TD & TP

Les exercices suivants seront étudiés en travaux dirigés puis testés en travaux pratiques. Chaque exercice doit-être étudié sur papier et aboutir à l'équivalent d'un pseudo-code informatique qui sera retranscrit en python sur machine. Chaque exercice doit-être abondamment testé sur machine.
A la fin de chaque TP, les tests effectués et les réponses demandées seront transposées dans document clair et correctement présenté. Ce document doit être envoyé au format PDF par mail à l'enseignant référant du groupe de TP avant la prochaine séance de TD.
Merci de rédiger le champ objet de votre email selon le modèle suivant : **[PEIP 2.g] TP n** où g sera remplacé par votre numéro de groupe et p par le numéro du TP.

I. Les variables

1. Déterminez les variables et leurs valeurs à l'issue des instructions suivantes.

a.
```
a = 0
b = 1
ab = 2
b = 3
ab = 4
```

b.
```
a = 1
b = 2
c = b
b = a
```

c.
```
a = 1
b = 2
c = a
b = a
c = b - a
```

d.
```
x = 3
y = 3 * x + 1
y = 3
x = 3 * y + 1
```

e.
```
a = 2
b = 4
x = a
a = b
b = x
```

f.
```
a = 2
b = 4
c = b
b = a
a = c
```

2. Expliquez pourquoi les séries d'instructions suivantes ne sont pas valides.

a.
```
a = 1
b = 1
2 = a
b = 2
```

b.
```
a = 1
a = b
```

3. Echangez les valeurs de deux variables.

a. Proposez une méthode qui utilise une variable auxiliaire appelée t. On pourra, par exemple, choisir la valeur 3 pour a et 5 pour b.

b. En TP, testez votre méthode, entre des variables de même type et des variables de types différents. Commentez.

c. En TP, après avoir pris soin de définir vos variables, testez l'instruction suivante : **a , b = b , a**. Commentez le résultat.

4. Déterminez les valeurs finales pour les variables m et n.

a.
```
m = 5
m = m + 1
n = m * 2
```

b.
```
m = 5
m, n = m + 2, m * 2
```

II. Premiers programmes

1. Ecrivez un programme qui calcule le prix TTC d'un montant HT donné. Le taux de TVA appliqué sera de 20%.

2. Nous souhaitons transporter nos 7 millions d'euros trouvés dans le jardin jusqu'à la banque. Pour prévoir un moyen de transport adapté, écrivez un programme qui calcule le poids du colis sachant qu'un billet de 200 euro pèse 1.1 grammes et que nous avons trouvé uniquement des billets de 200 euro.

3. Ecrivez un programme qui, pour une somme donnée en euros, sans centimes, affiche le nombre minimal de billets et de pièces nécessaires pour la composer (on utilisera le quotient et le reste de la division entière).
   Par exemple : 1949 = 3 x 500 + 2 x 200 + 0 x 100 + 0 x 50 + 2 x 20 + 0 x 10 + 1 x 5 + 2 x 2 + 0 x 1

III. Conditions & Boucles

1. Ecrivez un programme qui calcule et affiche les solutions d'une équation du second degré.

2. Reprenez le programme prixHT -> prixTTC pour qu'il répète autant de fois que nécessaire les opérations suivantes :
   - lecture au clavier d'un prix HT
   - calcul du prix TTC correspondant
   - affichage de ce prix

   La répétition sera arrêtée lorsqu'un prix négatif sera saisi. Le calcul sera alors remplacé par le message *Fin du programme*.

3. Reprenez le programme précédent afin qu'il affiche également le nombre de calculs qui ont été effectués et la somme des prix TTC.

4. **Calcul de factorielle** : écrivez un programme qui calcule itérativement, en utilisant une boucle for pour cet exercice, la valeur de la factorielle d'un nombre.

5. **Calcul de la monnaie** : reprenez le programme qui calcule la monnaie en utisant une liste qui contient les « types de monnaies » disponibles et une boucle for.

6. **Jeu de dés** : ce jeu se joue à deux joueurs, le gagnant étant celui qui le premier atteint 50 points.

