Une introduction au langage python

30 septembre 2020

Sommaire

python, c'est quoi ?

- √ Python est un langage de programmation.
- √ Un programme python – on dira un script – est interprété : son exécution nécessite un interpréteur python.
- √ Un programme (script) python est un fichier texte.
- √ Des interpréteurs python existent sur toutes les plates-formes (Windows, MAC-OS, linux)
- √ Logiciel libre : gratuit, re-distribuable (disponible sous licence open source).
- √ Large communauté de développeurs

Historique

- Créé en 1991 par Guido van Rossum (Néerlandais). van Rossum est le BDFL (dictateur bénévole à vie !) de la communauté.
- 1993 : sortie de Numerical Python, ancêtre de numpy (calcul numérique, algèbre matricielle).
- 2020 : versions en cours des branches 2.7 et 3.7

Quelques caractéristiques de python

- Variables typées (entière, réelle, chaine de caractères, liste, dictionnaire) : typage dynamique.
- Langage orienté objet : classe (attributs, méthodes), héritage.
- Le code peut être groupé en modules thématiques ou en paquets (packages).
- Il existe une vaste bibliothèque de modules python (algèbre, statistiques, graphiques, orientés vers les applications web, etc...)

Ressources en ligne

√ en français :

- site Python Software Foundation
  https://docs.python.org/fr/3.5/tutorial/
- un cours de l'université de Paris (document pdf, 278 pages), contient des conseils d'installation, choix d'éditeurs de texte, ... https://python.sdv.univ-paris-diderot.fr/cours-python.pdf

√ en anglais :

- Tutoriel python, numpy, scipy, matplotlib
  http://www.scipy-lectures.org/index.html
- Livres électroniques http://www.e-booksdirectory.com/listing.php?category=265

Ces liens sont sur la page web de l'UE :
http://wwwens.aero.jussieu.fr/lefrere/master/SPE/

De l'aide sur le web

web

Les deux sites suivants sont de bonnes références :

- √ https ://docs.python.org/3/
- √ https ://docs.scipy.org/doc/numpy/
- √ stackoverflow.com/.../python (forum modéré questions/réponses)

fonction «help»

Pensez à utiliser le help en ligne de commande (sous ipython) :

- `help(nom_de_fonction)` ou `?nom_de_fonction`

Installer un environnement python : anaconda

- Il existe des distributions python comprenant un ensemble de ressources (bibliothèques de modules, interfaces utilisateur, éditeur).
- La distribution «anaconda» contient tout ce dont on a besoin pour le calcul, le graphique, les entrées–sorties : numpy–scipy, matplotlib, ipython, spyder.
- Sur votre ordinateur personnel : télécharger la version 3.x (64 bits) dédiée à votre système (Mac, Windows, linux) : `https://www.anaconda.com/downloads`.
- Installer la distribution (instructions suivant le système d'exploitation).

L'environnement python d'anaconda

- Une fois la distribution anaconda installée, vous disposez :
  - d'un interpréteur python,
  - de nombreux modules de fonctions
  - de différents environnements de travail (console ipython, integrated development environment (IDE) spyder, jupyter notebook)
- l'environnement le plus léger est la console texte ipython, ouverte depuis un terminal.
- Vous aurez également besoin d'un éditeur de texte pour écrire ou modifier vos programmes (gedit, nano, vi (linux), Notepad++ (Windows),BBedit (Mac-OS),...)

Nous utiliserons ici la console ipython («i» signifiant «interactive»), sous linux (Open Suse)

La console ipython est lancée : premiers pas...

Opérateurs arithmétiques de base

```
[1]: 3+2
     5
[2]: 2**3
     8
[3] -1**2
    -1
[4]: 3/2
     1.5
[5]: 3//2  # division entière
     1
[6]: 5 % 2
     1
```

Définition de variables :

```
[6]: x=3
[7]: y=2
[8]: z=x+y
[9]: z
     5
[9]: type(z)
     int
[10]: x, y = 3., 2.
[11]: z=x+y
[12]: type(z)
      float
```

Premiers pas...

