Bilan de l'année

1 Binaire et représentation des nombres

1.1 Les entiers

Vous devez savoir calculer à la main ou en python la représentation binaire d'un entier

```
bin(412)
```

```
'0b110011100'
```

```
def binaire(n):
    """
    Retourne la liste des bits de n, le bit de poids faible en première position
    """
```

1.2 Les flottants

```
0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 +0.1 + 0.1 + 0.1 + 0.1
```

```
0.9999999999999999
```

```
10 * 0.1
```

```
1.0
```

1.3 Encodage des données

Dans n bits, on peut encoder exactement 2^n données différentes, ou encore tous les entiers de 0 à $2^n - 1$

2 Types de données

2.1 Les types simples

```
type(True)
```

```
bool
```

```
type(12)
```

```
int
```

```
type(3.14159)
```

```
float
```

```
type(4 + 2j)
```

```
complex
```

2.2 Les conteneurs

```
s = "une chaine"
L = [1, 2, True]
t = (1, 4, s)  # un tuple
```

```
s[0], L[1:3]
```

```
('u', [2, True])
```

```
for i in range(len(s)):
    c = s[i]  # c est un caractere
```

3 Les algorithmes classiques

3.1 Accumulateurs

Ils permettent de calculer des sommes, des produits, de compter (les occurences, par exemple)

```
def nombre_diviseurs(n):
    nb_div = 0
    for i in range(1, n + 1):
        if n % i == 0:
            nb_div += 1
    return nb_div

nombre_diviseurs(24)
```

```
8
```

3.2 Construction de listes et de chaînes

```
L = []
for i in range(1, 12):
    L.append(i**3)

# a le même effet que
L = [i**3 for i in range(1, 12)]
```

```
s = ""
for i in range(97, 123):  # les numéros des caractères
    s = s + chr(i)  # chr(i) est un caractere
s
```

```
'abcdefghijklmnopqrstuvwxyz'
```

3.3 Recherches simples

Présence d'un élément dans une liste, minimum, maximum

3.4 Algorithmes plus complexes

Vous devez être capable d'expliquer, voire de re-coder les algorithmes suivants :

3.4.1 Recherche d'un mot dans une chaîne de caractère

```
def cherche_motif(chaine, motif):
    nb = 0
    long = len(motif)
    for i in range(len(chaine) - long + 1):
        if chaine[i: i + long] == motif:
            nb += 1
    return nb

cherche_motif("abaaabaa", "aa")
```

```
3
```

3.4.2 Recherche par dichotomie dans une liste triée

3.4.3 Résolution d'un système par pivot de Gauss

Savoir expliquer dans le cas le plus simple (on ne cherche pas de pivot, on suppose à chaque étape qu'il se situe sur la diagonale)

4 Algorithmes numériques

4.1 Résolution d'équations numériques

Pour résoudre $f(x) = 0$ d'inconnue x, vous devez savoir mettre en oeuvre une recherche par dichotomie, ou la méthode de Newton.

4.2 Intégrales

Savoir retrouver et implémenter les formules pour les méthodes des rectangles et des trapèzes dans deux cadres : on donne une fonction, un intervalle et un pas, ou alors on donne une liste d'abscisses et une liste d'ordonnées représentant la courbe d'une fonction.

4.3 Dérivation : méthode d'Euler

Savoir écrire et utiliser l'approximation $f'(x) \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$

Par exemple, pour résoudre l'équation différentielle $y'(t) + \ln(1 + y(t)) = t$, on écrit $y'(t) = -\ln(1 + y(t)) + t$ et on veut les valeurs de y au temps $t_0 = 0, t_1 = \frac{1}{100}, \frac{2}{100}, \ldots, 2$.

A chaque instant t_i, on a $y'(t_i) = -\ln(1 + y(t_i)) + t_i$ et de plus $y(t_{i+1}) = y(t_i) + y'(t_i)(t_{i+1} - t_i)$.

Pour la résolution numérique, on prend $y(0) = 1$.

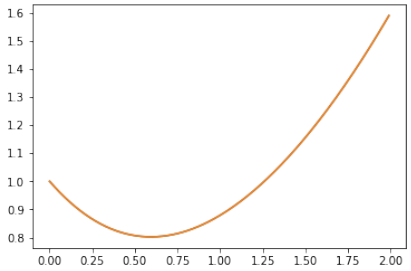
```
import math

def valeurs_y():
    h = 1/100  # le pas de temps
    t = 0
    y = 1  # valeur de y au temps t
    Y = [y]  # liste des valeurs de y calculées
    T = [t]  # liste des temps, utile pour le tracé
    for i in range(1, 200):
        y = y + (-math.log(1 + y) + t) * h
        t = t + h  # la différence des temps est h
        T.append(t)
        Y.append(y)
    return T, Y
```

```
T, Y = valeurs_y()
```

```
import matplotlib.pyplot as plt
```

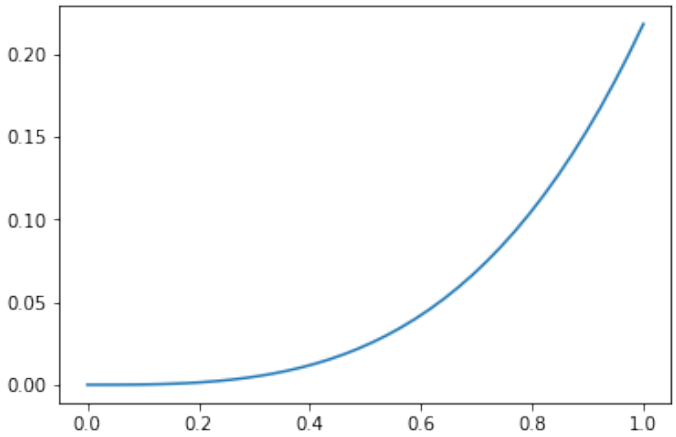
```
plt.plot(T, Y)
plt.show()
```



4.4 Utilisation basique de numpy

```
import numpy as np
```

```
    a = np.linspace(0, 1, 40)
    b = np.exp(a)
    a + b # opération terme à terme
    plt.plot(a, np.exp(a) - 1 - a - a**2/2)
    plt.show()
```



5 SQL

5.1 Structure de base

Une requête se construit avec la syntaxe suivante
SELECT une ou des données
FROM provenance des données
[WHERE condition(s)]
[GROUP BY une colonne, en cas d'utilisation d'une fonction d'aggrégation]
[ORDER BY une ou des colonnes]

5.2 Jointure

Dans le cas où les données à utiliser (dans le select ou dans d'autres parties de la requête) proviennent de deux tables, on utilise une jointure dans la clause FROM :

FROM table1 JOIN table2 ON champ1 = champ2

où on identifie deux champs (colonnes) : un dans la première table, un autre dans la deuxième. Pour savoir quels champs identifier, il faut se référer à la structure de la base, qui est explicite dans tous les cas.

Pour utiliser 3 tables ou plus, il suffit d'ajouter autant de clause JOIN ... ON que voulu