

# Correction des exercices

1

Extrait de Centrale 2015

- (a) [1, 2, 3, 4, 5, 6]  
(b) [1, 2, 3, 1, 2, 3]
- 

```
def smul(x, L):
    """renvoie la liste [ x*L[0], ... , x*L[-1] ]"""
    L2 = []
    for i in range(len(L)):
        L2.append( x * L[i] )
    return L2
```

ou bien, en compréhension (à ne pas faire ici) :

```
def smul(x, L):
    """renvoie la liste [ x*L[0], ... , x*L[-1] ]"""
    return [x * elem for elem in L]
```

3. Arithmétique des listes

(a)

```
def vsom(L1, L2):
    """renvoie la liste des sommes des éléments de L1 et L2"""
    assert len(L1) == len(L2), "Les listes n'ont pas la même longueur"
    L = []
    for i in range(len(L1)):
        L.append(L1[i] + L2[i])
    return L
```

(b)

```
def vdif(L1, L2):
    """renvoie la liste des différences des éléments de L1 et L2"""
    assert len(L1) == len(L2), "Les listes n'ont pas la même longueur"
    L = []
    for i in range(len(L1)):
        L.append(L1[i] - L2[i])
    return L
```

2

Extrait de Maths CCP 2015

- 21 = 16 + 4 + 1 s'écrit en binaire 10101.

$k$	1	2	3
$c_k$	6	5	2
$t_k$	[6]	[6,5]	[6,5,2]
$n_k$	25	2	0

- L'exécution de `mystere(256, 10)` renvoie [6, 5, 2].

3. (Avec  $b > 1$ )

**Première Solution :** On montre que l'on a  $n_{k+1} = \lfloor \frac{n_k}{10} \rfloor$  et par récurrence  $n_k = \lfloor \frac{n}{10^k} \rfloor$  : c'est vrai pour  $k = 0$  et si c'est vrai pour un  $k \geq 0$ ,

$$n_{k+1} \leq \frac{n_k}{10} \leq \frac{n}{10^{k+1}} < \frac{n_k}{10} + \frac{1}{10} < \underbrace{\frac{n_k}{10} + 1}_{\in \mathbb{Z}} + \underbrace{\frac{1}{10}}_{\in [0,1]}$$

donc  $n_{k+1} = \lfloor \frac{n}{10^{k+1}} \rfloor$ , ce qui établit la récurrence et donne directement la terminaison et le nombre de tours de boucle.

**Deuxième solution (plus simple) :**

- On a  $n_{k+1} = \lfloor \frac{n_k}{10} \rfloor$  donc lorsque  $n_k \neq 0$ ,  $n_{k+1} < n_k$  :  $(n_k)$  forme une suite entière naturelle strictement décroissante donc finie : **la boucle termine**.
- Par récurrence (à poser), on montre que  $n_k \leq \frac{n}{10^k}$ . Ainsi,  $n_k = 0$  lorsque  $\frac{n}{10^k} < 1$ , c'est-à-dire lorsque  $k > \log n$ . Donc  **$p \leq \log n + 1$**  (et même  $p = \lfloor \log n \rfloor + 1$ ).

4.

```
def somme_chiffres(n):
    """renvoie la somme des chiffres de n."""
    somme = 0
    while n > 0:
        somme += n % 10
        n //= 10
    return somme
```

À ne pas faire ici :

```
def somme_chiffres_python(n):
    return sum( int(car) for car in str(n) )
```

5.

```
def somme_rec(n):
    """fonction récursive renvoyant la somme des chiffres de n."""
    if n == 0:
        return 0
    return n % 10 + somme_chiffres(n // 10)
```

3

Extrait de Maths CCP 2016

1.

```
def gcd(a, b):
    """renvoie le pgcd de a et b par une méthode naïve"""
    diviseur = 1
    for i in range(2, min(a, b) + 1):
        if a % i == b % i == 0:
            diviseur = i
    return diviseur
```

2.

```
def euclide_rec(a, b):
    """Version récursive de l'algorithme d'Euclide"""
    if b == 0:
        return a
    else:
        return euclide_rec(b, a % b)
```

3.a Les divisions euclidiennes sont :  $8 = 5 \times 1 + 3$   $5 = 3 \times 1 + 2$   $3 = 2 \times 1 + 1$   $2 = 1 \times 2 + 0$ .

