



# Modulo PyEphem

## Utilizzo con Python – Speciale AstroPi



In questo tutorial faremo la conoscenza di **Ephem**, la libreria che permette di calcolare con precisione -in un dato istante- la posizione dei satelliti naturali ed artificiali e dei pianeti, e molto altro; dato poi che per le sfide di [astro-pi.org](http://astro-pi.org) utilizzeremo il linguaggio Python, vedremo in particolare **PyEphem** ed il suo utilizzo con **Thonny**, l'ambiente di programmazione integrato col quale potremo scrivere e testare i nostri programmi. Per questo tutorial non serve avere delle schede elettroniche.

Scarica i file sorgenti qua: [coderdojotrento.it/astropi2](https://coderdojotrento.it/astropi2)

## Thonny, l'IDE per Python

---

Se stai utilizzando un **Raspberry Pi**, sei fortunato: il linguaggio Python e Thonny sono già installati. Trovi Thonny qui: menù principale → Programmazione → “Thonny Python IDE” (ti ricordo che IDE sta per “Integrated Development Environment”, ovvero “Ambiente integrato di programmazione”: grazie a questi ambienti di lavoro abbiamo infatti tutto quello che ci serve per sviluppare i nostri programmi, dall'editor per scrivere il file di codice sorgente, al compilatore, al debugger, etc. etc.).

Se invece usi un PC, puoi installare il linguaggio Python da [python.it/download](https://python.it/download) : noi utilizzeremo la versione 3 (in questo momento siamo alla 3.6.3). Devi poi scaricare ed installare Thonny: trovi l'installer e le istruzioni sul sito ufficiale [thonny.org](https://thonny.org)

**N.B.** Se trovi difficoltà nell'installazione di Thonny o delle librerie, prova a seguire le istruzioni che trovi qui: [coderdojotrento.it/pyzero](https://coderdojotrento.it/pyzero)

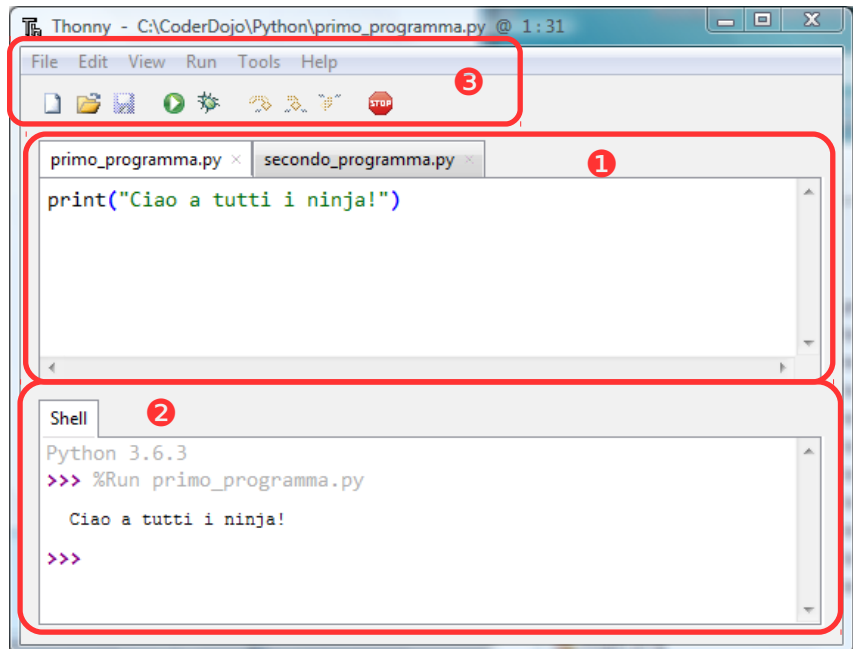




## L'IDE di Thonny

Thonny è molto semplice da utilizzare, ancor più della “Python 3 (IDLE)” che forse hai già utilizzato sul Raspberry Pi oppure sul PC: in una sola finestra sono infatti integrati sia i **tab che contengono il codice sorgente** (①), che la **shell Python** (②) che interpreta il sorgente Python e mostra l’output del programma.

Nel **menù e nella barra dei comandi** (③) troviamo al volo tutte le principali funzionalità per gestire i file e lanciare i nostri programmi.



