

## Chapitre 18

# Fonctions sous Python

### Contenu

1. **Mise en route**
  - a) Sous une bonne étoile
  - b) À la bonne surface
2. **Fonctions**
  - a) Intérêts
  - b) Structure
3. **Les fonctions existantes**
4. **Mes fonctions**
  - a) Constitution d'une bibliothèque
5. **Caisse à outils**
6. **Algorithmes**
7. **Évaluations**

### Liste des capacités

#### Notion de fonction

Écrire des fonctions simples ; lire, comprendre, modifier, compléter des fonctions plus complexes. Appeler une fonction.

Lire et comprendre une fonction renvoyant une moyenne, un écart type. Aucune connaissance sur les listes n'est exigée.

Écrire des fonctions renvoyant le résultat numérique d'une expérience aléatoire, d'une répétition d'expériences aléatoires indépendantes.

### Le saviez-vous ?

Le mathématicien anglais Alan Turing est considéré comme l'un des pères fondateurs de l'informatique moderne. Après avoir grandement contribué à casser les codes de la machine *Enigma*, qui cryptait les messages de l'armée nazie, il a posé les bases du fonctionnement des tout premiers ordinateurs.



## 1. Mise en route

### a) Sous une bonne étoile

1. À la lecture de fonction **toit()** dans le programme ci-contre, que permet-elle de tracer ?

On pivote de  $60^\circ$  vers la gauche, puis on avance de 20 après on pivote de  $120^\circ$  vers la droite et enfin on avance de 20. Cela ressemble à un toit.

2. Tracer le dessin obtenu si on répète que deux fois la fonction **toit()**.

M incliné vers la droite.


3. Saisir puis exécuter le programme proposé; que représente le dessin obtenu ?

Une étoile à 6 branches.

```
from turtle import*
def toit():
    left(60)
    forward(20)
    right(120)
    forward(20)

setheading(0)
down()
for i in range(6):
    toit()

mainloop()
```

4. Modifier le programme proposé en ajoutant une ligne à la fonction **toit()** et en modifiant le nombre de répétitions de telle sorte qu'il dessine la forme ZigZag ci-après. 

ZigZag

```
from turtle import*
def toit():
    left(60)
    forward(20)
    right(120)
    forward(20)
    left(60)
setheading(0)
down()
for i in range(3):
    toit()


mainloop()
```

Arc

```
from turtle import*
def toit():
    forward(20)
    left(30)

setheading(0)
down()
for i in range(5):
    toit()

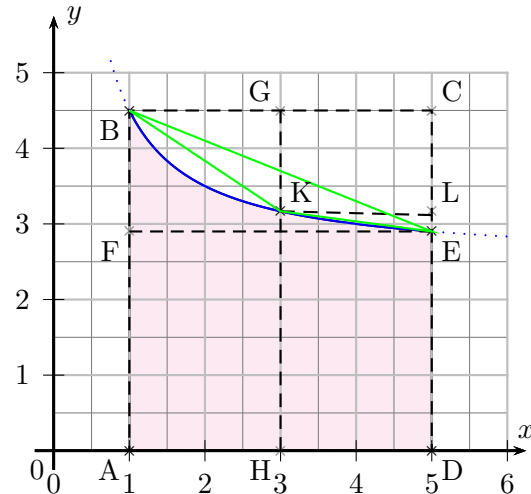
mainloop()
```

5. Écrire un programme Python reproduisant la figure suivante. 

## b) À la bonne surface

Pour un effet de style, la toiture d'un bâtiment suit une courbe modélisée par la fonction  $f(x) = \frac{2}{x} + 2.5$  sur l'intervalle  $[1; 5]$ , vous en trouverez une représentation ci-contre. Des façadiers doivent crépir le mur ; se pose la question de la connaissance de la surface pour connaître la quantité de crépi nécessaire et le montant de la facture qui sera adressée au client. Des calculs savants permettent d'obtenir le résultat  $\mathcal{A} \approx 13,22m^2$ . Mais en connaissant simplement l'aire d'un rectangle ou d'un trapèze, on peut en faire une très bonne approximation ...

Les points B, E et K sont sur la courbe représentative  $\mathcal{C}_f$  de la fonction  $f$ .



Pour faire une meilleure approximation, on décide de décomposer la figure en deux parties en prenant le milieu de  $[AD]$  noté H pour démarcation.

