Python

Les fonctions et les types de données avec le langage Python

Olivier SNOECK

Les fonctions internes

En programmation, les fonctions sont très utiles pour réaliser plusieurs fois la même opération au sein d'un programme. Elles rendent également le code plus lisible et plus clair en le fractionnant en blocs logiques.

Nous avons déjà utilisés certaines fonctions Python. Par exemple math.cos(angle) du module math renvoie le cosinus de la variable angle exprimé en radian. Nous avons aussi utilisé des fonctions internes à Python comme range() ou len().
Ce sont des fonctions internes à Python. Quelque part dans Python, il y a la programmation de ces fonctions. L'appel des fonctions se fait toujours avec le nom de la fonction et les parenthèses.

Les fonctions internes : exercice

Écrire une fonction **identifiant** prenant en argument deux variables **p** et **n**, correspondant à un prénom et un nom. Cette fonction permettra de créer l'identifiant d'une personne en prenant les 3 premières lettres de son prénom et en l'attachant aux 3 premières lettres de son nom de famille.

Cet identifiant devra être en minuscule même si les variables p et n contiennent des majuscules.

Sur la console voici un exemple d'utilisation de la fonction avec le résultat à obtenir après avoir rentré ces différentes instructions :

```
identifiant("Tata","Yoyo")
```

Le résultat est : `Identifiant : tatyoy`

Olivier
SNOECK

Les fonctions perso

Une fonction est très utilisée en langage Python :
1. La fonction est définie par le mot clef **def** puis la programmation après les **:** et l'indentation.

```
1         def permuter (a,b):
2                 a,b = b,a
```

2. L'appel de la fonction se fait simplement par **permuter (poids,mesure)**.
Ensuite, cette fonction va nous affecter les nouvelles valeurs des variables : **poids,mesure = permuter (poids,mesure)**

Olivier
SNOECK

Les fonctions

```
1  poids = 500
2  mesure = 490
3  def permuter (x, y):
4          print ("poids avant = ",x)
5          print ("mesure avant = ",y)
6          x, y = y, x
7          print ("poids après = ",x)
8          print ("mesure après = ",y)
9          return x, y
10
11 print ("poids au début = ",poids)
12 print ("mesure au début = ",mesure)
13 poids, mesure = permuter (poids,mesure)
14 print ("poids à la fin = ",poids)
15 print ("mesure à la fin = ",mesure)
```

et cette fonction nous permet d'obtenir :

```
poids au début = 500
mesure au début = 490
poids avant = 500
mesure avant = 490
poids après = 490
mesure après = 500
poids à la fin = 490
mesure à la fin = 500
```

Olivier
SNOECK

Les fonctions (variante)

```
1  poids = 500
2  mesure = 490
3  def permuter ():
4      global poids, mesure
5      print ("poids avant = ",poids)
6      print ("mesure avant = ",mesure)
7      poids, mesure = mesure, poids
8      print ("poids après = ",poids)
9      print ("mesure après = ",mesure)
10
11 print ("poids au début = ",poids)
12 print ("mesure au début = ",mesure)
13 permuter ()
14 print ("poids à la fin = ",poids)
15 print ("mesure à la fin = ",mesure)
```

et cette fonction nous permet d'obtenir :

```
poids au début = 500
mesure au début = 490
poids avant = 500
mesure avant = 490
poids après = 490
mesure après = 500
poids à la fin = 490
mesure à la fin = 500
```

Dans cette variante qui utilise des variables globales, il n'y a pas besoin de faire passer les variables.