## Installazione del modulo PyEphem

Prima di poter utilizzare le funzionalità della libreria **ephem**, dobbiamo installare il pacchetto Python che la contiene. Se usiamo il Raspberry Pi, dobbiamo aprire una shell di comando e lanciare l’istruzione “`sudo pip3 install ephem`” (①); il nostro Raspberry Pi scaricherà il modulo dalla rete, lo compilerà, configurerà ed installerà per noi: dopo un po’ di attesa (②) dovremmo vedere il messaggio finale che segnala l’avvenuta installazione (③).

```

pi@astropi-0: ~
File Edit Tabs Help
pi@astropi-0:~$ sudo pip3 install ephem
Downloading/unpacking ephem
  Downloading ephem-3.7.6.0.tar.gz (739kB): 739kB downloaded
  Running setup.py (path:/tmp/pip-build-4ye_k6ue/ephem/setup.py) egg_info for package ephem

Installing collected packages: ephem
  Running setup.py install for ephem
    building 'ephem._libastro' extension
      arm-linux-gnueabi-hf-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -g -fstack-protector-strong -Wformat -Werror=format-security -D_FORTIFY_SOURCE=2 -fPIC -Ilibastro-3.7.6 -I/usr/include/python3.4m -c ex
uild/temp.linux-armv7l-3.4/libastro-3.7.6/chap95_data.o build/temp.linux-armv7l-3.4/extensions/data/saturne.9910.o build/temp.linux-armv7l-3.4/extensions/data/jupiter.1020.o build/temp.linux-armv7l-3.4/extensions/data/uranus.1020.o build/temp.linux-armv7l-3.4/extensions/data/jupiter.9910.o build/temp.linux-armv7l-3.4/extensions/data/mars.9910.o build/temp.linux-armv7l-3.4/extensions/data/uranus.9910.o build/temp.linux-armv7l-3.4/extensions/data/saturne.1020.o build/temp.linux-armv7l-3.4/extensions/data/mars.1020.o -o build/lib.linux-armv7l-3.4/ephem/_libastro.cpython-34m.so

Successfully installed ephem
Cleaning up...
pi@astropi-0:~$

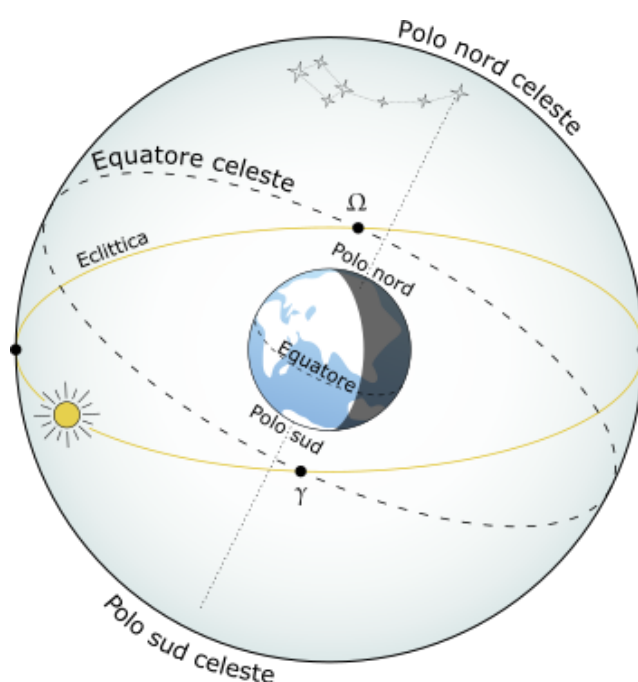
```

Se invece stiamo usando un PC, l'installazione è ancora più semplice (a patto che siano installati i "Build Tools" di Visual Studio): Thonny stesso infatti è in grado di cercare ed installare le librerie che vogliamo. Dal menù "Tools" selezioniamo la voce "Manage packages ...", nella casella di testo che compare scriviamo "ephem" e premiamo il tasto "Search". Una volta trovato il pacchetto, lo installiamo cliccando sul tasto "Install".

## Le coordinate celesti

Piccolo ripassino per ricordarci come vanno interpretate le informazioni che otterremo dalla libreria PyEphem: per "trovare" un corpo celeste in cielo (un pianeta, un satellite, o una stella), dobbiamo conoscere le sue **coordinate**.

In maniera simile a come facciamo sulla Terra utilizzando **latitudine** (altezza rispetto all'equatore) e **longitudine** (l'ampiezza dell'arco sull'equatore, a partire dal meridiano di Greenwich), possiamo localizzare un corpo celeste immaginando di proiettare sulla volta celeste i riferimenti che usiamo sulla Terra: avremo quindi l'**equatore celeste**, il **polo nord celeste** (coincidente con la **Stella polare**) ed il **polo sud celeste**.