Un tour se déroule comme suit :
Un joueur lance le dé et rejoue tant qu'il n'obtient pas le chiffre 1.
A chaque lancé, le nombre de point du joueur évolue ainsi :
- Si c'est un chiffre pair (2, 4 ou 6), le nombre de points augmente du chiffre obtenu.
- Si c'est le 3, le nombre de points est multiplié par deux.
- Si c'est le 5, le nombre de points est diminué de deux.

L'autre joueur prend donc la main selon les mêmes règles quand le chiffre 1 est tiré.

Réalisez un programme qui permet de jouer à ce jeu.

En TP, vous aurez besoin d'utiliser la fonction randint(min, max) pour simuler le lancé de dé. Consultez la documentation et tester cette fonction pour vous assurer qu'elle produit bien le résultat que vous attendez.

7. **Le nombre mystère** : l'utilisateur doit trouver un nombre choisi aléatoirement entre 0 at 100 par l'ordinateur, en moins de 10 essais. L'ordinateur indique, pour chaque proposition, si le nombre à trouver est inférieur ou supérieur à la valeur proposée.

Réalisez un programme qui permet de jouer à ce jeu.

Quelle technique de jeu employez-vous pour trouver ce nombre le plus rapidement possible ?

En voici deux probables :
1. Déterminer dans quel intervalle [a, b] se trouve le nombre caché et proposer un entier au hasard dans cet intervalle.
2. Déterminer dans quel intervalle [a, b] se trouve le nombre caché et proposer un entier « proche » du milieu de cet intervalle.

Inversons maintenant les rôles : l'utilisateur choisit le nombre et l'ordinateur doit le trouver. Réaliser le programme qui utilise la seconde technique.

IV. Manipulations de chaines, listes

1. Réalisez un programme qui à partir d'une liste de nombre **listePositions** et d'une chaine de caractères **s** affiche la chaine de caractères dont les positions dans **s** figurent dans **listePositions**.

Par exemple :
si listePosition = [2, 3, 8, 9]
et s = 'AZERTYUIOP'
le programme produit l'affichage : EROP

2. Réalisez un programme (avec une boucle) qui, à partir d'une liste de nombres donnée, affiche le couple formé par le plus grand élément de la liste et sa position.

3. Réalisez le programme ci-dessus avec les primitives max et index de Python.

4. Réalisez un programme qui, à partir d'une liste de nombres donnée, remplace toute suite d'éléments consécutifs par une seule occurrence de cet élément.

Par exemple :
si l = [1, 1, 1, 5, 6, 9, 50, 4, 4, 4, 1, 1]
le programme produit : [1, 5, 6, 9, 50, 4, 1]

5. Réalisez le programme ci-dessus en utilisant la méthode pop de Python.

V. Fonctions

Jeu du morpion : Le morpion est un jeu de réflexion à deux joueurs. Il se joue sur une grille carrée de 3x3 cases. Les joueurs doivent remplir chacun leur tour une case de la grille avec le symbole qui leur est attribué : O (rond) ou X (croix). Le gagnant est celui qui aligne en premier trois de ses symboles, horizontalement, verticalement ou en diagonale.

1. Donnez les grandes étapes nécessaires au bon déroulement du jeu.
2. Ecrivez
   - la fonction **initialiserGrille** qui retourne une grille initialisée. La taille de la grille sera définie par une variable TAILLE_GRILLE fixée à 3 en début de script.
   - la fonction **afficherGrille** qui prend une grille en paramètre et l'affiche selon le modèle

```
    0   1   2
    _   _   _
0 | O | X | X |
    _   _   _
1 |   | X | X |
    _   _   _
2 | O |   | O |

Joueur 2 (rond), indiquez la case où déposer votre symbole (l,c): 2,1
```

   suivant :
   - la fonction **grillePleine** qui prend une grille en paramètre et retourne un booléen indiquant si la grille est pleine.
   - la fonction **estLibre** qui prend en paramètres une ligne, une colonne, une grille et retourne un booléen qui indique si la case en disponible.
   - la fonction **placerPion** qui prend en paramètres le type de symbole à déposer, une ligne, une colonne, une grille et qui retourne une grille modifiée.
   - la fonction **coupGagnant** qui prend en paramètre une ligne, une colonne et la grille et qui retourne un booléen indiquant si le coup est gagnant.
3. A l'aide du travail réalisé ci-dessus, écrivez le programme principal qui permet de jouer au morpion.

