Les Fonctions

En Python, une **fonction** est un groupe d'instructions liées et structurées dont le but d'effectuer une tâche spécifique lors qu'elle est appelée.

Une **fonction** est utilisée pour appeler un seul code à plusieurs endroits dans un programme. On l'appelle aussi **méthode** ou **procédure**.

Donc une fonction est un sous-programme réalise une tache particulière, elle peut être étudiée/ testée séparément et réutilisée dans d'autres programmes.

Python fournit de nombreuses fonctions intégrées comme **print()**, **input()**, **compile()**, **exec()**, etc. mais il donne également la liberté de créer vos propres fonctions. Les fonctions aident à subdiviser le programme en morceaux plus petits afin de pratiquer la programmation modulaire. Au fur et à mesure que le programme grandit, les fonctions le rendent plus organisé et plus facile à gérer. En plus, cela évite les répétitions et rend le code réutilisable.

Les fonctions en Python sont :

Prédéfinies : prédéfinies sous Python et toujours accessibles.

Définies par l'utilisateur : définies par le programmeur et personnalisables.

Points essentiels :

- Un programme écrit sans fonction devient difficile à comprendre dès qu'il dépasse une centaine de lignes
- Les fonctions permettent de scinder le programme principal en plusieurs parties Une fonction peut elle-même, être découpée en plusieurs autres fonctions.
- Les fonctions permettent le partage de tâches : dans un projet important où travaille plusieurs développeurs, chacun est responsable d'une ou de plusieurs fonctions, qui sont ensuite utilisées par les autres programmeurs et mise en commun dans le programme principal.

Fonctions intégrées en python (prédéfinies)

L'interpréteur Python a diverses fonctions prédéfinies qui sont facilement utilisables. Nous n'avons pas besoin de définir ces fonctions pour les utiliser, nous pouvons les appeler directement.

Ces fonctions sont appelées fonctions **intégrées**.

- **Python input()** : Cette fonction python intégrée lit et renvoie généralement une ligne de chaîne.
- **Python int()** : Cette fonction python intégrée renvoie un entier à partir d'un nombre ou d'une chaîne.
- **print()** : Il est utilisé pour imprimer un objet donné
- **len()** : Cette fonction renvoie la longueur d'un objet.
- **list()** : Cette fonction est utilisée pour créer une liste Python **max()** : Cette fonction renvoie le plus grand élément.
- **min()** : Cette fonction renvoie le plus petit élément
- **range()** : Cette fonction renvoie la séquence d'entiers entre start et stop
- **set()** : Cette fonction renvoie un ensemble
- **str()** : Cette fonction renvoie une représentation informelle d'un objet. **type()** : Cette fonction renvoie le type d'objet

Les fonctions définies par l'utilisateur

Syntaxe :

La syntaxe pour définir une fonction en python est la suivante :

```
def nom_fonction (parametres) :
    bolc d_istructions
    return[expression]
```

Définir une fonction

La définition d'une fonction commence par une ligne d'introduction qui doit contenir : le mot clé **def** , suivi du nom de la fonction, suivi de tous les **paramètres** de la fonction entre parenthèses. Après cette ligne d'introduction, les instructions qui définissent la fonction doivent être dans un bloc **indenté** qui constitue le corps de la fonction (si une instruction n'est pas indentée : elle sera considérée comme ne faisant pas partie de la définition de la fonction et entraînera la fin de la définition du corps de la fonction).

Valeur retournée par la fonction

Le mot clé **return** permet d'indiquer la valeur retournée par la fonction.

La valeur retournée indiquée après le mot clé return est le résultat du travail de la fonction. Tout ce qui a pu être calculé localement dans la fonction est perdu s'il n'est pas retourné.

Si on ne met pas de return dans le corps de la fonction, ou bien un return sans spécifier de valeur, alors Python retourne automatiquement la valeur **None**.

Remarque **:** ne pas confondre la fonction **print()** et le mot-clé **return**. Ce n'est pas parce qu'une fonction affiche quelque chose à l'écran qu'elle retourne une valeur. Et le mot clé return ne provoque pas d'affichage à l'écran.

Appeler une fonction

Pour appeler une fonction il suffit d'indiquer le nom de cette fonction, suivi de ses arguments entre parenthèses (Les arguments remplacent les paramètres) pour que la fonction puisse effectuer sa tâche et retourner le résultat souhaité, les valeurs des arguments vont être affectées aux paramètres et ainsi être utilisées dans les

instructions du corps de la fonction lors de leur exécution. Ces arguments passés lors de l'appel, sont donc les valeurs particulières sur lesquelles on demande à la fonction de travailler.

