Les listes, les dictionnaires, les fonctions
2024 - 2025

1er avril 2025

1 Les listes et tuples :

Rappel :

- `Untuple=(1,2,3,4)`
- `Uneliste=[1,2,3,4]`

Une liste est modifiable et un tuple ne l'est pas. Les chaînes sont accessibles comme les tuples, il est parfois préférable de convertir certaines chaînes en liste pour faciliter la manipulation.

La création d'une liste vide :

```
1 maliste=[] # Je viens de créer une liste vide
```

Les fonctions utiles :

- `list.append('55')` : ajoute l'élément la chaîne de caractères `'55'` à la fin de la liste.
- `list.sort(reverse=True)` : range dans l'ordre croissant les éléments de la liste (l'ordre peut être décroissant si l'argument reverse est mis à True).
- `list.insert(3,55)` : insère dans la liste à l'indice 3 donné en premier argument, l'élément 55 donné en deuxième argument.
- `list.reverse()` : inverse l'ordre des éléments dans la liste.
- `list.remove(78)` : retire l'objet `78` passé en argument de la liste. S'il y est présent plusieurs fois, seule la première fois sera retirée.
- `sum(maliste)` : Fait la somme des élément de `maliste`
- `max(maliste)` : Trouve le maximum de `maliste`
- `min(maliste)` : Trouve le minimum de `maliste`
- `len(maliste)` : Trouve la taille de `maliste`

Exercice 1 : Réalisation d'un chenillard.

A l'aide d'une boucle `for`, d'une liste de 10 éléments et des fonction ci-dessus.

Considérer que chaque élément est un lampe. A l'itération 0, chaque lampe est éteinte donc chaque élément est égale à 0, puis chaque lampe devra s'allumer au fur et à mesure.

Exemple :

1. itération 0 : maliste= [0,0,0,0,0,0,0,0,0,0]
2. itération 1 : maliste= [1,0,0,0,0,0,0,0,0,0]
3. itération 2 : maliste= [0,1,0,0,0,0,0,0,0,0]
4. itération 3 : maliste= [0,0,1,0,0,0,0,0,0,0]
5. itération 4 : maliste= [0,0,0,1,0,0,0,0,0,0]

6. itération 5 : maliste= [0,0,0,0,1,0,0,0,0,0]

Lorsque le dernière élément est allumé, cela doit recommencer au début. Faites entrer le nombre d'instant en `input`. Modifiez le programme pour que le système fonctionne à rebours.

2 Les dictionnaires

Rappel :

Dans un dictionnaire, chaque fois que vous voudrez ajouter une donnée, vous devrez l'associer à une clé pour former le couple (clé, valeur).

`dico = {clé1 : valeur1, clé2 : valeur2, ...}`

Exemple :

```
dico = {variable : "Hey !", variable2 : "Salut !", variable3 : "Peuh !"}
dico[variable] # on obtient 'Hey !'
variable = 2
dico[variable] # on obtient "Salut !"
```

Exercice : Inscription internet

Lorsque vous vous inscrivez sur un site internet, les éléments suivants sont souvent demandés :

- Votre nom
- Votre prénom
- Votre date de naissance
- Votre adresse mail
- Votre mot de passe

Réaliser un programme qui demande à un utilisateur toutes ces informations puis créer une bibliothèque qui permette de les stocker.

3 Notions avancées : recursivité et gestion des erreurs

La récursivité : une fonction qui s'appelle elle-même :

La définition la plus simple d'une fonction récursive est la suivante : c'est une fonction qui s'appelle elle-même. Si dans le corps (le contenu) de la fonction, vous l'utilisez elle-même, alors elle est récursive.

Exceptions et erreurs

Lorsque quelqu'un d'autre utilisera votre programme il est possible qu'il commette des erreurs d'utilisation. Pour éviter ce problème on utilise les fonctions `try` et `except`

- Le `try` permet de tester une fonction, une variable ou même un programme
- Le `except` permet au programme de continuer si l'on connaît l'erreur qui sera commise et donc de réagir différemment

L'exemple le plus simple c'est de considérer la fonction factorielle $f(n) = n!$ ou encore le produit des nombres allant de 1 à n

```
def factoriel_recurs(n):
    """
    This function allow computing the factorial number of the entry 'n'. It
    is using a recursive method which permit to decrease the computation
    time and the necessary computer memory

    Input :
    ========

    n: The number that we want compute the factorial parameter

    Output :
```

```
    ====================

    The result of n!
    """
    # Ici nous avons une fonction récursive, à chaque nouveau tour, le fonction
    # enlève 1 à n se qui à pour effet d'obliger n à diminuer jusqu'à 1 et
    # donc de sortir de la boucle récursive.
    while True: # Déclenchement d'un boucle infinie
        try:    # Test du type de variable
            if n==1:    # Boucle if de test de la valeur de n
                return 1
                break   # Sortie de la boucle
            elif n==0:  # Boucle if de test de la valeur de n
                return 0
                break   # Sortie de la boucle
            else:
                return n*factoriel_recurs(n-1)  # Renvoie n que multiplie la
                                                # fonction elle même
                break     # Sortie de la boucle
        except TypeError: # Si une erreur de type survient
                          #  alors l'utilisateur devra
                          # rentrer un nouveau nombre 'n'
                print("Rentrez un nombre de type 'integer' ")
                n=eval(input("Donnez un nombre : "))

if __name__=='__main__': # Programme principal
    A=factoriel_recurs('pas un nombre')
    print(A)
```

Exercice

Réalisez une fonction récursive de la suite de Fibonacci.

$$F_{n+2} = F_{n+1} + F_n$$

avec $F_0 = 1$ et $F_1 = 1$