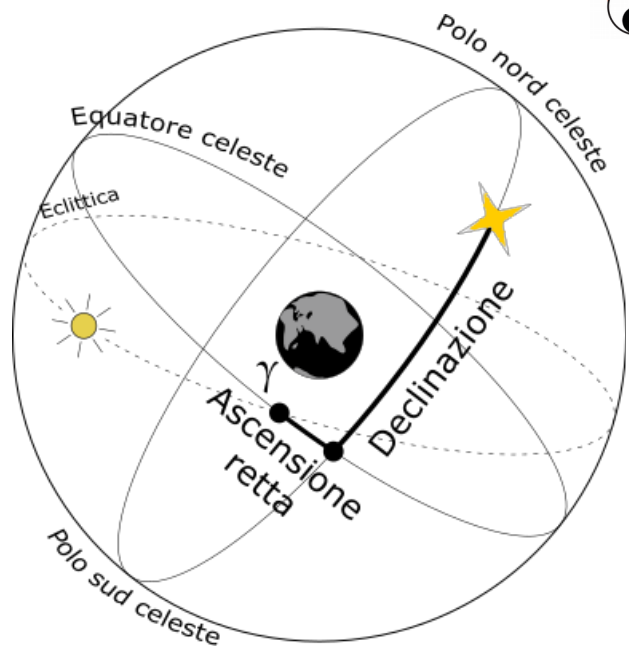




Per individuare ad esempio la stella a quattro punte del disegno qui a fianco, determineremo la sua **declinazione** (come la latitudine terrestre, è l'altezza rispetto all'equatore) e la sua **ascensione retta**: questa corrisponde alla longitudine, con l'unica differenza che non possiamo utilizzare il meridiano di Greenwich come riferimento, dato che la terra continua a girare e noi abbiamo bisogno di un punto di partenza stabile, come fosse disegnato e fisso sulla volta celeste.

Il punto che stiamo cercando si chiama  **$\gamma$  (gamma)**, ed è il punto in cui l'**eclittica** (la linea designata sulla volta celeste dal Sole nel suo moto apparente) incontra l'equatore celeste mentre entra nell'emisfero celeste settentrionale.

Il Sole passa per il punto  **$\gamma$**  quando si realizza l'equinozio di primavera: si trova nella costellazione dell'Ariete (ed infatti  **$\gamma$**  ne è il simbolo).



## I corpi celesti (“bodies”) in PyEphem

Cominciamo subito ad usare la libreria `ephem` come se fosse un **catalogo di corpi celesti** (“*bodies*” in inglese): lo schema da utilizzare nei nostri programmi sarà sempre quello indicato nell'immagine qui a fianco.

Vediamo le istruzioni, riga per riga:


[riga 1] **importiamo** la libreria **ephem**;

[riga 3] definiamo una **variabile nella quale memorizzare il corpo celeste** che ci interessa: `ephem` può fornirci dei riferimenti al Sole (Sun), i pianeti (Mars, Jupiter, Venus, Mercury, etc.), la luna (Moon) ed un centinaio tra le stelle più luminose (`ephem.star("Polaris")`);

[riga 4] chiediamo di effettuare il calcolo della posizione del corpo celeste invocando il metodo **compute()**;

[righe 5-6] se invocando `compute()` facciamo una specie di fotografia dell'oggetto, ora possiamo **consultare** la foto: possiamo chiedere alla libreria in quale costellazione si trova l'oggetto (riga 5) e quali sono le sue coordinate celesti.

Gli attributi **ra** e **dec** di riga 6 sono proprio l'ascensione retta (*right ascension*, in inglese) e la declinazione:

nella finestra della shell vediamo questi due angoli  espressi nel formato <gradi><minuti primi><minuti secondi><centesimi di secondo>. Della costellazione vien fornito il nome breve ed il nome esteso (sempre in inglese). Ma... Ophiuchus?

```

Thonny - /home/pi/...m01_body.py @ 7:1
File Edit View Run Tools Help
ephem01_body.py x
1 import ephem
2
3 sole = ephem.Sun()
4 sole.compute()
5 print(ephem.constellation(sole))
6 print('%s %s' % (sole.ra, sole.dec))
7

Shell
>>> %Run ephem01 body.py
('0ph', 'Ophiuchus')
17:32:59.47 -23:17:22.1
>>> |
  
```





# Lo zodiaco

Se vuoi ripassarti i segni dello zodiaco (e capire cosa sia questo Ofiuco), questo programmino fa per te: a partire dal primo gennaio -settimana per settimana- andremo a vedere la posizione del Sole nel corso del 2018.