Exercice 1

Testez ce programme :

```
def doubleur(a_locale_doubleur) :
        return 2*a_locale_doubleur
print("Hello")
nombre = 21
print (doubleur(nombre))
```

Exercice 2

Testez ce programme :

```
def my_function(fname):
        print("-> " + fname + " Snoeck")
my_function("Olivier")
my_function("Corinne")
my_function("Victor")
my_function("Cyriel")
```

Olivier
SNOECK

Exercice 3

Testez ce programme :

```
def compteur3():
        i = 0
        while i < 3:
                print(i)
                i = i + 1

def double_compteur3():
        compteur3()
        compteur3()

print("bonjour")
double_compteur3()
```

Exercice 4

Testez ce programme :

```
def compteur3(j):
        i=0
        while i < j:
                print(i) i = i + 1

def double_compteur3(variable):
        print(variable)
        compteur3(variable)
        print(variable)
        compteur3(variable)

print("bonjour")
toto=5
double_compteur3(toto)
```

Exercice 5

Make a program to solve a 2nd degree equation with the display of the discriminant, and the 2 possible results.
You have to use the math function :
`from math import *` or `from math import sqrt`
If the discriminant is negative, a "no solution in the real" is displayed.
Structure of the program :
In the main, we find the questions about the values of "a", "b" and "c".
Also, we find the call to the function.
Structure of the function :
We find all the calculations, tests and display

Olivier
SNOECK

Les fonctions récursives

Une fonction récursive est une fonction qui s'appelle elle-même jusqu'à ce qu'elle ne le fasse plus.

```
1 def fonction():
2     # ...
3     if condition:
4             # ne s'appelle pas elle-même
5             break
6     else:
7         fonction()
8     # ...
```

Par exemple, voici un décompteur :

```
1 def count_down(nombre):
2     """ Compte à rebours à partir d'un nombre """
3     print(nombre)
4     # Appeler la fonction count_down si le nombre suivant à décompter est positif
5     nombre_suivant = nombre - 1
6     if nombre_suivant > 0:
7         count_down(nombre_suivant)
8 count_down(3)
```

Olivier
SNOECK

Le calcul du factoriel

Si on pose

$$f(n) = n!$$

alors,

$$f(n) = 1 \ x \ 2 \ x \ 3 \ x \ 4 \ x \ .... \ x \ (n-1) \ x \ n$$

Voici un exemple de programme de calcul de factoriel, utilisant le principe de la récursivité.

```
1 def factorielle(n):
2     if n == 0:
3         return 1
4     else:
5         return n  * factorielle(n-1)
6 factorielle(5)
```

1. Modifiez ce programme pour observer toutes les étapes du calcul du factoriel.
2. Créez un nouveau programme de calcul de factoriel en utilisant une liste (sans récursivité).

Les fonctions +

Il est possible de préciser une valeur lors de la définition de la fonction :

```
1 def affichage(prenom, age = 25):
2     print("Bonjour, je m'appelle ",prenom," et j'ai ",age, " ans.")
3
4 affichage("Olivier")
```

Voici ce que j'obtiens sur la console :

```
Bonjour, je m'appelle Olivier et j'ai 25 ans.
```

Mais je peux également préciser cette valeur lors de l'appel à une fonction :

```
1 def affichage(prenom, age = 22):
2     print("Bonjour, je m'appelle ",prenom," et j'ai ",age, " ans.")
3
4 affichage("Olivier",52)
```

Voici ce que j'obtiens sur la console :

```
Bonjour, je m'appelle Olivier et j'ai 52 ans.
```

Olivier
SNOECK

Les fonctions +

Il est possible de passer un nombre arbitraire d'arguments avec le symbole * lors de la définition de la fonction.

```
1 def sommenombres(*nombre):
2     somme=0
3     for n in nombre:
4             somme += n
5     print("Somme = ", somme)
6
7 sommenombres (10,25)
```

Voici ce que j'obtiens sur la console :

```
Somme = 35
```

Et je peux faire passer une liste plus grande...

```
1 sommenombres (10,25,15,50)
```

Voici ce que j'obtiens sur la console :

```
Somme = 100
```

Olivier
SNOECK

Les fonctions +

Il est également possible de séparer les données d'une liste avec le caractère * mais cette fois-ci dans l'appel de la fonction.

```
1 def sommenombres(a , b , c):
2     somme=a + b + c
3     print("Somme = ", somme)
4
5 liste=(10, 15, 25)
6 sommenombres (*liste)
```

Voici ce que j'obtiens sur la console :

```
Somme = 50
```

Exercice 6

Objectifs :
En mathématiques, la suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent. Elle commence généralement par les termes 0 et 1 (parfois 1 et 1) et ses premiers termes sont 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, etc.

