Le langage Python, synthèse, partie 2

Les fonctions

Une fonction permet de contenir un ensemble d'instructions à exécuter.
Une fonction a un nom et permet d'effectuer une tâche.

La définition et l'appel d'une fonction

Une fonction peut être appelée régulièrement au sein de votre programme mais doit avoir été préalablement définie. La définition de la fonction indique ce qu'il faut faire lorsque la fonction est appelée afin d'être exécutée. La définition est aussi appelée l'implémentation, on la reconnaît grâce au mot clé « def ».
L'appel d'une fonction correspond à l'utilisation proprement dite. Une fonction peut être appelée au sein du programme principal ou même au sein d'une autre fonction.
Ci-dessous un exemple de fonction dont le rôle est d'afficher un message, qui est définie (une seule fois bien sûr), mais qui est utilisée plusieurs fois.

Exemple de mise en œuvre d'une fonction (la déclaration et deux appels de fonctions).

```
#On commence à définir la fonction
def msg_derivee_fct():
   print "la dérivée d'une grandeur par rapport au temps correspond à la vitesse de variation de cette grandeur."
#On n'est plus dans la définition de la fonction car la ligne n'est plus indentée
print "la dérivée de l'énergie par rapport au temps est la puissance."
print "pour rappel :"
msg_derivee_fct()                          #On appelle la fonction
print "la dérivée de la vitesse par rapport au temps est l'accélération."
print "Nous avions vu l'interprétation de la dérivée :"
msg_derivee_fct()                          #On appelle la fonction
```

Fonction sans paramètre et sans valeur de retour

L'exemple montré ci-dessus n'a pas de paramètre. En effet, tant au niveau de la définition de la fonction qu'au niveau de ses appels, il n'y a rien entre les parenthèses.
Les parenthèses permettent d'identifier une fonction mais aussi d'y placer des paramètres éventuels. Des parenthèses vides décrivent alors des fonctions sans paramètre.
Les paramètres sont également appelés des arguments, ils permettent de transmettre des informations à la fonction pour que celle-ci puisse effectuer sa tâche.
Par ailleurs l'exemple ci-dessus n'a pas non plus de valeur de retour, autrement dit l'appel de la fonction permet d'exécuter un certain nombre d'instructions mais la fonction ne renvoie pas un résultat qui pourrait être exploité.
Ci-dessous vous avez un autre exemple de mise en œuvre d'une fonction sans paramètre ni valeur de retour. La fonction se contente d'afficher la table de 5.

Deuxième exemple de mise en œuvre d'une fonction (la déclaration et deux appels de fonctions).

```
#On commence à définir la fonction
def affiche_table_5():
   facteur=1
   while facteur<=10:
      print facteur," fois 5 :",5*facteur
      facteur=facteur+1
print "voici la table de 5"
affiche_table_5()
print "et encore une fois :"
affiche_table_5()
```

Le nom de la fonction est « affiche_table_5 », elle est définie au début et est appelée ici deux fois dans le programme.

Fonction avec un paramètre mais sans valeur de retour

Une fonction avec un paramètre doit recevoir une valeur au moment de l'appel pour qu'elle puisse s'exécuter. Par exemple lorsqu'on veut utiliser une fonction pour générer un délai, il est commode de préciser le délai que l'on souhaite attendre lors de l'appel de fonction. On peut par exemple imaginer alors une fonction nommée attendre_en_sec() où il suffit d'indiquer le nombre de secondes à attendre entre les parenthèses. Ainsi,

pour générer un délai de 10 secondes on écrit : attendre_en_sec(10)
pour générer un délai de 3 secondes on écrit : attendre_en_sec(3)

On comprend que le nombre de secondes d'attente sera paramétrable, c'est pour ça que l'information transmise à la fonction au moment de l'appel est appelée paramètre.
Pour définir une fonction avec un paramètre, il faut placer une variable entre les parenthèses de la définition et utiliser la variable au sein de l'implémentation de la fonction. Cette variable peut avoir n'importe quel nom valable et sa valeur* représente l'information qui sera transmise à la fonction au moment de l'appel. La variable utilisé au sein de la définition de la fonction est une variable locale à la fonction (ainsi que toutes les autres variables déclarées de façon classique au sein de la fonction).
Ci-dessous nous allons définir une fonction dont le rôle est d'afficher la table d'un entier passé en paramètre :

Exemple de mise en œuvre d'une fonction avec un paramètre

```
#On commence à définir la fonction
def affiche_table(nb):
   facteur=1
   while facteur<=10:
      print facteur," fois ",nb," : ",nb*facteur
      facteur=facteur+1
print "voici la table de 5"
affiche_table(5)
print "et la table de 9 :"
affiche_table(9)
```

Pour comprendre l'exécution du programme, il faut commencer par se pencher sur les quatre lignes concernées, ce sont les quatre dernières :

print "voici la table de 5"	→	Du texte est affiché sur écran
affiche_table(5)	→	La fonction affiche_table() est appelée, la constante 5 est communiquée à la variable « nb » de la fonction
print "et la table de 9 :"	→	Du texte est affiché sur écran
affiche_table(9)	→	La fonction affiche_table() est appelée, la constante 9 est communiquée à la variable « nb » de la fonction

Une fonction peut recevoir une variable comme paramètre au moment de l'appel

Il est possible de transmettre une variable à une fonction au moment de l'appel.
Ci-dessous on trouve un exemple presque identique au précédent sauf que l'appel de la fonction fait apparaître une variable entre parenthèses (en non une constante).

