Université Hassan 1er
École Nationale des Sciences Appliquées – Berrechid

POLYCOPIE DES TRAVAUX PRATIQUES

PROGRAMMATION PYTHON

PROFESSEUR : LAHCEN MOUMOUN

Sommaire

Introduction

Python est un langage polyvalent et intuitif, largement utilisé dans des domaines variés tels que le développement web, l'analyse de données et l'intelligence artificielle. Son adoption croissante est due à sa syntaxe claire et accessible, ce qui en fait un excellent choix pour les débutants et un outil puissant pour les développeurs expérimentés.

Ce polycopié a pour objectif d'accompagner les étudiants dans leur apprentissage du langage Python, en leur proposant une série de travaux pratiques. Ils y découvriront comment exploiter les concepts fondamentaux de la programmation tout en développant leurs compétences en algorithmique et en manipulation de données.

TP N°1 : Les éléments de base du langage Python

Exercice n°1

Un triplet d'entiers naturels strictement positifs (a, b, c) est appelé "triplet de Pythagore" si $a^2+b^2 =c^2$

Écrire une fonction « havePythagoricianFriend(a,b,m,n) » qui prend quatre entiers naturels a, b, m et n en paramètre, et qui retourne True s'il existe un entier c tel que $m \leq c \leq n$ et tel que le triplet (a,b,c) soit un triplet de Pythagore.

Par exemple, « havePythagoricianFriend(3,4,1,10) » retourne True et «havePythagoricianFriend(3,4,1,4)» retourne False.

Exercice n°2

Soit la séquence nucléotidique suivante :

ACCTAGCCATGTAGAATCGCCTAGGCTTTAGCTAGCTCTAGCTAGCTG

En utilisant un dictionnaire, écrire un programme qui répertorie tous les mots de 2 lettres (successifs) qui existent dans la séquence (AA, AC, AG, AT, etc.) avec leur nombre d'occurrences puis qui les affiches à l'écran.

Exercice n°3

Sachant qu'une année bissextile est divisible par 4 (mais non par 100) ou par 400 et que le mois de février correspondant compte 29 jours.

- Écrire la fonction qui prend un entier correspondant à une année en paramètre et qui retourne vrai ou faux si celle-ci est ou non bissextile.
- En utilisant les listes, écrire une deuxième fonction qui retourne le nombre de jours à partir de deux paramètres : le nom du mois et l'entier correspondant à l'année.
- Réécrire cette deuxième fonction en utilisant le dictionnaire dnbjm = {'janvier': 31, 'février': 28, ...}

Elément de correction

Exercice n°1

```
def havePythagoricianFriend(a,b,m,n):
    for c in range (m,n+1):
        if (a**2+b**2)==c*2:  return True
    return False
```

Exercice n°2

```
phrase=input("donner une phrase  :")
dictMot=dict()
for i in range(1, len(phrase)):
   mot=phrase[i - 1:i + 1]
   if mot in dictMot: dictMot[mot]+= 1
   else:  dictMot[mot]=1
print(dictMot)
```

Exercice n°3

```
def bissextile (annee):
    if annee % 400==0:  return True
    if annee % 100==0:  return False
    if annee % 4==0:    return True
    return False
def nbJour(mois, annee):
    lnj=[31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    lnm=['janvier','février','mars','avril','mai','juin','juillet',
          'août', 'septembre', 'octobre', 'novembre','décembre']
       if mois not in lnm:  return -1
       indx=lnm.index(mois)
       if indx!=1 or not bissextile(annee):  return lnj[indx]
       return 29
    def nbJour2(mois, annee):
       dnbjm={'janvier':31,'février':28,'mars':31,'avril':30,'mai':31,
         'juin':30,'juillet':31,'août':31,'septembre':30,'octobre':31,
         'novembre':30,'décembre':31}
       if mois not in dnbjm: return -1
       if mois=='février' and bissextile(annee): return 29
       return dnbjm[mois]
    m=input("Donner mois  : ")
    an=int(input("Donner année  : "))
    print(nbJour2(m,an))
```

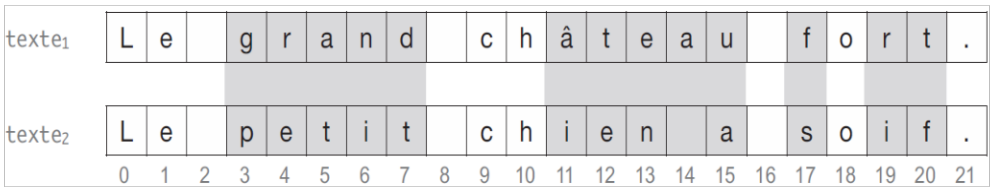
TP N°2 : Dictionnaire , liste et fonctions

Dans ce TP, on s'intéresse à des textes de grande taille auxquels plusieurs auteurs apportent des modifications au cours du temps. Ces textes peuvent par exemple être des programmes informatiques développés par de multiples auteurs. Il est important de pouvoir efficacement gérer les différentes versions de ces programmes au cours de leur développement et limiter le stockage et la transmission d'informations redondantes. nous traitons le problème avec une hypothèse simplificatrice : les textes comparés ont toujours la même taille.

