

Problème : classe polynôme

Un polynôme, c'est une expression de la forme $2x^2 + 3x + 4$, ou plus généralement de la forme $\sum_{n=0}^N \alpha_n x^n$.

- Les nombres α_n sont appelés les *coefficients* du polynôme.
- Le plus grand entier n tel que $\alpha_n x^n \neq 0$ est appelé le *degré* du polynôme.

Travail demandé :

- 1- Écrire une classe Polynôme avec un constructeur prenant pour argument une liste des coefficients et initialisant son attribut **coef** à cette liste.

Pour créer un polynôme, il faut lister ses coefficients par ordre de degré croissant.

Par exemple, pour le polynôme $X^3 + 2X - 3$,

```
>>> p = Polynome([-3, 2, 0, 1])
```

- 2- Ajoutez une méthode `__repr__`. Par exemple, pour le polynôme $3X^4 + 7X^3 + 2$, on obtiendra la chaîne :
« $3X^4 + 7X^3 + 2X^0$ »
- 3- Ajoutez une méthode `__call__(self, v)` permettant d'évaluer et de renvoyer la valeur du polynôme en un point v .
- 4- Ajoutez une méthode `degre()` permettant de renvoyer le degré du polynôme.
- 5- Ajoutez une méthode `__getitem__(self, i)` permettant d'obtenir le coefficient correspond au degré i de votre polynôme. Vous pouvez écrire `P[i]` plutôt que `P.__getitem__(i)`.
- 6- Ajoutez une méthode `__add__(self, other)` pour effectuer et renvoyer la somme de deux polynômes. Vous serez en mesure de faire la somme de deux polynômes P et Q avec la syntaxe $P + Q$.
- 7- Ajoutez une méthode `mult_constant(self, a)` pour effectuer et renvoyer la multiplication du polynôme par une constante a .
- 8- Ajoutez une méthode `mult_monome(self, d)` pour effectuer et de renvoyer la multiplication du polynôme par un monôme X^d .
- 9- Ajoutez une méthode `__mul__(self, other)` pour effectuer et renvoyer la multiplication de deux polynômes. Vous serez en mesure de faire le produit de deux polynômes P et Q avec la syntaxe $P * Q$.
- 10- Ajoutez une méthode `__pow__(self, n)` pour calculer la puissance $n^{\text{ième}}$ d'un polynôme. Vous serez en mesure de faire la puissance d'un polynôme P avec la syntaxe $P**n$.

Problème : Gestion des expressions post-fixées

L'objet de ce problème est de gérer des expressions post fixées, expressions algébriques constituées d'éléments appelés opérandes et d'opérateurs. La notation préfixée est souvent appelée notation polonaise.

Partie I :

Avant de réaliser le traitement des expressions post-fixées, nous avons besoin d'une structure de données, une pile, de type LIFO (le dernier entré est le premier sorti). Une pile P sera caractérisée par une capacité. Ainsi le premier élément de la pile désigne le nombre d'éléments existants dans la pile.

Une pile est entièrement définie par deux opérations :

1. **empiler** une valeur : c'est l'insérer au début de P
2. **dépiler** une valeur, si P est non vide, c'est sélectionner la première valeur de la pile et la supprimer de P

Travail demandé :

Une Pile sera définie comme une classe dont les attributs sont :

- **capacity** : le nombre maximum d'éléments qu'on peut empiler dans la pile
- **P** : une liste tels que :
 - $P[0] = n$, contient le nombre d'éléments présents dans la pile
 - $P[1]$ = la liste des éléments présents dans la pile.

1. Ecrire la classe Pile avec un constructeur prenant en paramètre la capacité maximum de la pile (par défaut = 100).
2. Ajouter une méthode `estVide()`, vérifiant si la pile est vide.
3. Ajouter une méthode `estPleine()`, vérifiant si la pile est pleine.
4. Ajouter une méthode `sommet()`, retournant le sommet de la pile.
5. Ajouter une méthode `empile()` prenant en paramètre une variable X et l'ajouter à la pile. Cette méthode traite l'exception en cas où la pile est pleine.
6. Ajouter une méthode `depile()`, permettant de dépiler la pile et retournant la variable supprimée. Cette méthode traite l'exception en cas où la pile est vide.
7. Ajoutez la méthode `__add__` pour effectuer la superposition de deux piles.

