Prontuario di Python per il laboratorio di calcolo

Introduzione

Python è un linguaggio di programmazione *interpretato*. Significa che il codice sorgente che scriviamo viene passato ad un altro programma, detto *interprete*, che lo esegue.

In generale, Python è un linguaggio di più alto livello rispetto a C. Il concetto di "linguaggio di programmazione di alto livello" non è rigoroso, ed ha senso solo in un contesto relativo: dire che il linguaggio X è di livello più alto rispetto a Y significa che con X è più difficile (o, in alcuni casi, impossibile) interagire con le primitive del sistema operativo e/o dell'hardware rispetto ad Y. Nel caso specifico di Python *vs.* C, le due principali differenze (per quanto concerne questo corso) sono:

1. In Python non esistono i puntatori (cioè variabili che contengono *indirizzi di memoria*, li vedremo meglio alla fine del corso)
2. La memoria in cui risiedono le variabili che utilizziamo è gestita (*managed*) direttamente da Python (in maniera cosiddetta trasparente). Non esiste quindi una distinzione tra *stack* e *heap*, né funzioni per allocare/deallocare dinamicamente la memoria (cioè funzioni equivalenti a `malloc` e `free` in C).

Nel corso utilizzeremo Python principalmente come uno strumento per creare grafici (più o meno interattivi) e per semplici analisi dati.

Eseguire un programma scritto in Python

I programmi scritti in Python (spesso definiti *script*) devono essere passati ad un interprete per essere eseguiti. Sui sistemi che utilizzate questo si riduce a passare lo script come primo argomento del programma `python3`. Ad esempio, se il nostro script si chiama `prova.py`, per lanciarlo dovremo eseguire il seguente programma:

```
$ python3 prova.py
```

L'importanza di indentare correttamente

Sarete sorpresi di scoprire che i codici Python non possono essere indentati come vi pare. In Python, infatti, il cosiddetto *whitespace* (spazi e tablature) è parte integrante del codice. In Python, infatti, i blocchi di codice (che in C sono delimitati dalle parentesi graffe) devono essere *indentati* (cioè spostati in avanti tramite l'inserimento di una certa quantità di spazi o tablature). In effetti, non basta che istruzioni appartenenti allo stesso blocco siano indentate, ma devono anche essere indentate allo stesso modo, cioè con la stessa quantità di *whitespace*: se utilizzate due spazi per la prima riga, dovrete usare due spazi anche per le seguenti.

Ecco un esempio in C:

```c
int a = 1;
if(a == 1) {
    printf("a è diverso da zero!\n");
    a = 0;
}
```

Che in Python diventa:

```python
a = 1
if a == 1:
    print("a è diverso da zero!")
    a = 0
```

Mentre l'indentazione in C è opzionale (ma raccomandata: migliora la leggibilità del codice), in Python è obbligatoria: provate a toglierla e vedrete che l'interprete se ne lamenterà!

Nella maggior parte degli script che vedremo nelle esercitazioni non ci sono blocchi di codice oltre a quello principale, e quindi non avremo la necessità di indentare alcunché (con alcune eccezioni).

Il comando `import`

All'inizio di quasi ogni script Python si trovano alcune linee del tipo

```
import os, sys
import numpy as np
```

Il comando `import` serve per *importare* delle librerie nel nostro script in modo da poterle utilizzare. Se vogliamo importare più librerie è sufficiente scrivere i nomi di tutte le librerie uno dopo l'altro, separandoli con delle virgole.

In generale, tutto ciò che è contenuto in una libreria (funzioni, variabili o classi, di cui parleremo più avanti nel corso) può essere utilizzato subito dopo che questa è stata importata utilizzando la sintassi `mylib.mystuff`, dove `mylib` è la libreria importata e `mystuff` il nome dell'oggetto cui si vuole accedere. Ad esempio, dopo la prima riga dell'esempio sopra è possibile accedere alla versione di Python che stiamo utilizzando con il comando `sys.version_info`.

Alcune volte, soprattutto nel caso di librerie a cui si accede spesso, può far comodo importare una libreria dandole un altro nome. Per far ciò si utilizza la particella `as`. Nell'esempio sopra si accederà alle funzionalità della libreria `numpy`utilizzando l'alias `np` (ad esempio, `np.zeros(100)` invece di `numpy.zeros(100)`).

Algebra lineare (e non solo): la libreria `numpy`

La libreria `numpy` (solitamente importata con l'*alias* `np`) permette di leggere, manipolare e stampare facilmente vettori, matrici e, più in generale, tensori di qualunque ordine. Nonostante non faccia parte della libreria standard di Python, `numpy` è *de facto* lo standard per lavorare con oggetti di questo tipo.

Gli *array* di `numpy`

Cominciamo con un **nota bene:** gli oggetti base di `numpy` si chiamano *array* ma sono, in generale, diversi dagli array che imparerete ad usare in C. Non confondete i due concetti!

In `numpy` un *array* può essere visualizzato come una griglia (in generale multidimensionale) di valori. Se definiamo d la dimensione dell'array, allora per $d = 1$ si ha un vettore, per $d = 2$ una matrice e così via. Nel corso utilizzeremo più che altro *array* unidimensionali (cioè con $d = 1$), che chiameremo vettori. Una volta inizializzato un vettore x, si può accedere al suo elemento i-esimo con la sintassi `x[i]`. Ad esempio, possiamo modificare il valore del primo elemento utilizzando il comando `x[0] = 1` (ricordatevi che tutti gli indici partono da 0!). Diversamente dal C, potete anche utilizzare indici negativi per accedere agli *array*: `x[-1]` indica l'ultimo elemento dell'array, `x[-2]` il penultimo e così via.

