PLS - Big data e network tra fisica e biologia

A.A. 2020/2021

- claudia.sala3@unibo.it
- alessandra.merlotti2@unibo.it
- tommaso.matteuzzi2@unibo.it

Potete trovare queste lezioni sul sito github del gruppo di biofisica:

https://github.com/UniboDIFABiophysics/PLSBigDataNetworks (https://github.com/UniboDIFABiophysics/PLSBigDataNetworks)

Introduzione a Python

https://www.python.org (https://www.python.org)

Un ottimo tutorial online lo potete trovare alla pagina:

http://www.scipy-lectures.org/ (http://www.scipy-lectures.org/)

Programmare serve per **risolvere problemi**.

Risolvere problemi significa avere l'abilità di schematizzarli, pensare creativamente alle possibili soluzioni ed esprimerle in modo chiaro ed accurato.

Il processo di imparare a programmare è un'eccellente opportunità di mettere in pratica l'abilità di risolvere problemi.

Un **programma** è una sequenza di istruzioni che specificano come risolvere il problema sotto studio.

Python è un linguaggio interpretato di alto livello.

Questa è una maniera complicata per dire che evita moltissimi dei dolori della programmazione classica.

"Interpretato" significa che non sono necessari step di compilazione. Questo rende più semplice il ciclo sviluppo - esecuzione - correzione

"Di alto livello" significa che è molto diverso dal linguaggio macchina. Python ha come principale linea guida il concetto di **leggibilità**, cercando il più possibile di rendere semplice agli sviluppatori scrivere librerie semplici ed intuitive.

Python supporta di base tutte le operazioni matematiche più comuni, compreso il supporto per la matematica complessa, e tutti i costrutti più comuni di programmazione (e molti altri) come:

- if, elif, else
- cicli for e while
- definizione di funzioni e classi

Utilizzare Python

Esistono molti modi diversi di lavorare tramite Python.

Il fondamentale è tramite la **shell di python**, un terminale che esegue le righe di comando mano a mano che vengono inserite, ed è estremamente utile per testare il codice al volo prima di inserirlo dentro un programma. Il terminale di python si può lanciare da terminale digitando **python**.

```
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Programmiamo in python!")
Programmiamo in python!
>>>
```

Il secondo metodo è scrivere uno (o più) **script** contenente i comandi che si vogliono eseguire, e farlo eseguire da python come un programma qualsiasi tramite il comando

python nomedelfile.py

Le cose più interessanti si trovano però andando a cercare fra i vari programmi disponibili.

Uno degli editor più famosi per il calcolo scientifico in python è **Spyder** (http://code.google.com/p/spyderlib/ (http://code.google.com/p/spyderlib/)), che fornisce un'interfaccia molto simile a quella di Matlab e di RStudio. È un programma molto solido e funzionale, ed è un'ottima piattaforma su cui lavorare.

Un altro approccio è l'utilizzo di un IDE (Integrated Development Environment) completo, quale PyCharm, Wing o Eclipse. Il mio personale preferito, molto più semplice ma con tutte le funzioni che mi servono è **Atom** (https://ide.atom.io/ (https://ide.atom.io/)).

Fra tutti lo strumento però in assoluto più potente è sicuramente **IPython** (http://ipython.org (http://ipython.org)). IPython fornisce tre programmi:

- La IPython shell, che è un terminale python potenziato, che può tranquillamente sostituire la shell di sistema
- La QTConsole, che contiene un ibrido fra una shell ed un sistema di gestione tramite GUI, e permette l'utilizzo dei grafici inline invece che in finestre separate.
- Il Notebook, ovvero quello che sto usando in questo momento, che è un'interfaccia web ad un server di esecuzione che permette di mescolare testo, codice, grafica e formule in un unico insieme. In particolare quello che sto usando in questo momento è la versione **3.6** di IPython

Procurarsi python

Python arriva di base installato con un gran numero di librerie sia sotto Linux che sotto macOSX, mentre invece va installato da zero su Windows.

Installare nuove librerie è molto semplice sfruttando il programma **pip**, che permette di dire a python quale libreria si desidera e lui la scaricherà ed installerà sul sistema.

