POO, Piles

Ce TP est consacré à la *Programmation Orientée Objet*, POO en abrégé et à son application à la construction de la structure de Pile.

1 Principe de la POO

1.1 Généralités

La POO consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Il possède une structure interne et un comportement, et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et de réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes.

Concrètement, un objet est une structure de données valuées et cachées qui répond à un ensemble de messages. Cette structure de données définit son état tandis que l'ensemble des messages qu'il comprend décrit son comportement :

- Les données — ou champs — qui décrivent sa structure interne sont appelées ses *attributs* ;
- L'ensemble des messages forme ce que l'on appelle l'interface de l'objet ; c'est seulement au travers de celle-ci que les objets interagissent entre eux. La réponse à la réception d'un message par un objet est appelée une *méthode* (méthode de mise en œuvre du message) ; elle décrit quelle réponse doit être donnée au message.

La structure interne des objets et les messages auxquels ils répondent sont définis par des modules logiciels. Ces mêmes modules créent les objets via des opérations dédiées. Deux représentations existent de ces modules : la classe et le prototype.

La classe est une structure informatique particulière dans le langage objet. Elle décrit la structure interne des données et elle définit les méthodes qui s'appliqueront aux objets de même famille (même classe) ou type. Elle propose des méthodes de création des objets dont la représentation sera donc celle donnée par la classe génératrice. Les objets sont dits alors instances de la classe.

1.2 En Python

En Python, la création d'une classe commence par le mot clé `Class` suivi du nom de la classe, de deux points et d'une indentation. Toute la partie du script indentée contient les méthodes de création des instances de cette classe et les méthodes de cette classe.

Lors de la programmation de méthodes dans une classe, on devra faire référence à l'instance de l'objet sur lequel on travaille : le mot clé `self` permet ceci en Python.

La méthode de création d'un objet est également conventionnelle en Python : elle est définie par `def __init__` suivi au minimum de l'argument `self` et éventuellement d'autres arguments. Les attributs des objets de cette classe sont définis dans cette méthode. Ils sont codés sous la forme `self.attribut`

Par exemple le code suivant permettrait de définir un objet de type Carte en Python, et une méthode qui renvoie un booléen indiquant si une carte est un honneur (10, V, D, R ou As) :

```
Class Carte:
      def __init__(self,h,v):
            self.hauteur=h
            self.valeur=v

      def honneur(self):
            return self.valeur in ('10', 'Valet', 'Dame', 'Roi', 'As')
```

Le code suivant créerait alors le Valet de Carreau : `C = Carte('Valet', 'Carreau')`
L'expression suivante serait évaluée à `True` : `C.honneur()`

On verra dans ce TP une autre méthode spéciale : `def __repr__(self):` appelée lors de l'instruction `print`.

Il ne s'agit là que d'une micro-introduction à la programmation orientée objet ; un semestre entier (en école...) ne suffira même pas à poser complètement les principes, techniques et intérêts de ce paradigme de programmation.

2 Premiers pas en Programmation Orientée Objet : classe Vecteur

Question 1. Ouvrir le fichier `vecteur.py` et l'exécuter.
Taper dans la console `V = Vecteur(1, 2, 3)` puis `print(V)`.
Créer une instance `W` de la classe Vecteur représentant le vecteur (4, 5, 6).
Tester dans la console `V.norme()` puis `V.somme(W)`.

Question 2. Écrire une méthode `multipliescalaire(self, k)` qui renvoie à partir du vecteur $\vec{u}$ et du scalaire k le vecteur $k\vec{u}$.

Question 3. Écrire une méthode `produitvectoriel` qui calcule le produit vectoriel de deux vecteurs.

Question 4. Écrire une méthode `testorthogonal` qui teste si deux vecteurs $\vec{u}$ et $\vec{v}$ sont orthogonaux.

3 Création d'une classe Pile

La notion de classe permet de créer de nouveaux types de structures. Nous allons nous en servir pour créer une structure pile de deux manières différentes.