Nota Bene: se provate ad accedere ad un elemento che non esiste il programma genererà un errore! Per esempio, se il vettore x ha 10 elementi, il comando `x[11] = 5` farà terminare lo script con un errore.

Alcune funzioni utili

Durante il corso faremo un uso molto limitato delle potenzialità di `numpy`, limitandoci essenzialmente a queste due funzioni:

1. `np.loadtxt('myfile.dat')` permette di inizializzare un array `numpy` con valori letti dal *file* `myfile.dat`. Nel contesto di questo corso nella maggior parte dei casi inizializzeremo due vettori a partire da file contenenti due colonne. Questa operazione si effettua col comando `x, y = np.loadtxt('myfile.dat',`

`unpack=True)`, dove `myfile.dat` è il nome del *file*. Una volta eseguito il comando, le variabili `x` ed `y` conterranno i dati della prima e seconda colonna del file. **Nota Bene:** questo comando dà errore se il *file* contiene più di due colonne. In questo caso bisogna specificare esplicitamente quali colonne si vogliono leggere. Ad esempio, per salvare in `x` la prima colonna ed in `y` la terza si deve eseguire il seguente comando: `x, y = np.loadtxt('myfile.dat', unpack=True, usecols=(0, 2))` (le colonne sono indicizzate partendo dallo 0!).

2. `np.linspace(a, b)` restituisce un vettore di numeri equispaziati nell'intervallo $[a, b]$. Di default, il vettore contiene 50 numeri. Questo numero può essere cambiato passando un terzo argomento alla funzione. Ad esempio, per generare 100 numeri equispaziati nell'intervallo $[0, 10]$ è sufficiente eseguire il comando `np.linspace(0, 10, 100)`.

Graficare con Python: la libreria `matplotlib`

La libreria `matplotlib` (solitamente importata con l'*alias* `plt`) permette di generare figure e grafici di varia natura. Noi utilizzeremo solo in parte le moltissime opzioni e funzionalità che possiede. In quel che segue considereremo `x` ed `y` come due array di `numpy` aventi la stessa lunghezza. La maggior parte degli script che utilizzano `matplotlib` sono composti (grossomodo) di tre parti, come descritto nelle prossime sezioni.

Setup iniziale

È conveniente cominciare con i comandi che controllano l'aspetto del grafico. Alcune funzioni utili sono:

- `plt.title('titolo')` imposta il titolo della figura
- `plt.xlabel('etichetta asse x')` e `plt.ylabel('etichetta asse y')` impostano le etichette (*label*) degli assi x ed y
- `plt.xlim(a, b)` e `plt.ylim(a, b)` impostano il *range* degli assi x ed y

Aggiunta dei *dataset*

Si possono aggiungere curve e *dataset* al grafico utilizzando il comando `plt.plot(x, y)`, dove `x` ed `y` devono essere *array* della stessa lunghezza contenenti le ascisse e le ordinate dei dati che vogliamo graficare. Lo stesso comando può essere ripetuto più volte (con gli stessi o con altri *array*). Il terzo argomento (opzionale) di `plt.plot` permette di specificare il formato con cui la curva verrà visualizzata. Alcuni esempi sono `'x'` per utilizzare delle croci, `'o'` per dei cerchi, `'-'` per una linea continua. Ulteriori parametri opzionali di `plt.plot` permettono di controllare alcune proprietà del *dataset* da graficare. Ad esempio il parametro `label='nome dataset'` permette di impostare l'etichetta che apparirà in legenda. Un elenco esaustivo delle opzioni possibili è consultabile all'indirizzo https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.plot.html.

Visualizzazione (e/o salvataggio su file)

Una volta aggiunti tutti i *dataset* possiamo caricare la legenda col comando `plt.legend()` e mostrare la figura con `plt.show()`. Opzionalmente è anche possibile salvare la figura su *file* con il comando `plt.savefig('prova.png')` *prima* (o al posto di) chiamare `plt.show()`.

Un semplice esempio completo è il seguente:

```
import numpy as np
import pylab as plt

# prepara due array di esempio
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
```

```
# aggiungiamo titolo ed etichette agli assi
plt.title("Prova grafico")
plt.xlabel("Asse x")
plt.ylabel("Asse y")

# aggiungiamo il dataset
plt.plot(x, y, label='prova dataset')
# salviamo la figura nel file grafico.png
plt.savefig('grafico.png')
# mostriamo il grafico
plt.show()
```

Link per approfondire

La maggior parte della documentazione online su Python e le sue librerie sono in lingua inglese. Ecco una lista (non esaustiva) di utili risorse:

- Un libro utile per i principianti è Python 101, consultabile gratuitamente all'indirizzo http://python101.pythonlibrary.org/
- La documentazione ufficiale di Python è consultabile all'indirizzo https://docs.python.org/3/
- Il sito ufficiale di `numpy` non è di facilissima navigazione, specialmente per programmatori "in erba", ed è quindi raccomandabile, per gli interessati, seguire tutorial introduttivi più semplici. Ne trovate un esempio all'indirizzo https://www.guru99.com/numpy-tutorial.html
- Il sito ufficiale di `matplotlib` contiene molte guide, tutorial ed esempi ed è consultabile all'indirizzo https://matplotlib.org/