Per installare pip:

```
conda install -c anaconda pip
```

Per installare la libreria math:

```
>>> pip install math
```

Oppure direttamente da conda:

```
conda install -c anaconda math
```

Una comoda alternativa sono le librerie all-inclusive. Queste sono normalmente a pagamento, ma arrivano con python e tantissime librerie preinstallate e configurate per il sistema in uso. Fra le principali ci sono:

- **Enthought Python Distribution**, probabilmente la più rodata, è disponibile per tutti i sistemi ed offre una versione gratuita per chi fà parte dell'università
- **Anaconda**, concorrente moderno della EPD, è gratuita e forse la migliore disponibile al momento
- **Python(x, y)**, gratuito e ben testato, ma purtroppo presente solo per windows
- **Sage**, completamente gratuito, che comprende oltre a python anche un gran numero di programmi per la matematica accessibili da tramite python

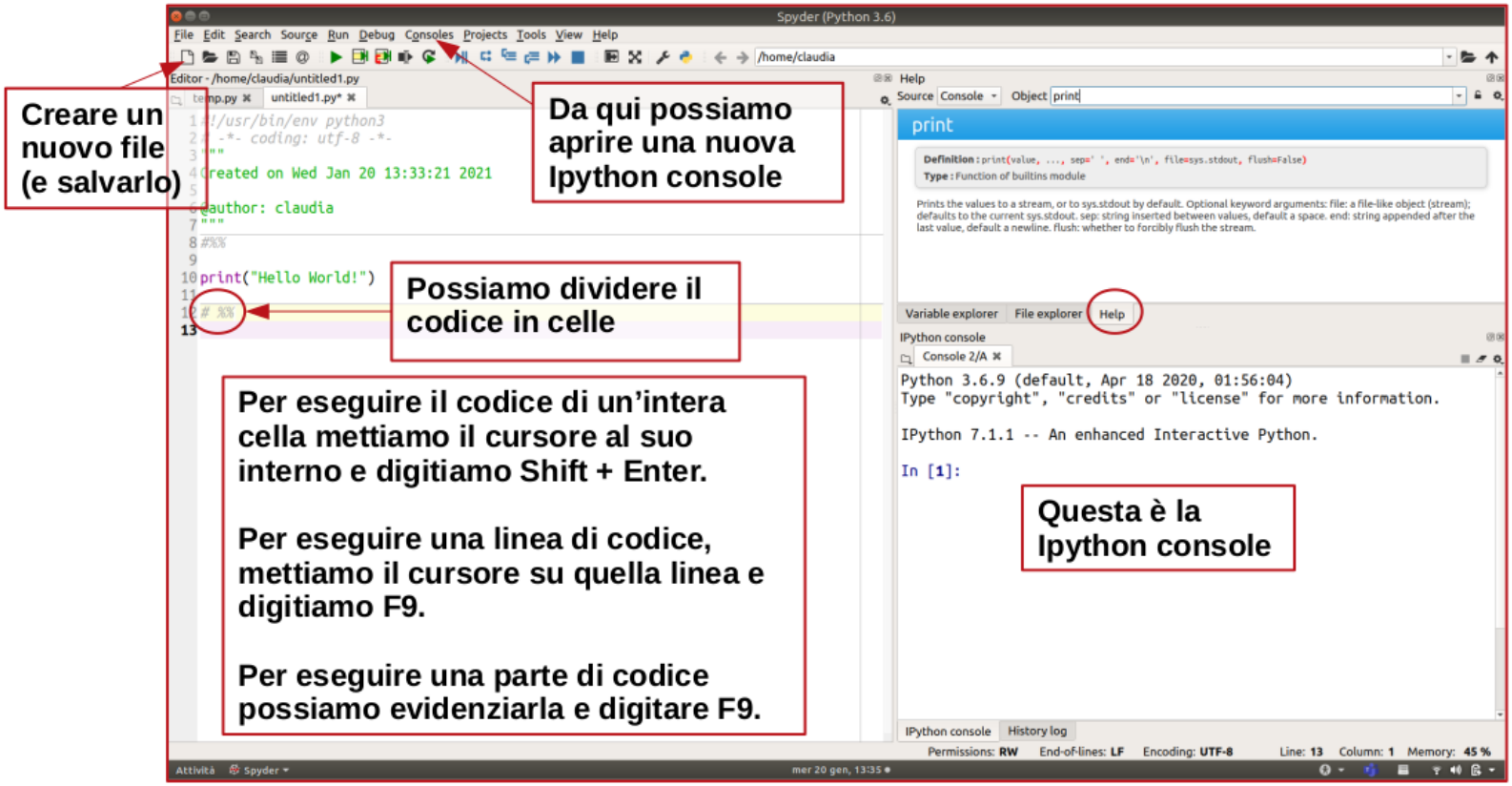
- **Wakari**, un sito che offre un ambiente di sviluppo integrato sul browser. Molto completo ma richiede la connessione internet non installando nulla sul pc.
- In modo simile i siti **pythonanywhere** e **tutorialspoint** forniscono un servizio gratuito per provare una shell di python online.
- Il servizio **PiCloud**, che fornisce accesso a una rete di supercomputer amazon, fornisce anche la possibilità di lavorare online con dei notebook che girano sui loro server.