Useremo la libreria **datetime** per fare qualche calcolo con le date ed **ephem** come prima, con la differenza che alla funzione `compute()` adesso passiamo una data specifica [riga 10 evidenziata].

```
ephem02_costellazioni.py x
1 import ephem
2 import datetime
3
4 sole = ephem.Sun()
5 primo_gennaio = datetime.datetime.strptime("2018-01-01", "%Y-%m-%d")
6 sette_giorni = datetime.timedelta(days=7)
7
8 for num_settimane in range(0, 53):
9     data = primo_gennaio + sette_giorni * num_settimane
10    sole.compute(data)
11    print('Data %s (settimana n°%d): %s' % (data, num_settimane, ephem.constellation(sole)))

Shell
Data 2018-12-03 00:00:00 (settimana n°48): ('Oph', 'Ophiuchus')
Data 2018-12-10 00:00:00 (settimana n°49): ('Oph', 'Ophiuchus')
Data 2018-12-17 00:00:00 (settimana n°50): ('Oph', 'Ophiuchus')
Data 2018-12-24 00:00:00 (settimana n°51): ('Sgr', 'Sagittarius')
Data 2018-12-31 00:00:00 (settimana n°52): ('Sgr', 'Sagittarius')

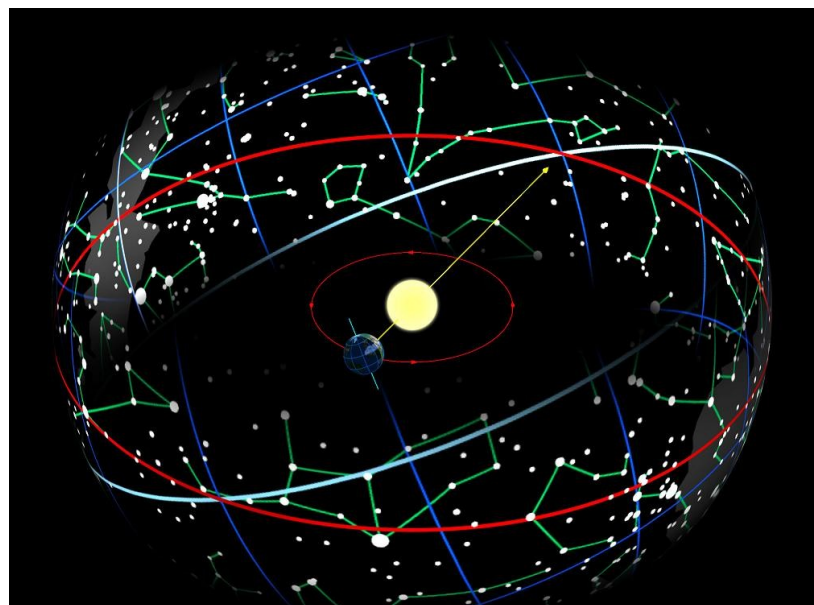
>>>
```

Ed ecco di nuovo l'Ofiuco! È una costellazione che sta fra Scorpione e Sagittario, che è attraversata dall'eclittica ma che non compare tra le dodici sorelle più famose che fanno parte dello "zodiaco astrolo-gico" (vedi la spiega-zione su [Wikipedia](#)).

Nel disegno qui a fianco si può vedere (in rosso) l'eclittica, ovvero il percorso che il Sole sembra compiere sulla volta celeste, mentre la Terra gli ruota attorno.

In verde sono evidenziate le varie costellazioni ed in azzurro è segnato pure l'equatore celeste ed il reticolo di meridiani e paralleli celesti.

È anche bene evidenziata l'inclinazione (di circa 23.5°) dell'eclittica rispetto all'equatore celeste: invertendo il punto di vista, ritroviamo quella inclinazione andando a vedere com'è posizionato l'asse di rotazione terrestre rispetto al piano dell'eclittica.

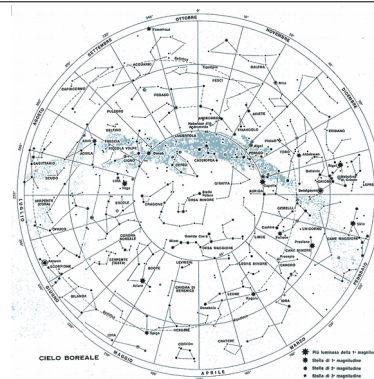




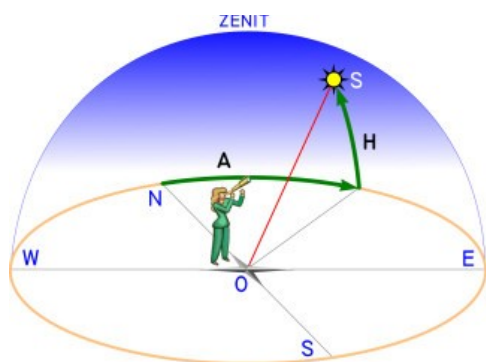
# L'osservatore



Se vogliamo trovare un oggetto nel cielo utilizzando le coordinate ascensione retta e declinazione, dobbiamo disporre di una mappa delle stelle simile a quella indicata qui a fianco: questa griglia di meridiani e paralleli celesti disegnata sopra le "stelle fisse" è un **sistema di riferimento assoluto**. Ma a meno di disporre di strumenti sofisticati o di conoscere per bene le costellazioni (che peraltro di giorno non si possono vedere), non è comodo per noi osservatori terrestri.



Per sfruttare un sistema di riferimento più pratico, Ephem permette di **definire un osservatore**: dopo aver indicato dove questo *Observer* si trova sulla Terra, Ephem ci fornirà le coordinate dell'oggetto che vogliamo osservare nel **sistema di riferimento alt-azimutale**.



