CHAPITRE

9 Les fonctions Python

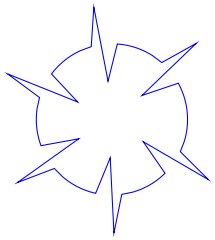


Table des matières

En classe de première, on utilise souvent des ***fonctions*** pour ***factoriser*** du code. L'appel de la fonction permet alors d'utiliser ce code.

1 Le principe sur un exemple !

Le code suivant :

```
1 L = [8, 5 , 6 , -1, 45, 0, 20]
2 som = 0
3 for i in range(len(L)):
4     som = som + L[i]
5 print(som)
```

permet le calcul de la somme des éléments d'une liste. Mais finalement ce code ne change pas si on change la liste `L` : il servira toujours à calculer la somme des éléments de la liste proposée en préambule....

On a donc :

- En ***entrée***, une liste Python
- En ***sortie***, un nombre(entier ou décimal)

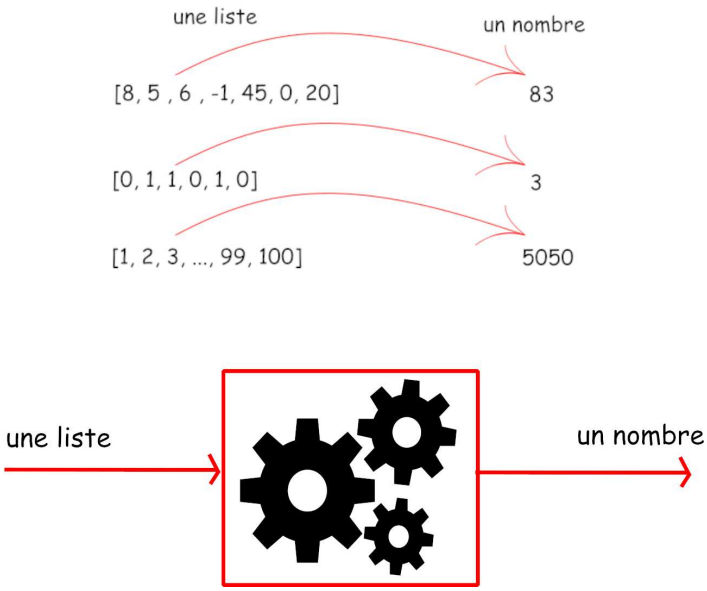


FIGURE 1 – On fait le parallèle avec les fonctions mathématiques

La définition des fonctions Python reprend le concept mathématique des fonctions qui transforme un nombre en un autre. Une fonction f telle que :

$$f(Liste) = Somme$$

La syntaxe Python qui crée une fonction est donnée par le code ci-contre :

```python
#-- VERSION CLASSIQUE --#

L = [8, 5 , 6 , -1, 45, 0, 20]
som = 0
for i in range(len(L)):
    som = som + L[i]
print(som)
```

```python
#-- VERSION FONCTION --#
def somme_de_liste(une_liste):
    n = len(une_liste)
    som = 0
    for i in range(n):
        som = som + une_liste[i]
    return som
```

Cette fonction :

- est une fonction `Python` car elle est déclarée avec la particule **`def`** ;
- porte le nom ***implicite*** de `somme_de_liste` (en snake_case) ;
- prend en paramètre un argument `une_liste` qui doit contenir une liste ;
- retourne une valeur égale à la somme des éléments de cette liste.

Depuis la version Python 3.5, on peut annoter les types de paramètres de sorties et d'entrées. Cela donnerait par exemple :

```python
def somme_de_liste(une_liste:list)->float:
    n = len(une_liste)
    som = 0
    for i in range(n):
        som = som + une_liste[i]
    return som
```

2 Utiliser une fonction

Une fonction peut avoir aucun, un ou plusieurs ***paramètres*** en entrée.

