

Exercice 1

- 1) (a) $\overline{11011}^2 = 2^4 + 2^3 + 2^1 + 2^0 = 16 + 8 + 2 + 1 = \boxed{27}$
- (b) $\overline{123}^8 = 8^2 + 2 \times 8^1 + 3 \times 8^0 = 64 + 16 + 3 = \boxed{83}$
- (c) $\overline{FA}^{16} = 15 \times 16^1 + 10 \times 16^0 = 240 + 10 = \boxed{250}$
- 2) Avec des divisions euclidiennes successives (ou bien une décomposition en sommes de puissances de la base), on arrive sans problème aux résultats suivants :
- (a) $75 = 64 + 8 + 2 + 1 = 2^6 + 2^3 + 2^1 + 2^0$ s'écrit ainsi $\overline{1001011}^2$ en base 2.
- (b) $75 = 64 + 8 + 3 = 8^2 + 8^1 + 3 \times 8^0$ s'écrit ainsi $\overline{113}^8$ en base 8.
- (c) $75 = 64 + 11 = 4 \times 16^1 + 11 \times 16^0$ s'écrit ainsi $\overline{4B}^{16}$ en base 16.
- 3) (a) Groupons les bits par paquets de 4 :
- $\overline{1010}^2$ donne $8 + 2 = 10$ soit A en base 16.
 - $\overline{11}^2$ donne $2 + 1 = 3$.
- Ainsi, l'entier représenté par $\overline{111010}^2$ s'écrit $\overline{3A}^{16}$ en base 16.
- (b) Là, c'est l'opération inverse :
- $\overline{F}^{16}$ donne 15 soit $\overline{1111}^2$ en base 2.
 - $\overline{D}^{16}$ donne 13 soit $\overline{1101}^2$ en base 2.
 - $\overline{E}^{16}$ donne 14 soit $\overline{1110}^2$ en base 2.
- Ainsi, l'entier représenté par $\overline{EDF}^{16}$ s'écrit $\overline{111011011111}^2$ en base 2.

Exercice 2

- 1) Le seuls codes corrects sont le code B et le code C.

A) Toutes les variables reçoivent la valeur initiale de a, et la valeur de b est effacée.

Voici en effet les contenus des variables a, b et c à chaque étape :

a	b	c
a	b	None
a	b	a
a	a	a
a	a	a

B) C'est un code vu en cours, donc il est forcément correct.

Pour ceux qui auraient le mauvais goût d'en douter, voilà la preuve :

a	b	c
a	b	None
a	b	a
b	b	a
b	a	a

- C) Ce code est peut-être un peu tordu, mais il fonctionne quand même.

En effet :

a	b
a	b
a+b	b
a+b	a
b	a

- D) On a repris le code précédent en modifiant la dernière instruction. Du coup, ça ne marche plus !

En effet :

a	b
a	b
a+b	b
a+b	a
2a+b	a

- 2) On a vu en cours que le plus simple est de taper le code suivant : $a, b = b, a$

Exercice 3

- 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 : '))

# Minutes et Secondes
m=d//60
s=d%60

# Heures et minutes
h=m//60
m=m%60

# Jours et heures
j=h//24
h=h%24

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

Exercice 4

- 1) (a) Comme $x = 10 > 1$, on effectue le bloc d'instructions après ALORS.
On incrémente un entier N jusqu'à ce que $10^N \geq 10^6$, puis on l'affiche.
Autrement dit, on renvoie le premier entier N tel que $10^N \geq 10^6$.

Ainsi,

L'algorithme affiche 6.

- (b) Là, $x \leq 1$ donc on effectue le bloc d'instructions après SINON.

Ainsi,

L'algorithme affiche "x doit dépasser 1".

- (c) Comme $x = 20 > 1$, on effectue le bloc d'instructions après ALORS comme dans le premier cas, qui renvoie le premier entier N tel que $20^N \geq 10^6$, soit $N = 5$.

Ainsi,

L'algorithme affiche 5.

- 2) Cet algorithme affiche le plus petit entier naturel N tel que $x^N \geq 10^6$ si $x > 1$, et renvoie le message "x doit dépasser 1" sinon.

Il permet tout simplement de résoudre l'inéquation $x^N \geq 10^6$.

- 3) Voilà le travail :

```
x=eval(input("Donnez un réel x>1 : "))
if x>1:
    N=0
    while x**N<10**6:
        N=N+1
    print(N)
else:
    print("x doit dépasser 1")
```

Exercice 5

Il vaut mieux commencer par attribuer une valeur négative à la variable dans laquelle on va stocker le nombre testé. Ensuite, une boucle while permet de réitérer les demandes.

```
from math import log
x=0
while x<=0:
    x=eval(input("Entrez un réel strictement positif : "))
print(log(x))
```

Exercice 6

- 1) Il manque une parenthèse fermante à la fin de la première ligne.

Ceci provoque l'erreur `SyntaxError: unexpected EOF while parsing`

De plus, la variable `total` n'a pas été créée (via une affectation) avant d'être utilisée la première fois dans le calcul à l'avant-dernière ligne.

Ceci provoque l'erreur `NameError: name 'total' is not defined`

Pour y remédier, il faut affecter la variable `total` avant l'entrée dans la boucle.

```
n=int(input())
total=0
for i in range(n):
    if i%2==0:
        carre=i**2
    else:
        carre=0
    total=total+carre
print(total)
```

- 2) Ce programme demande un entier n à l'utilisateur, puis il affiche la somme des carrés des entiers pairs compris entre 0 et $n - 1$ (donc 0 si $n < 1$).

Exercice 7

- 1) Les deux `L.pop()` successifs après le `if` vont fausser les calculs car on ne va pas stocker dans la variable `m` la valeur à laquelle on l'a comparée, en cas de modification.

- (a) Suite à la première instruction du programme, 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]`.

On exécute donc l'instruction `m=L.pop()` : ainsi, `m=1` et `L=[]` d'où `len(L)=0`.

On sort alors de la boucle et le programme affiche 1 ... qui n'est pas le maximum.

- (b) La première valeur prise par `m` étant le maximum de `L`, les tests `if m<L.pop()` vont simplement vider la liste terme par terme. Le programme affiche donc 2.

- (c) De la même façon, le programme affiche 4.

- (d) Problème : comme la condition `if m<L.pop()` est vérifiée, 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`.

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

Exercice 8

- 1) On utilise évidemment une liste, c'est la structure adaptée au problème.
- 2) On crée une liste initialement vide. On lui ajoute ensuite les résultats successivement rentrés à l'aide d'une boucle, qui est interrompue lorsque l'on saisit 6.

```
Lancers=[]
res=0
while res!=6:
    res=eval(input("Résultat : "))
    Lancers.append(res)
print(Lancers)
```

- 3) L'algorithme le plus rapide consiste à parcourir la liste Lancers et à incrémenter un des termes de la liste [N1,N2,N3,N4,N5] selon la valeur de l'élément ... cette astuce permet d'éviter d'écrire 5 tests.

```
# On vire d'abord le 6
Lancers.pop()
# On initialise les effectifs
Liste=[0,0,0,0,0]
# On parcourt la liste Lancers
for i in Lancers:
    Liste[i-1]=Liste[i-1]+1
print(Liste)
```

Beaucoup plus simple mais moins efficace : on utilise deux boucles imbriquées pour comptabiliser les 1, puis les 2, les 3, etc...

```
Liste=[]
for i in (1,2,3,4,5):
    N=0
    for res in Lancers:
        if res==i:
            N=N+1
    Liste.append(N)
print(Liste)
```