3.b  $F_{n+2} = F_{n+1} + F_n$  avec  $F_n < F_{n+1}$  si  $n \geq 2$ .

Donc **le reste de la division euclidienne de  $F_{n+2}$  par  $F_{n+1}$  est  $F_n$ .**

Il faudra donc  $n - 1$  divisions euclidiennes pour arriver à  $F_4 = 3 = F_3 + F_2 = 2 + 1$  et donc

$n$  divisions euclidiennes dans le calcul du pgcd de  $F_{n+2}$  et  $F_{n+1}$  par l'algorithme d'Euclide.

3.c On a donc  $u_n = n$  et  $v_n = 2(F_{n+1} - 1)$  car pour tous les entiers entre 2 et  $F_{n+1}$ , il y a deux restes de division euclidienne calculés. Comme  $v_n \sim \frac{2}{\sqrt{5}}\varphi^n$  avec  $\varphi > 1$ ,  $u_n = o(v_n)$ .

4

```
def fibo(n):
    "renvoie le nombre de Fibonacci F_n"
    f, f1 = 0, 1
    for i in range(n):
        # ici, f contient F_i et f1 contient F_(i+1)
        f, f1 = f1, f + f1
    return f
```

5

```
def gcd_trois(a, b, c):
    "renvoie le pgcd de a, b, c"
    return euclide(a, euclide(b, c))
```

4

Extrait de Maths CCP 2017

1.  $\text{len}(A)$ ,  $A[1]$  et  $A[2][1]$  renvoient respectivement 4, [4, 5, 6] et 8.

2.

```
def difference(x, y):
    "renvoie le vecteur x - y"
    n = len(x)
    assert n == len(y), "tailles incompatibles"
    vecteur = []
    for i in range(n):
        vecteur.append(x[i] - y[i])
    return vecteur
```

3.

```
def norme(x):
    "renvoie la norme infinie du vecteur x"
    maxi = 0
    for xi in x:
        if abs(xi) > maxi:
            maxi = abs(xi)
    return maxi
```

4.

```
def itere(x, A):
    "renvoie la norme infinie du vecteur x"
    n = len(x)
    assert n == len(A) == len(A[0]), "tailles incompatibles"
    vecteur = []
    for j in range(n):
        somme = 0
        for i in range(n):
            somme += x[i] * A[i][j]
        vecteur.append(somme)
    return vecteur
```

5.

```
def probaInvariante(A, epsilon):
    p = len(A)
    mu = [1 / p for _ in range(p)]
    mul = itere(mu, A)
    while norme(difference(mul, mu)) > epsilon:
        # au début du k_e tour de boucle,
        # mu contient mu_(k - 1) et mul contient mu_k
        mu, mul = mul, itere(mul, A)
    return mul
```

Notons que l'exemple proposé dans l'énoncé renvoie le premier  $k$  tel que  $\|\mu_k - \mu_{k+1}\|_\infty \leq \varepsilon$  ce qui n'est pas ce qui était demandé.

5

Extrait de Maths CCP 2018

1. (a)

```
def factorielle(n):
    if n == 0:
        return 1
    else:
        return n * factorielle(n - 1)
```

(b)

```
N = 1
while (2 * N + 3) * factorielle(N + 1) < 10 ** 6:
    N += 1
```

2. (a)

```
def L(i, x, a):
    """renvoie l'évaluation en a du i-ème polynôme de
    Lagrange associé aux x[k]."""
    P = 1
    for k in range(len(x)):
        if k != i:
            P *= (a - x[k]) / (x[i] - x[k])
    return P

def lagrange(x, y, a):
    """renvoie l'évaluation en a du polynôme
    d'interpolation de Lagrange associés aux points (x[k], y[k])."""
    s = 0
    for i in range(len(x)):
        s += L(i, x, a) * y[i]
    return s
```

(b) La matrice cherchée est une matrice de Vandermonde :

$$V = \begin{pmatrix} 1 & x_0 & \dots & x_0^n \\ 1 & x_1 & \dots & x_1^n \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \dots & x_n^n \end{pmatrix}$$

La complexité d'une résolution par pivot de Gauss est  $O(n^3)$ .