Partie II

Principe de l'évaluation :

Pour calculer une expression arithmétique codée en post-fixée, il suffit :

- d'empiler les entiers successifs que l'on rencontre
 - lorsque l'on détecte un opérateur, il faut :
 - dépiler les derniers entiers ;
 - leur appliquer l'opérateur
 - empiler le résultat
8. Ecrire une fonction récursive `estEntier(expr)` qui vérifie si la chaîne **expr** est un nombre entier .
 9. Ecrire une fonction python `evaluation (exp)` qui calcule une expression arithmétique codée en post-fixée.

```
>>> evaluation(['7', '4', '+'])
```

```
11
```

10. On s'intéresse à présent aux expressions infixées, comme par exemple $(1+(5*(6+2)))$.

Ecrire une fonction `estBienParentesee(exp)` qui renvoie True si exp est bien parenthésée, False sinon.

```
>>> estBienParentesee(['(', '1', '+', '(', '5', '*', '(', '6', '+', '2', ')', ')', ')'])
```

```
True
```

Problème: Forme géométrique

- I. Un point est défini par ses coordonnées X et Y.
 1. Ecrire une classe Point dotée des attributs X et Y de type réels.
Par exemple, pour créer un point :

```
>>> p1 = Point(2.1 , 3.0)
```
 2. Ajouter une méthode `__str__` qui renvoie une chaîne de la forme '`<x,y>`'.

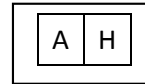
```
>>> print(p1)
```

```
'<2.1 , 3.0>'
```
 3. Ajouter une méthode `distance` qui renvoie la distance entre le point courant et un autre point.
- II. Un cercle est défini par un rayon (de type réel) et un centre (de type Point).
 1. Ecrire une classe Cercle doté des attributs rayon et centre avec un constructeur prenant en paramètre la valeur du rayon, l'abscisse et l'ordonnée du centre.
Par exemple, pour créer un cercle :

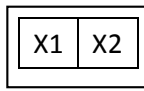
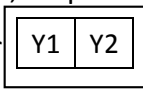
```
>>> c1 = Cercle(4 , 1.3 , 4.1)
```
 2. Ajouter une méthode `__repr__`, qui renvoie une chaîne de la forme :
'Cercle, ray =rayon, centre =<x,y>'
 3. Ajouter une méthode `périmètre`, qui renvoie le périmètre du cercle. Utiliser la variable PI du module math
 4. Ajouter une méthode `surface`, qui renvoie la surface du cercle.
 5. Ajouter une méthode `distance`, qui renvoie la distance entre le centre du cercle et le point d'origine.
- III. Un Cylindre est défini par une base et une hauteur.
 1. Ecrire une classe Cylindre qui hérite de la classe Cercle et doté en plus de son attribut hauteur.
 2. ui renvoie la surface du cylindre ($2 \times \text{Base} + \text{latérale}$).
 3. Ajouter une méthode `volume` qui calcule et renvoie le volume du cylindre
- IV. Un segment est défini par deux points dans le plan origine et extrémité.
 1. Ecrire la classe Segment doté des attributs origine et extrémité de type Point avec un constructeur prenant en paramètres deux tuples p1 et p2 tels que $p1 = (x1, y1)$ avec x1 et y1 les coordonnées du point origine (respectivement p2 pour extrémité)
 2. Ajouter la méthode `__len__` qui calcule et renvoie la longueur du segment (utiliser la méthode `distance` de la classe Point)
 3. Ajouter la méthode `plot` qui affiche le segment dans une figure (utiliser la fonction `plot` du module `matplotlib.pyplot`)
- V. Une droite est définie par deux Points dans le plan et une équation cartésienne $y = ax + b$
 1. Ecrire la classe droite qui hérite la classe Segment doté en plus des attributs a et b (les coefficients de l'équation cartésienne). Le constructeur de la classe prend en paramètres seulement les tuples p1 et p2 et initialise les attributs a et b.
 2. Ajouter la méthode `__str__` qui affiche un objet Droite sous la forme ' $y = ax + b$ '
 3. Ajouter la méthode `__contains__` qui prend en paramètre un objet Point et qui renvoie True si l'objet Point appartient à la droite courante, False sinon.
 4. Ajouter la méthode `parallèle` qui prend en paramètres une autre droite et qui renvoie True si les deux droites sont parallèles, False sinon.
- VI. Modifier la classe Point pour calculer le nombre d'instance (objet) créé.