1. Si deux textes ne sont pas égaux mais ont la même longueur n , on souhaite compter le nombre de positions qui diffèrent, c'est à dire déterminer combien il existe de positions i ($0 \leq i < n$) telles que les caractères en position i sont différents dans les deux textes.

Écrire une fonction distance (texte1, texte2) qui calcule cette quantité.

2. Nous introduisons maintenant une structure de données tranche pour représenter un différentiel par positions fixes entre deux textes. La figure suivante présente un exemple de couple de textes (texte1, texte2) qui diffèrent sur 4 tranches (représentées par des zones grisées sur la figure). En dehors des tranches, les textes sont égaux.



Une tranche est un dictionnaire avec trois clés ‘debut’, ’avant’ et ‘apres’. La valeur associée à la clé ’debut’ est le premier indice de la tranche, les textes associés aux clés ’avant’ et ‘après’ représentent les textes (de même longueur) des tranches respectives des textes à comparés.

Un différentiel est une liste (potentiellement vide) de tranches [tr1, ..., trk] représentant des modifications touchant des zones distinctes de 2 textes.

On peut représenter le différentiel de la Figure ci-dessus, par la liste suivante :

[{‘debut’ : 3, ’avant’ : ’grand’, ‘apres’ : ’petit’) },

{‘debut’ : 11, ’avant’ : ’âteau’ , ‘apres’ : ’ien a’) },

{‘debut’ : 17, ’avant’ : ’f’ , ‘apres’ : ’s’) },

{‘debut’ : 19, ’avant’ : ’rt’, ‘apres’ : ’if’) }]

Écrire une fonction différentielle(texte1, texte2) qui calcule le différentiel du texte texte2 vis-à-vis du texte texte1, supposés de même longueur.

Elément de correction

1

```
def distance(texte1, texte2) :
    n = len(texte1) # Longueur commune
    dist = 0
    for i in range(n) :
        if texte1[i] != texte2[i] :
            dist += 1
    return dist
```

2

```
def tranche(arg_debut, arg_avant, arg_apres):
    return {'début': arg_debut, 'avant': arg_avant, 'après': arg_apres}
def différentiel(texte1, texte2) :
    n = len(texte1)
    egalcar = True
    arg_debut = 0
    arg_avant = []
    arg_après = []
    out = []
    for i in range(n) :
        if egalcar :
            if texte1[i] != texte2[i] :
                arg_debut = i
                arg_avant.append(texte1[i])
                arg_après.append(texte2[i])
                egalcar = False
        else :
            if texte1[i] != texte2[i] :
                arg_avant.append(texte1[i])
                arg_après.append(texte2[i])
            else :
                out.append(tranche(arg_debut, arg_avant, arg_après))
                arg_avant = []
                arg_après = []
                egalcar = True
    if not egalcar :
        out.append(tranche(arg_debut, arg_avant, arg_après))
    return out
```

TP N°3 :Classes et objet en Python

Exercice n°1

Considérons une classe Hor des objets qui représentent le temps horloge, avec trois attributs entiers : heure, minute, seconde (ces attributs sont inaccessibles).

1. Écrire le constructeur de la classe « Hor » en respectant les consignes suivantes :
    - Les arguments correspondants aux différents attributs de la classe sont considérés facultatifs
    - Il faut prévoir des assertions pour le test de validation des valeurs de attributs fournis (le nombre de minutes ou de secondes ne peut ni être négatif ni dépasser 59, le nombre d'heures peut être négatif)
2. Définir une méthode spéciale de surcharge de l'opérateur d'addition '+' (à gauche ou à droite) de deux objets Hor. Cette méthode doit toujours normaliser le résultat pour que le nombre de minutes ou de secondes ne doit jamais dépasser 59.