Variante au jeu du morpion : Nous souhaitons apporter une nouvelle fonctionnalité au jeu du morpion. Cette fonctionnalité est le « coup double » et permet de faire rejouer un joueur juste après son coup si celui-ci a déposé son symbole sur une case « coup double » de la grille.

Pour cela, vous utiliserez une grille supplémentaire indiquant si la case est une case « coup double » ou non. Le nombre de cases « coup double » sera fixé à 2 par une constante en début de programme et ces cases seront choisies aléatoirement avant le début du jeu.

Conservez le script écrit à l'exercice précédent et travaillez sur une copie du fichier pour la suite.

```
Joueur 2 (rond), indiquez la case où déposer votre symbole (l,c): 2,2
COUP DOUBLE: Même joueur, joue encore!!!
    0   1   2
    _   _   _
0 | X |   | O |
    _   _   _
1 |   | X |   |
    _   _   _
2 |   |   | O |
Joueur 2 (rond), indiquez la case où déposer votre symbole (l,c): 1,2
```

figure 1. **Morpion avec « coup double »**

1. Modifiez la fonction **initialisationGrille** afin qu'elle retourne un tuple de grilles initialisées. La première grille est la grille de jeu, la seconde celle qui indiquera si une case est « coup double ».
2. Ecrivez une fonction **placerCoupDouble** qui prend en paramètre la grille « coups doubles » et la retourne avec deux « coups doubles » placés aléatoirement.
3. Ecrivez une fonction **coupDouble** qui prend en paramètres une ligne, une colonne, la grille des « coups doubles » et retourne un booléen indiquant si le coup est un « coup double ».
4. Modifiez votre programme pour jouer à cette variante du jeu.

VI. Quelques petits problèmes

Carré magique : un carré magique est un carré contenant des valeurs successives, à partir de 1, rangées de telle façon que les sommes de chaque ligne, de chaque colonne et des deux diagonales principales soient égales.

Ecrire un programme permettant de créer des carrés magiques en utilisant la méthode suivante (qui ne marche que pour des carrés de taille impaire) :

- Ranger 1 dans la case juste en dessous du centre
- Les éléments suivants sont rangés, en ordre croissant, en diagonale en dessous à droite
- Si on rencontre un bord, on continue de la même façon sur le bord opposé
- Si on rencontre une case déjà remplie, on passe dans la case en diagonale en dessous à gauche, si elle est elle-même occupée, on passe à la case en dessous à droite et ainsi de suite jusqu'à trouver une case vide

Drapeau : dans des cases alignées et numérotées de 1 à n sont rangées n boules de couleur rouge, bleue ou blanche. Il s'agit de regrouper en tête toutes les boules bleues et en queue toutes les boules rouges, en procédant par échanges successifs du contenu de cases prises deux à deux, en faisant le moins d'échanges possibles, et en n'utilisant pas de cases supplémentaires.

Ecrire un programme qui respecte l'invariant suivant (les entiers r1, r2 et r3 étant des variables du programme) :

- Si $1 \leq i < r1$ alors couleur(i) = bleu
- Si $r1 \leq i < r2$ alors couleur(i) = blanc
- Si $r3 < i \leq n$ alors couleur(i) = rouge

Pour cela :

- Déterminer les valeurs initiales que doivent prendre les variables r1, r2 et r3 pour que l'invariant soit vérifié a priori
- Déterminer dans quelle condition (relative à r1, r2 et r3) le problème est résolu
- Mettre en place les itérations qui feront progresser vers la solution tout en préservant l'invariant

Min-Max : étant donnée une matrice A[M][N], on dit qu'un point (i, j) est un min-max pour cette matrice si la valeur A[i][j] est minimum pour la ligne i et maximum pour la colonne j, c'est à dire :

$$A[i][j] = \text{Min } \{A[i][p], 0 \leq p \leq N-1\} = \text{Max } \{A[q][j], 0 \leq q \leq M-1\}$$

