Université Paris-Saclay - Année 2022-2023
Cours accéléré analyse numérique - M2 AMS

TP - Quelques exercices d'introduction à python

1 Présentation.

Python est un langage de programmation interprété. Il en existe plusieurs versions. Dans les machines des salles de TP de l'IMO, il existe les versions 2.7 et 3.6. Nous allons travailler avec cette dernière, mais ces deux versions ne différent essentiellement que par quelques aspects de syntaxe.

Pour écrire et exécuter un programme en python, on peut : soit écrire le programme dans un fichier avec extension `.py`, puis l'exécuter en lançant dans un terminal la commande `python nomdufichier.py` ; soit utiliser un environnement de programmation tel que `spyder` ou `pyzo`, qui permet d'écrire le fichier et de l'exécuter au même endroit. On peut aussi utiliser des `notebooks` ; on en parlera très bientôt.

2 Syntaxe de base.

Exercice 1. [`print` et `format`]
Exécutez les commandes suivantes :

```
print(0)
print("bonjour")
a = 2
b = 3
print("a = ", a, " b = ", b)
print("a+b = {0:d}".format(a+b))
print("a*b = {0:f}".format(a*b))
print("a^b = {0:10.3e}".format(a**b))
print("{0:d} = {0:3d} = {0:9d}".format(a))
print("{0:d} = {0:3d} = {0:.9f}".format(a))
```

Exercice 2. [Listes et tuples]
Une liste permet de stocker plusieurs objets de type éventuellement différent. On accède à ses éléments à l'aide de crochets. Les indices commencent à 0. Un tuple s'utilise comme une liste mais on ne peut pas modifier ses éléments.

Exécutez les commandes suivantes :

```
print("\n Les listes \n et les tuples")
La = [1, 2, 3]
Tb = (1, 2, 3)
Lc = [4, 5, 6]
Td = (0, 1, 2)
La[0]=3
print("La = ",La)
Tb[0]=2
print("On peut ajouter deux listes : ", La+Lc)
print("On peut ajouter deux tuples : ", Tb+Td)
```

Exercice 3. [`range`]
`range` crée une liste d'entiers en progression arithmétique.

Exécutez les commandes suivantes :

```
r1=range(10)
r2=range(3,10)
r3=range(2,10,2)
r4=range(10,0,-1)
print(r1)
list(r1)
list(r2)
list(r3)
list(r4)
```

Exercice 4. [Les boucles inconditionnelles (`for`) et conditionnelles (`while`), les tests (`if-elif-else`)] Exécutez les commandes suivantes :

```
# Ceci est un commentaire

# Attention aux deux points a la fin de l'instruction for et a
# l'indentation dans la ligne suivante
for k in range(10):
    print(k)

for k in range(10,0,-1):
    print(k,"\n")

u = 100
U = [u]
while u>1:
  if u%2==0: # le symbole % pour modulo
      u = u//2 # division entiere
  else:
      u = 3*u+1
  U.append(u) # ajoute un element a une liste
print(U)

for n in range(10):
    if n==0:
        print(n,"est nul")
    elif n%2==1:
        print(n,"est impair")
    else:
        print(n,"est pair")
```

3 Le package `numpy`.

Le module Numpy permet de manipuler des tableaux à plusieurs dimensions. Il ajoute en effet le type `array`, qui est similaire à une liste, mais dont tous les éléments sont du même type, entiers, flottants ou booléens.

Le module Numpy possède des fonctions basiques en algèbre linéaire.

La commande `import` permet d'importer des modules de python. Les modules importants pour le calcul scientifique sont chargés avec les commandes suivantes, que l'on doit mettre en tête de son script :

```
import numpy as np
import numpy.random as rd
import numpy.linalg as nlin
import scipy as sc
import math
import scipy.linalg as slin
```

Exercice 5. [`array`, `zeros`, `ones` et `eye`]
`array` permet de créer une matrice, ou tableau, à partir d'une ou plusieurs listes.