Exercice n°2

La classe « Robot » modélise l'état et le comportement de robots virtuels, Elle est modélisée par le diagramme suivant :

Robot
- nom // nom du robot : chaîne de caractères - position // position : dictionnaire d'entiers x et y - direction // direction suivi : valeurs "Nord", "Est", "Sud" ou "Ouest"
+ __init__ (…) + __add__(pas) + droite () + __str__ () // retourner l'état du robot : nom, position et direction …

- La méthode «__add__ » permet avancer selon la direction du pas : x augmente (y reste fixe) en allant vers l'Est, x diminue (y reste fixe) en allant vers l'Ouest, y augmente (x reste fixe) en allant vers le Nord et y diminue (x reste fixe) en allant vers le Sud.

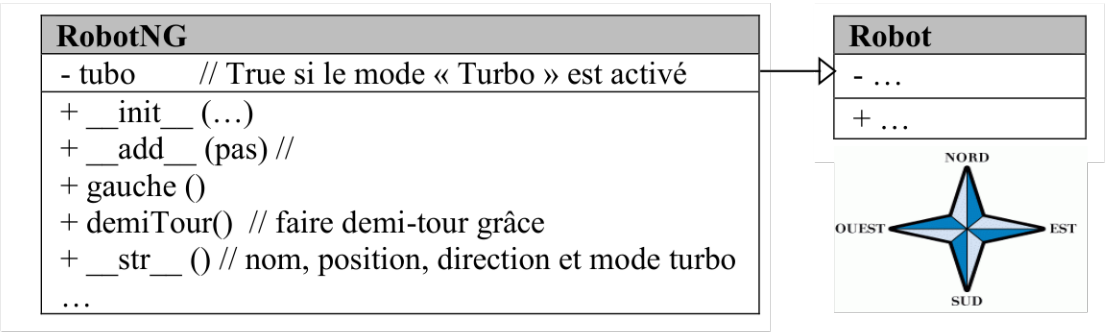
- La méthode « droite » permet de tourner à droite de 90° pour changer de direction (si sa direction était "Nord" elle devient "Est", si c'était "Est" elle devient "Sud", etc.). Les robots ne peuvent pas tourner à gauche.

1. Écrire le constructeur de la classe « Robot » sachant que le nom est obligatoire mais on peut ne pas spécifier la position et la direction, qui sont définis par défaut respectivement à {x =0, y= 0 } et "Est" (seul le nom peut être librement consulté alors que les deux autres attributs sont inaccessibles) ; il faut prévoir des assertions pour le test de validation des valeurs de ces variables d'instances.

2. Donner les éléments à ajouter à la classe « Robot » pour accéder à ses attributs inaccessibles

3. Définir une méthode spéciale de surcharge de l'opérateur d'addition '+' (à gauche) d'un pas

4. On considère une classe RobotNG (Robot Nouvelle Génération), dérivée de la classe robot, dont le diagramme est le suivant :



- La méthode « __add__ » permet d'avancer d'un pas donné (chaque pas est multiplié par 3 si le mode turbo est activé)

- La méthode « gauche » permet de tourner à gauche de 90° pour changer de direction (si sa direction était "Nord" elle devient "Ouest", si c'était "Ouest" elle devient "Sud", etc.).

Donner le code du constructeur de la classe « RobotNG »

5. Proposer un programme de test des différentes fonctionnalités des classes « robot » et « RobotNG »

Elément de correction

Exercice 1

```
class Hor:
        def __init__(self,heure=0, minute=0, seconde =0):
            assert heure>=0 and minute>=0 and seconde>=0
            assert minute <60 and seconde<60
            self.__heure=heure
            self.__minute=minute
            self.__seconde=seconde
        def __add__ (self , objet_a_ajouter ):
            nouvelle_Hor = Hor ()
            nouvelle_Hor.__minute = self.__minute
            nouvelle_Hor.__seconde = self.__seconde
            nouvelle_Hor.__seconde += objet_a_ajouter
            if nouvelle_Hor.__seconde >= 60:
                nouvelle_Hor.__minute += nouvelle_Hor.__seconde // 60
                nouvelle_Hor.__seconde = nouvelle_Hor.__seconde % 60
            return nouvelle_Hor
        def __radd__ (self , objet_a_ajouter ):
            return self.__add__(objet_a_ajouter)
        def __iadd__ (self , objet_a_ajouter ):
            nouvelle_Hor = objet_a_ajouter
            nouvelle_Hor.__heure += self.__heure
            nouvelle_Hor.__minute += self.__minute
            nouvelle_Hor.__seconde += self.__seconde
            if nouvelle_Hor.__seconde >= 60:
                nouvelle_Hor.__minute += nouvelle_Hor.__seconde // 60
                nouvelle_Hor.__seconde = nouvelle_Hor.__seconde % 60
            if nouvelle_Hor.__minute >= 60:
                nouvelle_Hor.__heure += nouvelle_Hor.__minute // 60
                nouvelle_Hor.__minute = nouvelle_Hor.__minute % 60
            return nouvelle_Hor
```

