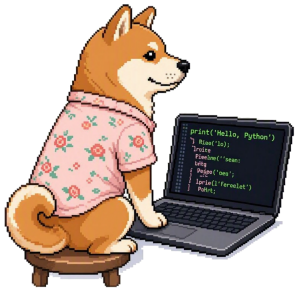
Informatique PT. 2026 - 2027

Mathématiques & Informatique - F. Gaunard
http://frederic.gaunard.com
Lycée Voltaire, Paris 11e.

1 Solution : Révisions d'algorithmique



Pour ce premier TP de rentrée, on propose des exercices, principalement issus de l'oral **Math II** qui invitent à manipuler des listes ou des tableaux de manière assez élémentaire.
On rappelle qu'un exercice de ce type doit être, le jour de l'oral, traité en moins de 30 minutes.

Exercice 1. *Nombres en binaire concaténés*, Oral Math II 2026

1.
```
def binaire(n):
    if n == 0:
        return '0'
    chaine = ''
    while n > 0:
        chaine = str(n % 2) + chaine
        n = n // 2
    return chaine
```

2.
```
def U(n):
    u = '0'
    for k in range(n):
        u += binaire(k)
    return u
```

3. Un entier $n \geq 1$ ayant k chiffres en binaire vérifie : $2^{k-1} \leq n < 2^k$. on a donc $k-1 \leq \log_2(n) < k$. Or, k est entier donc $\lfloor \log_2(n) \rfloor = k-1$. Donc, la longueur du terme U_n vaut :

$$|U_n| = 1 + \sum_{k=1}^{n} (\lfloor \log_2(k) \rfloor + 1) = n + 1 + \sum_{k=1}^{n} \lfloor \log_2(k) \rfloor.$$

4.
```
def occurrences_11(n):
    u = U(n)
    compteur = 0
    for i in range(len(u) - 1):
        if u[i:i+2] == '11':
            compteur += 1
    return compteur
```

5. La construction de U_n nécessite de stocker l'ensemble de la chaîne. La complexité en mémoire est donc $\mathcal{O}(n \log n)$. Il n'est cependant pas nécessaire de construire toute la chaîne pour compter les occurrences de `'11'`. On peut effectuer le comptage au fur et à mesure de la construction, en ne conservant que le dernier caractère déjà obtenu.

```
def occurrences_11(n):
    precedent = '0'
    compteur = 0
    for k in range(n):
        b = binaire(k)
        for c in b:
            if precedent == '1' and c == '1':
                compteur += 1
            precedent = c
    return compteur
```

Cette version évite de stocker U_n et permet donc de réduire la mémoire utilisée à $\mathcal{O}(\log n)$ en ne comptant que la mémoire nécessaire à la représentation binaire d'un entier.

Exercice 2. *Tri*, Oral Math II 2024

1.
```
def parcours(L):
    n = len(L)
    nb = 0
    for k in range(n-1):
        if L[k] > L[k+1]:
            L[k], L[k+1] = L[k+1], L[k]
            nb += 1
    return nb
```

2. À la fin d'un parcours, le dernier élément est un élément maximal de L.

3.
```
def tri1(L):
    nb = 1 # Nombre d'itérations égal à un,
       # car il en faut au moins une pour l'initialisation
    while parcours(L) != 0:
        nb += 1
    return nb
```

4. Le nombre de permutations décroit strictement à chaque itération; en effet lors d'une itération un élément maximal est placé en fin de liste et ne subit aucune permutation lors des itérations suivantes.
Si la liste est triée, aucune permutation n'est effectuée et la liste est parcourue une seule fois, ce qui donne une complexité linéaire. Si elle triée dans le sens décroissant, alors : il faut $n-1$ permutations, puis $n-2$ permutation, etc... et la liste est parcourue n^2 fois, ce qui conduit à une complexité quadratique.

5.
```
def tri2(L):
    n = len(L)
    while parcours(L) != 0 and n > 0:
        parcours(L[0:n])
        n -= 1
```

`tri2` améliore `tri1`, puisque à chaque étape elle opère sur une liste de longueur strictement plus petite.

Exercice 3. *Trains*, Oral Math II 2025, 2026

1.
```
def voeux(voyageurs, rt):
    train = [ [ ] for i in range(rt)]
    for voyageur in range(len(voyageurs)):
        j = voyageurs[voyageur]
        train[j].append(voyageur)
    return train
```

2.
```
def possible1(voyageurs, capacite):
    train = voeux(voyageurs, len(capacite))
    for i in range(len(capacite)):
        if len(train[i]) > capacite[i]:
            return False
    return True
```

3.
```
def possible2(voyageurs, capacite):
    train = voeux(voyageurs, len(capacite))
    surplus = 0
    for i in range(len(capacite)):
        nb = len(train[i]) + surplus
        if nb > capacite[i]:
            surplus = nb - capacite[i]
        else:
            surplus = 0
    return surplus == 0
```

4.
```
def affecte(voyageurs, capacite):
    train = voeux(voyageurs, len(capacite))
    affectation = [ [ ] for i in range(len(capacite))]
    surplus = [ ]
    for i in range(len(capacite)):
        candidats = surplus + train[i]
        affectation[i] = candidats[:capacite[i]]
        surplus = candidats[capacite[i]:]
    if len(surplus) > 0:
        return None
    return affectation
```