Écrivez un programme paramétrique qui fournit la série de Fibonacci pour obtenir les valeurs de n termes.

Olivier
SNOECK

Exercice 7

Sans le programmer, qu'affiche le programme suivant ?

```
1  def u(n):
2      if n==0:
3          return 2
4      else:
5          return 3*u(n-1)-2
6
7  print(u(0))
8  print(u(1))
9  print(u(2))
10
11 n=10
12 print(u(n))
```

Modifier le programme précédent pour qu'il calcule les termes de la suite (u_n) définie par l'expression : $u_n = \dfrac{2u_{n-1}^2 - 1}{u_{n-1}^2 + 2}$.

Réalisez le programme et afficher en particulier les termes u_{10} et u_{20}.

Les nombres

Python intègre 12 types numériques de données. Les voici :

- 1 type d'objet nul (None)
- 4 types numériques (int, float, complex, bool)
- 4 types de séquences (str, list, tuple, range)
- 1 type de mappage (dict)
- 2 types de set (set et frozenset)

Les types numériques

Les **nombres entiers** sont de type **int**.
On peut faire `print(type(15))` comme faire `print(type(-15))`, ce qui nous renvoie `<class "int">`.

Les types numériques

Les **nombres entiers** sont de type **int**.
On peut faire `print(type(15))` comme faire `print(type(-15))`, ce qui nous renvoie `<class "int">`.

Les **nombres réels** sont de type **float** (stocké sous la forme d'une mantisse et son exposant : $1.56 = 156x10^{-2}$).
On peut faire `print(type(1.56))`, comme faire `print(type(-1.56))`, ce qui nous renvoie `<class "float">`.

Les types numériques

Les **nombres entiers** sont de type **int**.
On peut faire `print(type(15))` comme faire `print(type(-15))`, ce qui nous renvoie `<class "int">`.

Les **nombres réels** sont de type **float** (stocké sous la forme d'une mantisse et son exposant : $1.56 = 156x10^{-2}$).
On peut faire `print(type(1.56))`, comme faire `print(type(-1.56))`, ce qui nous renvoie `<class "float">`.

Les **nombres complexes** sont de type **complex**.
On peut faire `print(type(1+6j))` qui nous renvoie `<class "complex">`.

Les types numériques

Les **nombres entiers** sont de type **int**.
On peut faire `print(type(15))` comme faire `print(type(-15))`, ce qui nous renvoie `<class "int">`.

Les **nombres réels** sont de type **float** (stocké sous la forme d'une mantisse et son exposant : $1.56 = 156x10^{-2}$).
On peut faire `print(type(1.56))`, comme faire `print(type(-1.56))`, ce qui nous renvoie `<class "float">`.

Les **nombres complexes** sont de type **complex**.
On peut faire `print(type(1+6j))` qui nous renvoie `<class "complex">`.

Un **chiffre booléen** est de type **bool**. Il existe **True** et **False**.
On peut faire `print(type(True))` qui nous renvoie `<class "bool">`.

Chaîne de caractères

Une **chaîne de caractères** est une collection ordonnée et modifiable de caractères. L'ordre des caractères est importants. Une chaîne de caractères est une séquence : il y a donc des indices...

```
1 for x in "ISTY":
2   print(x)
```

- Création d'une variable `"string"` : `a="Hello"` ou `a='Hello'`.
- Création sur plusieurs lignes : a="""Bonjour les loulous et vive Python""" ou '''Bonjour les loulous et vive Python'''.
- Affichage : `print("Hello")` ou `print('Hello')`.

Pour consulter la liste des méthodes disponibles, il faut afficher **dir(str)**.

La liste

Une **liste** (on tableau) est une séquence qui dispose d'une relation d'ordre : dans une liste, il peut y avoir des objets de types différents mais ils sont stockés dans un ordre précis. Une liste est un type de séquence particulier : elle est modifiable. Une liste est une séquence : il y a donc des indices...

```
1 list=["I","S","T","Y"]
2 for x in list:
3   print(x, end='')
```

Pour obtenir le résultat suivant : `ISTY`.