La console ipython intègre des commandes unix basiques.

? Déterminer le répertoire courant (pwd), lister le contenu de votre répertoire de travail (ls), se déplacer depuis ce répertoire (cd), créer le répertoire `plots` dans le répertoire de travail (mkdir).

```
[7]: pwd
     /home/mnics/3123456/SPE/
[8]: mkdir maths_SPE
[9]: ls
     maths_SPE/
[10]: ls -l
drwxr-xr-x 2 riw latmos 4096  4 sept. 21:14 maths_SPE/
[11]: cd maths_SPE
[11]: mkdir intro_python
[12]: cd intro_python
```

La directive «import» : Chargement du module numpy avec alias np

```
[13]: x
      3
[14]: s=sin(x)
NameError: name 'sin' is not defined

[15]: import numpy as np
[16]: s=np.sin(x)
      0.1411200080598672 1
[17]: type(s)
      numpy.float64
[18]: ly = np.log(y)
[19]: z = np.e**(np.complex(0,1)*np.pi)
      (-1+1.2246063538223773e-16j)
```

Les fonctions mathématiques courantes n'existent pas par défaut dans python. Il faut importer le (les) module(s) dont on a besoin.

Nommage des variables

- Un nom de variable ou de fonction (un identificateur) comprend un ou plusieurs caractères alphanumériques
- Un identificateur ne peut pas commencer par un caractère numérique.
- Un identificateur est sensible à la case (`x` différent de `X`)
- Les 33 mots clés de python sont à proscrire

```
and       del       from      None    True    as      elif    global
nonlocal  try       while     assert  else    if      not     break
except    import    or        with    class   False   in      pass
yield     continue  finally   is      raise   def     for     lambda
return
```

Types de variables

- Une variable simple a une valeur. Celle-ci peut être de différents types :

type	descriptif	taille en mémoire	valeur min	valeur max
int	nombre entier	variable	illimité	illimité
bool	booléen	1 octet	0 (False)	1 (True)
float	nombre réel	(4) 8 octets	10^{-308}	10^{308}
complex	nombre complex	2×(4) 8 octets	10^{-308}	10^{308}
str	chaine de caract.	variable	–	–

- le typage est dynamique : il est défini lors de l'affectation et modifiable par une nouvelle affectation.
- En plus des types de base, d'autres types de variables sont créés par les modules importés. Par exemple le module numpy comprend les types arange, fraction, décimal, ndarray, etc...

Types de variables

- Il est possible d'imposer un type de variable lors de l'affectation :

```
[1]: x = 10                 # entier
[2]: x = 10.                # réel
[3]: toto = int("300")      # conversion en entier
[4]: str(300)               # conversion d'un nombre
     '300'                  # en chaîne de caractères
[5]: float(300)             # conversion d'un entier
     300.0                  # en flottant (nombre réel)
[5]: bool(-123456789)       # conversion d'un entier en booléen
     True
```

- L'affectation n'est pas une égalité !

```
[1]: a = 1          # un entier de valeur 1
[2]: a = a + 1      # la valeur de a est augmentee de 1
[3]: a
     2
[4]: a == 1         # renvoie une expression booléenne
     False
```

- Une liste est une collection ordonnée de variables pouvant être hétérogène.
- Syntaxe : une suite délimitée par des crochets d'éléments séparés par des virgules,

```
[1]: feu_de_circulation = ["vert", "orange", "rouge"]
[2]: feu_de_circulation[1]
     'orange'
[3]: feu_de_circulation2 = [True,"vert", "orange", "rouge"]
[4]: feu_de_circulation[1]
     'vert'
[5]: type(feu_de_circulation2[0])
     bool
[6]: type(feu_de_circulation2[1])
     str
```

- Les éléments d'une liste sont repérés par un indice entier.
  - Le premier élément est repéré par l'indice 0
  - Le second élément est repéré par l'indice 1
  - Le dernier élément est repéré par l'indice -1

Manipulation de listes

? Créer une liste `liste1` comprenant deux éléments 'a' et 'b'.