Voici des exemples :

```
def politesse(): #pas de paramètres
    return "Merci beaucoup"
def est_nombre_pair(nombre): #un paramètre
    """Vérifie si un nombre est pair"""
    return nombre%2 == 0:
def calculer_aire_rectangle(longueur, largeur): #deux paramètres
    """Calcule l'aire d'un rectangle"""
    aire = longueur * largeur
    return aire
```

Exercice n°1

1. Recopiez ces fonctions dans un script python.
2. Exécutez ce script. Que se passe t-il ?

Les ***fonctions*** devraient retourner un résultat en général. Une ***procédure***, elle, exécute des actions mais ne retourne pas de valeur directement (ou retourne "void" dans certains langages).

Une fois la fonction déclarée, il faut être en mesure de l'utiliser.

Lorsque vous exécutez un script contenant une fonction, il ne se passe rien en apparence.

En effet, en exécutant votre code, vous mettez alors à la disposition de l'utilisateur une nouvelle fonction, un nouvel outil, que vous pouvez utiliser soit en console soit dans le programme principal.

Lorsque une fonction retourne une valeur, elle ne l'affiche pas dans la console !

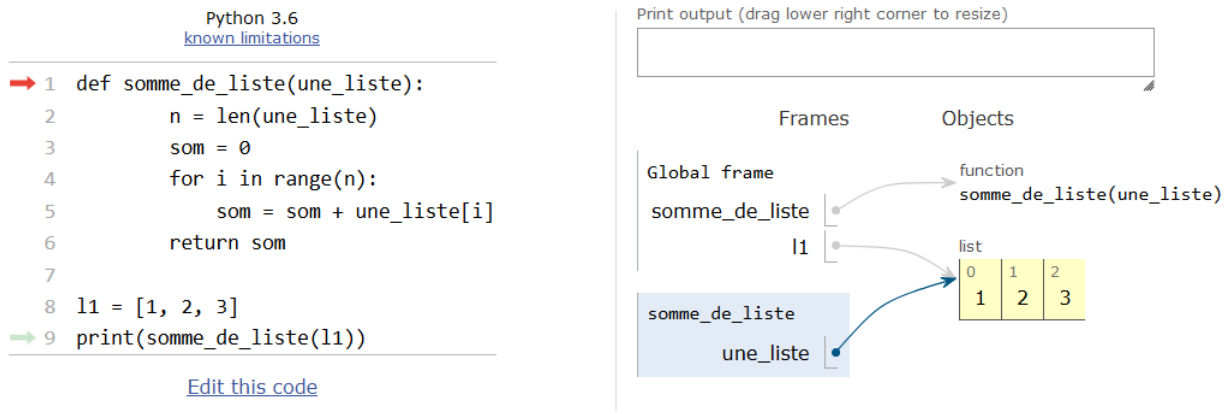
Donc utilisez des `print` pour visualiser le retour...

Exercice n°2

1. Recopier la fonction `somme_de_listes` puis exécutez votre script.
2. Calculer avec cette fonction la somme des listes suivantes : `L1 = [1, 3, 10,]`, `L2 = [i for i in range(100)], [5, 10, 1, 2]`

Il faut distinguer le paramètre des fonctions des variables ***globales*** du programme.

Lorsqu'on appelle la fonction sur la liste `L1`, la valeur de cette liste est transmise au paramètre `ma_liste` qui devient une variable ***locale*** à la fonction. Le site Python Tutor permet de visualiser ce transfert :



Voici une fonction avec un ***paramètre optionnel*** : si celui-ci n'est pas définie à l'appel de la fonction, sa valeur par défaut est appliquée :

```python
def calculer_prix_ttc(prix_ht:float, taux_tva = 0.20:float)->float:

    prix_ttc = prix_ht * (1 + taux_tva)
    return prix_ttc
```

Et ci-dessous, des exemples d'utilisation de cette fonction :

```python
# Utilisation
prix1 = calculer_prix_ttc(100)         # Utilise 20% par défaut
prix2 = calculer_prix_ttc(100, 0.055)  # TVA réduite à 5.5%

print(f"Prix TTC (20%) : {prix1} €")      # Affiche: 120.0 €
print(f"Prix TTC (5.5%) : {prix2} €")     # Affiche: 105.5 €
```

