Python - Module NumPy, fonctions, structure conditionnelle

1 Introduction à NumPy

Vous aurez remarqué que les fonctions mathématiques usuelles (exp, ln, cos, etc.) ne sont reconnues par défaut par Python. En effet, Python est avant tout un language de programmation, pas un logiciel de calcul scientifique ou formel. Heureusement, il existe un module (ou bibliothèque) qui contient une bonne partie des instructions de bases permettant d'utiliser Python pour du calcul scientifique : le module NumPy. Remercions au passage la communauté qui maintient à jour depuis plus de vingt ans.

Importation d'une commande Si vous avez besoin d'une commande specifique du module dans votre code, vous pouvez n'importer que celle-ci. Par exemple, si vous avez un programme qui ne nécessite que l'utilisation de la fonction exponentielle, vous appellerez :

```
from numpy import exp
print(exp(1))
```

Importation de la bibliothèque Si vous avez besoin de plusieurs commandes et fonctionnalités de la biliothèque, il est préférable d'importer l'intégralité de celle-ci. Chaque commande sera alors appelée `numpy.commande`. Pour alléger la notation, on définira une abbréviation pour `numpy`.

```
import numpy as np
print(np.exp(1))
```

Fonctions usuelles Les commandes de bases présentent dans NumPy incluent :

`exp` : fonction exponentielle
`cos` et `sin` : fonctions trigonométriques
`floor` : partie entière.
`log` : logarithme népérien
`sqrt` : racine carrée

Deux variables sont aussi prédéfinies dans le module NumPy : `pi` et `e`.

Exercice 1

Commenxer par importer le module NumPy : `import numpy as np`.

1. Pour chacune des commandes suivantes, prévoir sur feuille le résultat affiché par Python, puis entrez les commandes et comparez.

```
np.sqrt(2)**10      np.sin(180)          np.sin(np.pi)
np.cos(np.pi)       np.sqrt(-1)          np.sqrt(2)**2+2
np.sqrt(2)**2-2     np.floor(2/3)        np.floor(2-np.sqrt(2)**2)
np.exp(0)           np.log(1/np.e)       np.exp(np.sin(np.pi/2))-np.e
```

   Les résultats vous semblent-ils corrects ?

2. Déterminer les commandes Python permettant d'effectuer les calculs suivants :

$$\frac{e^\pi}{\pi^e}\ ; \qquad 8\sqrt{5}-3\sqrt{7}\ ; \qquad \left(\frac{2}{3}+\pi e\right)^3\ ; \qquad \sqrt{\left|\sin\left(\ln\left(\frac{2^{10}}{e^{-9}}\right)\right)\right|}.$$

2 La notion de fonction

Le mot **fonction** désigne des objets différents en informatique et en mathématiques. Si en mathématiques il renvoie à la notion de graphe (un espace de départ, un espace d'arrivée, et un sous-ensemble du produit cartésien de ceux-ci), en informatique il désigne une suite d'instruction, pouvant dépendre de paramètres, renvoyant ou non un autre objet.

Syntaxe La syntaxe générale pour une fonction en Python est :

```
def nom_de_la_fonction(argument1,argument2,...,argumentn):
    instruction1
    instruction2
    ...
    dernière instruction
```

Par exemple, on peut définir la fonction suivante dans un fichier .py et l'exécuter (la fonction sera alors connue du compilateur) :

```
def annoncer_train(d,h):
    print('Le train en direction de', d, ' partira à ', h)
    print('-----------------------------------------------')
```

Puis, dans la console, on peut appeler cette fonction :

```
annoncer_train("Melun",'7h16')
```

Le retour de valeur On peut demander à ce que la fonction renvoie un objet en sortie. On utilisera pour cela la commande `return(objet_de_sortie)` à la fin du bloc d'instructions.

```
def hypothenuse(x,y):
    from numpy import sqrt
    return(sqrt(x**2+y**2))
```

Que contiendra `hypothenuse(3,4)` ?

Quelques remarques :

— on signale à Python le début d'un bloc d'instruction par :. Toutes les instructions qui suivent ayant la même indentation (usuellement quatre espaces) appartiennent au même bloc d'instruction.
— Si une fonction requiert l'utilisation d'un module, on peut l'appeler au sein de la fonction.
— Une fonction peut avoir un nombre arbitraire de paramètres d'entrée. Le retour de valeur peut contenir n'importe quel objet.
— L'instruction `return` termine l'exécution : on ne peut le mettre au milieu du bloc d'instructions.

Exercice 2

1. Déterminer ce qu'afficherait le programme suivant, puis vérifier.

```
def myfunction(x):
    y=x
    z=x*y
    y=x+y+z
    z=z-y
    return y+z

x=4
y=10
z=myfunction(y)
print(x+y+z)
```

2. Plus généralement, identifier ce que renvoie `myfunction`.

Exercice 3

Écrire une fonction `secondes(h,m,s)`, prenant en argument un moment de la journéeexprimé en heures, minutes, secondes, et renvoyant le nombre de secondes écoulées depuis le début de la journée.