Per individuare un oggetto in questo sistema di riferimento, ci servono le coordinate **azimut** (rotazione orizzontale rispetto al Nord, misurata in senso orario: puoi aiutarti con una bussola, dato che l'azimut dei punti cardinali Nord, Est, Sud ed Ovest corrisponde rispettivamente a 0°, 90°, 180° e 270°) e **altezza** (di quanti gradi dobbiamo alzare lo sguardo rispetto all'orizzonte).

Nel disegno qui a fianco, l'azimut è l'angolo A, mentre l'altezza è H: se vuoi approfondire l'argomento, leggi la sezione "L'astronomo dilettante" del sito [giocomania.org](http://giocomania.org).

Nel programma qui sotto definiamo [righe 3-6] un osservatore terrestre che ha longitudine, latitudine ed altezza sul mare uguali a quelle di Trento; lo usiamo poi nella chiamata `compute()` di riga 9 per determinare la posizione della Luna rispetto ad un osservatore che sta appunto a Trento (azimut e altezza sono infatti due coordinate **relative**).

```
ephem03_osservatore.py x
1  import ephem
2
3  trento = ephem.Observer()
4  trento.lon = '11.121064'
5  trento.lat = '46.070888'
6  trento.elevation = 194
7
8  luna = ephem.Moon()
9  luna.compute(trento)
10 print(ephem.constellation(luna))
11 print('Azimut: %s - Altezza: %s' % (luna.az, luna.alt))

Shell
>>> %Run ephem03_osservatore.py
('Oph', 'Ophiuchus')
Azimut: 165:19:55.8 - Altezza: 23:34:21.4
```



# La traiettoria del Sole



Analizza il programma riportato qui sotto: anche aiutandoti con il nome del file o il titolo del capitolo (☺), riesci a capire quale output produce?

```
ephem04_percorso_sole.py ✕
1  import ephem
2  import datetime
3
4  trento = ephem.Observer()
5  trento.lon = '11.121064'
6  trento.lat = '46.070888'
7  trento.elevation = 194
8
9  sole = ephem.Sun()
10 sei_di_mattina = datetime.datetime.strptime("2018-01-01 06:00", "%Y-%m-%d %H:%M")
11 cinque_di_sera = datetime.datetime.strptime("2018-01-01 17:00", "%Y-%m-%d %H:%M")
12 quarto_d_ora = datetime.timedelta(minutes=15)
13
14 ora = sei_di_mattina
15 while ora <= cinque_di_sera:
16     trento.date = ora
17     sole.compute(trento)
18     print('Ora %s: Azimut: %s - Altezza: %s' % (ora, sole.az, sole.alt))
19     ora += quarto_d_ora

Shell
Ora 2018-01-01 16:30:00: Azimut: 245:24:11.5 - Altezza: -8:20:16.1
Ora 2018-01-01 16:45:00: Azimut: 247:55:52.4 - Altezza: -10:43:30.3
Ora 2018-01-01 17:00:00: Azimut: 250:26:12.4 - Altezza: -13:09:19.0
>>>
```

**Suggerimenti.** Come nel programma di pagina 4, facciamo ricorso alla libreria `datetime` per fare qualche calcolo sulle date e le ore: stavolta partiamo dalle sei di mattina del primo giorno del 2018 e lo incrementiamo di un quarto d'ora (un "delta" di tempo pari a 15 minuti) finché non arriviamo alle cinque di sera (gli estremi degli intervalli di analisi sono definiti a riga 10 e 11, ed il ciclo `while` che si estende da riga 15 a riga 19 "spazzola" tutta la giornata).

In maniera simile a quanto fatto nel programma di pagina 5 definiamo un osservatore terrestre posizionato a Trento, ma stavolta [riga 16 evidenziata] indichiamo anche l'ora precisa nella quale avviene l'osservazione: in questo modo possiamo analizzare sotto che angolo (azimut e altezza) il sole viene visto da Trento nel corso di tutta la giornata.

**Osservazioni.** La data scelta è in pieno inverno, quindi -guardando l'altezza- si nota come il sole sia visibile in cielo per poco tempo (solo dalle 7:00 alle 15:30) e rimane comunque sempre basso sull'orizzonte (non raggiunge nemmeno i 21°); prova a cambiare la data mettendo ad esempio il primo luglio, e vedrai che il sole rimane visibile per molto più tempo e raggiunge altezze maggiori.

**Sfida!** Se ora includi la libreria `math` e cambi il formato del `print()` di riga 18 in questo modo

```
print('%s;%s' % (math.degrees(sole.az), math.degrees(sole.alt)))
```

puoi anche provare a copiare l'output in un foglio di calcolo per poi visualizzare i dati in un grafico: le due colonne di dati sono organizzate come fossero in un file `.CSV` e rappresentano le coordinate della traiettoria espressa in gradi.