1. Tracer sur la figure ci-contre le rectangle ABCD.
2. Calculer sa surface. Que peut-on dire par rapport à celle du mur qui nous intéresse ?

$\mathcal{A}_{ABCD} = 4,5 \times 4 = 18$  elle est plus grand que celle du mur.

3. Tracer sur la figure ci-contre le rectangle ADEF.
4. Calculer sa surface. Que peut-on dire par rapport à celle du mur qui nous intéresse ?

$\mathcal{A}_{ADEF} = 4 \times 2,9 = 11,6$  elle est plus petite que celle du mur.

5. Tracer sur la figure ci-contre le trapèze ABED.
6. Calculer sa surface. Que peut-on dire par rapport à celle du mur qui nous intéresse ?

$\mathcal{A}_{ABED} = \frac{(4,5+2,9) \times 4}{2} = 14,8$  elle est plus grande que celle du mur mais plus proche que les autres.

7. Tracer les rectangles ABGH et HKLD. Calculer leur surface puis en faire la somme.

$\mathcal{A}_{ABGH} = 4,5 \times 2 = 9$  et  $\mathcal{A}_{HKLD} = 3,17 \times 2 = 6,34$  donc l'aire totale est environ 15,34.

8. Tracer les trapèzes ABKH et HKED. Calculer leur surface puis en faire la somme.

$\mathcal{A}_{ABKH} = \frac{(4,5+3,17) \times 2}{2} = 7,17$  et  $\mathcal{A}_{HKED} = \frac{(3,17+2,9) \times 2}{2} = 6,07$  donc l'aire totale est environ 13,24.

9. Quelle est l'approximation la plus proche ?

Celle faite par les trapèzes.

10. Écrire une fonction sous Python renvoyant l'image par la fonction  $f$  d'une valeur numérique donnée en paramètre.
11. Écrire une fonction sous Python renvoyant l'aire d'un trapèze avec en paramètre la petite et grande base ainsi que la hauteur.

```
def f():
    return 2/x+2.5

def Strapeze(b,B,h):
    return (b+B)*h/2
```

12. Soit le programme ci-contre.

a) Expliquer ce qu'il réalise.

Il définit les deux fonctions précédentes puis demande à l'opérateur de saisir le nombre d'intervalles souhaités (ou le nombre de trapèzes). Enfin il calcule l'aire de tous les trapèzes et en fait leur somme.

b) Vérifier vos résultats en prenant pour  $n$  les entiers 1 puis 2.

c) Tester ensuite avec des valeurs de plus en plus en grande 10, 100, 1000, ... Que constate-t-on ?

L'approximation de l'aire est de plus en plus fine, précise.

```
def f(x):
    return 2/x+2.5

def Strapeze(b,B,h):
    return (b+B)*h/2

n=int(input("Nb d'intervalles ="))
l=4/n
a=1
b=a+l
S=0
while b<=5:
    S=S+Strapeze(f(a),f(b),l)
    a=b
    b=b+l
print("Surface =",S)
```

Il faut rester prudent avec les approximations, tester par exemple avec  $n = 19, 20, 21, 22$ .

## 2. Fonctions

En informatique, la notion de fonction est très importante; elle permet de structurer un programme en le décomposant en sous-programme.



Labo 1.

### Fonctions

Une fonction réalise un bloc d'instructions et renvoie un résultat; elle ne sera exécutée que si elle est appelée dans le programme et ceux-ci éventuellement plusieurs fois. Elle possède généralement des paramètres qui prendront pour valeur les arguments donnés à la fonction.

*Exemple : on considère les algorithmes suivants calculant le volume de pyramides à base carrée. Soient  $c, h$  et  $v$  des variables réelles.*

*Algorithme sans fonction*

```
c ← 2
h ← 5
Afficher  $1/3 \times h \times c^2$ 
c ← .5
h ← 10
Afficher  $1/3 \times h \times c^2$ 
```

*Algorithme avec fonction*

```
Définir fonction
volpyr ←  $1/3 \times h \times c^2$ 

Afficher volpyr(2,5)
Afficher volpyr(.5,10)
```

*Ce qui se traduit par les programmes suivants :*

sans fonction

```
c=2
h=5
print(1/3*h*c**2)
c=.5
h=10
print(1/3*h*c**2)
```

avec fonction

```
def volpyr(c,h):
    return 1/3*h*c**2

print(volpyr(2,5))
print(volpyr(.5,10))
```

