Introduction à la Programmation sous Python

Partie 2

Les Fonctions

Fonctions prédéfinies

Les fonctions permettent de décomposer un programme complexe en une série de sous-programmes plus simples, lesquels peuvent à leur tour être décomposés eux- mêmes en fragments plus petits, et ainsi de suite…

Par exemple si nous disposons d'une fonction capable de calculer une racine carrée, nous pouvons l'utiliser un peu partout dans nos programmes sans avoir à la ré-écrire à chaque fois.

La fonction intégrée **input()** provoque une interruption dans le programme courant. L'utilisateur est invité à entrer des caractères au clavier et à terminer avec <Enter>

On peut l'invoquer en laissant les parenthèses vides. On peut aussi y placer en argument un message explicatif destiné à l'utilisateur.

```
print('Veuillez entrer un nombre positif quelconque : ')
nn = input()
print('Le carré de', nn, 'vaut', nn**2)
ou encore :
prenom = input('Entrez votre prénom (entre guillemets) : ')
print('Bonjour,', prenom)
```

Définir une fonction

Il arrivera souvent qu'une même séquence d'instructions doive être utilisée à plusieurs reprises dans un programme, et on souhaitera bien évidemment ne pas avoir à la reproduire systématiquement.

Nous avons vu diverses fonctions pré-programmées. Voyons à présent comment en définir nous-mêmes de nouvelles. La syntaxe Python pour la définition d'une fonction est la suivante:

```
def nomDeLaFonction(liste de paramètres):
        ...
        bloc d'instructions
        ...
```

Fonction simple sans paramètres

```
def table7():
        n = 1
        while n <11 :
                print n * 7,
                n = n+1
table7()
```

Pour utiliser la fonction que nous venons de définir, il suffit de l'appeler par son nom comme indiquer le programme.

Le script provoque l'affichage de :

7 14 21 28 35 42 49 56 63 70

Pourquoi définir une fonction ?

1. Créer une nouvelle fonction vous offre l'opportunité de donner un nom à tout un ensemble d'instructions. De cette manière, vous pouvez simplifier le corps principal d'un programme, en dissimulant un algorithme secondaire complexe sous une commande unique, à laquelle vous pouvez donner un nom très explicite, en français si vous voulez.

2. Créer une nouvelle fonction peut servir à raccourcir un programme, par élimination des portions de code qui se répètent. Par exemple, si vous devez afficher la table par 7 plusieurs fois dans un même programme, vous n'avez pas à réécrire chaque fois l'algorithme qui accomplit ce travail.

3. Une fonction est donc en quelque sorte une nouvelle instruction personnalisée, que vous ajoutez vous-même librement à votre langage de programmation.

Fonction avec paramètres

La fonction **table()** telle que définie ci-dessous utilise le paramètre **base** pour calculer les dix premiers termes de la table de multiplication de *base*.

```
def table(base):
        n = 1
        while n <11 :
                print n * base,
                n = n +1

table(13)
```

Pour tester cette nouvelle fonction, il nous suffit de l'appeler avec un argument.

```
>>> table(13)
13 26 39 52 65 78 91 104 117 130
>>> table(9)
9 18 27 36 45 54 63 72 81 90
```

Fonction avec plusieurs paramètres

La fonction **table()** est certainement intéressante, mais elle n'affiche toujours que les dix **premiers** termes de la table de multiplication, alors que nous pourrions souhaiter qu'elle en affiche d'autres. Nous allons l'améliorer en lui ajoutant des paramètres supplémentaires, dans une nouvelle version que nous appellerons cette fois **tableMulti()** :

```
def tableMulti(base, debut, fin):
        print( 'Fragment de la table de multiplication par', base, ':')
        n = debut
        while n <= fin :
                print (n, 'x', base, '=', n * base)
                n = n +1
tableMulti(8, 13, 17)
```

Cette nouvelle fonction utilise donc trois paramètres : la base de la table comme dans l'exemple précédent, l'indice du premier terme à afficher, l'indice du dernier terme à afficher.