# Il formato TLE per i satelliti terrestri

I fisici - soprattutto quando si fanno aiutare dai loro amici matematici - sono molto bravi nel **calcolare la traiettoria di un sasso**, a patto che vengano fornite la posizione e la velocità iniziale. In maniera analoga, conoscendo i parametri orbitali (inclinazione, eccentricità dell'orbita, velocità di rivoluzione, etc.) di un satellite terrestre in un dato istante di tempo, è possibile prevedere in maniera accurata la sua posizione negli istanti successivi.

Se quindi riusciamo ad ottenere dei dati aggiornati sulle caratteristiche attuali della traiettoria, possiamo **prevedere dove il satellite sarà nelle prossime ore o anche nei prossimi giorni**, ammesso che nel frattempo non avvengano correzioni nella traiettoria.

Se vogliamo avere dei dati aggiornati sulle caratteristiche della traiettoria di un satellite di nostro interesse, dobbiamo visitare il sito [celestrak.com/NORAD/elements](http://celestrak.com/NORAD/elements) che riporta i dati di un sacco di satelliti, stazioni orbitanti e pure di detriti che stanno girando attorno alla Terra. In particolare nel file [celestrak.com/NORAD/elements/stations.txt](http://celestrak.com/NORAD/elements/stations.txt) sono presenti i dati di tutte le stazioni spaziali: i dati sono raggruppati per righe e le prime tre riguardano non a caso la Stazione Spaziale Internazionale ("ISS" sta infatti per "International Space Station").

I parametri orbitali sono nel formato **TLE ("two-line element")** ed hanno questo aspetto:

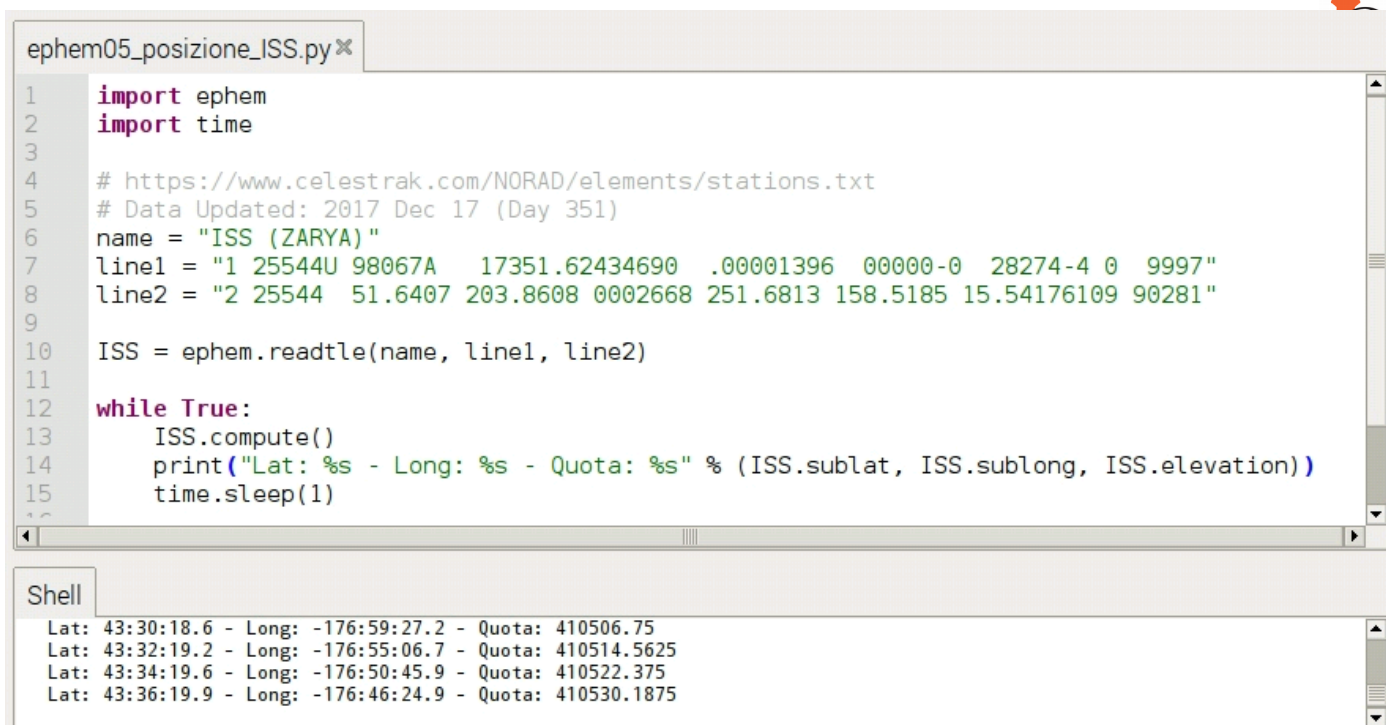
```
ISS (ZARYA)
1 25544U 98067A 17351.88704390 .00001256 00000-0 26159-4 0 9997
2 25544 51.6408 202.5505 0002666 252.5058 188.4660 15.54176482 90322
```

A parte la riga di intestazione che riporta il nome della stazione spaziale, nelle due righe successive sono riportati tutti i parametri che servono per descrivere la traiettoria: certo per noi è difficile capire il significato di tutti (se vuoi, puoi trovarli [qui](#)), ma l'importante è sapere che PyEphem può farlo al posto nostro!

