# Programmazione e Calcolo Scientifico
# Esercitazione Python 01

Prof. Stefano Berrone, Francesco Della Santa

24 Ottobre 2018

**Sommario**

Questa esercitazione ha l'obiettivo di far prendere dimestichezza allo studente con le nozioni di base di Python 3 illustrate nelle lezioni precedenti.

Prima di iniziare, si scarichino i file necessari dalla pagina del corso; questi file conterranno i codici di alcuni esempi precedentemente illustrati a lezione e che dovranno essere ampliati durante l'esercitazione.

Si consiglia, per praticità, di utilizzare l'IDE Pycharm per realizzare l'esercitazione.

## 1 Esercizi di Base

**Suggerimento:** questa esercitazione di appoggia agli esempi già illustrati in classe durante le lezioni. In caso di dubbi una consultazione del materiale fornito potrebbe risultare di aiuto. Non dimenticare inoltre di utilizzare le funzioni *help* e *dir* e, nel caso lavoriate nella shell python, di resettarla sempre dopo aver effettuato una modifica ai moduli (altrimenti la modifica non verrà considerata).

1. Crea un progetto di lavoro di nome *PCSpylaib01* ed al suo interno un *ambiente virtuale* avente per interprete una versione di Python 3 (*assolutamente* ***non*** *python 2*).

   In questa cartella copiare i file scaricati dalla pagina del corso e creare un pacchetto di nome *astrophysics* che li contenga e che, al momento di essere importato, importi "implicitamente" anche i moduli in questione. Per fare ciò si proceda quindi nel seguente modo:

   - creare una cartella *astrophysics* dentro la cartella progetto *PCSpylaib01*;
   - spostare i moduli scaricati in *astrophysics*;
   - creare in *astrophysics* un nuovo modulo *__init__.py* al cui interno è scritta solamente il comando

     ```
     from astrophysics import newtonmod, celestial, givencelestials
     ```

2. Aprire il modulo *celestial.py* e, leggendo i codici opportunamente commentati ed utilizzando la funzione *help*, cercare di capire il funzionamento delle classi *CelestialBody* e la sua sottoclasse *Star*.

Dopo aver letto il contenuto di *celestial.py* aprire il modulo *givencelestials.py* e leggere il codice opportunamente commentato che definisce l'oggetto `earth` della classe *CelestialBody*.

Per esercizio, si provi quindi a creare dei nuovi oggetti e modificare l'oggetto `earth` come descritto:

- lavorando nella shell python, si importi l'oggetto `earth` dal modulo *givencelestials*, per esempio con il comando

```
from astrophysics.givencelestial import earth
```

- Si aggiungano al modulo *newtonmod* le costanti:

$$\begin{aligned} &dist_{media}(Sole, Terra) = 1\,496 \cdot 10^8\,\mathrm{m} \\ &dist_{media}(Sole, Marte) = 2\,279 \cdot 10^8\,\mathrm{m} \\ &dist_{media}(Terra, Luna) = 3\,844 \cdot 10^5\,\mathrm{m} \\ &M_{luna} = 7\,342 \cdot 10^{22}\,\mathrm{kg}\,, \quad R_{luna} = 1\,737 \cdot 10^3\,\mathrm{m} \end{aligned} \tag{1}$$

- lavorando nella shell python, si creino tre oggetti `mars`, `moon` e `sun`, i primi due delle classi *CelestialBody* ed il terzo della sottoclasse *Star*, sfruttando i valori costanti definiti nel modulo *newtonmod.py* (prendere spunto dai comandi utilizzati per creare `earth` in *givencelestials*);
- lavorando nella shell python e utilizzando il metodo *info*, esplorare gli attributi degli oggetti `earth`, `mars`, `moon` e `sun`.
- lavorando nella shell python, utilizzando i metodi *add_satellite* o *add_assatellite*, si aggiornino gli attributi *satellites, satellites_dist, satelliteof, satelliteof_dist* degli oggetti `earth`, `mars`, `moon` e `sun` secondo i loro "rapporti satellitari" (Terra e Marte satelliti del Sole e Luna satellite della Terra).

  Utilizzando poi il metodo *info*, esplorare nuovamente gli attributi degli oggetti `earth`, `mars`, `moon` e `sun` osservando i cambiamenti;
- verificato di aver svolto correttamente i punti precedenti dell'esercitazione, aggiungere gli oggetti `mars`, `moon` e `sun`, più gli aggiornamenti dei rapporti satellitari, nel modulo *givencelestials* (basterà riscrivere all'interno del modulo i codici lanciati nella shell.).

  Così facendo, per fututi esercizi, si potrà importare direttamente questi oggetti da *givencelestials* senza bisogno di ricrearli

3. Modificare la classe *CelestialBody* (modulo *celestial*) aggiungendo:

   - l'attributo di istanza $g$ che indichi l'accelerazione gravitazionale del corpo in questione, ricordando la formula:

$$g = \frac{G \cdot M_{corpo}}{R_{corpo}}\,; \tag{2}$$

   - un metodo *weight(self, m)* che, data in input una generica massa $m$ posta in prossimità della superficie del corpo, restituisca in output il

suo peso in kilogrammi-forza (ricordarsi della funzione *newton_to_kgf* nel modulo *newtonmod* e che la forza peso di un corpo di massa $m$, in prossimità della superficie di un corpo celeste, è sempre $F = mg$). Dopo aver creato questo metodo, si resetti la shell e si importino gli oggetti `earth`, `mars`, `moon` e `sun` da *givencelestials*, verificando quale sia il peso in kg-forza su ognuno di questi corpi celesti di un uomo con massa 80 kg.