Fragment de la table de multiplication par 8 :
13 x 8 = 104
14 x 8 = 112
15 x 8 = 120
16 x 8 = 128
17 x 8 = 136

Utilisation des fonctions dans un script

Les fonctions peuvent aussi s'utiliser dans des scripts. Veuillez essayer vous-même le petit programme ci-dessous, lequel calcule le volume d'une sphère à l'aide de la formule : *V*= 4/3* pi*r^3

```
def cube(n):
        return n**3
def volumeSphere(r):
        return 4 * 3.1416 * cube(r) / 3

r = input('Entrez la valeur du rayon : ')
print ('Le volume de cette sphère vaut', volumeSphere(r))
```

Ce programme comporte trois parties : les deux fonctions **cube()** et **volumeSphere()**, et ensuite le corps principal du programme.
Dans le corps principal du programme, il y a un appel de la fonction **volumeSphere()**.
A l'intérieur de la fonction **volumeSphere()**, il y a un appel de la fonction **cube()**.

Variables locales, variables globales

Les variables locales sont des variables définies à l'intérieur du corps d'une fonction, ils ne sont accessibles qu'à la fonction elle-même.

Les variables définies à l'extérieur d'une fonction sont des *variables globales*. Leur contenu est visible de l'intérieur d'une fonction, mais la fonction *ne peut pas les modifier*.

```
>>> def mask():
...         p = 20
...         print(p, q)
...
>>> p, q = 15, 38
>>> mask()
20 38
>>> print(p, q)
15 38
```

Variables locales, variables globales

Une fonction est capable de modifier une variable globale.
Pour atteindre ce résultat, il vous suffira d'utiliser l'instruction ***global***. Cette instruction permet d'indiquer – à l'intérieur de la définition d'une fonction – quelles sont les variables à traiter globalement.

```
>>> def monter():
...         global a
...         a = a+1
...         print(a)
...
>>> a = 15
>>> monter()
16
>>> monter()
17
>>>
```

Essayez le même exercice en supprimant l'instruction ***global***

Exercices

1. Définissez une fonction **compteCar(ca,ch)** qui renvoie le nombre de fois que l'on rencontre le caractère **ca** dans la chaîne de caractères **ch**. Par exemple, l'exécution de l'instruction :
   print (compteCar('e','Cette phrase est un exemple')) doit donner le résultat : **7**
2. Définissez une fonction **indexMax(liste)** qui renvoie l'index de l'élément ayant la valeur la plus élevée dans la liste transmise en argument. Exemple d'utilisation :
   serie = [5, 8, 2, 1, 9, 3, 6, 7]
   print indexMax(serie)
   4
3. Définissez une fonction **nomMois(n)** qui renvoie le nom du nième mois de l'année. Par exemple, l'exécution de l'instruction :
   print nomMois(4) doit donner le résultat : **Avril**
4. Définissez une fonction **inverse(ch)** qui permette d'inverser l'ordre des caractères d'une chaîne quelconque. (La chaîne inversée sera renvoyée au programme appelant).
5. Définissez une fonction **compteMots(ph)** qui renvoie le nombre de mots contenus dans la phrase **ph** (On considère comme mots les ensembles de caractères inclus entre des espaces).

Exercices

1. Définissez une fonction **changeCar(ch,ca1,ca2,debut,fin)** qui remplace tous les caractères **ca1** par des caractères **ca2** dans la chaîne de caractères **ch**, à partir de l'indice **debut** et jusqu'à l'indice **fin**.
   Exemples de la fonctionnalité attendue :
   >>> phrase = 'Ceci est une toute petite phrase.'
   >>> print changeCar(phrase, ' ', '*', 8, 12)
   Ceci est*une*toute petite phrase.

2. Définissez une fonction **eleMax(liste,debut,fin)** qui renvoie l'élément ayant la plus grande valeur dans la **liste** transmise. Les deux arguments **debut** et **fin** indiqueront les indices entre lesquels doit s'exercer la recherche.

3. Définissez une fonction **maximum(n1,n2,n3)** qui renvoie le plus grand de 3 nombres n1, n2, n3 fournis en arguments. Par exemple, l'exécution de l'instruction : print(maximum(2,5,4)) doit donner le résultat : 5.

