TP 5

Exercice 1. Écrivez une classe `Domino` pour représenter une pièce de domino. Les objet sont initialisés avec les valeurs des deux faces, A et B. Ajoutez une méthode `.affiche_points()` qui affiche les valeurs des deux faces, et une méthode `.totale()` qui retourne la somme de deux valeurs.

```
>>> d = Domino(4, 6)
>>> d.affiche_points()
face A: 4, face B: 6
>>> x = d.totale()
>>> print(x)
10
```

Exercice 2. Écrivez une classe `CompteBancaire`. Les objets sont initialisés avec le nom du titulaire et le solde. L'argument solde doit être facultatif et avoir une valeur prédéfinie zéro. Ajoutez deux méthodes `.depot(somme)` et `.retrait(somme)` pour changer le solde. Ajoutez une méthode `.affiche()` qui montre le solde courant.

```
>>> compte1 = CompteBancaire('Jean', 1000)
>>> compte1.retrait(200)
>>> compte1.affiche()
Le solde du compte de Jean est 800 euros.
>>> compte2 = CompteBancaire('Marc')
>>> compte2.depot(500)
>>> compte2.affiche()
Le solde du compte de Marc est 500 euros.
```

Exercice 3. *Generateurs.* Écrivez une classe `TableMultiplication` qui crée des objets initialisés avec un nombre entier. Ajouter une méthode `.prochain()` qui renvoie à chaque appel un nouveau terme de la table de multiplication correspondante.

```
>>> tab = TableMultiplication(3)
>>> tab.prochain()
0
>>> tab.prochain()
3
>>> tab.prochain()
6
>>> tab.prochain()
9
```

Exercice 4. Écrivez une classe `Fraction` qui crée des objets initialisés avec deux nombres entiers `.num` et `.denom` pour le numérateur et le dénominateur. Ajoutez une méthode `.affiche()` qui affiche une représentation de la fraction. Ajoutez des méthodes spéciales pour pouvoir utiliser les opérateurs `+`, `-`, `*`, `/`.

Pour réduire la fraction, vous pouvez employer la fonction `math.gcd(a, b)` du module `math`, qui calcule le plus grand commun diviseur entre deux entiers `a` et `b`.

```
>>> f = Fraction(3, 4)
>>> f.affiche()
3/4
>>> g = Fraction(1, 2)
>>> g.affiche()
1/2
>>> r1 = f + g
>>> r.affiche()
5/4
>>> r2 = f / g
>>> r2.affiche()
3/2
```

Exercice 5. Écrivez des méthodes spéciales `.__repr__()` adaptées pour les classes des exercices précédents.

Exercice 6. On se propose de représenter les polynômes en une indéterminée avec un nombre quelconque de termes :

$$c_0 + c_1 \times x + c_2 \times x^2 + \cdots + c_n \times x^n.$$

▷ Écrivez une classe `Poly` qui crée des objets initialisés avec un nombre arbitraire d'arguments (qui représentent les coefficients) et les stocke dans une liste `.coeff`. Par exemple, on représentera le polynôme $p(x) = 3 + 4x - 2x^2$ avec :

```
>>> p = Poly(3, 4, -2)
```

et on récupérera ses coefficients avec :

```
>>> p.coeff
[3, 4, -2]
```

▷ Ajoutez une méthode `.evalue(x)` qui renvoie la valeur du polynôme évalué à `x`. Par exemple, pour calculer $p(2) = 3 + 4 \times 2 - 2 \times 2^2 = 3$:

```
>>> p.evalue(2)
3
```

Essayez d'utiliser la fonction `enumerate()` pour itérer à la fois sur les coefficients et les exposants.

▷ *Bonus.* Ajoutez des méthodes spéciales pour les opérateur `+` et `*`, qui devront calculer respectivement la somme et le produit de deux polynômes. Par exemple, si $p1(x) = 1 + 4x$ et $p2(x) = 3 - 3x + 2x^2$ on aura

$$p3(x) = p1(x) + p2(x) = 4 + x + 2x^2$$
$$p4(x) = p1(x) * p2(x) = 3 + 9x - 10x^2 + 8x^3$$

```
>>> p1 = Poly(1, 4)
>>> p2 = Poly(3, -3, 2)
>>> p3 = p1 + p2
>>> p3.coeff
[4, 1, 2]
>>> p4 = p1 * p2
>>> p4.coeff
[3, 9, -10, 8]
```

Astuce : utilisez la fonction `zip_longest()` du module `itertools` pour itérer sur des objets de taille différente.