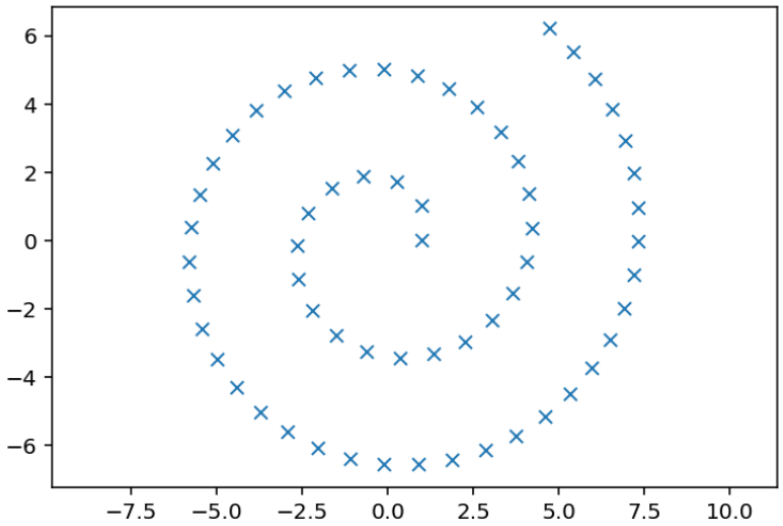
Exercice 4. *Suite de points et nombre de tours*, Oral Math II 2019

noindent On commence par importer les packages dont on aura besoin

```
import matplotlib.pyplot as plt
import numpy as np
```

1.
```
def A(N):
    L = [ ]
    M = [1,0]
    for i in range(N+1):
        L.append(M)
        x = M[0]-(M[1]/(np.sqrt(M[0]**2+M[1]**2)))
        y = M[1]+(M[0]/(np.sqrt(M[0]**2+M[1]**2)))
        M = [x, y]
    return L
```

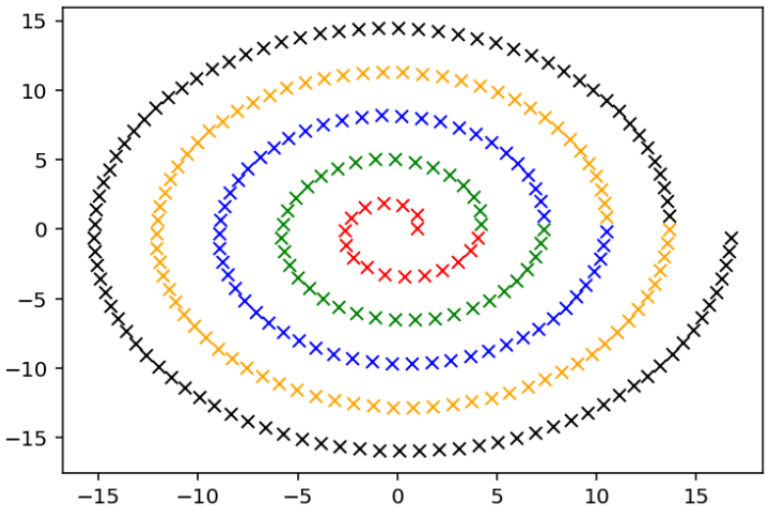
2.
```
def afficher(N):
    Res = A(N) # liste des An
    X = [p[0] for p in Res] # abscisses
    Y = [p[1] for p in Res] # ordonées
    plt.plot(X,Y,'x')
    plt.axis('equal')
    plt.show()
```
Pour $N = 60$, on a la figure ci-dessous



3. D'abord une fonction auxiliaire qui calcule le point A_{n+1} à partir de A_n
```
def aux(M):
    x = M[0]-(M[1]/(np.sqrt(M[0]**2+M[1]**2)))
    y = M[1]+(M[0]/(np.sqrt(M[0]**2+M[1]**2)))
    return [x, y]

def tours(m):
    T = []
    n = 0
    M = [1,0]
    for i in range(m):
        T.append(n)
        while M[1] >= 0: # itérer tant qu'on est dans le demi-plan supérieur,
            n += 1
            M = aux(M)
        while M[1] < 0: # itérer tant qu'on est dans le demi-plan inférieur
            n += 1
            M = aux(M)
    return T
```

4.
```
L = tours(6) # la liste des indices des points
             # qui commencent les 6 premiers tours
N = L[-1] # indice du point commençant le 6eme tour
points = A(N) # liste des An des cinq premiers tours (et début du 6e)

# partition des points en fonction du nombre de tours effectués
partition = [[points[i] for i in range(L[k],L[k+1])] for k in range(5)]

couleurs =['red','green','blue','orange','black']
```

```
i = 0
for tour in partition:
    abscisses = [p[0] for p in tour]
    ordonnees = [p[1] for p in tour]
    plt.plot(abscisses,ordonnees,'x',color = couleurs[i])
    i += 1
plt.show()
```



5.
```
L = tours(11)
N = L[-1]
points = A(N) # liste de tous les points ayant effectué N tours
for i in range(len(L)-1):
    courant = points[L[i]][0] # abscisse du point commençant le tour i
    suivant = points[L[i+1]][0] # abscisse du point commençant le tour i+1
    distance = suivant-courant
    print(distance)
# effectivement ces distances ont proches de pi
```