Un autre exemple de fonction retournant une liste :

```python
def filtrer_reussis(notes, seuil = 10):
    """
    Filtre les notes supérieures ou égales au seuil.

    Paramètres:
        notes (list): Liste des notes
        seuil (float): Note minimale pour réussir

    Retour:
        list: Liste des notes >= au seuil
    """
    notes_reussies = []
    for note in notes:
        if note >= seuil:
            notes_reussies.append(note)

    return notes_reussies
```

et un exemple d'utilisation :

```python
# Utilisation
toutes_notes = [8, 12, 15, 9, 14, 7, 16]
notes_validees = filtrer_reussis(toutes_notes)
print(f"Notes validées : {notes_validees}")  # Affiche: [12, 15, 14, 16]
```

On peut rencontrer des fonctions qui retournent plusieurs valeurs :

```python
def statistiques_liste(nombres):
    """
    Calcule plusieurs statistiques sur une liste.

    Paramètres:
        nombres (list): Liste de nombres

    Retour:
        tuple: (minimum, maximum, moyenne)
    """
    if len(nombres) == 0:
        return None, None, None

    minimum = min(nombres)
    maximum = max(nombres)
    moyenne = sum(nombres) / len(nombres)

    return minimum, maximum, moyenne
```

et un exemple d'utilisation :

```python
valeurs = [12, 8, 15, 10, 14]
mini, maxi, moy = statistiques_liste(valeurs)

print(f"Minimum : {mini}")      # Affiche: 8
print(f"Maximum : {maxi}")      # Affiche: 15
print(f"Moyenne : {moy}")       # Affiche: 11.8
```

3 Chaîne de fonctions

Le chaînage de fonctions consiste à utiliser le résultat d'une fonction comme argument d'une autre fonction. Cette technique permet de composer des traitements complexes à partir de fonctions simples. Voici deux exemples :

```python
def extraire_nombres(liste_mixte:list)->list:
    """
    Extrait uniquement les nombres d'une liste mixte.
    """
    nombres = []
    for element in liste_mixte:
        if isinstance(element, (int, float)):
            nombres.append(element)
    return nombres
```

```python
def filtrer_positifs(liste_nombres:list)->list:
    """
    Garde uniquement les nombres positifs.
    """
    positifs = []
    for nombre in liste_nombres:
        if nombre > 0:
            positifs.append(nombre)
    return positifs

def calculer_somme(liste_nombres:list)->float:
    """
    Calcule la somme des nombres d'une liste.

    """
    total = 0
    for nombre in liste_nombres:
        total += nombre
    return total
```

et son utilisation :

```python
# Chaînage : extraire → filtrer → sommer
donnees = [12, "texte", -5, 8, None, -3, 15, "autre", 7]

# Le retour de extraire_nombres devient l'entrée de filtrer_positifs
# Le retour de filtrer_positifs devient l'entrée de calculer_somme
resultat = calculer_somme(filtrer_positifs(extraire_nombres(donnees)))

print(f"Somme des nombres positifs : {resultat}")  # Affiche: 42
```

Les calculs mathématiques se prêtent bien à ce principe(en maths cela s'appelle composition...)

```python
def doubler(x):
    return x * 2

def ajouter_dix(x):
    return x + 10

def mettre_au_carre(x):
    return x ** 2
```

qui pourrait permettre les calculs suivants :

```python
# Chaînage : démonstration de l'ordre d'exécution
valeur = 3
# Cas 1 : (3 * 2 + 10)² = 16² = 256
resultat1 = mettre_au_carre(ajouter_dix(doubler(valeur)))
print(f"((3×2)+10)² = {resultat1}")  # Affiche: 256

# Cas 2 : (3² + 10) * 2 = 19 * 2 = 38
resultat2 = doubler(ajouter_dix(mettre_au_carre(valeur)))
```

```python
print(f"((3²)+10)×2 = {resultat2}")  # Affiche: 38

# Cas 3 : (3 + 10)² * 2 = 169 * 2 = 338
resultat3 = doubler(mettre_au_carre(ajouter_dix(valeur)))
print(f"((3+10)²)×2 = {resultat3}")  # Affiche: 338
```