Typage des paramètres

```
>>> def afficher3fois(arg):
...     print(arg, arg, arg)
...

>>> afficher3fois(5)
5 5 5

>>> afficher3fois('zut')
zut zut zut

>>> afficher3fois([5, 7])
[5, 7] [5, 7] [5, 7]

>>> afficher3fois(6**2)
36 36 36
```

Dans cet exemple, vous pouvez constater que la même fonction **afficher3fois()** accepte dans tous les cas l'argument qu'on lui transmet, que cet argument soit un **nombre**, une **chaîne** de caractères, une **liste**, ou même une **expression**.

Valeurs par défaut pour les paramètres

Dans la définition d'une fonction, il est possible de définir un argument par défaut pour chacun des paramètres. On obtient ainsi une fonction qui peut être appelée avec une partie seulement des arguments attendus.

```
>>> def politesse(nom, vedette ='Monsieur'):
...         print("Veuillez agréer ,", vedette, nom, ", mes salutations cordiales.")
...
>>> politesse('Dupont')
Veuillez agréer , Monsieur Dupont , mes salutations cordiales.

>>> politesse('Durand', 'Mademoiselle')
Veuillez agréer , Mademoiselle Durand , mes salutations cordiales.
```

Arguments avec étiquettes

Dans la plupart des langages de programmation, les arguments que l'on fournit lors de l'appel d'une fonction doivent être fournis **exactement dans le même ordre** que celui des paramètres qui leur correspondent dans la définition de la fonction.
Python autorise cependant une souplesse beaucoup plus grande. On peut faire appel à la fonction en fournissant les arguments correspondants dans **n'importe quel ordre, à la condition de désigner nommément les paramètres correspondants**

```
>>> def oiseau(voltage=100, etat='allumé', action='danser la java'):
...         print('Ce perroquet ne pourra pas', action)
...         print('si vous le branchez sur', voltage, 'volts !')
...         print("L'auteur de ceci est complètement", etat)
...

>>> oiseau(etat='givré', voltage=250, action='vous approuver')
Ce perroquet ne pourra pas vous approuver
si vous le branchez sur 250 volts !
L'auteur de ceci est complètement givré

>>> oiseau()
Ce perroquet ne pourra pas danser la java
si vous le branchez sur 100 volts !
L'auteur de ceci est complètement allumé
```

Exercices

1. Définissez une fonction **volBoite(x1,x2,x3)** de manière à ce qu'elle puisse être appelée avec un, deux, ou trois arguments. Si un seul est utilisé, la boîte est considérée comme cubique (l'argument étant l'arête de ce cube). Si deux sont utilisés, la boîte est considérée comme un prisme à base carrée (auquel cas le premier argument est le côté du carré, et le second la hauteur du prisme). Si trois arguments sont utilisés, la boîte est considérée comme un parallélépipède. Par exemple :

   print(volBoite()) résultat: -1 (indication d'une erreur)
   print(volBoite(5.2)) résultat: 140.608
   print(volBoite(5.2, 3)) résultat: 81.12
   print(volBoite(5.2, 3, 7.4)) résultat: 115.44

Exercices

2. Définissez une fonction **eleMax(liste,debut,fin)** qui renvoie l'élément ayant la plus grande valeur dans la liste transmise. Les deux arguments **debut** et **fin** indiqueront les indices entre lesquels doit s'exercer la recherche, et chacun d'eux pourra être omis (dans ce cas la chaîne est traitée d'une extrémité à l'autre). Exemples :

```
>>> serie = [9, 3, 6, 1, 7, 5, 4, 8, 2]
>>> print(eleMax(serie))
9
>>> print(eleMax(serie, 2, 5))
7
>>> print(eleMax(serie, 2))
8
>>> print(eleMax(serie, fin =3, debut =1)
6
```

Modules de fonctions

- Un programme Python est écrit dans un fichier portant l'extension .py. Cependant, pour la plupart des programmes, le recours à un seul fichier n'est pas très pratique car le nombre de lignes de code augmente très rapidement au fur et à mesure de l'ajout de fonctionnalités dans l'application.
- Comment utiliser une même fonction dans plusieurs programmes différents ?
- Comment avoir la définition d'une fonction dans un fichier différent de celui qui contient le programme principal ?
- → Les modules permettent de découper logiquement le code source de l'application à travers plusieurs fichiers. Un module peut même être partagé entre plusieurs applications afin de créer une bibliothèque logiciel réutilisable. Par exemple, Python fournit déjà une bibliothèque standard utilisable par toutes les applications sous la forme de plusieurs modules.