Basi del linguaggio

Vediamo subito il primo programma di default, Hello World:

In [1]:
```
print("Hello, World!")
```

```
Hello, World!
```


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
/home/claudia
Editor - /home/claudia/untitled1.py
temp.py
untitled1.py*
#!/usr/bin/env python3
-*- coding: utf-8 -*-
"""
Created on Wed Jan 20 13:33:21 2021
@author: claudia
"""
#%%
print("Hello World!")
# %%
Help
Source Console Object print
print
Definition : print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
Type : Function of builtins module
Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments: file: a file-like object (stream); defaults to the current sys.stdout. sep: string inserted between values, default a space. end: string appended after the last value, default a newline. flush: whether to forcibly flush the stream.
Variable explorer
File explorer
Help
IPython console
Console 2/A
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
Type "copyright", "credits" or "license" for more information.
IPython 7.1.1 -- An enhanced Interactive Python.
In [1]:
IPython console
History log
Permissions: RW
End-of-lines: LF
Encoding: UTF-8
Line: 13
Column: 1
Memory: 45 %
Attività
Spyder
mer 20 gen, 13:35
Creare un nuovo file (e salvarlo)
Da qui possiamo aprire una nuova Ipython console
Possiamo dividere il codice in celle
Per eseguire il codice di un’intera cella mettiamo il cursore al suo interno e digitiamo Shift + Enter.
Per eseguire una linea di codice, mettiamo il cursore su quella linea e digitiamo F9.
Per eseguire una parte di codice possiamo evidenziarla e digitare F9.
Questa è la Ipython console

Possiamo usare la funzione print per stampare (quasi) qualsiasi cosa.

Per esempio usiamo print per vedere i risultati di alcune operazioni matematiche

In [2]:
```
print(1)
print(1 + 2)
print((4 + 5j) * (2 + 3j))
print(4 ** 4)
```

```
1
3
(-7+22j)
256
```

Nelle vecchie versioni di Python (2.7 o inferiori) la divisione fra numeri interi restituiva un intero (come in C). Nella nuova versione (python 3, quello che utilizzeremo per queste lezioni) questo comportamento è stato modificato, ed ora la divisione fra numeri restituisce il valore decimale come ci si attenderebbe

In [3]:
```
2 / 3
```

Out[3]:
```
0.6666666666666666
```

Per avere la divisione intera si utilizza un operatore specifico, usando il doppio segno di divisione.

In [4]:
```
2 // 3
```

Out[4]:
```
0
```

Per ottenere le stesso comportamento nelle vecchie versioni si può utilizzare il comando:

```
from __future__ import division
```

Per utilizzare funzioni matematiche più avanzate abbiamo bisogno di utilizzare una delle librerie che arrivano con python, la libreria **math**, che ci mette a disposizione una lunga lista di funzioni. per utilizzare math basta scrivere

In [5]:
```
import math
```

Ed ora abbiamo accesso a tutte le funzioni necessarie. Se avessimo bisogno di lavorare con i numeri complessi, esiste una libreria gemella chiamata **cmath**.

Per avere più informazioni sulla libreria che andiamo ad usare possiamo utilizzare due comandi molto utili:

- **dir** per sapere quali siano le sue funzioni
- **help** per averne una descrizione (il risultato di **help(math)** è piuttosto lungo, quindi eviterò di mostrarlo)

In [6]:
```
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acos
h', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cos
h', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor
', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfini
te', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'mo
df', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau
', 'trunc']
```

dir mi ritorna una lista con i nomi di tutte le funzioni presenti dentro math. Per usarle basta scrivere

```
nomelibreria.nomefunzione
```

In [7]:
```
math.exp(1)
```

Out[7]:
```
2.718281828459045
```

Se voglio sapere cosa faccia di preciso una di quelle funzioni basta usare la funzione help

In [8]:
```
help(math.log1p)
```

```
Help on built-in function log1p in module math:

log1p(...)
    log1p(x)

    Return the natural logarithm of 1+x (base e).
    The result is computed in a way which is accurate for x near zero.
```

In [9]:
```
print(math.pi)
print(math.log1p(math.pi))
```

```
3.141592653589793
1.4210804127942926
```

In [10]:
```
math.log( math.pi + 1.0 )
```

Out[10]:
```
1.4210804127942926
```

Se avete bisogno di poche funzioni e non volete digitare ogni volta il nome della libreria potete importare solo una parte dei nomi, che saranno poi disponibili senza bisogno di fare riferimenti alla librerie.

