INTRODUCTION A LA PROGRAMMATION PYTHON

#5. Fonction

> **Objectifs:**
>
> - Introduire le concept de programmation fonctionnelle
> - Programmer en utilisant des fonctions

Lorsqu'on veut utiliser plusieurs fois une fonctionnalité, plutôt que de la coder à chaque fois, il est plus intéressant de:

- **définir** une **fonction** (une seule fois)
- puis d'**appeler** cette fonction (à chaque fois qu'on désire la fonctionnalité)

Voici un exemple en python:

```
# définition de la fonction
'''
fonction qui calcule le double d'un nombre
et affiche un message informatif
    - paramètre: n un nombre (int ou float)
    - pas de retour
'''
def double( n ):
    print("le double de", n, "est: ")
    print(2*n)

# corps du programme avec deux appels de fonction

# appel de fonction 1
double( 3 )

# appel de fonction 2
double( 7 )
```

```
le double de 3 est:
6
le double de 7 est:
14
```

La fonction définie précédemment n'est toutefois pas optimale.

En effet, l'utilisateur ne voudra pas forcément faire un `print`, ou pas forcément sous cette forme, ou voudra combiner le résultat à d'autres opérations...

On préférera faire un **retour de fonction**, c'est à dire que la fonction va retourner son résultat:

```
# définition de la fonction
'''
fonction qui calcule le double d'un nombre
et le renvoie
    - paramètre: n un nombre (int ou float)
    - retour: double de n (int ou float)
'''
def double( n ):
    return 2*n   # mot clef return  renvoit le résultat de la fonction

# corps du programme

# appel de fonction 1
double( 3 )     # le résultat est bien retourné mais est non utilisé donc perdu

# appel de fonction 2
```

```
a = double( 3 ) # le retour de la fonction est stocké dans la variable a
print( a )      # puis affiché

# appel de fonction 3
print( double(3))   # le retour de la fonction est imprimé

# affectation du résultat d'une comparaison avec 3 appels de fonctions
isTrue = ( double(3)+double(4) == double (3+4) )
print( isTrue )
```

```
6
6
True
```

On peut implémenter une fonctionnalité quelconque par une fonction qui:

- a des paramètres (des entrées)
- réalise la fonctionnalité avec les données passées en paramètres
- et retourne le résultat de la fonction (la sortie)

```
def maFonction ( e1 , e2 , ... , en): # paramètres entre parenthèse séparés par des virgules
    ...
    ...
    return s # le retour de la fonction
```

Mission 5.1.

> On reprend la mission 2.3 mais cette fois ci avec une fonction.
>
> Programmer une fonction qui renvoie un message de bienvenue : `messageBienvenue( nom , prenom, annee ) -> message`
>
> > paramètres: nom (str), prénom (str) , année de naissance (int)
> > retour: message (str) de bienvenue mise en forme et contenant le nom, le prénom ainsi que l'âge (fin d'année)
>
> Puis l'utiliser en demandant les informations nécessaires pour trois personnes.

Mission 5.2.

> Programmer une fonction qui retourne le carré d'un nombre : `carre( n ) -> n_square`
>
> > paramètre: n (int ou float)
> > retour: n_square (int ou float) le carré de n
>
> Puis l'utiliser pour:
>
> > afficher les carrés des 20 premiers entiers
> > afficher les entiers dont le carré est inférieur à un nombre à demander à l'utilisateur

Mission 5.3.

> On continue sur la mission 4.2.
>
> Programmer une fonction qui retourne le plus grand diviseur d'un nombre: `plusGrandDiviseur(n) -> d`
>
> > paramètre: n (int)
> > retour: d (int) le plus grand diviseur de n (différent de n)
>
> Puis l'utiliser pour:
>
> > afficher le plus grand diviseur d'un nombre demandé à l'utilisateur
> > afficher les dix premiers nombres premiers

Vidéo

>