La structure Pile, que nous étudions en cours, est une structure de données linéaire, qui fonctionne sur le modèle de la pile d'assiettes. Les seules opérations possibles sont le fait de déposer un élément en haut de la pile (on parlera de la fonction `push`) ou d'enlever l'élément du haut de la pile (on parlera de la fonction `pop`). Les piles fonctionnent donc sur le principe du LIFO (*Last in First Out*) :

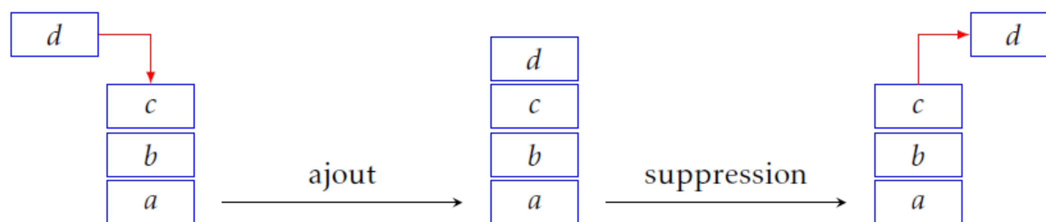

FIGURE 1 – Empiler - dépiler

Dans un premier temps, on va se servir de la structure `list` pré-implémentée en Python. Nos piles seront représentées en mémoire par des listes. Le sommet de la pile sera le dernier élément de la liste, ainsi empiler un élément consistera à ajouter un élément à la fin d'une liste, et effectuer un `pop` consistera à supprimer le dernier élément de la liste en renvoyant sa valeur. Le seul attribut d'un objet pile sera une liste de valeurs.

On s'inspirera évidemment de la partie précédente.

Question 5. Créer une classe `Pile`, et créer une méthode de constructeur de pile vide à l'aide de la syntaxe `def __init__(self)`.

Question 6. Créer une méthode `estvide(self)` qui renvoie un booléen indiquant si la pile est vide ou non.

Question 7. Créer une méthode `push(self, x)` qui empile l'élément `x` sur la pile. Il s'agira donc de rajouter un élément à la fin de la liste qui représente la pile.

Question 8. Créer une méthode `pop(self)` qui renvoie une erreur si la pile est vide, et qui dépile le sommet de la pile en renvoyant sa valeur dans le cas contraire.

Question 9. Créer la méthode `__repr__(self)` qui va être appelée lors de l'affichage d'une pile `P` à l'aide de l'instruction `print(P)`. On souhaite que la pile contenant les valeurs 1, 2 et 3, empilées dans cet ordre, soit affichée de la manière suivante :

sommet
|3|
|2|
|1|
fond

Question 10. Créer une fonction (en dehors de la classe `Pile`, il ne s'agit pas d'une méthode) qui inverse les deux premiers éléments d'une pile, en ne se servant que des méthodes définies sur les piles.

Question 11. Créer la pile obtenue en empilant successivement 1 puis 2 puis 3 puis 4. Afficher la. Tester la fonction précédente, et afficher le résultat.

4 Une classe pile alternative

Il est également possible de créer une classe Pile sans se servir de classes déjà pré-implémentées comme la classe List. C'est l'objectif de cette partie.

Pour ce faire, nous allons créer à la main une structure de liste chaînée ; pour ce faire nous allons créer une classe Cellule, qui est donnée dans le document joint `PilesAlt.py` ; et une deuxième classe Pile, qui se servira de cette structure, dont certains éléments sont également donnés dans le fichier joint.

On rappelle qu'une liste chaînée est une structure linéaire, constituée de cellules. Une cellule contient une valeur et un pointeur qui dirige soit vers la cellule suivante, soit vers la valeur `None`, auquel cas la cellule est la « dernière » de la liste chaînée.

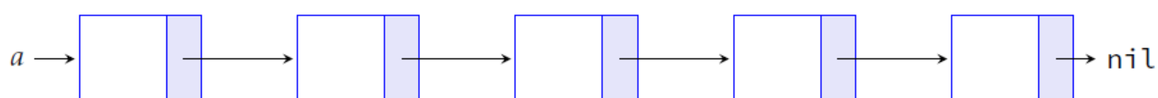

Figure 2 – Représentation chaînée

Question 12. Compléter la méthode `push` fournie dans le fichier joint.

Question 13. Créer une méthode `pop`.

Question 14. Créer la méthode `__repr__(self)` qui va être appelée lors de l'affichage d'une pile `P` à l'aide de l'instruction `print(P)`.

5 Recherche d'une enveloppe convexe plane

5.1 Introduction

Cette section, qui reprend une partie du problème posé au concours d'entrée à l'École Polytechnique en 2015 en section MP et PC, étudie un algorithme de recherche d'enveloppe convexe d'un nuage de points du plan.