? Créer une liste `liste2` comprenant trois éléments 1, 2 et 3.

? Créer une liste `liste3` dont le premier élément est `liste1` et le second `liste2`.
  - Afficher les deux premiers éléments de liste2
  - Extraire (afficher) liste2 à partir de liste3.
  - Afficher le deuxième élement du premier élément de liste3.

? Créer `liste3` contenant les éléments de `liste1` et `liste2`

Manipulation de listes

```
[4]: liste1 = ['a','b']
[5]: liste2 = [1,2,3]
[6]: liste3 = [liste1,liste2]
[7]: liste3[1]
     [1, 2, 3]
[8]: liste3[0][1]
     'b'
[9]: liste2[0:2]
     1, 2
```

Les tuples

Définition

Un tuple est une collection ordonnée d'objets non modifiables, éventuellement hétérogènes.

Syntaxe : une suite délimitée par des parenthèses d'éléments séparés par des virgules.

```
[1]: feu_de_circulation = (True, "vert", "orange", "rouge")
[2]: feu_de_circulation[0]
     True
[3]: feu_de_circulation[1] = "bleu"
TypeError: 'tuple' object does not support item assignment
```

Les tuples sont utiles pour définir des constantes.
Les éléments d'un tuple sont repérés par un indice entier.

Certaines fonctions (i.e. commandes) renvoient des tuples.

Les dictionnaires

Définition

Un dictionnaire (dict) python est une collection non-ordonnée d'objets accessibles par un nom, i.e. une «clé». C'est un type de données permettant de stocker des couples (clé : valeur).
syntaxe : Collection de couples {clé : valeur} entourée d'accolades.

```
[1]: ma_voiture = {
        "marque" : "Renault",
        "couleur": "rouge",
        "consommation": 8.5,
        "vitesse_max" : 150
      }
[2]: ma_voiture['vitesse_max']
     150
[3]: ma_voiture[0]
KeyError: 0
```

Types itérables

Les listes, les tuples, les dict sont des variables itérables : il est possible de «parcourir» les éléments dans une boucle.

```
[4] for i in liste2:
        print(i)
1
2
3
```

```
[4] for f in feu_de_circulation:
        print(f)
True
vert
orange
rouge
```

Les objets iterables sont :

- les listes (list)
- les tuples (tuple)
- les dictionnaires (dict)
- les chaînes de caractères (string)
- ...

Itération d'un dictionnaire

Un dictionnaire est itérable (on peut le parcourir), mais il n'est pas ordonné.

- *?* Afficher les clés et valeurs du dictionnaire `ma_voiture` (utiliser la syntaxe `print(cle,':',valeur)`)

Itération d'un dictionnaire

```
[3]: for nom in ma_voiture:
         print(nom,':',ma_voiture[nom])
marque : Renault
couleur : rouge
consommation : 8.5
vitesse_max : 150
```

Plusieurs façons de définir des chaînes de caractères

```
[1]: str1 = "une chaine de caractères \n sur deux lignes"
[2]: str2 = "une chaine sans retour ligne"
[3]: print(str2)
une chaine sans retour ligne
[4]: print(str1)
une chaine de caractères
 sur deux lignes
[4]: str1[0:5]
     'une c'
[5]:  m,n = 1,2;
[6]: print("m = %d et n = %d" %(m,n))
m = 1 et n = 2
[6]: str3 = """ une chaîne
     sur plusieurs
     lignes """
```

Les instructions de contrôle

- Un script python est constitué d'une suite d'instructions qui seront exécutées séquentiellement
- Certaines instructions peuvent être exécutées sous condition, ou peuvent être répétées plusieurs fois.
- On utilisera pour ce faire une «instruction composée»

instruction composée : définition

Une instruction composée comprend :

- une ligne d'en-tête comprenant un mot clé terminée par deux-points « : »
- Un bloc d'instructions indenté par rapport à la ligne d'en-tête. On utilise habituellement trois/quatre espaces par indentation.