Exemples :

Exemple 1 : fonction sans paramètres, ne retournant pas de valeur (fonction simple)

```
def ma_fonction () :
    print ('bonjour')

ma_fonction()
```

Appel fonction

Affichage après exécution : bonjour

Exemple 2 : fonction avec paramètres, retournant une valeur

```
def min (a,b) :
    if (a < b):
        return a
    else:
        return b

print('le plus ptit nombre est:' , min(26,79))
```

Appel fonction

Affichage après exécution : Le plus petit nombre est : 26

Exemple 3 : fonction avec paramètres, sans retour de valeur

```
def min (a,b) :
    if (a < b):
        print('le plus ptit nombre est:' , a)
    else:
        print('le plus ptit nombre est:' , b)

min(129,15)
```

Appel fonction

Affichage après exécution : Le plus petit nombre est : 15

Exemple 4 : fonction sans paramètres, retournant une valeur

```
def p_carre () :
    pp= 3.14 ** 2
    return pp

s = p_carre() * 2
print (s)
```

Appel fonction

Affichage après exécution : 19.7192

Arguments positionnels et arguments par mot-clé

Arguments positionnels

Lorsqu'on définit une fonction def fct(x, y): les arguments x et y sont appelés arguments positionnels. Il est strictement obligatoire de les préciser lors de l'appel de la fonction. De plus, il est nécessaire de respecter le même ordre lors de l'appel que dans la définition de la fonction.

Exemple :

1)
```
def sust (a,b) :
    return a-b

print('a -b = ' , sust(18,4))    #  a=18 , b=4  -->  a - b = 18 - 4
```

Affichage après exécution : a – b = 14

2)
```
def sust (a,b) :
    return a-b

print('a -b = ' , sust(4,18))    #  a=4 , b=18  -->  a - b = 4 - 18
```

Affichage après exécution : a – b = -14

3)
```
def sust (a,b) :
    return a-b

print('a -b = ' , sust(18))
 ou
print('a -b = ' , sust(4))
```

Affichage après exécution : erreur

Arguments par mot-clé

Un argument défini avec une syntaxe **def fct(arg=val):** est appelé argument par motclé (en anglais, keyword argument). Le passage d'un tel argument lors de l'appel de la fonction est facultatif.

```
def  asri (x=0,y=0, z=0) :
    return x,y,z

apl1 = asri()
apl2 = asri(98)
apl3 = asri(41, 6)
apl4 = asri(56, 71, 194)
apl5 = asri(z=38)
apl6 = asri(y=23, z=44, x=11)

print("le resultal de l'appel 1 de la fonction asri est: " , apl1)    # l'affichage est:  0,0,0
print("le resultal de l'appel 2 de la fonction asri est: " , apl2)    # l'affichage est:  98,0,0
print("le resultal de l'appel 3 de la fonction asri est: " , apl3)    # l'affichage est:  41,6,0
print("le resultal de l'appel 4 de la fonction asri est: " , apl4)    # l'affichage est:  56,71,194
print("le resultal de l'appel 5 de la fonction asri est: " , apl5)    # l'affichage est:  0,0,38
print("le resultal de l'appel 6 de la fonction asri est: " , apl6)    # l'affichage est:  11,23,44
```

Remarque :

Python permet même de rentrer les arguments par mot-clé dans un ordre arbitraire

Point essentiel !!! :

Que se passe-t-il lorsque nous avons un mélange d'arguments positionnels et arguments par mot-clé?

```
def fct(a, b, x=0, y=0, z=0)) :
    return a,b,x,y,z

appel_1 = fct()
appel_2 = fct(740)
appel_3 = fct(23,61)
appel_4 = fct(42,945,z=37)
appel_5 = fct(z=8, x=402, y=77)

print("le resultat de l'appel 1 de la fonction fct est: " , appel_1)  # erreur , passage des argument positionnel (a et b), obligatoire!!!
print("le resultat de l'appel 2 de la fonction fct est: " , appel_2)  # erreur , passage des argument positionnel (a et b), obligatoire!!!
print("le resultat de l'appel 3 de la fonction fct est: " , appel_3)  # 23,61,0,0,0
print("le resultat de l'appel 4 de la fonction fct est: " , appel_4)  # 42,945,0,0,37
print("le resultat de l'appel 5 de la fonction fct est: " , appel_5)  # erreur , passage des argument positionnel (a et b), obligatoire!!!
```

Remarque : Les arguments positionnels doivent toujours être placés **avant** les arguments par mot-clé.

Variables locales et variables globales

Lorsqu'on manipule des fonctions, il est essentiel de bien comprendre comment se comportent les variables. Une variable est dite **locale** lorsqu'elle est créée dans une fonction. Elle n'existera et ne sera visible que lors de l'exécution de la fonction.

Une variable est dite **globale** lorsqu'elle est créée dans le programme principal. Elle sera visible partout dans le programme.

