

Exercice 1

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

c	3	2	1	0
p	1	2	4	8

Quand on tape $f(3)$, on obtient donc 8.

- 2) Si l'on tape $f(-4)$, la variable c va d'abord prendre la valeur -4 puis diminuer de 1 à chaque itération. Comme elle reste ainsi strictement négative, elle n'atteindra jamais la valeur 0 si bien que la boucle `while` ne s'arrête pas.

On obtient donc une boucle infinie lorsque l'on tape $f(-4)$... avec un carré rouge dans la console, et si tout se passe bien un blocage de la machine qui nous oblige à redémarrer le noyau !

- 3) Il y a plusieurs façons de procéder. La plus simple consiste à utiliser la fonction f de l'énoncé, qui fait le travail pour les entiers positifs, pour définir une nouvelle fonction qui sait aussi gérer les entiers négatifs.

```
def g(n):
    if n >= 0:
        return f(n)
    else:
        return 1/f(-n)
```

Exercice 2

- 1) Soit n la longueur de ch : cette chaîne $ch = '0.a_1a_2 \dots a_{n-2}'$ représente le nombre réel

$$x = \sum_{i=1}^{n-2} a_i 2^{-i}$$

Notons que a_1 correspond à $ch[2]$, a_2 correspond à $ch[3]$, ..., a_{n-2} correspond à $ch[n-1]$. De manière générale, a_i est la valeur de $ch[i+1]$, ce qui conduit au code suivant :

```
x=0
n=len(ch)
for i in range(1,n-1):
    x=x+eval(ch[i+1])*2**(-i)
print(x)
```

- 2) On applique l'algorithme vu en cours : on multiplie x par 2 puis on lui retire sa partie entière, qui donne le premier chiffre après la virgule de sa représentation binaire. On réitère le procédé tant que $x > 0$, en ajoutant les chiffres obtenus les uns après les autres à droite d'une chaîne initialement égale à $'0.'$.

```
ch='0.'
while x>0:
    n=int(2*x)
    x=2*x-n
    ch=ch+str(n)
print(ch)
```

Exercice 3

1) Voici la procédure demandée :

```
def Tableur(f,a,b,n):
    pas=(b-a)/(n-1)
    X=[a+k*pas for k in range(n)]
    Y=[f(x) for x in X]
    return Y

# Variante avec numpy
X=numpy.linspace(a,b,n)
```

2) Et voilà le travail :

```
def f(x):
    return 2*x**3+x-5

print(Tableur(f,0,1,11))
```

Exercice 4

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 5

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, $p \geq 1$ au début de chaque itération donc $p \geq -1$ à l'intérieur de la boucle.

De ce fait, $v = 2p + 3c \in \mathbb{Z}$ et $v \geq -2$: 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 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 × 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) La variable `i` est un variant de boucle : en effet,

- ❶ elle est à valeurs entières car elle démarre à 100 et diminue uniquement de 1
- ❷ elle est strictement décroissante puisqu'elle baisse de 1 à chaque étape
- ❸ elle est minorée par 0 tout au long du déroulement de la fonction.

Ceci nous assure de la terminaison de l'appel `fonction()`.

2) La boucle se déroule jusqu'à ce que la variable `i` atteigne la valeur 0.

Il y a donc au total N itérations comprenant chacune un nombre fini et fixé d'opérations, ce qui nous donne une complexité en $\mathcal{O}(N)$.

Exercice 8

1) • Avant l'entrée dans la boucle, on a $i = 1 = 2^0$ donc la proposition $\mathcal{P}(0)$ est vraie.

• Supposons que $\mathcal{P}(k)$ est vraie pour un entier $k \in \mathbb{N}$.

On a donc $i = 2^k$ à l'issue du k^{e} passage dans la boucle.

Si l'on effectue un passage supplémentaire – le $(k+1)^{\text{e}}$ – la variable i est multipliée par 2 donc $i = 2 \times 2^k = 2^{k+1}$, si bien que la proposition $\mathcal{P}(k+1)$ est vraie.

• Ceci montre que la proposition $\mathcal{P}$ est bien un invariant de boucle.

2) De ce fait, elle est encore vraie en sortie de boucle : à ce moment-là, on a donc $i = 2^k$ avec $k \in \mathbb{N}$ et $i \geq n$ pour la première fois.

On en déduit que cette fonction renvoie la plus petite puissance de 2 supérieure ou égale à un entier $n \in \mathbb{N}$.

Exercice 9

1) La variable i est un variant de boucle : en effet,

- ❶ elle est à valeurs entières puisqu'elle démarre à 0 et augmente uniquement de 1
- ❷ elle est strictement croissante puisqu'elle augmente de 1 à chaque étape
- ❸ elle est majorée par $n - 1$ tout au long du déroulement de la fonction.

Ceci nous assure de la terminaison de l'appel fonction(L).

2) *Examinons ce qui se passe dans le pire des cas.*

La boucle se déroule jusqu'à ce que la variable i atteigne la valeur $n - 1$.

Il y a au total $n - 1$ itérations comprenant chacune un nombre fini et fixé d'opérations, ce qui nous donne une complexité en $\mathcal{O}(n)$.

3) • Avant l'entrée dans la boucle, on a $i = 0$ et la proposition $\mathcal{P}(0)$ est vraie.

• Supposons que $\mathcal{P}(i)$ est vraie pour un entier $i \in \mathbb{N}$.

On a donc $L[0] < L[1] < \dots < L[i]$ à l'issue du i^{e} passage dans la boucle.

Si l'on effectue un passage supplémentaire – le $(i + 1)^{\text{e}}$ – on a alors $i + 1 \leq n - 1$ donc l'élément $L[i+1]$ existe bien.

De plus $L[i] < L[i+1]$, si bien que $L[0] < L[1] < \dots < L[i+1]$.

De ce fait, la proposition $\mathcal{P}(i + 1)$ est vraie.

• Ceci montre que la proposition $\mathcal{P}$ est bien un invariant de boucle.

4) Elle est donc encore vraie en sortie de boucle, ce qui nous permet d'affirmer que notre fonction renvoie le plus grand entier $i \in \llbracket 0 ; n - 1 \rrbracket$ tel que $L[0] < L[1] < \dots < L[i]$.

Exercice 10

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) Ooooooh ! C'est un des algorithmes du chapitre X ! Il suffisait d'apprendre le cours ...

```
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)
```