TD Python

1°) Afficher la table de multiplication de 55 à l'écran

2°) Demander à l'utilisateur un nombre et afficher la table de multiplication de ce nombre à l'écran

3°) Créer une liste contenant les chiffres 1 à 5

4°) Créer une fonction qui retourne le plus petit élément d'une liste en utilisant une boucle for

5°) Créer une fonction qui retourne le plus petit élément d'une liste en utilisant une boucle while

6°) employes = {"Pierre": 2500, "Marie": 5000, "Julien": 1200}

Ecrire la fonction qui additionne les nombres et renvoie donc 8700

7°) Le challenge *FizzBuzz* est un classique pour évaluer les bases de programmation. L'objectif de cet exercice est d'écrire un programme qui affiche les nombres de 1 à 100 avec les exceptions suivantes :

- Pour les multiples de 3, on affiche 'Fizz' à la place du nombre.
- Pour les multiples de 5, on affiche 'Buzz' à la place du nombre.
- Pour les multiples de 3 et 5, on affiche 'FizzBuzz' à la place du nombre.

Exemple

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
```

8°) Dans cet exercice, nous allons **calculer le volume d'une sphère** ayant pour rayon **x centimètres**, x étant donné par l'utilisateur

La formule pour calculer le volume d'une sphère est :

$(4\pi/3) \times rayon^3$

rayon représentant la valeur du rayon (défini dans le code par la variable rayon).

Récupérez la valeur du volume de la sphère dans la variable volume

9°) Dans cet exercice, nous cherchons à **compter le nombre d'occurrences d'une lettre** dans une chaîne de caractère.

Ici, nous cherchons le nombre de fois que **la lettre "o"** apparaît dans la phrase **"Bonjour tout le monde".**

Votre script devra retourner dans ce cas-ci dans une variable resultat le nombre 4 !

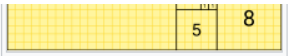
lettre_a_chercher = "o"
phrase = "Bonjour tout le monde"

10°) **créer une liste de nombres pairs allant de 1 à 100**

11°) Programme la Suite de Fibonnacci : un nombre est demandé à l'utilisateur et le programme retourne le nombre de Fibonacci

Définition formelle [modifier | modifier le code]

La suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est définie par $F_0 = 0, \quad F_1 = 1$, et la relation de récurrence $F_n = F_{n-1} + F_{n-2}$ pour $n \geqslant 2$. Le tableau suivant donne les 15 premiers termes de la suite de Fibonacci :



Une juxtaposition de carrés dont les côtés ont pour longueur des nombres successifs de la suite de Fibonacci : 1, 1, 2, 3, 5, 8, 13 et 21.

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	...	F_n
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	...	$F_{n-1}+F_{n-2}$

Dans cet article, nous avons fait commencer la suite à $n = 0$ avec $F_0 = 0$, comme le fait Édouard Lucas[1]. D'autres auteurs font débuter la suite à 1[2] : $f_0 = f_1 = 1, f_2 = 2, f_3 = 3, f_4 = 5$, etc. Le nombre F_n s'appelle parfois le *n*-ième nombre de Fibonacci[2] (bien qu'il soit techniquement le *n*+1-ième si on commence à 0).

12°) Suite de Syracuse

Le problème qui va suivre, appelé **conjecture de Syracuse**, a été proposé en 1928 par le mathématicien allemand Lothar Collatz. En 1952, lors d'une visite à Hambourg, L. Collatz expliqua son problème à Helmut Hasse. Ce dernier le diffusa en Amérique à l'université de Syracuse : la suite de Collatz prit alors le nom de « suite de Syracuse ».

Partie A

Dans cette suite, chaque terme se déduit du précédent par le procédé suivant :
- Si le terme est pair, alors on le divise par 2.
- Si le terme est impair, on le multiplie par 3 et on ajoute 1.

On note (u_n) le terme de rang n de cette suite de nombres.

Programmer en Python cette suite. Vous demanderez à l'utilisateur un nombre pour u_0 et vous arrêterez si vous tombez sur 1 ou au bout de 1000 boucles (soit u_{1000})

Essayez avec plusieurs u0. Qu'observez-vous.

Partie B

Dans votre programme, faire une boucle qui appelle votre fonction Syracuse pour tous les nombres de 1 à 1000. Quelle conjecture faites-vous sur la convergence de la suite ?

13°) Créer un dictionnaire dont les clés sont les lettres de l'alphabet et les valeurs sont leurs positions dans l'alphabet

14°) Créer une fonction coder qui prend une chaine de caractère et un dictionnaire en paramètre et qui retourne la chaîne codée avec le code contenu dans le dictionnaire. Ce code sera le dictionnaire de la question 13°).

```
def coderInfo(message,codage):
    messagecoder="";
    return messagecoder
```

15°) Le principe du tri par sélection/échange (ou *tri par extraction*) est d'aller chercher le plus petit élément du vecteur pour le mettre en premier, puis de repartir du second élément et d'aller chercher le plus petit élément du vecteur pour le mettre en second, etc...

1. **PROCEDURE** tri_Selection (Tableau a[1:n])
2. **POUR** i **VARIANT DE** 1 **A** n - 1 **FAIRE**
3. TROUVER [j] LE PLUS PETIT ELEMENT DE [i + 1:n];
4. ECHANGER [j] ET [i];
5. **FIN PROCEDURE**;

Programmer en Python le tri par sélection. Une liste sera passée en paramètre et devra donc être triée.

16°) Le principe du tri à bulles (*bubble sort* ou *sinking sort*) est de comparer deux à deux les éléments e_1 et e_2 consécutifs d'un tableau et d'effecteur une permutation si $e_1 > e_2$. On continue de trier jusqu'à ce qu'il n'y ait plus de permutation.

1. **PROCEDURE tri_bulle (TABLEAU a[1:n])**
2. **passage ← 0**
3. **REPETER**
4. **permut ← FAUX**
5. **POUR i VARIANT DE 1 A n - 1 - passage FAIRE**
6. **SI a[i] > a[i+1] ALORS**
7. **echanger a[i] ET a[i+1]**
8. **permut ← VRAI**
9. **FIN SI**
10. **FIN POUR**
11. **passage ← passage + 1**
12. **TANT QUE permut = VRAI**

Programmer en Python le tri à bulles. Une liste sera passée en paramètre et devra donc être triée.