4 La portée de variables

Une ***variable locale*** est une variable déclarée à l'intérieur d'une fonction. Elle n'existe que dans le contexte de cette fonction et n'est pas accessible en dehors de celle-ci.

```python
def calculer_surface(longueur, largeur):
    # 'surface' est une variable locale
    surface = longueur * largeur
    print(f"Surface calculée : {surface}")
    return surface

# Utilisation de la fonction
resultat = calculer_surface(5, 3)
print(f"Résultat : {resultat}")

# Tentative d'accès à la variable locale en dehors de la fonction
# print(surface)  # Ceci générerait une erreur : NameError
```

Dans l'exemple suivant, on définit la variable `message` à deux reprises, pourtant elles sont totalement différentes :

```python
def fonction_a():
    message = "Je suis dans la fonction A"
    return message

def fonction_b():
    message = "Je suis dans la fonction B"
    return message

# Appels des fonctions
print(fonction_a())  # Affiche: Je suis dans la fonction A
print(fonction_b())  # Affiche: Je suis dans la fonction B
```

Une **variable globale** est une variable déclarée en dehors de toute fonction, au niveau principal du programme. Elle est accessible depuis n'importe quelle partie du code, y compris à l'intérieur des fonctions.

```python
# Variable globale
total = 100

def ajouter_montant(montant):
    global total  # Déclaration nécessaire pour modifier
    total = total + montant
```

```
    print(f"Nouveau total : {total}")

print(f"Total initial : {total}")  # Affiche: 100
ajouter_montant(50)                # Affiche: Nouveau total : 150
print(f"Total final : {total}")    # Affiche: 150
```

Sans le mot-clé ***global***, Python créerait une nouvelle variable locale du même nom.

Un ***effet de bord*** est provoqué lorsqu'une fonction modifie l'état d'une variable globale.

En général, cet effet est indésirable...

5 Le coin des exercices

Exercice n°3

❶ Écrire une fonction Python qui prend en paramètre une chaîne de caractères et qui retourne le premier caractère de cette chaîne.

❷ Écrire une fonction Python qui prend en paramètre une chaîne de caractères et qui retourne le premier et le dernier caractère de cette chaîne .

❸ Écrire une fonction Python qui prend en paramètre une chaîne de caractères et un entier et qui retourne le caractère de cette chaîne situé au rang donné par l'entier.

Exercice n°4

Écrire la fonction `hypotenuse` qui prend en paramètres `a` et `b` et qui retourne la longueur de l'hypoténuse d'un triangle rectangle dont les côtés de l'angle droit mesurent `a` et `b`.

Exercice n°5

La fonction ci-dessous prend en paramètres un dictionnaire dont les clés sont des fruits ou des légumes dont le nombre dans le stock est donné par la valeur de ces clés. Par exemple `{courgette:250, carotte: 485, bananes:899,...}`. Compléter le code de la fonction pour qu'elle retourne le nombre total de légumes et fruits dans le stock.

```
def compte_stocke(stock:dict)->float:
    total = ...
    for cle, valeur in stock.items():
        total ......
    return total
```

Exercice n°6

Voici une fonction Python qui prend en paramètres une chaîne de caractères et un entier et qui retourne une chaîne de caractères.

❶ Rappelez le nom de l'opération + pour les chaînes de caractères.

❷ À quoi sert l'instruction `str(age)` ?

❸ Quelle instruction permettrait d'afficher dans la console : `Bonjour je m'appelle Charles et j'ai 15 ans.` ?

```
def presentation(chaine:str, age:int)-> str:
        return "Bonjour je m'appelle " + chaine + " et j'ai " + str(age) + "ans."
```

Exercice n°7

Écrire une fonction qui prend en paramètres une chaîne de caractères et un caractère et qui retourne le nombre de fois où le caractère est présent dans la chaîne.

```
assert compte_carac("informatique", 'i') == 2
assert compte_carac("informatique", 'a') == 1
assert compte_carac("informatique", 'o') == 0
```

Exercice n°8

Construire une fonction `comptage` qui prend en paramètre une liste d'entiers et qui retourne un dictionnaire où les clés sont les entiers présents dans la liste et leur valeur, le nombre de fois où cet entier est présent.

```
assert comptage([1, 1, 2, 5]) == {1:2, 2:1, 5:1}
assert comptage([11, 11, 21, 21, 21, 50]) == {11:2, 21:3, 50:1}
```