4. Definendo i metodi speciali *__gt__*, *__ge__* ed *__eq__* nella classe *CelestialBody*, ridefinire gli operatori >, >= e == analogamente a come già illustrato dal codice fornito per gli operatori < e <=.

5. Osservando le definizioni dei metodi *add_satellite* e *add_assatellite* della classe *CelestialBody*, definire i seguenti nuovi metodi:

   - Due metodi *remove_satellite(self, name)* e *remove_assatellite(self, name)* che, sfruttando il metodo *pop* dei dizionari, eliminino le relazioni satellitari tra l'oggetto considerato e gli oggetti con nome *name*.

## 2 Esercizi Extra

Questi esercizi sono stati principalmente pensati come esercitazioni extra per verificare l'effettiva conoscenza delle basi del linguaggio Python. Non devono essere svolte necessariamente durante l'esercitazione in aula, ma si consiglia di provare comunque a svolgerli tutti, eventualmente proseguendo il lavoro a casa.

1. Definendo i metodi speciali *__iadd__* e *__isub__* nella classe *CelestialBody*, sovrascrivere gli operatori += e −= affinché:

   **A += (B, d):** equivalga all'esecuzione di *A.add_satellite(B, d)*;

   **A −= 'name':** equivalga all'esecuzione di *A.remove_satellite('name')*.

2. Aggiungere un metodo *Gforce_satellites(self, name)* nella classe *CelestialBody* che calcoli la forza gravitazionale tra l'oggetto della classe ed un oggetto nell'elenco di satelliti o di cui è satellite[1];

3. Modificare, nella classe *CelestialBody*, il metodo *Gforce* aggiungendo dei *keyword arguments* `**typedist` affinché:

   - `corpo.Gforce(m, d)` restituisca la forza calcolata rispetto alla distanza $d$ tra i centri delle masse (versione già implementata nel codice);
   - `corpo.Gforce(m, d, typedist='surf')` restituisca la forza calcolata rispetto alla distanza $\hat{d} = d + R_{corpo}$ tra i centri delle masse.

4. Si crei una nuova classe *SolarSystem* in *celestials* tale che:

   - *__init__* prende in argomento: una stringa per il *nome*, un oggetto *Star* ed un numero variabile (anche nessuno) di oggetti *CelestialBody* che ricopriranno il ruolo dei pianeti del sistema;

[1] supponendo non possano esserci due oggetti con stesso nome in entrambi gli elenchi

- abbia un attributo *name* uguale alla stringa *name* passata nell'inizializzazione;
- abbia un attributo *star* uguale all'oggetto *Star* passato nell'inizializzazione;
- abbia un attributo *planets* uguale ad un dizionario *(nome pianeta, ogg. CelestialBody)* che contenga i pianeti passati nell'inizializzazione più i satelliti indicati nell'attributo *satellites* della stella;

  **Importante:** se i pianeti dell'inizializzazione già non erano elencati tra i satelliti della stella (attributo *satellites*), né la stella era indicata nel loro attributo *satelliteof*, allora aggiornare gli attributi di entrambi gli oggetti;
- abbia un attributo *planets_stardist* uguale ad un dizionario *(nome pianeta, distanza media dalla stella)* rispetto ai pianeti passati nell'inizializzazione più i satelliti indicati nell'attributo *satellites* della stella;

  **Importante:** poiché nell'inizializzazione non devono essere indicate le distanze dei pianeti inseriti, fare sì che con una chiamata alla funzione *input* (dentro __init__) l'utente possa immettere tale distanza interattivamente dalla shell. Si aggiornino poi, analogamente a come scritto nel punto precedente, anche le distanze negli attributi *satellites_dist* della stella e *satelliteof_dist* del pianeta.
- abbia un attributo *massa* uguale alla massa totale data dalla stella, i suoi pianeti ed i satelliti dei suoi pianeti (supponendo un pianeta non possa avere altri pianeti come satelliti e che una luna possa essere satellite solamente di un pianeta e nient'altro);
- abbia un attributo *radius* uguale alla massima distanza tra la stella ed uno dei suoi pianeti;
- abbia un attributo *obj_type* uguale alla stringa 'Oggetto Celeste (Sistema Solare)';
- abbia un metodo *info* analogo alla classe *CelestialBody*;
- abbia un metodo *add_planet(self, planet, stardist)* che aggiunga un pianeta al sistema solare. Dovranno essere anche aggiornati gli attributi dell'oggetto pianeta e dell'oggetto stella per descrivere la loro "relazione satellitare";
- abbia un metodo *remove_planet(self, name)* che rimuova il pianeta col nome indicato dal sistema solare. Dovranno essere anche aggiornati gli attributi dell'oggetto pianeta e dell'oggetto stella per descrivere la loro "relazione satellitare" terminata;
- ridefinendo il metodo speciale *__repr__* si faccia in modo di visualizzare la stringa "< SolarSystem object *nome*, mass: *valore massa*, radius: *valore raggio*, star class (yerks): *classe della stella* >".

5. Aggiungere al modulo *givencelestials.py* un oggetto *solsyst* della classe *SolarSystem* avente per stella il sole e per pianeti la Terra (con Luna annessa) e Marte.