Problème (Puzzle de domino) (Homework)

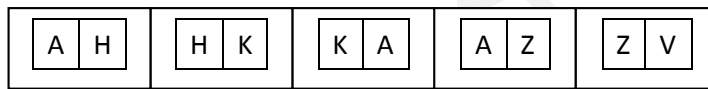
Un domino est un rectangle qui contient 2 lettres, par exemple



Pour simplifier, on considèrera qu'un domino ne peut pas être retourné. On cherche à construire à partir d'une liste de dominos, un puzzle : c'est à dire une séquence de dominos le long de laquelle si

 est suivi par  alors $X2 = Y1$ (les lettres voisines coïncident sur chaque paire des dominos consécutifs).

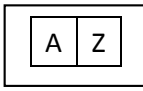
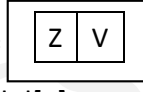
Un exemple de chaîne de dominos (puzzle) :



Le but de l'exercice est donc de construire une solution du puzzle à partir de n dominos selon la règle ci-dessus.

Travail demandé :

Partie A :

1. Définissez une classe **Domino** qui permette d'instancier des objets simulant les pièces d'un jeu de dominos. Le constructeur **`__init__()`** de cette classe initialisera les lettres présentes sur les deux faces du domino.
2. Ajoutez une méthode **`__repr__()`** pour afficher un objet de type Domino. Par exemple, pour un domino , on retourne la chaîne "(A, Z)".
3. Ajoutez une méthode **`__getitem__()`** permettant d'obtenir la valeur de la 1^{er} ou de la 2^{ème} face du domino, afin d'écrire **`d[1]`** plutôt que **`d.getitem(1)`**.
 - Par exemple, pour un domino $d =$ , on retourne
 - i. la valeur 'Z' pour un appel **`d[1]`**
 - ii. la valeur 'V' pour un appel **`d[2]`**
4. Ajoutez une méthode **`__eq__()`** permettant de tester l'égalité de deux dominos afin d'écrire **`d1 == d2`** plutôt que **`d1.__eq__(d2)`**
5. Ajoutez une méthode **`extraitLst()`** qui à partir d'une liste de dominos (passé en paramètres), renvoie une nouvelle liste qui comporte tous les éléments de la liste exceptés la première occurrence du domino courant.

Partie B :

1. Définissez une classe **Puzzle** qui permette d'instancier des objets simulant un puzzle de dominos. Le constructeur **__init__()** de cette classe prend pour argument un domino et initialisera le 1^{er} élément du puzzle (liste de domino) par le domino passé en paramètre.

2. Ajoutez une méthode **__repr__()** pour afficher un objet de type Puzzle.

▪ Par exemple pour un Puzzle p =

H	K
---	---

K	A
---	---

A	Z
---	---

On retourne la chaîne ' [(H , K) (K , A) (A , Z)] '

3. Ajoutez une méthode **__getitem__()** permettant d'obtenir le domino de la i^{ème} position du Puzzle p, afin d'écrire **p[2]** plutôt que **p.__getitem__(2)**.

▪ Par exemple pour un Puzzle p =

H	K
---	---

K	A
---	---

A	Z
---	---

On retourne le domino

A	Z
---	---

 pour un appel p[2].

4. Ajoutez une méthode **is_possible_to_add()**, qui prend en argument un domino et qui teste si ce domino peut compléter le puzzle.

▪ Par exemple pour un Puzzle p :

H	K
---	---

K	A
---	---

A	Z
---	---

i. Et un domino d =

Z	V
---	---

 , **p.is_possible_to_add(d)** retourne **True**
ii. Et un domino d =

A	H
---	---

 , **p.is_possible_to_add(d)** retourne **False**

5. Ajoutez une méthode **get_list_domino()** qui extrait d'une liste de dominos (passée en argument) , la liste des dominos pouvant compléter le puzzle.

6. Pour calculer une solution possible du puzzle courant, l'idée est de développer une méthode récursive **construire_solution()** dont le seul argument est une liste des dominos.

Pour compléter notre puzzle, on calcule d'abord la liste des dominos pouvant compléter le puzzle, et pour chacun d'eux, on l'ajoute au puzzle et on appelle la même méthode avec pour argument la liste des dominos privée du domino choisi.