Exécutez les commandes suivantes :

```
import numpy as np

print("\narrays\n")
x=np.array([0,1,2])
lx=[0,1,2]
print(x)
print(lx)
print("\n x est un objet de type",type(x))
print("\n lx est un objet de type",type(lx)) # les types sont differents

y=np.array(range(10))
z=np.array([[0,1,2],[3,4,5]])

print(y)
print(z)
```

Les fonctions `zeros` et `ones` permettent de créer des tableaux dont tous les coefficients sont respectivement 0 et 1, avec la structure souhaitée.

Exécutez les commandes suivantes :

```
print("\narrays avec 0 et 1\n")
x=np.zeros(3) # vecteur
y=np.zeros((3,1)) # matrice colonne
z=np.ones((1,3)) # matrice ligne
A=np.zeros((3,3)) # matrice carree 3*3
B=np.ones((3,4)) # matrice rectangulaire 3*4
print(x)
print(y)
print(z)
print(A)
print(B)
```

`eye` permet de créer des arrays à partir de l'identité.

Exécutez les commandes suivantes :

```
print("\narrays Identite\n")
I=np.eye(4)
J=np.eye(5,4)
K=np.eye(5,4,k=2)
L=np.eye(5,k=1)
print(I)
```

```
print(J)
print(K)
print(L)
```

Exercice 6. [`rand`]
`rand` permet de créer un array avec des coefficients pris de manière aléatoire entre 0 et 1.

Exécutez les commandes suivantes :

```
import numpy.random as rd

A=rd.rand(3,3)
print(A)
a,b=-2,2
B=a+(b-a)*rd.rand(3,3) # pour un tableau avec des coefficients aleatoires entre
                       # -2 et 2
print(B)
```

Exercice 7. [`linspace` et `arange`]
`linspace` et `arange` permettent de créer des vecteurs dont les coefficents sont en progression arithmétique.

Exécutez les commandes suivantes :

```
v=np.linspace(0,1,5)
w=np.arange(0,1,0.25)
z=np.arange(0,1.25,0.25)
print(v,"\n",w,"\n",z,"\n")
print("v=z")
```

Exercice 8. [`diag`]
Si l'argument de `diag` est unidimensionnel (vecteur), `diag` crée une matrice diagonale à partir de cet argument. Si l'argument de `diag` est bidimensionnel (matrice), `diag` crée un array unidimensionnel (vecteur) avec la diagonale de l'argument.

Exécutez les commandes suivantes :

```
v=np.linspace(0,1,5)
print(v,"\n")
A=np.diag(v)
print(A,"\n")
B=np.diag(v,k=1)
print(B,"\n")
C=np.diag(v,k=-2)
print(C,"\n")
w=np.diag(B)
print(w,"\n")
```

Exercice 9. [`shape`, `ndim` et manipulation de tableaux]
`shape` retourne les dimensions d'un tableau. `ndim` retourne la dimension de l'array (1 si vecteur, 2 si matrice,...).

Exécutez les commandes suivantes :

```
A=rd.rand(3,5)
v=rd.rand(4)
print(A,"\n")
```

```
print(v,"\n")

print(A.shape,A.ndim,"\n")
print(v.shape,v.ndim,"\n")
```

On peut extraire des éléments ou des parties d'un array comme suit. Attention, les indices dans les arrays comme dans les listes commencent à 0.

```
print(v[0 ],"\n")
print(v[:],"\n")
print(v[-1],"\n")
print(A,"\n")
print(A[1,1],"\n")
print(A[0,:],"\n")
print(A[:,1],"\n")
print(A[0:2,1:4],"\n")
```

Exercice 10. [Opérations sur les tableaux]
Exécutez les commandes suivantes :

```
T=[1,2,3] # liste
X=np.array(T) # numpy array

print(X,"\n")
print(X+1,"\n") # ajoute 1 a tous les elements de X
print(2*X,"\n") # multiplie par 2 tous les elements de X
print(X**3,"\n") # eleve au cube tous les elements de X

Y=np.empty(X.shape) # cree un tableau vide pour reserver la place memoire
print(Y)
print(Y.shape)

for k in range(3):
    Y[k]=np.sin(X[k]) # boucle pour remplir Y ;
                      # on peut faire mieux et on doit eviter les boucles

W=np.sin(X) # la bonne facon de faire

print(W)
print(Y)
print(W-Y)
```

D'autres fonctions peuvent s'appliquer à un numpy array : `np.tan, np.exp, np.cos`.