Une partie $\mathcal{E}$ du plan euclidien $\mathbb{R}^2$ est dite *convexe* si, et seulement si, pour tout couple de points (A, B) de $\mathcal{E}$, le segment $[\mathrm{A}, \mathrm{B}]$ est inclus dans $\mathcal{E}$. De manière générale, l'*enveloppe convexe* d'une partie $\mathcal{A}$ du plan est la plus petite partie convexe de $\mathbb{R}^2$ contenant $\mathcal{A}$. C'est l'intersection de toutes les parties convexes de $\mathbb{R}^2$ contenant $\mathcal{A}$. On peut prouver que l'enveloppe convexe d'un nuage de points est une ligne polygonale fermée dont les sommets sont des points du nuage.

La notion d'enveloppe convexe est importante en analyse fonctionnelle et en théorie de la mesure. Hors du champ strict des mathématiques, elle est notamment utile en robotique, en traitement d'images (reconnaissance de formes), en théorie des jeux (recherche d'équilibres), en physique statistique (théorie ergodique) et en informatique formelle (vérification de programmes).

Dans toute la suite, on supposera que l'on cherche l'enveloppe convexe d'un nuage de points $\mathcal{P}$ contenant au moins trois points et que les points sont en position dite générale, *i.e.* que $\mathcal{P}$ ne contient pas trois points alignés. On testera les différentes fonctions sur la famille de points suivantes, dont on donne une représentation graphique ci-après, avec leur enveloppe convexe :

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}
$(0,0)$	$(1,4)$	$(1,8)$	$(4,1)$	$(4,4)$	$(5,9)$	$(5,6)$	$(7,-1)$	$(7,2)$	$(8,5)$	$(11,6)$	$(13,1)$

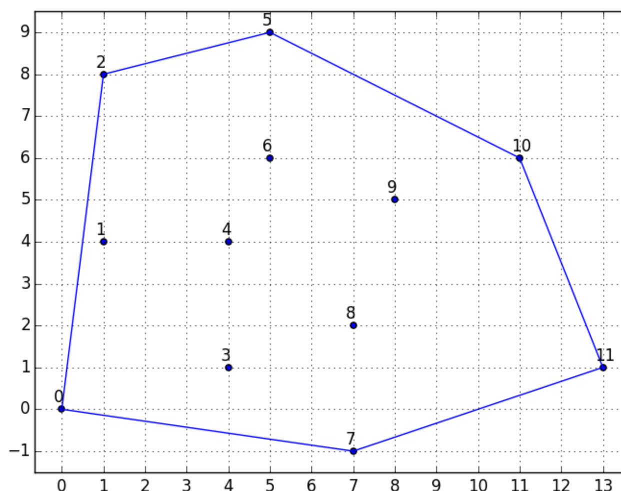

FIGURE 3 – Enveloppe convexe d'un nuage de points

Question 15. Rentrer le nuage de point exemple sous la forme d'un tableau `Numpy` bidimensionnel contenant deux lignes, la première correspondant aux abscisses des points du nuage et la deuxième à leurs ordonnées.

Question 16. Écrire une fonction prenant en argument un tableau `Numpy` du type de la question précédente et retournant un tableau de même taille où les points sont rangés par abscisses croissantes. On appellera génériquement `tab` un tel tableau.

5.2 Orientation d'un triangle

Le plan affine euclidien est rapporté à sa base canonique. Les coordonnées de ses points sont exprimées dans cette base. Un triangle (A, B, C) est *orienté positivement* si la base $\left(\overrightarrow{AB}, \overrightarrow{AC}\right)$ du plan euclidien qu'il définit a la même orientation que la base canonique. Par le calcul, cela signifie que son aire signée $\mathcal{A}\big((A, B, C)\big) = \frac{1}{2} \det\left(\overrightarrow{AB}, \overrightarrow{AC}\right)$ est positive.

Question 17. Écrire une fonction `orient(i, j, k)` prenant en paramètres un tableau `tab` représentant un nuage $\mathcal{P}$ de points P_i du plan et trois indices (distincts ou non), et renvoyant 1 si le triangle (P_i, P_j, P_k) est orienté positivement, -1 s'il est orienté négativement et 0 s'il est applati.

