Les sous-programmes

Les fonctions

Les procédures

Réflexion

- Supposons que nous avons un programme comme suit:

```
a=int(input("saisir le premier nombre"))
b=int(input("saisir le deuxième nombre"))
s=a+b
print(s)
……
……
c=int(input("saisir le premier nombre"))
d=int(input("saisir le deuxième nombre"))
s=c+d
print(s)
…..
…..
…..
e=int(input("saisir le premier nombre"))
f=int(input("saisir le deuxième nombre"))
s=e+f
print(s)
…..
```

Programme A

Réflexion

- Vous pouvez remarquer qu'il y'a des lignes qui se **répètent** à **plusieurs endroit** dans notre programme mais **les valeurs varient** d'un emplacement vers un autre.
- Vous pouvez remarquer que cette écriture n'est pas bonne, car il y'a des lignes qui reviennent à chaque fois.
- Pour éviter de répéter ces lignes, il faut utiliser ce qu'on appelle **sous-programme.**

Réflexion

- ***Comment faire (l'idée) ?***

  On regroupe les lignes qui se répètent à **part**, et on lui donne un **nom**. Par exemple:

X

```
a=int(input("saisir le premier nombre"))
b=int(input("saisir le deuxième nombre"))
s=a+b
print(s)
```

Puis on **remplace** ces lignes qui reviennent dans notre **programme A** par **X**, donc au lieu d'écrire 9 lignes, on va écrire 3 lignes:

```
X
.....
.....
X
.....
.....
.....
X
.....
```

Programme A

Définition d'un sous programme

- Un sous programme dans le langage python, on appelle une **fonction.**
- Une **fonction** donc est un **ensembles de lignes de code** qui effectuent une certaine tâche(calcul ou autre).
- Ces lignes sont regroupés dans un **bloc** et on lui donne un **nom**. (c'est la **définition** de la fonction)
- Le nom de la fonction est écrit à chaque fois qu'on en a besoin de l'utiliser (on dit : on fait **appel** à la fonction).

L'avantage des fonctions

- L'avantage d'une fonction c'est qu'on peut la **réutiliser plusieurs fois** dans un programme d'une manière simple avec un simple appel.
- Les fonctions rendent également le code **plus lisible** et **plus claire** en le fractionnant en blocs logiques.
- Dans le langage python , il existe plusieurs **fonctions prédéfinies** comme les fonctions **print**(), **input**(), **range**() ou **len**()…
- Dans cette partie du cours c'est à vous de **créer vos propres fonctions**

Types de fonction et utilisation

- Il existe deux types de fonctions:
  1. Fonction qui **ne retourne pas de valeur** , nommée dans certains autres langages procédures
  2. Fonction qui **retourne une valeur**

- Donc , lorsque on travaille avec les fonctions, on doit faire deux choses:
  1. **La Définition de la fonction**
  2. **L'appel de la fonction**

Comment définir une fonction?

- **Pour définir** une fonction, Python utilise le mot-clé **def suivi du nom de la fonction suivi des parenthèses qui sont obligatoires**
- **Les parenthèses peuvent contenir un ou plusieurs paramètres comme elle peuvent être vide**
- Si on souhaite que la fonction **retourne** (renvoie) une valeur, il faut utiliser le mot-clé return.
- **Si la fonction ne retourne pas de valeur, on n'pas besoin du mot return**
- ***Syntaxe***:

```
def nomfonction ( liste des paramètres ) :
    #instructions
    return valeur
```

Les paramètres d'une fonction

- Les paramètres dans la définition d'une fonction, **sont des variables** qui peuvent interagir avec d'autres programme **en prenant des valeurs spécifiés lors de l'appel**,

- ***Exemple:***
- La fonction **len() est une fonction déjà définie, qui** permet de **retourner la longueur** d'une chaine, elle prend **en paramètre une chaine** de caractères.
- Lorsqu'on **fait l'appel** de cette fonction, on lui **donne la valeur** de cette chaine de caractère,

```
s="Bonjour"
x=len(s)
```

Comment faire appel d'une fonction

- Une fonction ne s'exécute que lorsque cette dernière est **appelée**.
- **Le nombre de valeurs** passés entre parenthèses lors de l'appel doit se correspondre au **nombre de paramètres** lors de la définition,
- **Le type des valeurs** passés entre parenthèses lors de l'appel doit se correspondre au **type des paramètres** lors de la définition, successivement

- ***Syntaxe 1***: si la fonction **ne retourne pas** de valeur.

```
NomFonction(valeur1 , valeur2 , ....)
```

- ***Syntaxe 2:*** si la fonction **retourne une valeur**

```
variableX = NomFonction(valeur1 , valeur2 , ....)
```

Définition d'une fonction et son appel

- ***Exemple 1:*** définir une fonction qui retourne la somme de deux nombres données en paramètres.