Modules de fonctions

En Python, on peut distinguer trois grandes catégories de module en les classant selon leur éditeur :

- Les modules standards qui ne font pas partie du langage en soi mais sont intégrés automatiquement par Python ;
- Les modules développés par des développeurs externes qu'on va pouvoir utiliser ;
- Les modules qu'on va développer nous mêmes.

Importer un module de fonctions

Les fonctions intégrées au langage sont relativement peu nombreuses : ce sont seulement celles qui sont susceptibles d'être utilisées très fréquemment. Les autres sont regroupées dans des fichiers séparés que l'on appelle des modules.

Les modules sont des fichiers qui regroupent des ensembles de fonctions

Il existe un grand nombre de modules préprogrammés qui sont fournis d'office avec Python. Vous pouvez en trouver d'autres chez divers fournisseurs. Souvent on essaie de regrouper dans un même module des ensembles de fonctions apparentées, que l'on appelle des bibliothèques.

Le module ***math***, par exemple, contient les définitions de nombreuses fonctions mathématiques telles que ***sinus, cosinus, tangente, racine carrée***, etc.
Pour pouvoir utiliser ces fonctions, il vous suffit d'incorporer la ligne suivante au début de votre script :

```
from math import *
```

Importer un module de fonctions

Exemple

```
# Démo : utilisation des fonctions du module <math>

from math import *
nombre = 121
angle = pi/6            # soit 30°
                        # (la bibliothèque math inclut aussi la définition de pi)
print("racine carrée de", nombre, "=", sqrt(nombre))
print("sinus de", angle, "radians", "=", sin(angle))
```

Très nombreuses fonctions sont déjà disponibles pour réaliser une multitude de tâches, y compris des algorithmes mathématiques très complexes (Python est couramment utilisé dans les universités pour la résolution de problèmes scientifiques de haut niveau).
une liste détaillée est accessible sur le lien suivant:
https://docs.python.org/3/py-modindex.html

Modules de fonctions - Exemple

Par exemple, nous allons créer un fichier nommé **puissance.py** qui va définir 2 fonctions : **carre()** et **cube().** Un tel fichier est appelé un module et il va pouvoir être importé dans un autre fichier, et en particulier dans le fichier qui contient le programme principal.

```
def carre(valeur):
    resultat = valeur**2
    return resultat

def cube(valeur):
    resultat = valeur**3
    return resultat
```

Il est maintenant possible d'utiliser dans un programme principal les fonctions qui ont été définies dans le module puissance.py. Pour cela, il faut importer les fonctions à partir du module.

Modules de fonctions - Exemple

Exemple : on importe une seule fonction

```
from puissance import carre

a = 5
u = carre(a)
print("le carre vaut", u)
```

Avertissement

Le fichier `puissance.py` doit être dans le même répertoire que le programme principal (ou bien se trouver dans le « path » de Python).

Exemple : on importe explicitement les deux fonctions

```
from puissance import carre, cube

a = 5
u = carre(a)
print("le carre vaut", u)
v = cube(a)
print("le cube vaut", v)
```

Modules de fonctions - Exemple

Exemple : on importe toutes les fonctions

```
from puissance import *

a = 5
u = carre(a)
print("le carre vaut", u)
v = cube(a)
print("le cube vaut", v)
```

Avertissement

L'importation de toutes les fonctions avec `*` est fortement déconseillée. En effet, elle ne permet pas d'avoir une vision claire des fonctions qui ont été importées. Ceci est donc une source potentielle d'erreurs.

Exemple : on importe le module

```
import puissance

a = 5
u = puissance.carre(a)
print("le carre vaut", u)
v = puissance.cube(a)
print("le cube vaut", v)
```

Note

Dans ce cas, il faut préciser le nom du module devant la fonction.

Modules de fonctions - Exemple

Exemple : on importe le module et on lui donne un alias

```
import puissance as pu
a = 5
u = pu.carre(a)
print("le carre vaut", u)
v = pu.cube(a)
print("le cube vaut", v)
```

Exemple : on importe une fonction d'un module et on lui donne un alias

Il est aussi possible de donner un alias à une fonction comme dans l'exemple suivant :

```
from puissance import carre as ca

a = 5
u = ca(a)
print("le carre vaut", u)
```