FICHE N°3 : LES FONCTIONS

A) Qu'est-ce qu'une fonction pour Python ?

Définition : Une **fonction** est une suite d'instructions qui définissent un sous-programme et qui renvoient un résultat pouvant être utilisé autant de fois que nécessaire dans un programme plus général.

L'un des principaux intérêts d'une fonction est d'éviter d'écrire toujours la même séquence d'instructions lorsque l'on sait que cette séquence va servir plusieurs fois. Les fonctions permettent de gagner du temps et surtout de rendre un programme plus lisible.

Info : Les instructions print() et sqrt() sont des exemples de fonctions qui ont déjà été rencontrées dans la fiche n°2.

Exemple n°1 : Dans une station-service, on cherche à automatiser le calcul du prix payé pour un plein d'essence. Au lieu de créer une variable, on pourra donc utiliser une fonction pour systématiser le calcul (voir partie ci-dessous).

B) Fonction sans argument

Une fonction sans argument (ou sans paramètre) est une suite d'instructions qui ne nécessite pas que l'utilisateur précise une valeur pour utiliser la fonction.

Définir une fonction sans argument
Dans un algorithme

Fonction prix_au_litre():
  Retourner 1.52
Fin Fonction
 	
Fonction quantite_en_litre():
  Retourner 50
Fin Fonction
 	
Fonction total():
  Retourner prix_au_litre() × quantite_en_litre()
Fin Fonction
 	
Afficher total()

À retenir : Dans un algorithme, la définition d'une fonction sans argument commence toujours par la ligne :
Fonction nom_de_la_fonction():
Les parenthèses vides indiquent que la fonction n'a pas besoin de paramètres.
Les deux-points à la fin de la ligne sont importants.
Ensuite, la suite d'instructions (parfois très courte comme ici) est écrite avec une indentation (un alinea) pour montrer que l'on est à l'intérieur d'un bloc appartenant à un même groupe d'instructions.
L'instruction Retourner indique la valeur sauvegardée à la fin de l'appel à la fonction.
L'instruction Fin Fonction permet de montrer que l'on a fini le bloc d'instructions et que l'indentation n'est plus nécessaire.

Définir une fonction sans argument

Dans un programme Python

```
1   def prix_au_litre():
2     return(1.52)
3
4   def quantite_en_litre():
5     return(50)
6
7   def total():
8     return(prix_au_litre() * quantite_en_litre())
9
10  print(total())
```

À retenir : Dans un programme Python, la définition d'une fonction sans argument commence toujours par la ligne :
def nom_de_la_fonction():
Les parenthèses vides indiquent que la fonction n'a pas besoin de paramètres.
Les deux-points à la fin de la ligne sont **obligatoires**.
Ensuite, la suite d'instructions (parfois très courte comme ici) est écrite avec une indentation (un alinea) pour montrer que l'on est à l'intérieur d'un bloc. Avec Python, les indentations sont **obligatoires**. Elles ont également l'avantage de clarifier la lecture du programme. Lorsque l'on continue le programme sans indenter le texte, cela signifie que l'on a quitté le bloc d'instructions correspondant à la fonction.
L'instruction return() permet de savoir la valeur sauvegardée à la fin de l'appel à la fonction. Dès que Python exécute une instruction return(), l'appel à la

fonction est automatiquement terminé et le programme continue avec le bloc suivant.

C) « Print » ou « Return » ? Que choisir ?

À retenir :
- print(<argument>) **affiche** la valeur contenue dans <argument> et aucune valeur n'est sauvegardée. On ne peut donc pas s'en resservir par la suite.
- return(<argument>) n'affiche rien mais **retourne** la valeur de <argument> et la stocke dans une variable qui pourra ensuite être utilisée par la suite (dans une autre fonction par exemple ou dans un calcul). La valeur stockée pourra être affichée à l'aide de print().

Pour cela, dans les fonctions, on utilise return() alors qu'on utilise print() dès que l'on souhaite afficher un résultat.

D) Fonction avec argument(s)

Les programmes précédents utilisent encore beaucoup de lignes de code. On peut les améliorer en les raccourcissant. Pour cela, on va utiliser des arguments (ou paramètres) dans la définition d'une fonction.

Définir une fonction avec arguments
Dans un algorithme
Fonction total(prix_au_litre, quantite_en_litre):   Retourner prix_au_litre × quantite_en_litre Fin Fonction T1 ← total(1.52, 50) T2 ← total(1.43, 45) Afficher T1 Afficher T2

À retenir : Dans un algorithme, la définition d'une fonction avec argument(s) ressemble à celle d'une fonction sans argument. Il faut juste écrire le nom des arguments dans les parenthèses. Il peut y avoir un ou plusieurs arguments, ils sont alors séparés par des virgules.
On définit une fonction avec des arguments lorsqu'on en a besoin dans le bloc d'instructions de la fonction.

Les variables T1 et T2 font toutes les deux appel à la fonction total(). Dans T1, la variable prix_au_litre prend la valeur 1.52 et quantite_en_litre prend la valeur 50. La fonction total() renvoie donc la valeur $1,52 \times 50 = 76$. La variable T1 prend donc la valeur 76.0 de type *flottant* qui sera la valeur affichée.
La variable T2 prend la valeur 64.35 de type *flottant* également.

Attention : Les variables prix_au_litre et quantite_en_litre ne sont définies que pour les besoins de la fonction total() et n'existent pas en dehors de cette fonction.

