Exercice 5.57

Une pyramide de nombre est un graphe donc les sommets sont des nombres.Chaque sommet d'un même niveau a deux arrêtes vers le bas.Deux sommets voisins d'un même niveau sont reliés à un même sommet du niveau suivant.

```
1
2          3
3         2  4
4        2  7  5
5       9  5  4  6
```

Ici on peut suivre le chemin 3 - 4 - 7 - 5 mais pas 3 - 2 - 5 : les sommets 2 et 5 ne sont pas reliés.
Objectif :Déterminer la valeur maximale de la somme des chemins traversant une pyramide de nombre.

1. Quel est le chemin qui donne la somme maximale dans l'exemple ci-dessus ? ***

2. Choix d'une structure de données adaptée pour les pyramides :
   $T[i, j]$ est la valeur du nombre situé à l'étage i en partant du bas et en position j en partant de la gauche.
   ☛ Toujours sur l'exemple,T[1,1] = 9, T[1,2] = 5, T[2,1] = 2, T[3,1] = 2. Donner les valeurs de T[4,1] et T[3,2].

3. L[i, j] est la somme maximale de la sous pyramide dont le sommet est la case en position (i, j).

   a) Que vaut L[1, j] ?

   b) Montrer que pour i >1, `L[i,j] =T[i,j] + max(L[i-1, j], L[i-1, j+ 1])`.

4. a) Proposer un programme python récursif qui répond au problème.

   b) Est-ce qu'on effectue plusieurs fois le même calcul ? Est-ce optimal ?

5. Proposer une amélioration en utilisant la programmation dynamique.

Exercice 5.58

Avec les programmes précédents, on obtient bien la somme maximale, mais pas le chemin qui y mène. Que pourrait-on ajouter pour l'obtenir ?

Chapitre 6

Les paradigmes de programmation

Sommaire

6.1 Les paradigmes de programmation

NSI Tle - JB Duthoit

6.1.1 Introduction

Définition 6.8

Un ***paradigme*** est un point de vue particulier sur la réalité, an angle d'attaque privilégié face à un problème, un état d'esprit.

Remarque

Un paradigme a lui seul ne permet qu'une vision limitée de la réalité.
Il existe des milliers de langages informatiques, souvent classés par catégorie suivant leur fonctionnement ; ces catégories sont appelés ***paradigmes de programmation*** .

6.1.2 La programmation impérative

Jusqu'à présent nous avons vu un seul paradigme de programmation : la programmation impérative.
La programmation impérative respecte plusieurs points :

- La séquence d'instructions (les instructions d'un programme s'exécutent étape par étape). On parle d'instructions *séquentielles*.
- l'affectation (on attribue une valeur à une variable, par exemple : x = 15)
- l'instruction conditionnelle (if / else)
- les boucles (while et for)

La programmation impérative est la plus courante et la plus ancienne : elle a été inventé conjointement aux premiers ordinateurs.

Exemple

- C
- Pascal
- COBOL
- Fortran
- Python :-)

Exercice 6.59

Construire en langage python deux fonctions qui permettent à terme un tri d'un tableau :

```
def insere(t,v,i):
    '''
    insère v dans le tableau t[,...,i[ supposé trié
    '''
```

```
    pass

def tri_insertion(t):
    '''
    trie le tableau t dans l'ordre croissant
    en utilisant la fonction insere
    '''
    pass
```

Mais la programmation impérative est loin d'être le seul paradigme de programmation.

6.1.3 le paradigme orienté objet

Programmer avec un langage orienté objet amène le programmeur à identifier les "acteurs" qui composent le problème, puis de déterminer ce qu'est et ce que doit savoir faire chaque acteur.
Ce type de programmation est très puissant, et offre un excellent outil de modularité.
Les programmes orientés objets sont modifiables et réutilisables.

Exemple

- C++
- Java
- Python :-)

Reprenons l'exemple :

```
liste = [2,1,6]
liste.sort()
print(f"Le minimum est {liste[0]}")
```

Lorsque l'on crée la liste, on crée un objet liste qui est une instance de la classe liste. Cette classe possède beaucoup de méthodes publiques, dont *sort()* qui ordonne les éléments de façon croissante.

6.1.4 Paradigme fonctionnel

Une fonction accepte des données et renvoie des données.

Une fonction est un objet de première classe sur lequel d'autres fonctions peuvent opérer.

☞ Ce paradigme ne contient pas la notion de variable ! **Un programme en langage fonctionnel ne produit jamais d'effets secondaires !**.

Ce paradigme propose un point de vie "transformationnel" : tout comportement doit être vu comme un enchaînement de transformations sur un état initial et produisant un état final.

Reprenons notre exemple de recherche du minimum d'une liste :

```
def minimum(liste):
        long = len(liste)
        if lon == 0:
```

```
                print("La liste est vide")
        else:
                min = liste[0]
                i = 1
                while i < long:
                        if liste[i] < min:
                                min = list[i]
                        i = i + 1
                print(f"Le minimum est {min}")

liste = [2,1,6]
print(f"Le minimum est minimum(liste))
```

Remarque

La fonction *min* est native dans Python, on aurait pu juste écrire :

```
liste = [2,1,6]
print(f"Le minimum est min(liste))
```

Exemple

- Lisp
- Ocaml
- LOGO
- Python :-)

Attention aux effets de bord !

Considérons le programme suivant :

```
l = [4,7,3]
def ajout(i):
  l.append(i)
```

Le programme ci-dessus respecte-t-il la logique du paradigme fonctionnel ?
Et celui-ci :

```
def ajout(i,l):
        x = l + [i]
  return x
```

☞ Le paradigme fonctionnel va amener le programmeur non pas à modifier une valeur existante, mais plutôt à créer une nouvelle grandeur à partir de la grandeur existante : une grandeur existante n'est jamais modifiée, donc aucun risque d'effet de bord.

Selon vous, le programme ci-dessus respecte-t-il le paradigme fonctionnel ?

6.1.5 Utiliser le "bon" paradigme de programmation

Il est important de bien comprendre qu'un programmeur doit maitriser plusieurs paradigmes de programmation (impératif, objet ou encore fonctionnelle).
En effet, il sera plus facile d'utiliser le paradigme objet dans certains cas alors que dans d'autres situations, l'utilisation du paradigme fonctionnelle sera préférable.