Exercice 2

```
class Robot:
        listDir=["Nord", "Est", "Sud" ,    "Ouest"  ]
        def __init__(self, nom, x=0,y=0,direction="Est"):
            assert isinstance(nom, str) and len(nom)!=0
            assert isinstance(x, int)
            assert isinstance(y, int)
            assert isinstance(direction, str) and direction in
   Robot.listDir
            self.nom=nom
            self.__position={"x":x, "y":y}
            self.__direction=direction
        def __str__(self):
            return "Nom: {0} Position : {1}  Direction: {2}".format(
                self.nom, self.__position.items(), self.__direction)
        def _getPosition(self):
            return self.__position
        def _setPosition(self, position):
            self.__position = position
        position = property(_getPosition(), _setPosition())
```

```
        def _getDirection(self):
            return self.__direction
        def _setDirection(self, direction):
            self.__direction = direction
        direction = property(_getDirection(), _setDirection())
        def __add__(self, pas):
            if self.direction=="Est":
                self.__position[0]+=pas
            elif self.direction=="Ouest":
                self.__position[0]-=pas
            elif self.direction == "Nord":
                self.__position[1] -= pas
            elif self.direction == "Sud":
                self.__position[1] += pas
    class RobotNG(Robot):
        def __init__(self, nom, x=0,y=0,direction="Est", turbo=False):
            super(RobotNG, self).__init__(nom, x,y, direction)
            self.turbo=turbo
```

TP N°4 : Classes et recharge d'opérateurs

Nous considérons un polynôme de degré n est de la forme :

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 \ldots + a_n x^n$$

Un polynôme est stocké sous la forme d'une List : [$a_0, a_1, a_2, a_3 \ldots, a_n$]. Par exemple le polynôme : $7 + 2.3x - 9.12 x^2 + 7.8 x^4$ est stocké sous la forme : [7, 2.3, -9.12, 0, 7.9]

Une liste représentant un polynôme doit contenir au moins une valeur, éventuellement nulle.

A. On désire une classe Polynomial permettant de les manipuler de manière « naturelle » les polynômes. Cette classe doit contenir les méthodes suivantes :

- Méthode d'initialisation, permettant de créer une instance de la classe Polynome avec comme argument une list contenant les coefficients du polynôme tel que défini précédemment ;
- Méthode de représentation textuelle, retournant la représentation d'un polynôme sous forme de chaîne lisible. Par exemple pour le polynôme [23.4, 1.1, 11.3, 0, 0, 4, 12.85] , il faut retourner "23.4+1.1x+11.3x^2+4x^5+12.85x^6"
- 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 degré le plus élevé du polynôme ayant un coefficient non nul, ou 0 si tous les coefficients sont nuls)
- Méthode simplifie(self) qui retourne un polynôme équivalent en ayant supprimé les coefficients nuls des degrés supérieurs au degré du polynôme :

  Exemple simplifie([4, 0, 12, -2, 0, 4, 9.3, 0, 0, 0]) retourne [4, 0, 12, -2, 0, 4, 9.3]
- 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 scalaire(self, x) qui renvoie le polynome p multiplié par l'entier x
- Méthode coefmonome(self, n) qui renvoie le ceofficient a du monôme aX^n

B. Il faut maintenant compléter la définition de la classe avec les fonctions (à tester bien sûr):

- __sub__(self,other) pour définir l'action de l'opérateur - (soustraction).
- __mul__(self,other) pour définir l'action de l'opérateur * (multiplication).
- __pow__(self,other) pour définir l'action de l'opérateur ** (exponentiation). Tester cette méthode par un affichage des puissances successives du polynôme (X + 1) et voir apparaître le triangle de Pascal.
- __eq__(self,other) pour vérifier l'égalité de deux polynômes (opérateur ==).
- __repr__(self,other) pour définir le rendu en chaîne de caractères.
- derive(self) pour définir la dérivation.
- primitive(self) qui renvoie une primitive du polynôme p
- eval_horner(self,x) pour évaluer le polynôme en x avec la méthode de Horner.Cette méthode permet de calculer la valeur d'un polynôme de degré n en ne faisant que n additions et n multiplications au lieu de n(n+1)/2 . La méthode consiste à multiplier le coefficient de plus haut degré par x, puis à lui ajouter le second coefficient. On multiplie le résultat par x, auquel on ajoute le troisième coefficient, etc. Par exemple le polynôme $-2X^3 + 3X^2 - 5X + 6$ peut s'écrire sous la forme $(((-2)X + 3)X - 5)X + 6$.