Définir une fonction avec arguments

Dans un programme Python

```
1  def total(prix_au_litre, quantite_en_litre):
2    return(prix_au_litre * quantite_en_litre)
3
4  T1 = total(1.52, 50)
5  T2 = total(1.43, 45)
6
7  print(T1, T2)
```

À retenir : Dans un programme écrit avec Python, la définition d'une fonction avec argument(s) ressemble à celle d'une fonction sans argument. Il faut juste écrire le nom des arguments dans les parenthèses. Il peut y avoir un ou plusieurs arguments, ils sont alors séparés par des virgules.
Les variables T1 et T2 font toutes les deux appel à la fonction total(). Dans T1, la variable prix_au_litre prend la valeur 1.52 et quantite_en_litre prend la valeur 50. La fonction total() renvoie donc la valeur $1,52 \times 50 = 76$. La variable T1 prend donc la valeur 76.0 de type *flottant* qui sera la valeur affichée.
La variable T2 prend la valeur 64.35 de type *flottant* également.

Exemple n°2 : La fonction perimetre_carre() renvoie le périmètre d'un carré de côté c.

Algorithme	Programme Python

Fonction perimetre_carre(cote):
  perimetre ← 4*cote
  Retourner perimetre
Fin Fonction
	`def perimetre_carre(cote):`
	`  perimetre = 4*cote`
	`  return(perimetre)`

E) Exercices

✓ **Exercice 1 :**

1) Écrire, en utilisant la syntaxe d'un algorithme, une fonction aire_carre() avec un argument qui renvoie l'aire d'un carré de côté c.
2) Programmer et tester cette fonction avec Python.

✓ **Exercice 2 :**

1) Écrire en utilisant la syntaxe d'un algorithme une fonction perimetre_rectangle() avec deux arguments qui renvoie le périmètre d'un rectangle.
2) Écrire en utilisant la syntaxe d'un algorithme une fonction aire_rectangle() avec deux arguments qui renvoie l'aire d'un rectangle.
3) Programmer et tester les deux fonctions précédentes avec Python.

✓ **Exercice 3 :**

On peut utiliser la notion de fonction en informatique pour définir une fonction au sens mathématique. Par exemple, pour définir la fonction f sur $\mathbb{R}$ par $f(x) = 3x^2 + 2x - 4$, on peut utiliser les syntaxes suivantes.

Algorithme	Programme Python
Fonction f(x): Retourner $3x^2$ + 2x - 4 Fin Fonction	`def f(x):` `  return(3*x**2 + 2*x - 4)`

1) Définir la fonction g sur $\mathbb{R}$ par $g(x) = -2x^3 + 4x - 7$ en utilisant les deux syntaxes.
2) À l'aide de Python et de la fonction print(), afficher l'image des nombres –3 ; 0 ; $\frac{5}{2}$ et $\frac{5}{7}$ par la fonction g.

✓ **Exercice 4 :**

La valeur absolue d'un nombre x est définie sur Python par la fonction abs(x) (l'utilisation de cette fonction ne nécessite pas la librairie *math*).
1) Tester la fonction abs(x) sur Python avec différentes valeurs de x.
2) Écrire, en utilisant la syntaxe d'un algorithme, une fonction distance() qui prend deux nombres réels en arguments et qui calcule la distance entre ces deux réels (dans un algorithme, on pourra utiliser la notation abs(x) ou bien la notation mathématique $|x|$).
3) Programmer et tester cette fonction à l'aide de Python.

✓ **Exercice 5 :**

On considère un triangle ABC rectangle en A.

Écrire une fonction qui prend en arguments les longueurs AB et AC et qui renvoie la longueur BC.

Indication : on pensera à utiliser la fonction sqrt() de la librairie math.

✓ **Exercice 6 :**

1) On considère le programme ci-dessous.

```
1  from math import sqrt
2  def cote_angle_droit(a, b):
3    c = sqrt(a**2 - b**2)
4    return(c)
5
```

a) Que retourne la fonction cote_angle_droit() avec $a = 5$ et $b = 3$?

b) Que se passe-t-il avec $a = 3$ et $b = 5$? Justifier.

c) On suppose que a et b sont deux réels strictement positifs. Quel est le but de la fonction cote_angle_droit() ?

2) On suppose que a et b sont deux réels strictement positifs tels que $a > b$. Le programme continue de la façon suivante.

```
6  def perimetre(a, b):
7    c = cote_angle_droit(a, b)
8    P = a + b + c
9    return(P)
10
11 def aire(a, b):
12   c = cote_angle_droit(a, b)
13   A = (b*c)/2
14   return(A)
```

a) Sans utiliser Python, calculer puis interpréter la valeur affichée par les instructions print(perimetre(5, 3)) et print(aire(5, 3)).

b) Faire la vérification avec Python. Commenter le résultat.

✓ **Exercice 7 :**

On considère le programme ci-contre.

1) Sans utiliser Python, calculer les quatre valeurs affichées par ce programme.

2) Vérifier ensuite avec Python.

```
1  def f(x):
2    return(3*x**2 - 2*x + 2)
3
4  def g(x):
5    return(-2*x + 1)
6
7  def h(x):
8    return(x**2 - 1)
9
10 print(f(g(h(1))))
11 print(h(g(f(1))))
12 print(f(g(h(0))))
13 print(h(g(f(0))))
```