Exercice 4

L'aire d'un triangle de côtés a, b, c est donné par la formule de Héron :

$$\mathcal{A} = \sqrt{p(p-a)(p-b)(p-c)}$$

où p est le demi-périmètre du triangle.

1. Définir une fonction `demi_perimetre` prenant en arguments trois flottants ou entiers `a,b,c` et calculant le demi-périmètre d'un triangle de côtés de longueurs a, b, c.
2. Définir une fonction `Aire` prenant en arguments trois flottants ou entiers `a,b,c` et calculant l'aire du triangle correspondant.

Exercice 5 Moyennes

Étant donnés deux réels a, b, il existe plusieurs façons de calculer la moyenne de ces deux nombres :

— La moyenne arithmétique : $m = \dfrac{a+b}{2}$.

— La moyenne géométrique : $g = \sqrt{ab}$.

— La moyenne harmonique : $h = \dfrac{2}{\frac{1}{a}+\frac{1}{b}}$.

— La moyenne quadratique : $q = \sqrt{\dfrac{a^2+b^2}{2}}$.

1. À quelles conditions sur a, b ces différentes moyennes sont-elles définies ?
2. Écrire une fonction Python prenant comme arguments deux flottants ou entiers `a,b` et renvoyant, dans l'ordre, ces quatre moyennes.
3. Évaluer cette fonction pour différents couples a, b. Que peut-on conjecturer sur la comparaison entre ces différentes moyennes ?

3 Instructions conditionnelles

Test simple

Les structures conditionnelles permettent de créer des morceaux de programmes dans lesquels une instruction est exécutée sous certaines conditions de l'état courant. La syntaxe Python est :

```
if condition:
    bloc_d_instructions
```

Comme pour tout bloc d'instruction Python, elle repose sur :

— l'usage des deux points `:` pour signifier le début d'un bloc d'instruction.

— l'indentation signifiante : on indente de quatre caractères l'ensemble du bloc.

La `condition` est généralement un objet de type booléen. Si `condition=True`, le programme exécute le bloc d'instruction, si `condition=False` il ignore l'ensemble du bloc. Si on veut exécuter d'autres instructions, on peut bien entendu enchaîner deux instructions conditionnelles successives, mais la syntxe la plus élégante et courante est de proposer une alternative :

```
if condition:
    bloc_d_instructions_1
else:
    bloc_d_instructions_2
```

Exemple : comprendre le script suivant.

```
def pos(x):
    if x>=0:
        print(f"{x} est positif")
    else:
```

```
        print(f"{x} est strictement négatif")

pos(3)
po(-2.4)
pos(-0)
```

Exercice 6

Écrire une fonction `absolue(x)` qui renvoie la valeur absolue d'un entier ou flottant `x`, sans utiliser la commande `abs()`.

Exercice 7

Créer une fonction `racine` qui prend en argument un flottant ou entier `x`, renvoie un message d'erreur de votre choix si celui-ci est négatif, et renvoie $\sqrt{x}$ si c'est possible. Tester votre programme avec plusieurs valeurs.

On rappelle que la biliothèque numpy contient la fonction `sqrt`. Il vous faut donc importer cette commande dans votre fonction.

Exercice 8

L'arrondi à l'unité d'un réel x est $\lfloor x \rfloor$ si $x < \lfloor x \rfloor + \frac{1}{2}$, et $\lfloor x \rfloor + 1$ sinon. Écrire une fonction `arrondi` prenant en argument un flottant `x` et renvoyant son arrondi à l'unité. Tester votre fonction avec plusieurs valeurs.

Tests imbriqués

Si on souhaite tester plus d'une alternative, plutôt que de multiplier les imbrications de tests, on peut utiliser la commande `elif`, contraction de *else if*. La syntaxe est alors :

```
if condition1:
    bloc_d_instructions_1
elif condition2:
    bloc_d_instructions_2
elif condition3:
    :
    :
else:
    dernier_bloc
```

Exercice 9

Écrire une fonction `parité` qui prend en argument un entier `n` et écrit "n est pair" si c'est le cas, "n est impair sinon."

Exercice 10 Des dates

1. Une année est bissextile si ou bien elle est divisible par 4 mais pas par 100, ou bien elle est divisible par 400. Écrire une fonction prenant en argument une année et renvoyant « Cette année est bissextile » si elle l'est et « Cette année n'est pas bissextile » sinon.
2. Créer une fonction `existe` qui prend en argument trois entiers `j,m,a`, et indique s'ils correspondent à une date qui existe (j pour jour, m pour mois, a pour année). Votre fonction devra prendre en compte les années bissextiles : $(29, 2, 2020)$ existe, mais pas $(29, 2, 2021)$.

Exercice 11 Opérateurs logiques

1. Écrire une fonction `negation(P)` qui renvoie la négation de la variable booléenne `P`, sans utiliser la commande `not`.
2. Écrire une fonction `fun_et(P,Q)` qui calcule et renvoie le booléen P et Q sans utiiliser la commande `and`. De même, écrire une fonction `fun_ou(P,Q)` renvoyant P ou Q sans utiiliser la commande `or`.