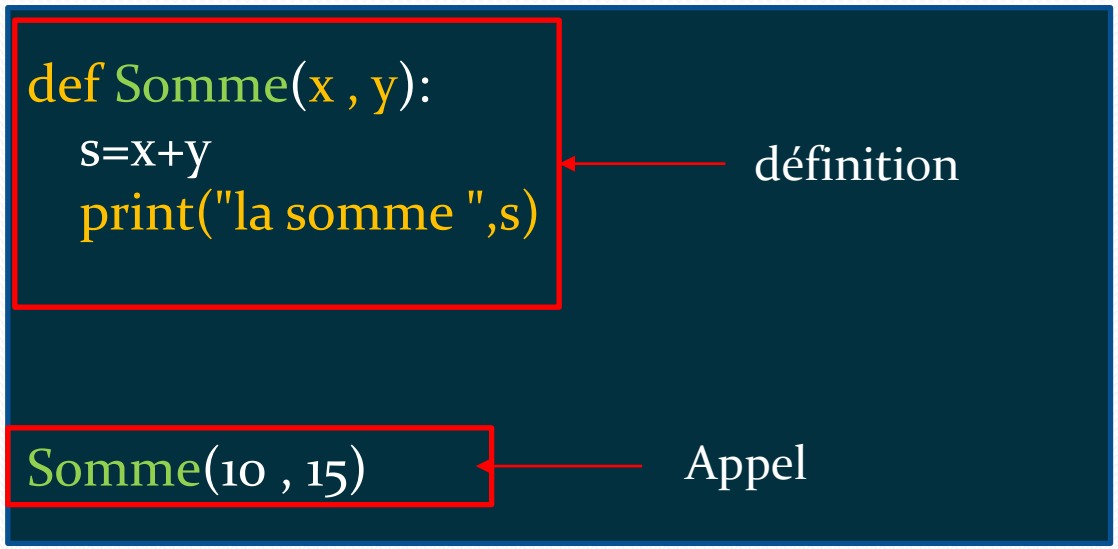
```
def Somme(x , y):
    s=x+y
    return s

X= Somme(10 , 15)

print("le somme des deux nombres : ", X)
```

définition

Appel

Définition d'une fonction et son appel

- ***Exemple 2:*** définir une fonction qui **affiche** la somme de deux nombres données en paramètres.



Définition d'une fonction et son appel

- ***Exemple 3:*** définir une fonction qui **affiche** la somme de deux nombres saisies par l'utilisateur.

```
def Somme():
    x=int(input("saisir le premier nombre"))
    y=int(input("saisir le deuxième nombre"))
    s=x+b
    print("la somme est :" ,s)

Somme()
```

définition

Appel

Définition d'une fonction et son appel

- ***Exemple 4:*** définir une fonction qui **retourne** la somme de deux nombres saisies par l'utilisateur.

```
def Somme():
    x=int(input("saisir le premier nombre"))
    y=int(input("saisir le deuxième nombre"))
    s=x+b
    return s

A=Somme()

Print("la somme est ", A)
```

définition

Appel

Définition d'une fonction et son appel

- ***Exemple 5:*** définir une fonction qui retourne la carré d'un nombre passé en **paramètre. Et faire appel de cette fonction dans un exemple.**

```
def Carre (x):
    return x^2

A=int(input("saisir un nombre de votre choix"))
print("le carré de :",x, "est :",Carre(A))
```

Le mot return dans une fonction

- Le mot return dans une fonction permet de :

1. **Renvoyer** une valeur
2. **Sortir** de la fonction

Donc tous **code écrit après le mot return , ne sera jamais exécuté.**

Les variables locales

- Une variable locale **est une variable définit à l'intérieur d'une fonction.**

- Une variable locale **ne peut pas être utilisée en dehors de la fonction.**

Les variables locales

- Exemple :
- **S** est une variable locale

```
def somme(x, y):
    s=x+y
    return s

print(s)
```

- Afficher une variable locale (s) en dehors de la fonction où a été déclarée.
- Une erreur sera signalée:

```
    print(s)
NameError: name 's' is not defined
```

Les variables globales

- **Une variable globale** est une variable définit en dehors des fonctions.
- Toutes les fonctions définies dans le programme peuvent utiliser une variable globale.
- **La valeur d'une variable globale ne peut pas être modifiée à l'intérieur d'une fonction.**

Les variables globales

- **Exemple:**

Variable globale →

Accès à la variable globale →

Appel de la fonction →

```
s=10
def Afficher():
    s=20


Afficher()
print(s)
```

- Résultat de l'affichage:

```
>>>
10
```

- La valeur de s est **10** au lieu de **20**

Les variables globales

- Exemple:

```
s=10
def Afficher():
    s=20
    print("la valeur est :" ,s)

Afficher()
print(s)
```

- Résultat:

```
>>>
la valeur est 20
10
```

- La valeur de **s à l'intérieur de la fonction** est **20** et la valeur de **s** en **dehors de la fonction est 10**

Modifier une variable globale depuis une fonction

- Dans certaines situations, il serait utile de pouvoir modifier la valeur d'une variable globale depuis une fonction, notamment dans le cas où une fonction se sert d'une variable globale et la manipule.
- Pour faire cela, il suffit d'utiliser le mot clef global devant le nom d'une variable globale utilisée localement afin d'indiquer à Python qu'on souhaite bien modifier le contenu de la variable globale et non pas créer une variable locale de même nom.

```
s=10
def Afficher():
    global s
    s=20
    print(s)

Afficher()
print(s)
```

Le résultat sera comme suit:

20

20