Langage Python

Les bases du langage 5 : Les fonctions

Formation POEC Cybersécurité
Théo Hubert

Syntaxe

- *Fonctions*

Lorsqu'une tâche doit être réalisée plusieurs fois par un programme avec seulement des paramètres différents, on peut l'isoler au sein d'une fonction.
En effet une **fonction** désigne en **programmation** un « sous-**programme** » permettant d'effectuer des opérations répétitives.

Par exemple, nous avons déjà utilisé la fonction `print()` définie par le développeur à l'origine à Python.

```
print("Bonjour")
t = range(4)
mot = str(200)
nbr = int(mot)
```

Il est particulièrement intéressant d'utiliser des fonctions lors des cas suivants :

- On répète plusieurs fois une suite d'instructions dans notre code avec des paramètres qui varie
- On cherche à résoudre un problème particulièrement complexe
- On cherche à gagner en lisibilité

Syntaxe

- *Fonctions*

Syntaxe :

La syntaxe Python pour la définition d'une fonction est la suivante :

=> On utilise le mot-clé **`def`** pour indiquer qu'il s'agit d'une fonction
=> Si on souhaite que la fonction renvoie quelque chose, il faut utiliser le mot-clé **`return.`**
=> On utilise **deux-points** comme les boucles for et while ainsi que les tests if, un bloc d'instructions est donc attendu. De même que pour les boucles et les tests, **l'indentation de ce bloc d'instructions** (qu'on appelle le corps de la fonction) est **obligatoire**.
=> Vous pouvez choisir n'importe quel nom pour la fonction que vous créez, à l'exception des mots-clés réservés du langage, et à la condition de n'utiliser aucun caractère spécial ou accentué (le caractère souligné « _ » est permis).
=> Par convention on utilise des **minuscules** (les majuscules sont réservés aux classes que l'on découvrira dans un prochain cours).

```
def nom_fonction(parametre1, parametre2, ...):
  bloc d'instruction
  return resultat
```

```
def nom_fonction2():
  bloc d'instruction
```

Syntaxe

- *Fonctions*

Exemples :

```
def double_nbr(x):
  result = x * 2
  return result

print(double_nbr(2))
```

```
4
```

Ici la fonction **double_nbr()** prend en entrée un argument x puis va retourner la valeur de cet argument multiplié par 2.

```
def affiche_alert(message, code_erreur):
  print("Erreur :", str(code_erreur), " message : ", message)

affiche_alert("erreur graphique", 28452)
```

```
Erreur : 28452  message :  erreur graphique
```

Ici la fonction **affiche_alert()** prend en entrée 2 arguments message et code_erreur et va afficher le code de l'erreur ainsi que le message.

Syntaxe

- *Fonctions*

Exemples :

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

print("bonjour")
compteur3()
compteur3()
```

```
bonjour
0
1
2
0
1
2
```

Ici la fonction **compteur3()** ne prend aucun arguments en entrée et affiche la valeur d'une variable i tant que celle-ci est strictement inférieur à trois puis elle augmente la valeur de cette variable de 1.

Syntaxe

- *Fonctions*

Exemples :

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

def double_compteur3():
    compteur3()
    compteur3()

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

```
bonjour
0
1
2
0
1
2
```

Ici la fonction **double_compteur3()** ne prend aucun arguments en entrée et fait appel à 2 reprises à la fonction **compteur3()**.

Syntaxe

- *Fonctions*

Particularités de Python :

Une première particularité des fonctions en Python est que vous n'êtes pas obligé de préciser le type des arguments que vous lui passez, dès lors que les opérations que vous effectuez avec ces arguments sont valides. En effet Python est un langage au « typage dynamique ».

Une deuxième particularités de Python est que les fonctions sont capables de renvoyer plusieurs objets à la fois.

```
def multi(a):
    b = a+1
    c = a+2
    return b, c

print(multi(2))
print(multi(2)[0])
print(multi(2)[1])
```

```
(3, 4)
3
4
```

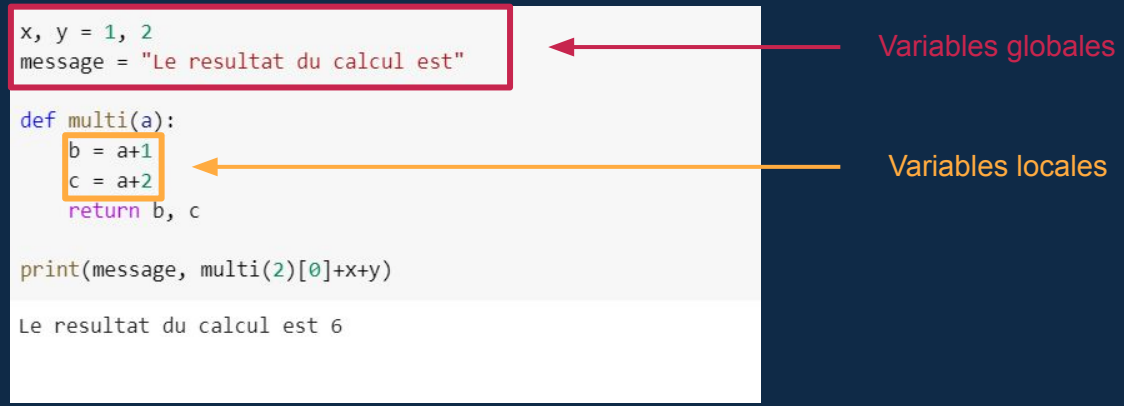
Syntaxe

- *Fonctions*

Variables locales et globales :

Lorsqu'on manipule des fonctions, il est essentiel de bien comprendre comment se comporte 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 ladite fonction.
- Une variable est dite **globale** lorsqu'elle est créée dans le programme principal. Elle sera visible partout dans le programme.



Syntaxe

- *Fonctions*

Variables par défaut des 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, titre ="Monsieur"):
    print("Veuillez agréer,", titre, nom, ", mes salutations distinguées.")

politesse("Dupont")
```

```
Veuillez agréer, Monsieur Dupont , mes salutations distinguées.
```

```
politesse('Durand', 'Mademoiselle')
```

```
Veuillez agréer, Mademoiselle Durand , mes salutations distinguées.
```

Syntaxe

- *Fonctions*

Emplacement des fonctions dans un script :

```
import numpy as np

def carre(n):
    return n**2

def aire_cercle(r):
    return np.pi * carre(r)

r = float(input("Entrez la valeur du rayon : "))
print("L'aire de ce cercle est", aire_cercle(r))
```

```
Entrez la valeur du rayon : 5
L'aire de ce cercle est 78.53981633974483
```

Ici on remarque que les deux parties du programme ont été disposées dans un certain ordre :

- d'abord la définition des fonctions,
- et ensuite le corps principal du programme.

Cette disposition est nécessaire, car **l'interpréteur** exécute les lignes d'instructions du programme **l'une après l'autre**, dans l'ordre où elles apparaissent dans le code source. Dans le script, **la définition des fonctions** doit donc **précéder leur utilisation**.

Syntaxe

Travaux Pratique

Références

https://courspython.com/fonctions.html

https://python.developpez.com/cours/apprendre-python3/?page=page_11

Fin

La suite :
POO 5 (programmation orientée objet)