CHAPITRE IX - Python - Fonctions

Les fonctions sont des sous-programmes qu'on peut appeler en cas de besoin. Cela a pour effet d'alléger le corps principal du programme et facilite sa compréhension. Cela aide en outre sa conception en permettant de répartir le contenu en plusieurs blocs, à traiter au fur et à mesure ou à confier en partie à un autre programmeur.

Écriture d'une fonction

def *nom_fonction* **(**arg1, arg2, ..**):**
Bloc d'instructions
return *valeur*

Remarques

- Les **« : »** et **l'indentation** sont obligatoires comme pour les tests, les boucles.
- Les **« () »** sont obligatoires, mais il n'y a pas nécessairement d'**argument** (ou paramètre).

Conseils

- « **return** *valeur*» en fin de bloc d'instructions n'est pas systématique : il renvoie un résultat si besoin.
- nom de fonction : éviter les mots réservés (print, …), les caractères spéciaux ou accentués.
  Le caractère souligné « _ » est permis.

Ex1 : Multiplications

Tester ce qui suit directement dans "console python " fenêtre du bas (ou "shell").

1. **Une fonction sans argument**

- Que fait ce script ? Compléter le commentaire
- Le taper (sans le copier), puis faire Enter.
  A l'invite de commande >>>, taper table7() et vérifier.

```
>>>def table7():    # Que fait cette fonction ? ……
        n=1
        while n<11:
            print(n*7,' ',end='')
            n=n+1
```

2. **Passer un argument**

- Modifier le script précédent (flèche Haut) comme suit.
- Deviner le résultat : Compléter le commentaire
- Vérifier en tapant table(4), puis table(8)

```
>>> def table(x):   # Résultat ?……..
...     n = 1
...     while n <11 :
...         print (n * x,' ',end='')
...         n = n +1
...
```

3. **En quatre lignes !**

A partir de ce qui précède, écrire (toujours en console) une fonction table_entier(x), permettant d'écrire les 10 premiers multiples des entiers de 1 à x, comme ci-contre où x=10

Idée : Une fonction peut en appeler une autre.

```
>>> table_entier(10)
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
>>>
```

Appeler le professeur pour validation

Ex2 : Arguments et valeurs

Le programme doit simuler le lancé de deux dés à six faces puis afficher les chiffres et la somme obtenu.

Compléter le nom et les instructions de la fonction manquante.

Tester

```
from random import*
def …....
    ..
    ..
    ..
# corps du programme
D1=randrange(1,7)
D2=randrange(1,7)

s=somme2D(D1,D2)

print('Résultats : Dé1=',D1, '; Dé2=',D2,'; Total=', s)
```

NB : Le nom d'une variable passée comme argument est indépendant du nom de l'argument dans la fonction. Ces noms peuvent être identiques si vous le voulez, mais ils ne désigne pas la même mémoire.

Ex3 : Morceau de pendu

Vous devez créer une fonction « nbessais(*m,L,n*) ». Les paramètres d'entrée sont : "m" un mot à trouver (en majuscules) , "L" la lettre proposée, et "n" le nombre d'essais restant avant la proposition.

En sortie la fonction doit afficher le nombre d'essais restant après la proposition.

Par exemple : nbessais('BARBECUE',' Z',5) doit afficher 4

Ex4 : Autre morceau de pendu (en plus pour les rapides)

Vous devez créer une fonction « affichemot(m,L) ». Les paramètres d'entrée sont : "m" Le mot à trouver et "L" la chaîne formée des lettres déjà proposées.

En sortie la fonction doit afficher le mot à trouver avec des « _ » à la place des lettres manquantes

Par exemple : affichemot('MAISON',' CARTON') donne _A_ _ON

Portées des variables

Une variable définie à **l'intérieur d'une fonction est locale :**

Elle est détruite dès qu'on sort de la fonction.

Une variable définie à **l'extérieur d'une fonction est globale :**

Elle est accessible depuis l'intérieur de fonction, mais sans pouvoir en modifier le contenu.

Ex5 : Quel est le truc ?

Travailler avec l'interpréteur, dans la console Python

1. a) Taper les deux premières lignes. Qu'affichera print(p) ? Tester

   b) Taper la suite et deviner ce qu'affiche **truc()**. Tester

   c) Qu'affichera cette fois **print(p) ?** Vérifier, expliquer

```
>>>p=15
>>>print(p)     # résultat ?……...

>>>def truc():
        p=8
        print (p)

>>>truc()    # résultat ?……...
```

2. Comment faire alors pour modifier tout de même une variable globale avec une fonction ?

   a) Méthode 1 : Insérer l'instruction ***global p*** au début la fonction rend la variable *p* globale.

   Faire la modification puis tester avec « >>> p=15 », « >>>truc() » puis « >>>print(p) » .

   b) Méthode 2(préférable) :

   insérer ***return p*** en fin de fonction. Ensuite dans le programme, taper **p=truc()** changera forcément p.

Ex6 : Mini test

Déterminer ce que vont afficher ces programmes : soit un message d'erreur, soit le résultat. Puis tester.

def demo(v):
  v=v+10
  print (v)
 	
# *corps du programme* 	
v=20
print (v)
demo()
print (v)
 	
# résultats ? ……. 	
	def demo(v):
	  v=v+10
	  print (v)
	
	# *corps du programme*
	v=20
	print (v)
	demo(v)
	print (v)
	
	# résultats ? …….
		def demo(v):
		  v=v+10
		  print (v)
		
		# *corps du programme*
		v=20
		print (v)
		demo(0)
		print (v)
		
		# résultats ? …….
			def demo():
			  global v
			  v=v+10
			  print (v)
			# *corps du programme*
			v=20
			print (v)
			demo()
			print (v)
			
			# résultats ? …….

Écriture dans un programme

Dans un programme on écrit d'abord les fonctions , puis le corps principal du programme.

De plus il est conseillé de les documenter : c'est à dire de décrire entre guillemets leur fonctionnement

Plan d'un programme

Descriptif du programme

##Liste des fonctions

''' Avec la documentation adéquate'''

Programme principal

On commente et on utilise les fonctions précédentes