Pour consulter la liste des méthodes disponibles, il faut afficher **dir(list)**.
Les plus connus sont : `'append'`, `'clear'`, `'copy'`, `'count'`, `'extend'`, `'index'`, `'insert'`, `'pop'`, `'remove'`, `'reverse'`, `'sort'`

Exercice sur une liste

Réalisez une liste contenant les 7 journées de la semaine.

1. proposez une méthode n'affichant que les jours de la semaine (du lundi au vendredi inclus).
2. proposez une 2nde méthode n'affichant que les jours du week-end (...).
3. proposez une méthode qui affiche la liste entière mais, inversée...

   Supposons que le lundi soit le 2 :
4. proposez une méthode dans laquelle on passe une journée (par exemple, samedi) et qui nous renvoie la date de tous les samedis du mois (7, 14, 21 et 28) .

Olivier
SNOECK

Le tuple

Une **tuple** *(en français n-uplet)* est une séquence qui dispose d'une relation d'ordre : dans un tuple, il peut y avoir des objets de types différents mais ils sont stockés dans un ordre précis.
Un tuple est un type de séquence particulier : il est non modifiable. Le tuple définit une sorte de constante.
Un tuple est une séquence : il y a donc des indices...

```
1 tuple=("I","S","T","Y")
2 for x in tuple:
3   print(x, end='')
```

Pour obtenir le résultat suivant : `ISTY`.
Les tuples sont dits `"hashable"` : ceci signifie que nous pouvons les trier et s'en servir de clefs pour les dictionnaires (partie suivante).
Pour consulter la liste des méthodes disponibles, il faut afficher **dir(tuple)**.

Exercice sur un tuple

Réalisez un tuple contenant les 7 journées de la semaine.

1. proposez une méthode n'affichant que les jours de la semaine (du lundi au vendredi inclus).
2. proposez une 2nde méthode n'affichant que les jours du week-end (...).
3. proposez une méthode qui affiche le tuple entier mais, inversé...
   Supposons que le lundi soit le 2 :
4. proposez une méthode dans laquelle on passe une journée (par exemple, samedi) et qui nous renvoie la date de tous les samedis du mois (7, 14, 21 et 28) .
   Le tuple présente l'intérêt, par rapport à la liste, d'indiquer au programmeur qu'il est immuable.
5. Quel est l'intérêt du tuple par rapport à la liste ? Vous pouvez dire, non...
6. Peut-on ajouter une élément à la fin d'un tuple ?

Le dictionnaire

Une **dictionnaire** est une collection non ordonnée de relations entre clés et valeurs. Une clé de dictionnaire est hashable. Les clés doivent être immuables (inchangeables) mais les valeurs peuvent changer. Une clé joue le rôle d'indice d'une liste.

```
1 ecole={} # Création du dictionnaire
2 ecole["Nom"]="ISTY"
3 ecole["Rue"]="Rue Roger Salengro"
4 ecole["Ville"]="Mantes la ville"
5 print(ecole) # Affichage du dictionnaire complet
6 print (ecole.get("Ville")) # Affichage de la donnée du dictionnaire correspondant à la clé
↪  Ville
```

Pour obtenir le résultat suivant :

```
'Nom': 'ISTY', 'Rue': 'Rue Roger Salengro',
'Ville': 'Mantes la ville'
Mantes la ville.
```

Pour consulter la liste des méthodes disponibles, il faut afficher **dir(dict)**.

Exercice sur le dictionnaire

Soit le programme suivant :

```
1 mes_fruits = {"poire": 3, "pomme": 4, "orange": 2}
2 print ("Stock de fruits :")
3 for fruit, qte in mes_fruits.items():
4         print (f"{fruit} : {qte}")
```

1. Testez ce programme.
2. Proposez une méthode permettant à l'utilisateur de saisir de nouveaux fruits et ainsi que de nouvelles quantités.
3. Modifiez la précédente méthode afin de ne pas avoir de doublon de fruits (1 fois poire, 1 fois pomme,....).