

TP2A - BIBLIOTHÈQUES Python

Le but de ce TP est d'introduire le concept de bibliothèques en Python et de commencer à en utiliser une particulière, la bibliothèque `numpy`.

1 Qu'est-ce qu'une bibliothèque ?

1.1 Calcul d'une racine

Nous avons évité jusqu'à présent le calcul de racine carrée. La raison en est très simple : c'est passablement compliqué. Mais c'est pour nous une bonne occasion de découvrir le concept de bibliothèque.

Exercice 1

★

1. Recopier le code suivant :

```
1 def sqrt(x):
    assert x >= 0
    if x == 0:
        return 0
5 steps = 10
  y = 1
  for i in range(steps):
    y = y - (y*y - x)/(2*y)
  return y
```

Ce code utilise un algorithme, appelé la méthode de Newton, pour calculer une approximation de la racine

2. Vérifier les valeurs renvoyées par `sqrt(4)`, `sqrt(9)`, `sqrt(16)`, `sqrt(0.25)`, `sqrt(1)` et `sqrt(0)`. *carrée.*
3. Vérifier enfin le résultat de `sqrt(2)*sqrt(2)`, `sqrt(3)*sqrt(3)` et `sqrt(0.5)*sqrt(0.5)`.
4. Que se passe-t-il si on exécute `sqrt(-1)` ?

1.2 Création d'une bibliothèque

Comme nous avons pu le voir, calculer une racine est compliqué. Les fonctions Python sont déjà une aide : plutôt que devoir réécrire l'algorithme à chaque fois, on peut plus simplement faire appel à `sqrt(x)`. Mais il reste un soucis : à chaque fois que l'on aura besoin de la racine, il va falloir recopier la définition de la fonction `sqrt`. Les bibliothèques servent précisément à résoudre cette difficulté.

Exercice 2

★★

1. Enregistrer le fichier précédent sous le nom `utils.py` et le fermer.
2. Créer un nouveau fichier dans le même dossier appelé `TP2a.py` et taper le code suivant :

```
1 import utils
  print(utils.sqrt(4))
```

3. Exécuter le code. Que se passe-t-il ? Sauriez-vous expliquer le rôle de la ligne `import utils` ?
4. Remplacer le code par les variantes suivantes :

```
1 from utils import sqrt
  print(sqrt(4))
```

```
1 from utils import *
  print(sqrt(4))
```

```
1 import utils as u
  print(u.sqrt(4))
```

Y a-t-il des différences à l'exécution ? Commenter.

1.3 Bibliothèques usuelles

L'avantage des bibliothèques est donc dans la *réutilisation* du code. On peut écrire un fichier et réutiliser les fonctions définies autant de fois que l'on veut dans autant de projets que l'on veut. Mais cela va plus loin.

On peut également réutiliser le code fait par d'autres personnes.

Exercice 3

★

Taper le code suivant :

```
1 from math import sqrt
   print(sqrt(4))
```

Que se passe-t-il ? Y a-t-il un fichier `math` dans votre dossier ? Comment expliquer cela ?

Exercice 4

★★

En utilisant la bibliothèque `math`, écrire les trois fonctions suivantes :

1. `def cote(a)` : qui calcule le côté d'un carré d'aire a ;
2. `def surface(r)` : qui calcule la surface d'un disque de rayon r ;
3. `def angle(a,h)` : qui calcule l'angle (en degrés) dans un triangle rectangle d'hypoténuse h et de côté adjacent à l'angle de longueur a .
On pourra utiliser `math.acos`, `math.pi` et `math.sqrt`.

2 La bibliothèque Numpy

2.1 Introduction

Cette année, nous utiliserons principalement (et presque exclusivement) la bibliothèque `numpy`. Elle contient de nombreuses fonctions spécialisées dans le *calcul numérique*, c'est-à-dire qu'elle permet de faire des simulations numériques et des calculs d'approximations. La bibliothèque `numpy` implémente beaucoup de fonctions semblables à `math` mais avec un accent sur la rapidité et l'efficacité numérique.

Presque toute l'année, nous importerons la bibliothèque `numpy` de la manière suivante :

```
1 import numpy as np
```

Exercice 5

★★

Réimplémenter les trois fonctions de l'exercice 4 en utilisant `numpy`.

2.2 Les tableaux numpy

On pourra utiliser `np.arccos`, `np.pi` et `np.sqrt`.

L'outil principal que nous utiliserons cette année est celui des *tableaux numpy*. Pour nous, il s'agit d'un nouveau type de variables. Il y avait jusqu'à les nombres, les chaînes de caractères, les booléens et désormais, il y aura les tableaux `numpy`.