Nel programma che segue compaiono le tre righe che descrivono la ISS [righe da 6 a 8] e a riga 10 viene usato il metodo **readtle** per ottenere il riferimento al satellite. Oltre a tutti i parametri che caratterizzano un "body", per i satelliti artificiali PyEphem fornisce anche latitudine e longitudine del punto sulla Terra sopra il quale il satellite sta passando, e la sua quota di volo [vedi riga 14].







```
ephem05_posizione_ISS.py
1 import ephem
2 import time
3
4 # https://www.celestrak.com/NORAD/elements/stations.txt
5 # Data Updated: 2017 Dec 17 (Day 351)
6 name = "ISS (ZARYA)"
7 line1 = "1 25544U 98067A 17351.62434690 .00001396 00000-0 28274-4 0 9997"
8 line2 = "2 25544 51.6407 203.8608 0002668 251.6813 158.5185 15.54176109 90281"
9
10 ISS = ephem.readtle(name, line1, line2)
11
12 while True:
13     ISS.compute()
14     print("Lat: %s - Long: %s - Quota: %s" % (ISS.sublat, ISS.sublong, ISS.elevation))
15     time.sleep(1)
16
Shell
Lat: 43:30:18.6 - Long: -176:59:27.2 - Quota: 410506.75
Lat: 43:32:19.2 - Long: -176:55:06.7 - Quota: 410514.5625
Lat: 43:34:19.6 - Long: -176:50:45.9 - Quota: 410522.375
Lat: 43:36:19.9 - Long: -176:46:24.9 - Quota: 410530.1875
```

Prova a trascrivere il programma ed a lanciarlo dopo aver aggiornato le linee del TLE consultando il sito del NORAD: puoi confrontare i dati calcolati con uno dei siti che riportano la posizione aggiornata della stazione spaziale. Prova questo [isstracker.com](http://isstracker.com) oppure [quello ufficiale dell'ESA](#) Siamo bravi, vero?, a calcolare la traiettoria del sasso!

N.B. Il programma contiene al suo interno un "while True" che è un ciclo infinito: per interromperlo, devi utilizzare il tasto rosso di stop dell'interfaccia di Thonny.

## Giorno e notte sulla ISS

Nell'ultimo programma di questo tutorial determineremo un'informazione che può essere molto utile per i team che partecipano alla sfida di [astro-pi.org](http://astro-pi.org) : sapere se la ISS sta passando sopra l'emisfero terrestre illuminato dal Sole, o se invece sta volando sopra un punto immerso nella notte. Data la sua elevata velocità di orbita (quasi 30 mila Km/h), infatti, nell'arco delle 24 ore "sperimenta" molte albe e tramonti: compiendo una rivoluzione completa ogni ora e mezza circa, la stazione spaziale vede sorgere il Sole 16 volte al giorno!

Abbiamo già tutti gli strumenti per fare questo calcolo: dobbiamo solo combinare il calcolo delle coordinate del punto che sta sotto la ISS (attributi sublat e sublong) con il concetto di altezza del sole (l'angolo sotto il quale un osservatore a Terra vede il Sole, spostando lo sguardo verso l'alto a partire dall'orizzonte), e spiegare il significato di "crepuscolo".

Il **crepuscolo mattutino** è quell'intervallo di tempo durante il quale si percepisce la luminosità diffusa dall'attraversamento dell'atmosfera da parte dei raggi solari, anche se il Sole non è ancora sorto; in maniera simile, c'è anche il **crepuscolo serale**: dopo il tramonto, anche se non vediamo direttamente il Sole, possiamo godere di un po' di luce diffusa e non ci troviamo direttamente in piena notte.

Se siamo sulla terraferma possiamo dire che il limite del crepuscolo è quando il Sole sta circa 6° sotto l'orizzonte (*domestic twilight*, in inglese). Se invece fossimo in mare aperto, senza montagne e altri ostacoli attorno, noteremmo che il giorno si estende un po' di più: possiamo infatti far partire il crepuscolo nautico ("*nautical twilight*") quando il Sole è 12° sotto l'orizzonte.