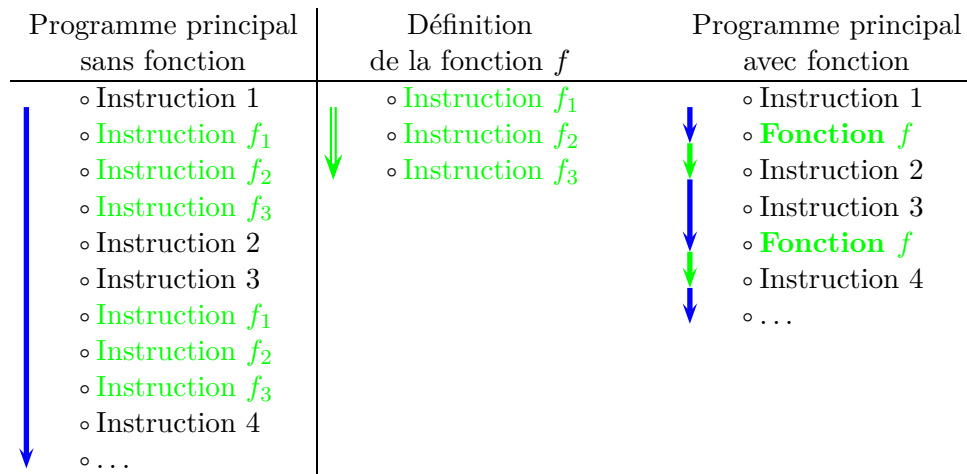
*La fonction volpyr prend en paramètres les valeurs du côté de la base et la hauteur de la pyramide et renvoie le volume de la pyramide.*

### a) Intérêts

- Lisibilité : les fonctions rendent le programme principal plus facile à lire.
- Répétabilité : les fonctions permettent d'éviter la redondance du programme. Il est assez fréquent d'avoir à répéter des blocs d'instructions, on les regroupe alors dans une fonction.
- Flexibilité : il est plus rapide de modifier une fonction que de modifier l'ensemble des parties du programme où elle est utilisée.

### b) Structure

Les schémas ci-dessous montrent la différence entre l'exécution d'un programme sans fonction, à gauche, et d'un programme avec fonction, à droite ; où les instructions  $f_1$ ,  $f_2$ ,  $f_3$  sont regroupées dans une fonction  $f$ .



## 3. Les fonctions existantes

Il existe déjà un très grand nombre de fonctions stockées dans des modules que l'on importe en début de programme. Quelques exemples :

- le module **math** intègre la fonction `sqrt()` qui permet de calculer la racine carrée d'un nombre ;
- le module **random** intègre la fonction `random()` qui permet de renvoyer un nombre aléatoire compris entre 0 et 1 ;
- ...

## 4. Mes fonctions

Le programmeur peut créer ses propres fonctions et enrichir une ou des bibliothèques personnelles. Les lignes de création d'une fonction sont les suivantes :

- la première ligne du bloc d'instructions commence par le mot clé `def`, suivi du nom de la fonction avec entre parenthèses les éventuels paramètres et terminée par `;` ;
- on place une indentation pour les instructions ;
- la dernière ligne du bloc d'instructions commence par le mot clé `return`, suivi entre parenthèses du(des) résultat(s) renvoyé(s) par la fonction.

Exemple :

*Fonction calculant le volume d'un pavé*

Définir fonction  
 $volpav \leftarrow L \times l \times h$   
 Renvoyer le volume

```
def volpav(a,b,c):
    return a*b*c
```

Le mot clé `def` permet la construction d'une fonction.

Le mot clé *return* permet à la fonction de renvoyer un résultat.

Les paramètres sont des variables nécessaires au bon fonctionnement de la fonction ; certaines d'entre-elles n'admettent aucun paramètre.

*Remarques :*

- on choisit le nom de la fonction en rapport avec ce qu'elle réalise pour rendre le programme principal plus lisible, plus intelligible ;
- on retrouve le double-point : et l'indentation pour indiquer le début et la fin de la fonction ;
- de nombreuses bibliothèques de fonctions sont disponibles ; il faut au préalable les importer avec le mot clé *import*.

### a) Constitution d'une bibliothèque

#### Aire d'un carré

L'aire d'un carré est obtenue en élevant au carré la longueur d'un côté donc on peut traduire cela avec une fonction telle que :

```
def Scarre(c):
    return c**2
```

#### Bibliothèque de fonctions calculant des aires

On crée un fichier *Surfaces.py* dans lequel on stocke les fonctions calculant les aires. On importe le module **math** pour avoir le nombre  $\pi$ .

```
from math import*

def Scarre(c):
    return c**2

def Sdisque(r):
    return pi()*r**2

def Srectangle(l,h):
    return l*h

def Striangle(b,h):
    return b*h/2

def Strapeze(b,B,h):
    return (b+B)*h/2
```

#### Utilisation de la bibliothèque *Surfaces*

En début de programme j'importe la bibliothèque désirée puis je peux utiliser les fonctions qui y sont définies.

Ce programme demande à l'utilisateur de renseigner une longueur, une base et une hauteur puis affiche l'aire d'un rectangle de longueur  $l$  et de hauteur  $h$ , l'aire d'un triangle de base  $b$  et de hauteur  $h$ , l'aire d'un carré de côté  $l$ , l'aire d'un disque de rayon  $b$  et enfin l'aire d'un carré de côté 5.

```
from Surfaces import*

l=float(input("longueur ="))
b=float(input("base ="))
h=float(input("hauteur = "))

print(Srectangle(l,h))
print(Striangle(b,h))
print(Scarre(1))
print(Sdisque(b))
print(Scarre(5))
```

## 5. Caisse à outils

*Créer une fonction simple*  $\Rightarrow$  on débute le bloc d'instructions par le mot clé *def*, on lui attribue un nom explicite puis entre parenthèse on indique les paramètres. Après indentation, on écrit les instructions et pour terminer on renvoie éventuellement un résultat par le mot clé *return*.

Voici une fonction qui simule le lancer d'un dé de  $n$  faces (valeur donnée en paramètre) renvoyant un entier naturel compris entre 1 et le nombre total de faces.

Définir fonction  
 $\text{lancerde}(n) \leftarrow$  nombre entier entre 1 et  $n$

```
def lancerde(n):
    return(randint(1,n))
```

Dans le programme, il faudra appeler le module **random** pour que la commande *randint* soit opérationnelle. On peut également passer par l'usage d'une liste qui contient l'ensemble des issues, puis utiliser la commande **choice** qui permet de choisir aléatoirement une valeur contenue dans la liste.

```
def lancerde(n):
    de=[i+1 for i in range(n)]
    return(choice(de))
```

*Application : écrire une fonction simulant le lancer d'un dé cubique truqué sachant que la face 5 à deux fois plus chance de sortir que les autres faces et renvoie 1 si l'on lit la face 5.*

On écrit la loi de probabilités correspondante :

1	2	3	4	5	6
p	p	p	p	2p	p

donc  $p = \frac{1}{7}$  et  $P(5) = \frac{2}{7}$ , on considère que si l'on obtient une valeur décimale comprise entre 0 et  $\frac{2}{7}$  on renvoie 1 sinon 0.

Définir fonction  
 $r \leftarrow 0$   
 $d \leftarrow$  Nb aléatoire  
**si**  $d \leq \frac{2}{7}$  **alors**  
     $r \leftarrow 1$   
**fin**  
Renvoyer  $r$

```
def de_truque():
    r=0
    d=random()
    if d<= 2/7:
        r=1
    return(r)
```

```
def de_truque():
    DE=[1,2,3,4,5,5,6]
    if choice(DE)==5:
        return(1)
    else:
        return(0)
```

En utilisant une liste on obtient la fonction suivante :

*Modifier ou compléter une fonction*  $\Rightarrow$  suivant le cahier des charges on modifie les instructions déjà présentes ou l'on ajoute les instructions manquantes.

Voici une fonction qui calcule la vitesse d'un coureur à pied, à partir de deux paramètres qui sont la distance parcourue en kilomètres et le temps en heures ; elle renvoie une vitesse exprimée en kilomètre par heure.

Définir fonction  
 $v \leftarrow d/t$

```
def vitesse(d,t):
    return(d/t)
```

Mais dans les plans d'entraînement pour les courses de fond, les consignes ne sont pas des vitesses exprimées en *km/h* mais des allures exprimées en *min/km*. Pour simplifier on conservera les minutes décimales.

Définir fonction  
 $v \leftarrow d/t$   
 $a \leftarrow 60 \times t/d$

```
def vitesse(d,t):
    v=d/t
    a=60*t/d
    return(v,a)
```

Application :

le programme ci-contre renvoie la moyenne de deux valeurs. Modifier ce programme pour qu'il renvoie la moyenne de trois valeurs.

```
def moy(a,b,c):
    return((a+b+c)/3)
```

```
def moy(a,b):
    return((a+b)/2)
```

## 6. Algorithmes

*Fonction renvoyant la moyenne d'une série de valeurs*

Pour déterminer la moyenne d'une série de valeurs, on doit ajouter toutes les valeurs de la série puis diviser cette somme par l'effectif total. La commande **len** donne le nombre de valeurs dans la liste donc l'effectif total ; la commande **sum** qui effectue la somme des valeurs d'une liste.

Définir fonction  
 $\bar{x} \leftarrow \text{somme des valeurs} / \text{Nb des valeurs}$

```
def moy(a):
    return(sum(a)/len(a))
```

*Fonction renvoyant l'écart type d'une série de valeurs*

Pour déterminer l'écart type d'une série, on effectue la moyenne des écarts positifs des valeurs par rapport à la moyenne. On doit penser à importer la collection **math** pour pouvoir effectuer le calcul de la racine carrée.

$N \leftarrow \text{effectif total}$   
 $m \leftarrow \text{moyenne}(\text{serie})$   
**pour**  $i$  variant de 1 à  $N$  **faire**  
    |  $\text{ecarts}[i] \leftarrow (\text{serie}[i] - m)^2$   
**fin**  
 $s \leftarrow \sqrt{\frac{\text{somme}(\text{ecarts})}{N}}$   
Afficher  $s$

```
from math import*

def moyenne(maliste):
    return sum(maliste)/len(maliste)

def ecartype(maliste):
    N=len(maliste)
    m=moyenne(maliste)
    ecarts=[(maliste[i]-m)**2 for i in range(N)]
    return (sqrt(sum(ecarts)/N))
```

Pour rappel, l'algorithme suivant a été utilisé dans le chapitre sur l'échantillonnage. On détermine la fréquence des échantillons dont la fréquence observée du caractère étudié est contenue dans l'intervalle de confiance.

*Simulation de  $N$  échantillons de taille  $n$ .*

On fait appel à des fonctions qui permettent dans l'ordre de calculer le nombre de succès, tirage inférieur à la proportion  $p$ , puis de calculer la fréquence observée dans l'échantillon. Enfin on comptabilise les échantillons pour lesquels l'écart entre proportion et fréquence est suffisamment faible pour en calculer la proportion. Pour exécuter le programme, saisir par exemple la commande `repet_echan(30,100,0.7)` ; donc  $N = 30$ ,  $n = 100$  et  $p = 0,7$ .



```

Définir fonction nombre_succes
nb_succes ← 0
pour compteur allant jusqu'à n faire
    si valeur aléatoire < p alors
        | nb_succes ← nb_succes + 1
    fin
fin
Renvoyer nb_succes

```

```

Définir fonction frequence_succes
Renvoyer nombre_succes(n,p)/n

```

```

Définir fonction repet_echan
s ← 0
pour i allant jusqu'à N faire
    f ← frequence_succes(n,p)
    si abs(p - f) ≤ 1/√n alors
        | s ← s + 1
    fin
fin
Renvoyer s/N

```

```

from math import*
from random import*

def nombre_succes(n,p):
    nb_succes = 0
    for compteur in range(n):
        if random()<p:
            nb_succes = nb_succes + 1
    return nb_succes

def frequence_succes(n,p):
    return nombre_succes(n,p)/n

def repet_echan(N,n,p):
    s = 0
    for i in range(N):
        f = frequence_succes(n,p)
        if abs(p-f) <= 1/sqrt(n):
            s = s+1
    return s/N

```

## 7. Évaluations

### *Devoir en temps libre n° 18 : Fonctions sous Python*

#### Exercice n°1 : Et le plus grand est ...

On souhaite écrire une fonction qui renvoie le maximum de trois valeurs.

1. Écrire un algorithme qui permet d'obtenir le maximum de trois valeurs appelées  $a$ ,  $b$  et  $c$ .

Comparer  $a$  et  $b$ , si  $a$  est le plus grand, le comparer avec  $c$ , le maximum est le plus grand des deux. Sinon comparer  $b$  et  $c$ , le maximum est le plus grand des deux.

2. Compléter la fonction ci-dessous pour qu'elle renvoie le maximum des trois valeurs passées en argument.

```

def max(a,b,c):
    if a>b:
        if a>c:
            return a
        else:
            ...

```

```

def max(a,b,c):
    if a>b:
        if a>c:
            return a
        else:
            return c
    else:
        if b>c:
            return b
        else:
            return c

```

3. Tester et vérifier la fonction.
4. Maintenant que nous avons une fonction opérationnelle, nous allons essayer de l'optimiser en diminuant le nombre de lignes. Compléter la fonction ci-dessous pour qu'elle renvoie le maximum des trois valeurs passées en argument.

```
def max(a,b,c):
    if ... :
        return a
    elif ... :
        ...
```

```
def max(a,b,c):
    if a>b and a>c:
        return a
    elif b>c:
        return b
    else:
        return c
```

5. Tester et vérifier la fonction.
6. En vous inspirant de la fonction précédente, écrire , tester et vérifier une fonction qui renvoie le minimum des trois valeurs passées en argument.

```
def min(a,b,c):
    if a<b and a<c:
        return a
    elif b<c:
        return b
    else:
        return c
```

7. Pour être encore plus efficace, écrire une fonction qui renvoie le maximum et le minimum des trois valeurs passées en argument en utilisant une liste.

```
def extrema(a,b,c):
    d=[a,b,c]
    d.sort()
    return d[0],d[2]
```

8. Le fin du fin, écrire une fonction qui renvoie le maximum et le minimum des trois valeurs passées en argument en utilisant les commandes existantes.

```
def extrema(a,b,c):
    return min(a,b,c),max(a,b,c)
```

## Exercice n°2 : *HT ou TTC ?*

En France, le taux courant de la T.V.A. appliqué sur la majorité des biens est de 20 %.

1. Un article coûte 72 € hors taxe. Quel est son prix toutes taxes comprises (TTC) ?

$$72 \times \left(1 + \frac{20}{100}\right) = 72 \times 1,2 = 86,4 \text{ €}$$

2. La variable *pht* contient un réel positif correspondant au prix hors taxe de l'article. Parmi les instructions suivantes, lesquelles permettent de calculer le prix TTC ?

a)  $p \leftarrow 0,2 \times pht$       b)  $p \leftarrow 1,2 \times pht$       c)  $p \leftarrow pht + 20/100 \times pht$       d)  $p \leftarrow 1,02 \times pht$

Les propositions *b* et *c*.

3. On considère la fonction Python ci-contre.

- a) Déterminer ce que renvoient les instructions suivantes :  
**prix(50,False)** et **prix(86.4,True)**.

**prix(50,False)** renvoie 60 soit le prix TTC et  
**prix(86.4,True)** renvoie 72 soit le prix HT.

- b) Expliquer le rôle de cette fonction.

Si l'on connaît un prix hors taxe, la TVA n'est pas appliquée donc mention False alors la fonction renvoie le prix TTC.

Si l'on connaît un prix TTC, la TVA est déjà appliquée donc mention True alors la fonction renvoie le prix hors taxe.

```
def prix(a,b):
    if b:
        h=a/1.2
        return h
    else:
        p=a*1.2
        return p
```

## Devoir surveillé n° 18 : Fonctions sous Python

### Exercice n°1 : Quelle heure est-il ?

7 pts

2 pts

1. Compléter le programme ci-dessous afin que la variable *c* contienne la somme des carrés des variables *a* et *b* et que la variable *d* contienne le produit des inverses des variables *a* et *b*.

```
a,b=2,5
c=...
d=...
```

```
a,b=2,5
c=a*a+b*b
d=1/(a*b)
```

2. On donne la fonction **duree()** qui a pour arguments un nombre *h* d'heures, un nombre *m* de minutes et un nombre *s* de secondes.

1 pt

- a) Que renvoie **duree(2,40,6)** ?

$$2 \times 3600 + 40 \times 60 + 6 = 7200 + 2400 + 6 = 9606s$$

1 pt

- b) Quel est le rôle de cette fonction ?

Transformer une durée exprimée en heures, minutes et secondes en secondes.

1 pt

- c) Modifier cette fonction pour qu'elle renvoie en minutes une durée saisie en heures, minutes et secondes.

```
def duree(h,m,s):
    d=3600*h+60*m+s
    return(d)
```

```
def duree(h,m,s):
    d=60*h+m+s/60
    return(d)
```

2 pts

- d) Donner une fonction renvoyant en heure, minutes et secondes une durée saisie en heure décimale.

```
def duree(h):
    hh=int(h)
    m=int((h-hh)*60)
    s=((h-hh)*60-m)*60
    return(hh,m,s)
```