Exercice 6

★

1. Exécuter les lignes suivantes :

```
1 tab = np.zeros(5)
   print(tab)
```

Que se passe-t-il ? Que se passe-t-il si on change le 5 en un autre nombre ? Quel est le rôle de la fonction `np.zeros` ?

2. Remplacer la fonction `np.zeros` par la fonction `np.ones`. Que se passe-t-il ?
3. Utiliser le code suivant :

```
1 tab = np.array([1,2,3])
  print(tab)
```

Que fait la fonction `array` ?

4. Recopier le code suivant :

```
1 n = 10
  tab = np.zeros(n)
  for k in range(n):
    tab[k] = k*k
5 print(k)
```

À votre avis, que peut signifier la syntaxe `tab[k]` ? Rappeler quel est l'ensemble des valeurs parcourues par `k` dans la boucle ? Quel est le numéro du premier élément du tableau `tab` ?

Exercice 7

★★

Écrire une fonction `def fibonacci(n)` : qui prend un entier $n \geq 1$ et renvoie un tableau numpy contenant les n premiers nombres de la suite de Fibonacci, qui est définie par

$$\begin{cases} u_0 = 1 \\ u_1 = 1 \\ \forall n \in \mathbb{N}, u_{n+2} = u_{n+1} + u_n \end{cases} .$$

2.3 Manipulations élémentaires

Exercice 8

★★

Exécutez le code suivant :

```
1 print(np.arange(7))
  print(np.arange(2,6))
  print(np.arange(1,9,2))
  print(np.linspace(1,5,3))
5 print(np.linspace(1,5,5))
  print(np.linspace(1,5,9))
```

Que font les fonctions `np.arange` et `np.linspace` ?

Exercice 9

★

Exécutez le code suivant :

```
1 t1 = np.arange(1,11,1)
  t2 = np.arange(11,1,-1)
  resultat = t1 + t2
  print(t1)
5 print(t2)
  print(resultat)
```

Que se passe-t-il lorsque l'on additionne 2 tableaux numpy ?

De manière générale, on peut utiliser les opérations algébriques usuelles sur des tableaux numpy de même taille et cela va faire l'opération terme à terme.

Exercice 10

★★

Sans écrire de boucles, écrire une fonction `def carres(n)` : utilisant des tableaux numpy et renvoyant un tableau contenant les n premiers entiers carrés en commençant par 0.

Exercice 11

★★

1. Préparer un tableau `numpy` `angles` contenant les valeurs 0 , $\frac{\pi}{6}$, $\frac{\pi}{4}$, $\frac{\pi}{3}$ et $\frac{\pi}{2}$.
2. Exécuter `np.cos(angles)`. Que constate-t-on ?

Exercice 12

★

Certaines fonctions sont au contraire prévues pour agir directement sur un tableau.

1. Construire un tableau `numpy` `A` contenant les nombres 1 , 42 , -2 , 0.75 et 0.25 dans cet ordre.
2. Exécuter les fonctions suivantes une à une et déterminer leurs rôles :

```
1 np.min(A)
  np.max(A)
  np.mean(A)
  np.sum(A)
5 np.prod(A)
```

3. Exécuter également les fonctions suivantes : `np.cumsum` et `np.cumprod`. Qu'est-ce qui les distinguent des fonctions précédentes ?

2.4 Utilisation comme matrices

Les tableaux `numpy` peuvent avoir plusieurs *dimensions*. Pour l'instant, nous les avons toujours présentés en tableau à 1 ligne et plusieurs colonnes, mais il est tout à fait possible de faire des tableaux à plusieurs lignes et plusieurs colonnes. Cela permet notamment de représenter des *matrices*.

Exercice 13

★

1. Créer un tableau `numpy` `A` arbitraire. Puis exécuter `np.shape(A)`. Que fait la fonction `np.shape` ?
2. Exécuter maintenant :

```
1 A = np.eye(5)
  print(np.shape(A))
```

Que s'affiche-t-il ? Comment l'interpréter ? Exécuter directement `print(np.eye(5))`. Que fait `np.eye` ?

3. Que fait `np.ones((2,3))` ?
4. Que fait `np.array([[1,2,3],[4,5,6]])` ?

Exercice 14

★★

1. Afficher le résultat de :

```
1 np.ones((3,3))*np.ones((3,3))
```

Est-ce le résultat attendu ? Quel problème cela pose ?

2. Exécuter :

```
1 np.dot(np.ones((3,3)), np.ones((3,3)))
```

Que fait `np.dot` ?

3. Que font les fonctions `np.linalg.inv` et `np.linalg.det` ? Vérifiez-le sur quelques exemples.