Exemple d'appel d'une fonction en utilisant une variable comme paramètre	Commentaires

def affiche_table(nb):
  facteur=1
  while facteur<=10:
  print facteur," fois ",nb," : ",nb*facteur
  facteur=facteur+1
nb_choisi=input("Veuillez donner un nb entier SVP")
print "voici la table de ",nb_choisi
affiche_table(nb_choisi)
	Lorsque la dernière ligne est exécutée, c'est-à-dire : affiche_table(nb_choisi)
	
	la valeur de la variable *nb_choisi* sera communiquée à la variable *nb* de la définition de la fonction.

Il est possible d'utiliser le même nom de variable comme paramètre de l'appel et au sein de la définition de la fonction. Cependant la variable dans la fonction (variable locale) ne représente pas celle dans le programme principal (variable globale).

* Le passage de paramètre se fait par valeur ou par copie pour les types primitifs. Il est à noter qu'il existe également ce qu'on appelle le passage par référence.

Fonction sans valeur de retour mais avec plusieurs paramètres

Une fonction peut avoir plusieurs paramètres, ceux-ci seront placés entre les parenthèses et séparés par une virgule. Lors de l'appel, tous les paramètres devront être fournis à la fonction et dans le même ordre que le prévoit la définition de la fonction.
Imaginons une fonction nommée *affiche_n_lignes_de_table* qui reçoit deux paramètres :

- un premier paramètre qui est le nombre de lignes souhaitées, c'est aussi le nombre de facteurs ou le plus grand facteur. Dans l'exemple précédent ce nombre n'était pas paramétrable, il valait 10 (on affichait la table jusque 10).
- un deuxième paramètre qui est le nombre dont on souhaite afficher la table.

Exemple d'utilisation d'une fonction à deux paramètres	Commentaires

def affiche_n_lignes_de_table(facteur_max,nb):
  facteur=1
  while facteur<=facteur_max:
  print facteur," fois ",nb," : ",nb*facteur
  facteur=facteur+1
print "un petit test ..."
affiche_n_lignes_de_table(15,7)
nb_choisi=input("Veuillez donner un nb entier SVP ")
choix_facteur=input("Quel facteur max voulez-vous ? ")
print "voici la table de ",nb_choisi,"de 1 jusque ",choix_facteur
affiche_n_lignes_de_table(choix_facteur,nb_choisi)
	A l'exécution, la table de 7 (multipliée jusque 15) est affichée.
	Ensuite,
	Le programme lit deux nombres entiers introduits par l'utilisateur et appelle la fonction *affiche_n_lignes_de_table* en transmettant ces deux nombres entiers comme paramètres.
	ATTENTION, l'ordre doit être conservé.

Fonction avec valeur de retour

Normalement une « vraie » fonction renvoie un résultat et donc possède une valeur de retour. Les fonctions qui ne possèdent pas de valeur de retour sont parfois appelées procédures. L'avantage d'une fonction qui renvoie un résultat (donc avec une valeur de retour) et que ce résultat peut être exploité en dehors de la fonction.
On peut par exemple affecter la valeur de retour de la fonction à une variable.
Imaginons que l'on veuille calculer la température en Celcius d'une température donnée en Fahrenheit. On peut alors créer une fonction nommée *conversion_fahrenheit_celcius* qui reçoit comme paramètre une température en fahrenheit et qui calcule la température Celcius équivalente avant de fournir le résultat comme valeur de retour.
Pour la conversion on peut se baser sur la formule suivante : T(°C) = (T(°F) - 32)/1,8

Exemple d'utilisation d'une fonction avec une valeur de retour	Commentaires

def conversion_fahrenheit_celcius(temp_F):
  temp_C= (temp_F - 32)/9.0*5.0
  return temp_C
print "un petit test ..."
temp_test=conversion_fahrenheit_celcius(82)
print "82 Fahrenheit correspondent à ",temp_test," Celcius"
temp_F_choisie=input("Une température en Fahrenheit SVP ")
temp_C_convertie=conversion_fahrenheit_celcius(temp_F_choisie)
print temp_F_choisie," F --> ",temp_C_convertie," C"
	Le mot clé *return* termine la fonction et donne le résultat
	La valeur de retour est affectée à *temp_test*
	Le résultat est affecté à *temp_C_convertie*

Fonction avec plusieurs valeurs de retour

En python, contrairement à beaucoup de langage, il est possible d'avoir plusieurs valeurs de retour pour une fonction.
Cette fonctionnalité un peu particulière provient de la possibilité de faire des affectations multiples. En python on peut par exemple écrire : a,b = 8, 20
Ça signifie que *a* est affecté de la valeur 8 et *b* de la valeur 20
On peut alors imaginer une fonction nommée *min_max* qui a plusieurs valeurs de retour et dont l'instruction *return* serait par exemple : return min, max
L'appel de la fonction sera alors liée à une affectation multiple comme par exemple :
nb_min, nb_max = *min_max(nb1, nb2, nb3, nb4, nb5, nb6)*