or	congiunzione OR inclusivo
is	identità
in	appartenenza

Tipi di dati

- **Numeric:** Integer `x = 3` e Floating point `x = 1.5`
- **Complex** `y = 2 + 3j`
- **Strings** `a = 'pippo'`
- **Boolean** `b = True` o `b = False`
- **None** tipo di dato riservato a `None`, rappresenta una variabile vuota

Data una variabile assegnata `x`, la funzione `type(x)` restituisce il tipo di `x`.

```
> x = 3
> type(3)
[1] float
```

```
> a = "pippo"
> type(a)
[1] str
```

```
> b = True
> type(b)
[1] bool
```

Tuple

- Le tuple sono tipi di dato (immutabili), che possono contenere al loro interno oggetti di tipo diverso contemporaneamente.
- Una tuple si costruisce con la funzione `tuple()` e si accede al suo elemento i-esimo usando `[i]`.

```
> t = (1, 2, 3)
> len(t)
[1] 3
> t[-1]
[1] 3
> t[0]
[1] 1
> t[1:]
[1] (2,3)
```

Liste

- Le liste sono tipi di dato, che possono contenere al loro interno oggetti di tipo diverso contemporaneamente.
- Una lista si costruisce con la funzione `list()` e si accede al suo elemento i-esimo usando `[i]`.
- A differenza delle tuple le liste sono oggetti mutabili (**Attenzione!!**).

```
> L = ['pippo', 1]
> L.append([1, 3, 1])
> L[-1]
[1] 1 3 1
> L[0]
[1] 'pippo'
> len(L)
[1] 3
```

Condizione If-else

- Il comando `if` serve per eseguire codice solamente quando è verificata una condizione.
- La condizione può essere un'espressione logica o un valore booleano.
- Il comando `else` serve per eseguire codice nel caso in cui non sia verificata la condizione.
- Si possono concatenare più `if` con il comando `elif`.

```
if (condizione1):
   espressione1
elif (condizione2):
   espressione2
else:
   espressione3
```

Cicli while e for

- Con il ciclo `while` si esegue l'espressione finché è verificata la condizione.

```
while (condizione):
    espressione
```

- Con il ciclo `for`, dato una lista/vettore `v`, si esegue l'espressione facendo scorrere l'indice `i` in `v`.

```
for i in v:
    espressione
```

- Il comando `range(n)` crea una 'lista' di numeri che vanno da 0 ad n-1.

Funzioni user-defined

- Spesso si ripete più volte una serie di comandi in uno stesso script.
- In questo caso, è opportuno definire una funzione per alleggerire e rendere più leggibile il codice .

```
def nome_function(x1,x2,...):
    comandi
    return output
```

- Per richiamare la funzione è necessario scrivere `f(x1, x2, ...)`.
- Le variabili definite all'interno della funzioni sono locali e vengono cancellate una volta terminata l'esecuzione della stessa.

Funzioni user-defined

```
> def media(x1, x2):
      m = (x1+x2)/2
      return m
> media(1,5)
[1] 3.0
> m
NameError: name 'm' is not defined
```

Funzioni e liste (I)

- Fare sempre attenzione a modificare liste e array (li vedremo più avanti) all'interno di una funzione.

```
>L = [1, 2, 3]
> def mod_list(lista):
      lista[1] = 200
      return lista
> new_L = mod_lista(L)
```

- Cosa si ottiene se si stampano le due liste?

Funzioni e liste (I)

- Fare sempre attenzione a modificare liste e array (li vedremo più avanti) all'interno di una funzione.

```
>L = [1, 2, 3]
> def mod_list(lista):
      lista[1] = 200
      return lista
> new_L = mod_lista(L)
```

- Cosa si ottiene se si stampano le due liste?

```
> new_L
[1] [1, 200, 3]
> L
[1] [1, 200, 3]
```

Funzioni e liste (II)

- Per evitare questo fenomeno, creare sempre una copia di una lista prima di applicarci una funzione:

```
> L2 = L.copy()
> L2 = L[:]
```

- Per liste più "profonde" è possibile usare:

```
> L2 = copy.deepcopy(L)
```

Librerie

- Alcune funzioni sono già scritte e disponibili in apposite librerie, che possono quindi essere importate.
- Una libreria molto utilizzata è `math`, che contiene tutte le funzioni trigonometriche, la radice quadrata, il logaritmo...
- Per importare una libreria ci sono 3 modi, che si presentano con una sintassi leggermente diversa
  1. `import math`
  2. `from math import *funzione*`
  3. `import math as *nome*`

Esempi librerie

- Nel primo caso, importiamo tutta libreria e ogni chiamata ad una funzione dovrà essere preceduta da `math.`:

```
> import math
> math.sqrt(2)
```

- Nel secondo caso, importiamo la funzione specifica dalla libreria:

```
> from math import sqrt
> sqrt(2)
```

- Il terzo caso è simile al primo, ma math è abbreviato da una parola di nostra scelta:

```
> import math as m
> m.sqrt(2)
```