Toutes les instructions consécutives au même niveau d'indentation appartiennent au même bloc.

Instruction(s) conditionnelle(s) : mots clés `if`, `else`, `elif`

```
if condition1:
    instructions1
else:
    instructions2
```

? Afficher la température. Si celle-ci est négative afficher "il gèle", sinon "il ne gèle pas"

```
# Saisir la température en C
# Attention!!! T est une chaine
  de caractères
[1]: T = input("Entrez T : ")
```

```
[1]: T = input("Entrez T : ")
[2]: print("la température est de %s C" %T)
[3]: if float(T) <= 0:
        print("il gèle !!!")
     else:
        print("il ne gèle pas")
```

Boucle sur les éléments d'un objet «itérable» : le mot clé `for`

```
for i in liste:
    instruction1
    instruction2
    etc...
```

? Afficher les 5 premières lignes de la table de multiplication de 6.

```
[5]: li = [1,2,3,4,5]
[6]: print("table de multiplication de 6")
[7]: for i in li:
         print("%d x 6 = %d " %(i,i*6))
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
```

Les opérateurs de comparaison

Opérateur	Signification
x == y	x est égal à y
x != y	x est différent de y
x > y	x est plus grand que y
x < y	x est plus petit que y
x >= y	x est plus grand ou égal à y
x <= y	x est plus petit ou égal à y

```
[1]: 3 < 5
True
[2]: 3 > 4
False
```

Opérateur logique `and`

```
[3]: 2 < 3 and 3 < 4
True
[4]: 2 < 3 and 3 > 4
False
```

Opérateur logique `or`

```
[5]: 2 < 3 or 3 < 4
True
[6]: 2 < 3 or 3 > 4
True
```

L'itérateur range

La fonction «range» est un itérateur : génère une suite de nombres entiers en progression arithmétique. L'usage le plus fréquent est de décrire une itération sur les indices d'un tableau.
Il y a deux syntaxes possibles :

- `range(stop)` où `stop` est un entier décrivant le nombre d'entiers à générer, de 0 à `stop - 1`,
- `range([start],stop[, step])` où `start` est l'entier de départ, `step` le pas d'un entier au suivant.

? Pour n compris entre 1 et 5, afficher la parité de n.

L'itérateur range

```
[6]: for i in range(1,6):
         if i/2 == i//2:   # ou i % 2 == 0
            print("%d est pair" %i)
         else:
            print("%d est impair" %i)
1 est impair
2 est pair
3 est impair
4 est pair
5 est impair
```

L'itérateur range

? Écrire une boucle permettant de calculer la somme des n premiers entiers positifs. Comparer à n(n+1)/2.

L'itérateur range

```
[1]: n = int(input("Entrer un entier : ")
[2]: s = 0
[2]: for i in range(1,n+1):
         s = s + i
[3]: print   ("somme des %d premiers entiers = %d" %(n,s))
[4]: print("%dx%d/2 = %d" %(n,n+1,s))
somme des 10 premiers entiers = 55
10x11/2 = 55
```

Les modules

- Dans la version basique, le langage python est extrêmement pauvre.
- Il est nécessaire de charger les ressources logicielles dont on aura besoin.
- Les ressources logicielles sont organisées en modules thématiques.
- Un module est un fichier – ou un ensemble de fichiers – contenant du code python logiquement organisé.
- Un module comprend généralement des définitions de types de variables, des fonctions, des classes (objets).
- Les modules numpy et scipy (calcul), et matplotlib (graphique) contiennent la quasi totalité des outils nécessaires pour le calcul et la présentation graphique de données.