In [11]:
```
from math import radians, log1p, modf

print(log1p(1.0))
```

```
0.6931471805599453
```

Potreste trovare alcuni tutorial su internet che propongono di importare le funzioni da una libreria con il seguente comando:

```
from math import *
```

Meglio evitarlo!

Questo comando importa infatti tutti i nomi di funzioni dalla libreria, sovrascrivendo possibili funzioni già esistenti. Questo comportamento potrebbe riservarvi delle sorprese molto spiacevoli, e di cui è difficile rendersi conto!

Inclusi nel linguaggio di base arrivano anche molte strutture dati estremanente utili. Una lista non esaustiva include:

- liste
- stringhe
- set
- dizionari
- deque
- heap

Una menzione particolare la meritano i dizionari, che permettono di memorizzare dei dati in modalità random (ovvero non sequenziale) accedendovi tramite nomi esplicativi

In [12]:
```
rubrica = dict()

rubrica['Ludovico Fabbri'] = ('334-5678901', 'Via Larga 34, Bo')

rubrica['Mario Rossi'] = ('333-4567891', 'Via Stretta 43, Bo')

print(rubrica)
```

```
{'Ludovico Fabbri': ('334-5678901', 'Via Larga 34, Bo'), 'Mario Rossi': ('333-
4567891', 'Via Stretta 43, Bo')}
```

In [13]:
```
print(rubrica['Ludovico Fabbri'])
```

```
('334-5678901', 'Via Larga 34, Bo')
```

Se provo a cercare un elemento assente, Python si lamenta in modo rumoroso.

Questo è un design intenzionale, meglio fallire presto in modo chiaro che non dare risultati inaspettati!

In [14]:
```
rubrica['enrico giampieri']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-14-dcf302016889> in <module>
----> 1 rubrica['enrico giampieri']

KeyError: 'enrico giampieri'
```

In [15]:
```
try:
    print(rubrica['enrico giampieri'])
except KeyError:
    print("non trovato")
```

```
non trovato
```

Funzioni

È possibile definire le proprie funzioni tramite il comando **def**. In questo caso creerò una funzione che prende due oggetti e ne restituisce la somma.

A python non interessa di che oggetti si tratti, fintantochè sia definita fra di loro una somma. Posso usare questa funzione per sommare indifferentemente fra di loro numeri, stringhe, liste o distribuzioni di probabilità.

In [16]:
```
def mia_funzione(a, b):
    return a + b

print(mia_funzione(1, 2))
print(mia_funzione('hello ', 'world!'))
print(mia_funzione([1, 2, 3], [4, 5, 6]))
```

```
3
hello world!
[1, 2, 3, 4, 5, 6]
```

Iterazioni

Python supporta le iterazioni sia con il costrutto **while** che con il **for**.

Il for in pyhton è un ciclo speciale estremamente potente, ed è quindi il modo principale di scrivere cicli.

Esempio: Vorrei stampare tutti gli elementi di una lista.

Soluzione 1) Posso usare gli **indici** per accedere agli elementi della lista.

Questo metodo non è molto conveniente, specialmente se la lista è lunga!

In [17]:
```
lista_ingredienti = ['pane','pomodori','farina','acqua','sale','mozzarella']

print(lista_ingredienti[0])
print(lista_ingredienti[1])
print(lista_ingredienti[2])
print(lista_ingredienti[3])
print(lista_ingredienti[4])
print(lista_ingredienti[5])
```

```
pane
pomodori
farina
acqua
sale
mozzarella
```

Soluzione 2) Posso usare un ciclo **while**, ovvero fare un ciclo sopra gli indici della lista e usare tali indici per accedere agli elementi della lista, dal primo all'ultimo.

In [18]:
```
lista_ingredienti = ['pane','pomodori','farina','acqua','sale','mozzarella']

lunghezza = len(lista_ingredienti)

idx = 0

while idx < lunghezza:

    print(lista_ingredienti[idx])

    idx += 1
```

```
pane
pomodori
farina
acqua
sale
mozzarella
```

Soluzione 3) Posso usare il ciclo **for** per scorrere direttamente gli elementi della lista.

In [19]:
```
lista_ingredienti = ['pane','pomodori','farina','acqua','sale','mozzarella']

for ingrediente in lista_ingredienti:

    print(ingrediente)
```

```
pane
pomodori
farina
acqua
sale
mozzarella
```

In [ ]: