Exercice 1

1) On demande d'abord une durée en secondes, qui est stockée dans la variable d.
   Ensuite, on crée deux variables m et s contenant le quotient et le reste de la division de d par 60, si bien que d=60*m+s.
   Ceci permet de convertir la durée d en minutes et secondes : elle correspond ainsi à m minutes et s secondes.

2) Il suffit de reprendre la même idée que précédemment :
   - en calculant le quotient et le reste de la division de m par 60, on convertit la durée m minutes en h heures et m minutes.
   - en calculant le quotient et le reste de la division de h par 24, on convertit la durée h heures en j jours et h heures

   Le programme est alors complet, et on finit par un joli affichage.

```
d=eval(input('durée en secondes : '))

m=d//60
s=d%60

h=m//60
m=m%60

j=h//24
h=h%24

print('Cela fait',j,'jours',h,'heures',m,'minutes et',s,'secondes')
```

Exercice 2

1) Les deux `pop()` successifs après le `if` vont fausser les calculs car on ne remplace pas `m` par la valeur à laquelle on l'a comparée, comme on pourrait le croire à première vue.

   (a) Suite à la première instruction de la fonction, on a `m=2` et `L=[1,3]`.
   Comme `len(L)>0`, la boucle commence.
   Ensuite `L.pop()` renvoie 3, qui est bien supérieur à `m`. Et `L=[1]`.
   Après l'instruction `m=L.pop()`, on `m=1` et `L=[]` donc `len(L)=0`. On sort alors de la boucle et la fonction renvoie 1 ... qui n'est pas le maximum de `L`.

   (b) Suite à la première instruction, `m=2` et `L=[1]`. La boucle commence.
   Ensuite `L.pop()` renvoie 1. Or `1<m` : on ne fait donc rien. À ce moment, `L=[]` si bien que la boucle s'arrête et la fonction renvoie 2 ... qui est le maximum de `L`.

   (c) De même, la première valeur prise par `m`, 4, étant le maximum de `L`, les tests `if m<L.pop()` vont vider la liste terme par terme et la fonction renvoie 4.

   (d) Après la première instruction, `m=1` et `L=[2]`. La boucle démarre.
   Ensuite la condition `if m<L.pop()` est vérifiée, donc on va tenter d'effectuer un troisième `L.pop()` alors que la liste est déjà vide ... ce qui provoque une erreur.

2) L'usage inconsidéré de l'instruction `L.pop()` dans la boucle est la source des erreurs, car elle supprime le dernier élément de la liste sans le conserver. Il faut donc penser à le stocker dans une nouvelle variable que l'on compare ensuite à `m` sans nouveau `pop()`.

```
def maxi(L):
    m=L.pop()
    while len(L)>0:
        e=L.pop()
        if m<e:
            m=e
    return m
```

Exercice 3

Aucun souci, c'est un bête calcul de somme. Il faut juste faire attention au `range` pour que notre compteur aille bien de 1 à n ... et pas de 1 à $n-1$ ou de 0 à n (horreur !)

```
def Somme_Inverses(n):
    somme=0
    for i in range(1,n+1):
        somme=somme+1/i
    return somme
```

Exercice 4

1) Voici l'état des variables p et c au cours de l'exécution :

p	5	3	4	2	3	1	2	0
c	0	1	0	1	0	1	0	1

On rentre dans la boucle `while` avec les valeurs $p = 5$ et $c = 0$.
Quand elle s'achève, on a $p = 0$ et $c = 1$.

2) • Au cours de l'exécution du programme, on a toujours $c = 0$ ou $c = 1$ donc $c \in \mathbb{N}$.
De plus, $p \in \mathbb{Z}$ car on ne fait que lui ajouter ou retrancher des nombres entiers.
Enfin, $p \geqslant 1$ au début de chaque itération donc $p \geqslant -1$ à l'intérieur de la boucle.
De ce fait, $v = 2p + 3c \in \mathbb{Z}$ et $v \geqslant -2$: ainsi, v est une variable entière et minorée.

• Notons v, p, c les valeurs de nos variables au début d'une itération et v', p', c' leurs valeurs à la fin de cette itération.

– Si $c = 0$, alors $p' = p - 2$ et $c' = 1 = c + 1$ donc
$$v' = 2p' + 3c' = 2p - 4 + 3c + 3 = 2p + 3c - 1 = v - 1$$

– Si $c = 1$, alors $p' = p + 1$ et $c' = 0 = c - 1$ donc
$$v' = 2p' + 3c' = 2p + 2 + 3c - 3 = 2p + 3c - 1 = v - 1$$

Dans tous les cas, $v' = v - 1$ donc v est strictement décroissante.

Ainsi, la variable v est entière, strictement décroissante et minorée : c'est donc un variant de boucle. Elle nous assure que le programme se termine effectivement.

Exercice 5

1) La création d'une liste nulle U de bonne taille va nous permettre d'appliquer textuellement la relation de récurrence.

```
def SuiteU(N):
    U=(N+1)*[0]
    U[0]=1
    for i in range(N):
        U[i+1]=U[i]/2+1/U[i]
    return U
```

2) Pour représenter les termes d'une liste `L` en fonction de leurs indices, il suffit d'un simple `plot(L)`. Voici alors le code pour représenter les 20 premiers termes de notre suite de Héron :

```
# On n'oublie pas d'importer la bibliothèque
import matplotlib.pyplot as pl
# On récupère la liste [u0,...,u19]
U=SuiteU(19)
pl.plot(U)
pl.title("Vingt premiers termes de la suite de Héron")
pl.show()
```

3) Il faut évidemment appeler la fonction `SuiteU` pour récupérer la liste $[u_0, u_1, \dots, u_n]$, qui n'existe plus en dehors de cette fonction.

```
def MoyenneU(n):
    # On récupère la liste [u0,...,un]
    U=SuiteU(n)
    # On calcule la somme des termes
    S=0
    for e in U:
        S=S+e
    return S/(n+1)
```

Exercice 6

1) Comme on l'a vu en cours, il faut créer une variable, initialement égale à 1, que l'on multiplie successivement par tous les entiers de 1 à 99.

$p = 1$
POUR i ALLANT DE 1 A 99
    $p = p \times i$
Afficher p

2) Voici ce que cela donne en `Python` :

```
p=1
for i in range(1,100):
    p=p*i
print("Le produit vaut",p)
```

Attention avec l'instruction `range(a,b)` : les valeurs prises vont de `a` à `b-1`.

Exercice 7

1) On n'oublie pas que l'échange du contenu de deux variables se fait à l'aide d'une égalité de couples, pour ne pas effacer des données.

```
def reindexation(f,xM,xN):
    if f(xM)>f(xN):
        (xM,xN)=(xN,xM)
    return (xM,xN)
```

2) Par hypothèse, il est inutile de réindexer le segment [MN] dans cette fonction.

```
def iteration(f,xM,xN):
    xR=2*xM-xN
    if f(xR)<f(xM):
        xN=xR
    else:
        xC1=(xM+xN)/2
        xC2=(xM+xR)/2
        if f(xC1)<f(xM):
            xN=xC1
        else:
            xN=xC2
    return xM,xN
```

3) Pas de difficulté ici, on calcule simplement la quantité proposée.

```
def ecart(xM,xN):
    return 2*abs(xM-xN)/(abs(xM)+abs(xN))
```

4) Attention ici, après le `while` on écrit la négation de la condition d'arrêt

« l'écart est inférieur à `eps` ou bien le nombre d'itérations atteint `Itmax` »

(avec des inégalités larges) ce qui donne

« l'écart est supérieur à `eps` et le nombre d'itérations est inférieur à `Itmax` »

avec des inégalités strictes.

```
def boucle(f,a,b,Itmax,eps):
    xM,xN=(a,b)
    nb_It=0
    while nb_It<Itmax and ecart(xM,xN)>eps:
        (xM,xN)=reindexation(f,xM,xN)
        (xM,xN)=iteration(f,xM,xN)
        nb_It=nb_It+1
    return (xM,xN)
```