Ecrire un programme qui trouve l'ensemble des éléments vérifiant cette propriété, en effectuant le travail suivant pour chaque ligne :

- Trouver les minimas de la ligne i et empiler les numéros de colonnes correspondants
- Pour chacun de ces numéros de colonnes j, vérifier si A[i][j] est un maximum pour la colonne j

VII. Récursivité

Mystère

Ecrire une fonction récursive « mystere(n) » où n est positif ou nul, qui affiche à l'écran la suite *n n-1 … 0 0 … n-1 n*, par exemple *mystere(4)* affiche :

4 3 2 1 0 0 1 2 3 4

Puissance

Ecrire une fonction récursive « puissance(x, n) » qui calcule x^n en se basant sur la définition suivante :

$x^0 = 1$
$x^n = (x^2)^{n/2}$ si n est pair
$x^n = x\ x^{n-1}$ si n est impair

Palindrome

Ecrire une fonction récursive « palindrome(s) » qui renvoie *True* si la chaine de caractères *s* est un palindrome, *False* sinon.
NB : un palindrome est une chaine de caractères qui peut se lire indifféremment dans les deux sens.

Nombres impairs

Ecrire une fonction récursive « nombres_impairs(l) » qui renvoie une liste constituée des nombres impairs contenues dans la liste numérique *l*.

Tri par insertion

Pour trier *n* entiers rangés dans une liste, il suffit de trier les *n-1* premiers entiers puis d'insérer le $n^{ème}$ à sa place dans les *n-1* déjà triés.
Ecrire une fonction récursive « trier_insertion(l) » selon l'algorithme ci-dessus (la fonction auxiliaire « inserer » sera également récursive).

Tri par fusion

Pour trier *n* entiers rangés dans une liste, il suffit de trier chaque moitié de la liste puis de les fusionner.
Ecrire une fonction récursive « trier_fusion(l) » selon l'algorithme ci-dessus (la fonction auxiliaire « fusionner » sera également récursive).

VIII.Introduction à la Programmation Fonctionnelle

Fonctions en arguments

Ecrire une fonction récursive « zero(f, eps, a, b) » qui renvoie le zéro *z* d'une fonction continue *f* strictement monotone entre *a* et *b*, en effectuant une recherche dichotomique.
Comme paramètres de la fonction *zero* sont également donnés deux réels *a* et *b* vérifiant $a < z < b$, et un réel positif petit *eps* qui vérifiera pour finir $|f(z)| < eps$.
Tester la fonction *zero* avec la fonction *cos* puis avec la fonction $x \rightarrow 2x^2 - 4$, dans les deux cas avec $eps = 0.001$, $a = 0$ et $b = 3$.

Fonctions d'ordre supérieur

a. Ecrire une fonction récursive « longueurs_chaines(l) » qui renvoie, étant donnée une liste de chaînes de caractères *l*, la liste des longueurs respectives de ces chaînes.

b. Réécrire la question précédente en utilisant la fonction « map »

c. En utilisant la fonction « filter », écrire une instruction qui renvoie, étant donnée une liste de chaînes de caractères, la liste des chaînes de longueur supérieure à 3.

d. En utilisant les listes en compréhension, écrire une instruction qui renvoie la liste des éléments définis par $\{x^2 \mid x \in \mathbb{N} \text{ et } x < 20 \text{ et } x \text{ pair}\}$

e. Ecrire une fonction récursive « intersection(l, m) » qui renvoie la liste des éléments communs aux deux listes *l* et *m*.

f. Réécrire la question précédente en utilisant la fonction « filter »

g. Soit une matrice *mat* de dimension *N*x*M* implémentée par une liste de *N* listes (les lignes) de longueur *M* (les colonnes). En utilisant les listes en compréhension imbriquées, construire une matrice dont les lignes et les colonnes sont échangées par rapport à la matrice *mat*.

Décorateurs

Reprendre votre programme du jeu de morpion de façon à donner une « trace » de l'exécution (affichage de chaque fonction appelée avec la liste de ses arguments) en utilisant un décorateur.

