Chap. 1 Introduction à la programmation Python Thème 1

Fonctions

Sommaire du chapitre 1

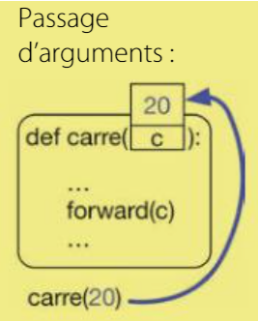
Une **fonction** permet d'encapsuler un bloc d'instructions et de lui donner un nom. On peut ensuite exécuter ce bloc en utilisant ce nom. On dit que l'on **appelle** cette fonction.
Une fonction peut avoir un ou plusieurs **paramètres**, qui permettent de transmettre des valeurs au bloc d'instructions. A l'intérieur de ce bloc, les paramètres sont traités comme des variables.
La fonction suivante dessine un carré dont la longueur du côté est passée en paramètre :

```
def carre(c) :      # c est le paramètre
    """ Dessine un carré de côté c """
    for cote in range(4) :
        forward(c)  # on avance de la valeur de c
        left(90)
```

I Appel de fonction

Pour appeler une fonction, on indique son nom suivi des **arguments** entre parenthèses. Les arguments sont des expressions et doivent correspondre aux paramètres.

```
carre(20)
carre(random()*100)
```



II Valeur de retour

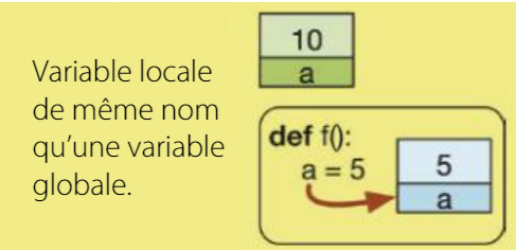
Une fonction peut **retourner une valeur**. L'instruction `return` définit la valeur retournée par la fonction, et *interrompt immédiatement son exécution* pour retourner la valeur au point d'appel.
Voici une fonction qui calcule et renvoie la plus petite de deux valeurs :

```
1 def min(a, b) :
2     """ Retourne le minimum de a et de b """
3     if a < b :
4         return a
5     # Pas besoin de 'else' : si a > b, la fonction a déjà cessé de
      s'exécuter à ce point
6     return b
7 print(min(10,20))
```

III Variables locales et variables globales

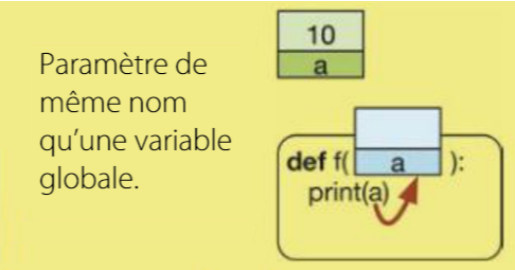
Lorsqu'une fonction utilise des variables, celles-ci sont normalement propres à la fonction et ne sont pas accessibles à l'extérieur de celle-ci. On dit qu'il s'agit de **variables locales**, par opposition aux **variables globales** qui sont utilisées dans le programme principal.
Dans l'exemple ci-dessous, la variable `a` de la ligne 1 n'est pas la même que celle de la ligne 3, bien qu'elle ait le même nom, c'est pourquoi le programme affiche 10.

```
1 a = 10        # variable globale
2 def f():
3     a = 5     # variable locale
4 f()
5 print(a)      # référence la variable globale : affiche 10
```



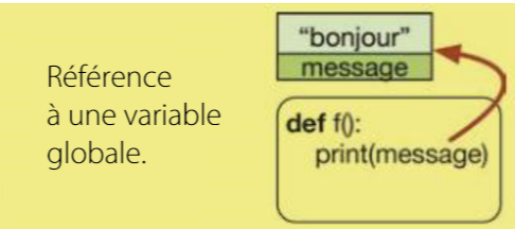
Les paramètres sont également des variables locales de la fonction :

```
1 a = 10        # variable globale
2 def f(a):     # paramètre = variable locale
3     print(a) # affiche la valeur du paramètre
4 f(5)          # affiche 5
5 print(a)      # référence la variable globale : affiche 10
```



Cependant, si l'on *référence* dans une fonction une variable qui n'a pas été affectée dans la fonction, Python regarde s'il existe une variable globale de ce nom. Si oui, il utilise sa valeur au lieu de provoquer une erreur de type « variable indéfinie ». Dans l'exemple suivant, la variable `message` référencée dans la fonction `f`, ligne 3, est la variable globale `message` de la ligne 1 :

```
1 message = "bonjour"    # variable globale
2 def f() :
3     print(message)     # référence à la variable globale !
4 f()                    # affiche "bonjour"
```

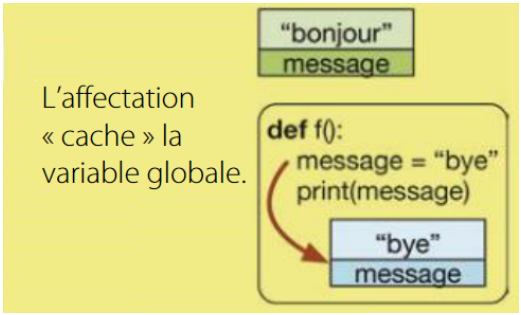


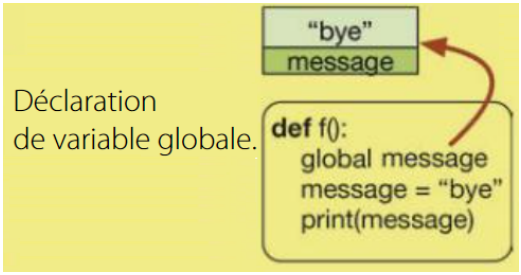
Cette règle du langage Python est une source potentielle d'erreurs, car il suffit d'ajouter une affectation pour créer une variable locale et « cacher » la variable globale de même nom.

```
1 message = "bonjour"    # variable globale
2 def f() :
3     message = "bye"    # affectation qui crée une variable locale
4     print(message)     # référence à la variable locale
5 f()                    # affiche "bye"
6 print(message)         # affiche "bonjour"
```

Le langage Python permet cependant d'affecter, à l'intérieur d'une fonction, une variable globale. Pour cela, il faut **déclarer** la variable comme étant globale au début de la fonction, avec l'instruction `global` :

```
1 message = "bonjour" # variable globale
2 def f() :
3     global message  # déclarer la référence à la variable globale
4     message = "bye" # affectation qui crée une variable globale
5     print(message)  # référence à la variable globale
6 f()                 # affiche "bye"
7 print(message)      # affiche "bye" (la valeur du message a changé)
```





Modifier ainsi une variable globale depuis une fonction s'appelle un **un effet de bord** et est considéré comme une mauvaise pratique, même si elle est parfois indispensable. Référencer une variable globale depuis une fonction est toléré lorsqu'il s'agit d'un élément de configuration du programme.
Par exemple, une variable booléenne globale `trace` peut servir à contrôler l'affichage de messages lors de l'exécution :

```
1 trace = True
2 def f() :
3     ...
4     if trace :
5         print("On a appelé f")
```

Recapitulatif

FONCTION

- **Définition de fonction**
  On définit une fonction nommée `fonction1` qui prend en entrée des `arguments`, exécute toutes les instructions indentées puis renvoie `valeurs_renvoyées` comme suit :

```
def fonction1(arguments) :
    bloc_instructions
    return valeurs_renvoyées
```

- **Appel de fonction**
  On appelle une fonction en exécutant :

```
fonction1(arguments)
```