## Scriviamo il programma

Per scrivere il nostro programma possiamo prendere il codice riportato nel paragrafo precedente (vedi pag. 7) e modificarlo così (guarda il testo completo nella pagina seguente):

- 1) dato che dobbiamo fare dei calcoli con gli angoli, oltre a `ephem` e `time` importiamo anche la libreria `math` [righe da 1 a 3];
- 2) trascriviamo i dati aggiornati della stazione spaziale presi da [qui](#), e prepariamo la variabile `ISS` [righe da 5 a 8];
- 3) prepariamo le variabili per il Sole e per l'angolo che definisce il crepuscolo (per come è orientato l'angolo *altezza*, 6° sotto l'orizzonte corrispondono al valore -6) [righe da 10 a 11];
- 4) cominciamo un ciclo infinito [righe da 13 a 24] all'interno del quale:
  - a) calcoliamo la posizione dell'ISS [riga 14];
  - b) costruiamo un osservatore che sta esattamente sotto la ISS (sublat e sublong della stazione spaziale, ma a quota zero sul livello del mare) [righe da 15 a 18];
  - c) calcoliamo la posizione del Sole rispetto a questo osservatore e prendiamo l'angolo *altezza* col quale l'osservatore vede il Sole [righe da 19 a 20];
  - d) decidiamo se è giorno o notte confrontando l'altezza del Sole con l'angolo del crepuscolo e stampiamo a video le informazioni acquisite [righe da 21 a 23]; dopo una pausa di un secondo, infine, ricominciamo il ciclo [riga 24].

```
ephem06_giorno_o_notte_ISS.py x
1  import ephem
2  import time
3  import math
4
5  name = "ISS (ZARYA)"
6  line1 = "1 25544U 98067A 17351.62434690 .00001396 00000-0 28274-4 0 9997"
7  line2 = "2 25544 51.6407 203.8608 0002668 251.6813 158.5185 15.54176109 90281"
8  ISS = ephem.readtle(name, line1, line2)
9
10 sole = ephem.Sun()
11 crepuscolo = -6 # il crepuscolo comincia 6 gradi sotto l'orizzonte
12
13 while True:
14     ISS.compute()
15     osservatore = ephem.Observer()
16     osservatore.lat = ISS.sublat
17     osservatore.long = ISS.sublong
18     osservatore.elevation = 0 # sul livello del mare, sotto la ISS
19     sole.compute(osservatore)
20     altezza_sole = math.degrees(sole.alt)
21     giorno_o_notte = "giorno" if altezza_sole > crepuscolo else "notte"
22     print("Lat: %s - Long: %s ed e' %s (infatti l'altezza del sole e' %d°)" %
23           (ISS.sublat, ISS.sublong, giorno_o_notte, altezza_sole))
24     time.sleep(1)
```

```
Shell
Lat: -23:43:12.0 - Long: 154:50:30.1 ed e' notte (infatti l'altezza del sole e' -9°)
Lat: -23:46:20.8 - Long: 154:59:27.7 ed e' notte (infatti l'altezza del sole e' -9°)
Lat: -23:43:29.5 - Long: 155:02:05.3 ed e' notte (infatti l'altezza del sole e' -9°)
Lat: -23:40:38.1 - Long: 155:04:42.7 ed e' notte (infatti l'altezza del sole e' -9°)
```



## Note sulle strutture del linguaggio Python qui utilizzate.



In quest'ultimo programma abbiamo usato due scritture un po' particolari: la prima è l'**espressione condizionale** che compare a riga 21 (alla variabile giorno\_o\_notte assegno il valore "giorno" se è soddisfatta una data condizione, altrimenti il valore "notte") e la seconda è la **continuazione implicita di riga** che si può vedere alle righe 22 e 23. In Python possiamo infatti "spezzare" su più righe un'istruzione molto lunga, a patto che ci siano delle parentesi che raggruppano gli elementi che stanno su righe diverse (in questo caso sono le parentesi degli argomenti del print che permettono di accorpare le due righe).

### Ringraziamenti Questo tutorial:

- \* si trova qua: [coderdojotrento.it/astropi2](http://coderdojotrento.it/astropi2)
- \* fa parte delle "risorse" del CoderDojo Trento
- \* è stato scritto con il supporto di CoderDolomiti
- \* è basato in larga parte [sul seminario web tenuto dal team di AstroPi in data 30/11/2017](#).



CoderDojo  
Trento

Coder  
Dolomiti



Crediti immagini: quelle di pag. 2 e 3, sono state preparate a partire da immagini di [Joshua Cesa \(Opera propria, CC BY 3.0\)](#); quella di pagina 4 è di [Tau'olunga \(Opera propria, CC BY-SA 3.0\)](#); la mappa delle stelle di pag. 5 è estratta dall'opera "[Osservare il cielo](#)" di [Roberto Mura \(Opera propria, Pubblico dominio\)](#) l'utilizzo di quella su altezza ed azimut di pag. 5 è stato concesso dall'autore [Giuseppe Grande \(da visitare assolutamente il suo sito Giocomania\)](#).