Exercice 11. [`sum` et `prod`]
Exécutez les commandes suivantes :

```
A=np.eye(4)
A[0,2]=3
A[1,0]=-1
print(A)

print(np.sum(A,0)) # somme par colonnes
print(np.sum(A,1)) # somme par lignes
print(np.sum(A)) # somme de tous les elements de A
```

```
print(np.sum(abs(A)))
print(np.prod(A,0)) # produit par colonnes,...
```

4 Algèbre linéaire.

Le sous-module de numpy `numpy.linalg` fournit certaines fonctions liées à l'algèbre linéaire. On peut le charger avec la commande

```
import numpy.linalg as nlin
```

Avec ce module, on peut calculer la transposée d'une matrice, le produit matriciel entre deux matrices, des normes d'une matrice ou d'un vecteur, le déterminant d'une matrice, l'inverse d'une matrice, résoudre un système linéaire,...

Exercice 12.
Exécutez les commandes suivantes :

```
A=rd.rand(4,4)
print(A,"\n")
print(np.transpose(A),"\n") # transposee de A
print(A*np.eye(4),"\n") # produit terme a terme entre A et l'identite
print(np.dot(A,np.eye(4)),"\n") # produit matriciel de A par l'identite

B=nlin.inv(A) # inverse de A
print(np.dot(A,B),"\n") # identite

y=np.arange(4)
b=np.dot(A,y)

yy=nlin.solve(A,b) # resout le systeme Ax=b, dont la solution est le vecteur y

print(y)
print(yy)
print(y-yy) # ce sont les memes vecteurs

print(nlin.norm(y)) # norme euclidienne
print(nlin.norm(y,1)) # norme 1
print(nlin.norm(y,np.inf)) # norme infini

M=np.array([[1,2],[3,4]])
print(nlin.det(M)) # determinant de la matrice M
```

5 Le package `matplotlib`.

Le module `matplotlib.pyplot` possède la fonction `plot`, qui permet de tracer des lignes brisées à partir de deux vecteurs de la même taille. On peut le charger avec la commande :

```
import matplotlib.pyplot as plt
```

Voici un exemple d'application.

Exercice 13.
Exécutez les commandes suivantes :

```
plt.figure(1)
plt.plot(X,np.sin(X),'b',label='sin') # b pour la couleur bleu
plt.plot(X,np.cos(X),'r--',label='cos') # r pour la couleur rouge (red),
                                        # -- pour le trace pointille
plt.plot(0,0,'kx') # k pour noir, x pour le symbole
plt.xlabel('x')
plt.ylabel('f(x)')
plt.axis([-0.2,6.4,-1,1])
plt.legend()
plt.title('Les fonctions sinus et cosinus')
plt.show()
```

6 Créer des fonctions.

Voici, dans trois exemples, comment créer des fonctions.

Exercice 14.
Exécutez les commandes suivantes :

```
def ma_fonction(n,y): # cette fonction s'appelle ma_fonction et elle a deux
                      # arguments en entree
    i=0
    Z=0
    while i<=n:
        Z+=y**i # le meme que Z=Z+y**i
        i+=1
    return Z # valeur retournee par la fonction ma_fonction

W=ma_fonction(4,2)
print(W)
```

On peut créer une fonction sans arguments en entrée ou en sortie, elle ne retourne alors aucune valeur.

```
def autre_fonc():
    x=np.linspace(0,1,100)
    plt.plot(x,x**2)
    plt.show()

autre_fonc()
```

Une autre manière de créer une fonction, avec la fonction python `lambda`.

```
encore_autre_fonc = lambda x,y : x**y
z=encore_autre_fonc(2,3)
print(z)
```