8 - La POO (suite)

Je m'étais rendu compte que seule la perception grossière et erronée place tout dans l'objet, quand tout est dans l'esprit...

Proust (le Temps retrouvé)

Objectifs :

Maîtriser la Programmation Orientée Objet.

Fichiers à produire : `TP8_1_m.py` `TP8_1.py` `TP8_2_m1.py` `TP8_2_m2.py` `TP8_2.py`.

8.1 -Programmation

Pour vous entraîner pour le partiel voici deux sujets qui ont déjà été proposés.

8.1.1 - Le jeu de Marienbad

Ce jeu – appelé également « jeu des allumettes » – nécessite deux joueurs et 21 allumettes.

Les 21 allumettes sont réparties en 6 tas, avec `i` allumettes dans le `i`^{e} tas : une allumette dans le premier tas, deux dans le deuxième, etc.

Chacun à son tour, les joueurs piochent dans un seul tas le nombre d'allumettes souhaité. *Le joueur qui prend la dernière allumette perd la partie.*

Votre programme devra faire jouer deux joueurs humains. Au cours du jeu, l'affichage se présentera simplement sous la forme suivante (si Bob est l'un des deux joueurs)

```
tas        : (1, 2, 3, 4, 5, 6)
allumettes : [0, 2, 3, 2, 4, 4]
-------- Prochain joueur : Bob --------
```

Le module de classe TP8_1_m

Ce module contient la classe `Marienbad` qui comprend :

- les attributs de la classe :
  - `tas` de type tuple d'entier,
  - `allumettes` de type liste d'entier,
  - `joueurs` de type tuple de chaîne de caractères,
  - `tour` un entier qui permet d'alterner les joueurs à chaque tour. (Indication : il sert d'indice à l'attribut `joueurs`).
- les méthodes :
  - un constructeur avec des valeurs par défaut adaptées (noms des joueurs : `"joueur1"` et `"joueur2"`),
  - `__str__()` qui affiche l'état du jeu en cours,
  - `verifie()` qui renvoie un booléen : vrai s'il est possible d'enlever `n` allumettes dans le tas `t`, faux sinon,
  - `enlever()` qui met à jour les tas en enlevant `n` allumettes dans le tas `t`,
  - `termine()` qui renvoie un booléen : vrai si le jeu est terminé, faux sinon.

Le module principal TP8_1

Dans le programme principal :

- Saisissez le nom des deux joueurs ;
- lancez le jeu (c'est une boucle tant que le jeu n'est pas terminé).

  À chaque tour vous devez :

  - afficher l'état du jeu,
  - demander au joueur en cours le numéro du tas `t` et le nombre d'allumettes `n` qu'il désire ôter,
  - vérifier si son choix est valide, sinon expliquer l'erreur et reposer la question,
  - supprimer `n` allumettes du tas `t`,
  - vérifiez si le jeu est terminé,
- annoncez le gagnant (ou le perdant).

8.1.2 - Les polynômes

Introduction

Un polynôme de degré n est de la forme :

$$a_0 + a_1x + a_2 x^2 + a_3 x^3 ... + a_n x^n$$

On va stocker les polynômes sous la forme de tuples : (a_0, a_1, a_2, a_3,..., a_n).

Par exemple le polynôme :

$$7 + 2.3\ x - 9.12\ x^2 + 7.8\ x^4$$

Est stocké sous la forme :

```
(7, 2.3, -9.12, 0, 7.9)
```

Un tuple représentant un polynôme doit contenir **au moins** une valeur, éventuellement nulle. Attention, un tuple à une seule valeur s'exprime avec une virgule (ex: `(3,)`).

Le module de fonctions TP8_2_m1

Ce module contiendra les fonctions suivantes :

`poly_calcul(p, x)` — Prend en paramètre un tuple polynôme `p` et une valeur numérique `x` et retourne l'évaluation numérique du polynôme avec cette valeur :

```
>>> poly_calcul((7,2,3),2)
23
```

`poly_coef(p)` — Prend en paramètre un tuple polynôme `p` et retourne le degré le plus élevé du polynôme ayant un coefficient non nul, ou 0 si tous les coefficients sont nuls ;

`poly_add(p1, p2)` — Prend en paramètres deux tuples polynômes `p1` et `p2`, et retourne le tuple polynôme correspondant à la somme de `p1` et `p2` (polynôme ayant comme coefficient pour chaque degré la somme des coefficients de `p1` et `p2` pour ce degré).

`poly_chaine(p)` — Prend en paramètre un tuple polynôme `p` et retourne une chaîne correspondant à une forme lisible de ce polynôme :

```
>>> poly_chaine((23.4, 1.1, 11.3, 0, 0, 4, 12.85))
"23.4+1.1x+11.3x^2+4x^5+12.85x^6"
```

`poly_simplifie(p)` — qui prend en paramètre un tuple polynôme `p` et retourne un tuple polynôme équivalent en ayant supprimé les coefficients nuls des degrés supérieurs au degré du polynôme :

```
>>> poly_simplifie((4, 0, 12, -2, 0, 4, 9.3, 0, 0, 0))
(4, 0, 12, -2, 0, 4, 9.3)
```

Le module de classe TP8_2_m2

Ce module définira la classe `Polynome`, et utilisera les fonctions du module de fonctions afin de définir les méthodes suivantes :

- méthode d'initialisation, permettant de créer une instance de la classe `Polynome` avec comme argument un tuple contenant les coefficients du polynôme tel que défini précédemment ;
- méthode de représentation textuelle, retournant la représentation d'un polynôme sous forme de chaîne lisible ;
- méthode d'addition, permettant d'ajouter une instance de la classe `Polynome` à une autre instance de cette classe, et retournant une nouvelle instance de `Polynome` correspondant à la somme des deux autres ;
- méthode `evaluer(self, x)` permettant de réaliser l'évaluation du polynôme avec la valeur `x` donnée et retournant le résultat de cette évaluation ;
- méthode `degre(self)` retournant le degré du polynôme.

Le module principal TP8_2

Ce module importe la classe `Polynome` et réalise les deux traitements suivant :

- Saisie des degrés d'un polynôme (on pourra utiliser la fonction standard `eval(`*`chaîne`*`)` afin de permettre à l'utilisateur de saisir l'ensemble des coefficients séparés par des virgules.
- Création d'un fichier texte `polyres.txt` contenant en première ligne la représentation textuelle lisible du polynôme, et dans les lignes suivantes, les valeurs entières de -10 à 10 suivies d'un caractère tabulation (`"\t"`) de l'évaluation de ce polynôme pour cet entier.

CQFP

- Synthèse de vos connaissances.
- ***Tout réviser pour le partiel !***