Le mot clé global

Le mot clé globale est utilisé pour déclarer qu'une variable a l'intérieur d'une fonction est une variable globale, les règles de base pour le mot-clé **global** en Python sont les suivantes :

- Lorsque nous créons une variable à l'intérieur d'une fonction, elle est **locale** par défaut.
- Nous utilisons un mot-clé **global** pour lire et écrire une variable globale à l'intérieur d'une fonction.
- Lorsque nous définissons une variable en dehors d'une fonction, elle est globale par défaut. *Vous n'avez pas besoin d'utiliser le mot-clé global.*
- L'utilisation d'un mot-clé global en dehors d'une fonction n'a aucun effet.

Exemple : que retournent les programmes suivants ?

```
x=10
def my_function():
        x=5
        print(x)
        return
#Appel de la fct
my_function()
print (x)
```

```
x=10
def my_function():
        global x
        x=5
        print(x)
#Appel de la fct
my_function()
print (x)
```

```
global x
x=10
def my_function():
        x=5
        print(x)
#Appel de la fct
my_function()
print (x)
```

Les modules

Une caractéristique clé de Python est son système de "modules", qui permet aux développeurs de créer des programmes bien structurés et modulaires.

Mais qu'est-ce qu'un module exactement et comment est-il utilisé en Python ?

Qu'est-ce qu'un module en python

En Python, un module est un fichier contenant des définitions de fonctions, de classes et de variables, ainsi que des instructions exécutables.

En d'autres termes, c'est une manière d'organiser le code de manière logique et cohérente.

Le nom du module est dérivé du nom du fichier (sans l'extension **.py**).

Par exemple, un fichier nommé ***calcul.py*** correspondrait au module ***calcul***.

Un des grands avantages des modules en Python est qu'ils permettent le partage et la réutilisation du code.

En effet, une fois qu'un module a été importé, les fonctions, classes et variables qu'il définit peuvent être utilisées dans le code comme s'ils avaient été définis localement.

Exemple :

```
def dire_bonjour():
    return "Bonjour !"

def dire_bonsoir():
    return "Bonsoir !"
```

Dans cet exemple, salutations.py est un module qui définit deux fonctions, dire_bonjour et dire_bonsoir.

Comment utiliser les modules

L'utilisation de modules en Python se fait grâce à l'instruction **import**.

Cette instruction permet d'importer un module et d'utiliser les fonctions, classes et variables qu'il définit.

Exemple :

Voici comment on pourrait utiliser le module salutations défini précédemment :

```
import salutations

# utilisation des fonctions définies dans le module

print(salutations.dire_bonjour())  # affiche "Bonjour !"
print(salutations.dire_bonsoir())  # affiche "Bonsoir !"
```

Il est également possible d'importer uniquement certaines fonctions ou classes d'un module, en utilisant la syntaxe : **from**……**import**……

Exemple :

```
# importation de la fonction dire_bonjour du module salutations
from salutations import dire_bonjour

# utilisation de la fonction
print(dire_bonjour())  # affiche "Bonjour !"
```

Il est à noter que Python dispose d'une vaste bibliothèque standard, qui comprend de nombreux modules utiles pour une variété de tâches, allant du traitement de fichiers à la programmation réseau, en passant par l'analyse de données. L'utilisation de ces modules peut considérablement accélérer le processus de développement.

Obtenir de l'aide

Pour obtenir de l'aide sur l'utilisation d'une fonction, il suffit d'appliquer la méthode suivante.

- Taper dans la console **import** suivi du nom du module.
- Taper **dir(nom_module)** afin d'obtenir la liste des fonctions du module.
- Taper la commande **help** suivi du nom du module.nom de la fonction entre parenthèses, afin d'obtenir la documentation associée à la fonction.

Exemples :

On importe le module math, puis on obtient la liste des fonctions de ce module.

```
>>>> import math        # importer le module math
>>>> dir (math)         # afficher les fonctions du module math
```

Pour obtenir l'aide de la fonction sqrt (permet de calculer la racine carré d'un nombre) qui fait partie du module math, on tape :

```
>>>> help (math.sqrt)
```

Pour voir les modules disponibles, saisissez:

```
>>>> help ('modules')
```

On peut aussi trouver de l'aide sur les modules et les bibliothèques sur le site officiel de Python.

Exercices :

Exercice1 :

Ecrire un programme Python qui définit et appelle une fonction *Bonjour* qui affiche le message : *Bonjour!*.

Exercice2 :

Ecrire un programme Python qui définit une fonction *Somme* qui prend deux nombres en arguments et renvoie leur somme.

Exercice3 :

Ecrire un programme Python qui définit une fonction *Positif* qui prend un entier comme argument, et revoie ***True*** s'il est positif et ***False*** sinon.

Exercice4 :

Créer une fonction qui prend un entier **N** comme argument et d'afficher s'il est premier ou non. Un nombre est dit premier s'il est divisible uniquement par 1 et par lui-même.