Les modules numpy et matplotlib

```
[1]: import numpy as np
[2]: import matplotlib.pyplot as plt
[3]: np.sin(1)
0.8414709848078965
# Les fonctions np.sin, np.cos, etc s'appliquent aux éléments
# d'un vecteur.
# arange: une extension numpy à la fonction range pour les réels
[4]: X = np.arange(0,100,0.1)
[5]: Y = np.cos(X)*np.sin(X)
[6]: plt.plot(X,Y)
[7]: plt.show(block=False)
```

Les modules numpy et matplotlib

Il existe un racourci permettant de charger à la fois matplotlib.pyplot et numpy lors d'une session ipython : pylab

la commande «magique» :

```
[1]: %pylab
[2]: ...
```

ou, au lancement de ipython depuis un terminal :

```
>> ipython --pylab
```

La commande «magique» %paste permet de coller un bloc d'instructions depuis un éditeur dans la console ipython en conservant les indentations originelles.

Les tableaux numpy

- Le type de base de numpy est le tableau (array).
- un tableau est une collection ordonnée de variables homogènes
- Il existe plusieurs façon de créer des tableaux :

```
[1]: import numpy as np
[2]: X = np.array([1, 2, 3, 4, 5])
[3]: type(X)
     numpy.ndarray
[4]: Y = np.ndarray(len(X),dtype=float)
     # Le tableau Y est créé mais n'est pas initialisé
[5]: for i in range(len(X)):
        Y[i] = np.log10(X[i])
# ou plus simplement:
[5]: Y = np.log10(X)
[6]: Z1 = np.ones((len(X),len(Y)),dtype=int)
[7]: Z2 = np.zeros_like(Z1)
```

Méthodes et attributs des tableaux numpy

Quelques attributs et méthodes des tableaux numpy :

```
[8]: X.shape
     (5,)
[9]: X.ndim
     1
[10]: X.dtype
     dtype('int64')
[11]: X.min()
     1
[12]: X.sum()
      15
[13]: X.cumsum()
      array([ 1,  3,  6, 10, 15])
```

et d'autres encore :

```
[13]: X.mean()
      3.0
[14]: X.std()
      1.4142135623730951
```

Extraction d'un tableau

Extraction (slicing) d'une partie d'une matrice M :

```
[1]: M = np.array([[1,2,3],[4,5,6]])
[2]: M
array([[1, 2, 3],
       [4, 5, 6]])
[3]: M[:,0]          # extrait la première colonne
array([1, 4])
[4]: M[1,:]          # la deuxième ligne
array([4, 5, 6])
[5]: M[:,0:2]        # les deux premières colonnes
array([[1, 2],
       [4, 5]])
```

Duplication d'une variable

Une propriété troublante des tableaux numpy :

```
[14]: X = np.array([1, 2, 3, 4, 5])
[15]: Y = X
[16]: X
    array([1, 2, 3, 4, 5])
[17]: Y
    array([1, 2, 3, 4, 5])
[18]: X[0] = -1
[19]: X
    array([-1,  2,  3,  4,  5])
[20]  Y
    array([-1,  2,  3,  4,  5])
# le tableau Y est identique
# au tableau  X !!!
[21] id(X) == id(Y)
     True
```

```
[21]: Y = X.copy()
[22]: X[1] = 10
[23]: X
    array([-1, 10, 3, 4, 5])
[24]: Y
    array([-1, 2, 3, 4, 5])
# Cette fois, le tableau Y
# n'est pas modifié si X
# est modifié.
[25]: id(X) == id(Y)
      False
```

numpy : autres fonctionnalités

Numpy contient bien d'autres fonctionnalités

- Lecture/écriture de fichiers textes
  `np.loadtxt()`
  `np.savetxt()`
- interpolation
  `np.interp()  # interpolation linéaire`
- calcul d'histogramme
  `np.histogram()`, `np.histogram2()`
- génération de nombres aléatoires `np.random.rand()`, `np.random.randn()`

Le module scipy s'appuie sur numpy et contient de nombreuses fonctions mathématiques :
Les fonctions scipy sont groupées en sous-modules :

- stats (scipy.stats)
- ndimage (scipy.ndimage)
- io (scipy.io)
- signal (scipy.signal)
- interpolate ...
- linsolve, ode
- fftpack
- integrate
- ...