NB : l'attribut *__name__* d'une fonction permet de récupérer son nom sous forme de *string* tandis que la notation **args* permet de désigner un nombre variable d'arguments pour une fonction.

IX. Classes - Objets

Pile : Créer une classe Pile permettant de gérer une pile de nombres entiers (Last In, First Out).

Pour cela :

- on définira un constructeur à un argument qui précisera le nombre maximum d'éléments de la pile
- les nombres entiers seront conservés dans une liste
- le nombre maximum et le nombre courant d'éléments seront stockés dans deux autres attributs
- on définira la méthode booléenne pleine (resp. vide) qui vérifie si la pile est pleine (resp. vide)
- on définira une méthode empiler à un argument entier qui ajoute l'entier sur la pile
- on définira une méthode depiler sans argument qui renvoie le sommet de pile tout en dépilant
- on définira une méthode getCourant sans argument qui renvoie le sommet de pile

Utilisation : Créer un objet p de la classe Pile, empiler et depiler des nombres entiers.

File : Quelle serait la modification à faire à la classe Pile pour la transformer en une classe File (First In, First Out) ?

Formes

- Créer une classe Point décrivant un point du plan par deux coordonnées entières.
- Donner un constructeur et une méthode additionner prenant un argument Point et rendant un Point résultat de l'addition (coordonnée à coordonnée) de l'argument avec l'objet de référence.
- Redéfinir la méthode __str__ de façon à afficher les coordonnées d'un Point sous forme de couple. Tester ainsi la fonction print sur un objet Point.
- Créer une classe Segment décrivant un segment du plan par un point origine et un point final.
- Donner un constructeur et deux méthodes d'accès à ses attributs.
- Créer une classe Cercle décrivant un cercle du plan par un point origine et un rayon.
- Donner un constructeur et deux méthodes d'accès à ses attributs.

Dans cette question on utilisera la classe Pile de l'exercice 1. qui gèrera une pile de Point (et non de nombres entiers). Définir dans chacune des classes Segment et Cercle les deux méthodes suivantes :

- Dessiner avec un argument Pile dont le rôle est d'afficher le type de forme concerné ainsi que ses attributs Point translatés par la transformation courante donnée par le sommet de pile.
- Deplacer avec un argument Point dont le rôle est d'ajouter l'argument Point aux attributs Point de la classe concernée.

Tester ce programme pour voir s'il correspond bien à vos attentes (créer une pile, empiler des Point, créer, dessiner et déplacer un Segment, dépiler un Point de la pile, créer, dessiner et déplacer un Cercle)

X. Classes abstraites – Héritage - Polymorphisme

1. Modifier le programme du TD/TP n°7 de façon à introduire une classe abstraite *Forme* mère de *Segment* et *Cercle* (*Forme* ne contiendra à ce stade que les deux méthodes abstraites *dessiner* et *deplacer*)

2. Créer une classe *Image*, fille de *Forme* et collection de plusieurs segments, cercles et autres formes que l'on pourrait introduire plus tard. Décrire une *Image* par un point origine, une liste des formes qui la composent et un nombre de formes qui la composent. Donner un constructeur avec un argument *Point*. Définir la méthode *ajouter* qui ajoute une forme à l'image, *dessiner* dont on extrapolera le rôle et *deplacer*.

3. Tester le programme précédent de la façon suivante :
    a. Créer une pile de *Point* et une image contenant un segment et un cercle
    b. Dessiner cette image puis la déplacer d'un point donné
    c. Dessiner à nouveau cette image
    d. Créer une seconde image contenant un autre segment et un autre cercle
    e. Ajouter cette seconde image à la première image
    f. Dessiner à nouveau la première image

4. Modifier le programme précédent de la façon suivante :
    a. Enrichir la classe *Forme* par un point origine (car ce descripteur est commun à toutes les sous-classes), un constructeur et un accès en lecture au point origine
    b. Définir la méthode *deplacer* de *Forme* de façon à ce qu'elle exprime le comportement commun à toutes les sous-classes et reporte le code spécifique dans chaque sous-classe
    c. Modifier en conséquence les sous-classes (en particulier en implémentant la méthode abstraite relative au code spécifique)
    d. Tester ce nouveau programme