5.3 L'algorithme de Graham-Andrew

L'algorithme de recherche de l'enveloppe convexe étudié ici a été conçu par R. Graham en 1972 et nous en traitons une simplification proposée par A. Andrew quelques années plus tard. On a besoin pour cela que le nuage de points soit trié selon les abscisses croissantes (comme c'est le cas dans l'exemple fourni en début de partie). L'idée de l'algorithme est de balayer le nuage de points horizontalement de gauche à droite par une droite verticale (figurée en pointillés sur les figures ci-dessous) tout en mettant à jour l'enveloppe convexe des points de $\mathcal{P}$ situés à gauche de cette droite. Plus précisément, l'algorithme visite chaque point de $\mathcal{P}$ une fois, par ordre coirsant d'abscisse (donc par ordre croissant d'indice de colonne). À chaque nouveau point P_i visité, il met à jour le bord de l'enveloppe convexe du sous-nuage $\{P_0, \ldots, P_i\}$, soit les points situés à gauche de P_i. On remarque que les points P_0 et P_i appartiennent nécessairement à ce bord. L'enveloppe convexe partielle est séparée en une partie supérieure, contenant ceux des points situés au-dessus de la droite $(P_0 P_i)$ — y compris P_0 et P_i —, et une partie inférieure contenant ceux situés en dessous — également y compris P_0 et P_i. Par exemple, au stade de la seconde figure ($i = 6$), l'enveloppe convexe supérieure est la ligne polygonale $[P_0, P_2, P_5, P_6]$ et l'enveloppe inférieure de $[P_0, P_3, P_6]$.

Sur le plan informatique, les indices des sommets des enveloppes inférieure et supérieure sont stockés dans deux piles d'entiers séparées, appellées respectivement `enveloppe_i` et `enveloppe_s`. La mise à jour des enveloppes se fait de la manière suivante : prenons l'insertion du point P_6 réalisée dans la figure du milieu. À

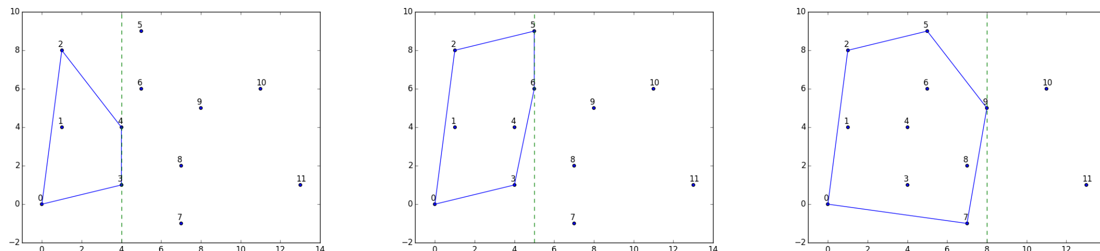

Figure 4 – Algorithme de Graham-Andrew

l'étape précédente (non représentée), on a `enveloppe_i=[0,2,5]` et `enveloppe_s=[0,3,5]`. On constate que le triangle (P_5, P_2, P_6), constitué, dans l'ordre, du dernier élément de la pile `enveloppe_i`, de l'avant-dernier élément de cette pile et du point en train d'être traité, est orienté positivement. On empile alors P_6 (en pratique, son indice) sur la pile, qui devient ainsi `enveloppe_i=[0,2,5,6]`. Symétriquement, pour l'enveloppe inférieure, on constate que le triangle (P_5, P_3, P_6) est lui aussi orienté positivement. Au contraire de ce qui se passe pour l'enveloppe supérieure, cela signifie que P_5 doit disparaître. On dépile donc P_5 de l'enveloppe inférieure, qui devient `enveloppe_s=[0,3]`. Le triangle (P_3, P_0, P_6) est, lui, orienté négativement, donc on empile P_6 et l'on a `enveloppe_s=[0,3,6]`. Si, en cours d'algorithme, l'enveloppe (supérieure ou inférieure) ne contient plus qu'un élément, on empile le nouveau point.

Question 18. Écrire une fonction `majsup(tab, enveloppe_s, i)` prenant en argument le tableau `tab`des coordonnées du nuage de points $\mathcal{P}$, la pile des indices des points de l'enveloppe supérieure `enveloppe_s` et l'indice i du point P_i à insérer et qui met à jour la pile `enveloppe_s` en un temps en $\mathcal{O}(i)$.

Question 19. Écrire une fonction `majinf(tab, enveloppe_i, i)` effectuant le même travail sur l'enveloppe inférieure.

Question 20. Écrire une fonction `Convexe(tab)` prenant en argument le tableau `tab` des coordonnées du nuage de points $\mathcal{P}$ et qui renvoie une liste unique contenant les points de $\mathcal{P}$ appartenant au bord de l'enveloppe convexe de $\mathcal{P}$.

Question 21. Écrire une fonction `Convexe(tab)` prenant en argument le tableau `tab` des coordonnées du nuage de points $\mathcal{P}$ et qui renvoie une figure du type de la figure 5.1, les points étant numérotés par ordre croissant des abscisses. On trouvera dans le fichier `Enveloppe.py` le code ayant généré 5.1.

Question 22. Évaluer la complexité de l'algorithme de Graham-Andrew.