Lecture de fichier texte : loadtxt

La fonction numpy.loadtxt() permet de lire des fichiers textes contenant des tableaux – sous la forme de colonnes de nombres – et éventuellement un en-tête descriptif.

```
[1]: import numpy as np
# Lecture d'un fichier texte contenant des colonnes de nombres
# les trois premières lignes sont sautées (entête)
[2]: data = np.loadtxt('file.txt',skiprow = 3)
# les données sont stockées dans le tableau data
```

Structure d'un script

- Un script python est constitué d'une suite d'instructions python. Celles-ci seront exécutées séquentiellement.
- Bonne pratique : les deux premières lignes sont des directives indiquant l'interpréteur du script (python) et le codage des caractères (utf-8)
- Les lignes suivantes sont les «import» des modules dont on aura besoin
- La suite du script contient vos instructions
- Sauver le fichier qui suit sous le nom : `log_vs_dl3.py`
- Pour l'exécuter

```
run log_vs_dl3.py       # depuis la console ipython
python log_vs_dl3.py    # depuis un terminal
./log_vs_dl3.py         # depuis un terminal comme une commande
                          si le fichier est exécutable.
```

Un script python

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-0.5,1,100)
l1 = np.log(1+x)
d3 = x - x**2/2 + x**3/3
plt.figure(1)
plt.clf()
plt.plot(x,l1,'blue',linewidth=2,label="log(1+x)")
plt.plot(x,d3,'red',label="DL ordre 3")
plt.xlabel('x',fontsize = 14)
plt.ylabel('log(1+x)/DL(3)',fontsize = 14)
titre = "comparaison log(1+x)/DL(3)"
plt.title(titre,fontsize=15)
plt.grid(); plt.legend()
plt.show(False)
```

python ? Analyser ce script ? Que réalise t-il ?

Ecriture d'un script

- Un dernier exemple : lecture d'un fichier et tracé d'une colonne de données en fonction de l'autre
- Placer le fichier de données «exoplanets_clean.tsv» dans le répertoire `data/` (créer le si besoin est).

? Écrire un script python permettant de

  - charger les données du fichier `exoplanets_clean.tsv` dans un tableau data.
  - extraire le rayon orbital et la période de révolution dans les tableaux `R` et `P`.
  - tracer le cube du rayon en fonction du carré de la période.
  - Enregistrer le graphique sous le nom «Loi_de_Kepler.png» dans le répertoire `plots/`. Si ce répertoire n'existe pas, créez le.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import numpy as np
import matplotlib.pyplot as plt

# Définition d'une fonction simple
def power(x,n):
   return x**n

fich = "exoplanets.tsv"  # nom du fichier
data = np.loadtxt(fich,skiprows=3)
R = data[:,1]; P = data[:,2]
plt.figure(2)
plt.clf()
plt.loglog(power(R,3),power(P,2),'x',color='blue',linewidth=2)
plt.xlabel('$R^3$',fontsize = 14)
plt.ylabel('$T^2$',fontsize = 14)
titre = "T$^2$ vs R$^3$ (troisième loi de Kepler)"
plt.title(titre,fontsize=15)
plt.grid();
if not os.path.isdir("../plots/"):  # Si le répertoire n'existe pas,
   os.mkdir("../plots/")            # on le crée.
plt.savefig("../plots/Loi_de_Kepler.pdf")
plt.show(False)
```