C. Reprendre les questions précédentes en considérant que le polynôme est représenté par un dictionnaire : Le degré et le coefficient d'un monôme correspondent respectivement à la clé et à la valeur d'un élément du dictionnaire.

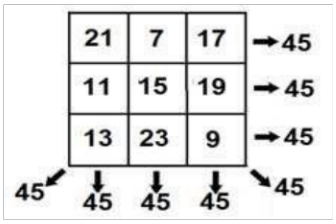
TP N°5 : Liste de liste et tableau NumPy

Un **carré magique** d'ordre n (n est entier strictement positif) est une matrice carrée d'ordre n (n lignes et n colonnes), qui contient des nombres entiers strictement positifs. Ces nombres sont disposés de sorte que les sommes sur chaque ligne, les sommes sur chaque colonne et les sommes sur chaque diagonale principale soient égales. La valeur de ces sommes est appelée : constante magique.

Un **carré magique normal** d'ordre n est un carré magique d'ordre n, constitué de tous les nombres entiers positifs compris entre **1** et $\mathbf{n^2}$.

Exemple :

Carré magique d'ordre 3, sa constante magique 45



A. On considère dans cette premier partie qu'une matrice carrée d'ordre n (n lignes et n colonnes) est représentée par **une liste qui contient n listes, toutes de même longueur n**.

Exemple : M = [[4,7,10,3],[3,2,9,6],[13,0,5,8],[7,1,6,25]] avec M[i] est la liste qui représente la ligne d'indice i dans M.

On considère les fonctions suivantes :

- La fonction « **sommeLigne(M,i)** », qui reçoit en paramètres une matrice carrée M contenant des nombres, et un entier i qui représente l'indice d'une ligne dans M. cette fonction retourne la somme des nombres de la ligne d'indice i dans M.
- La fonction « **sommeColonne(M,j)** » , qui reçoit en paramètres une matrice carrée M contenant des nombres, et un entier j qui représente l'indice d'une colonne dans M. Cette fonction retourne la somme des éléments de la colonne d'indice j dans M.

- La fonction « **sommeDiag1(M)** », qui reçoit en paramètre une matrice carrée M contenant des nombres, et qui retourne la somme des éléments de la première diagonale principale dans M.
- La fonction « **sommeDiag2(M)** », qui reçoit en paramètre une matrice carrée M contenant des nombres, et qui retourne la somme des éléments de la deuxième diagonale principale dans M. (La deuxième diagonale principale part du coin en haut à droite, jusqu'au coin en bas à gauche)

1. Donner le code de la fonction « **sommeLigne(M,i)** »
2. Écrire la fonction « **carreMagique(C)** » , qui reçoit en paramètre une matrice carrée C contenant des entiers strictement positifs, et qui retourne : True, si la matrice C'est un carré magique : les sommes sur chaque ligne, sur chaque colonne et sur chaque diagonale principale sont toutes égales False, sinon.

B. Dans cette partie, nous considérons qu'une matrice est représentée un tableau (array) de NumPy. Notre objectif est la construction d'un carré magique d'ordre n. En effet, Il existe plusieurs méthodes pour construire un carré magique d'ordre impaire et les carrés d'ordre pair. Si un carré est déjà construit, il est possible d'en dériver d'autres par permutation de ses colonnes et de ses rangées. On s'intéresse dans la suite à la méthode calculatoire pour construire un carré magique d'ordre impaire n>=3 :

On part de la matrice carrée (d'ordre n) P=$\begin{bmatrix} 1 & 1 & \dots & 1 \\ 2 & 2 & \dots & 2 \\ \vdots & \vdots & \ddots & \vdots \\ n & n & \dots & n \end{bmatrix}$ et on note P^T sa transposée et U la matrice carrée d'ordre n composée uniquement de 1 (toutes les cases sont égales à 1). On pose les matrices :

$$A = \left(P + P^T + \frac{n-3}{2}U\right) modulo\ n \text{ et } B = (P + 2P^T - 2U)\ modulo\ n$$

Alors le carré magique s'obtient avec $M = nA + B + U$

1. Ecrire la fonction «**matP(n)**» qui renvoie la matrice P d'ordre n définie ci-dessus.
2. Ecrire la fonction «**carreMagique(n)**» qui construit et renvoie la matrice magique d'ordre n en suivant les étapes décrites ci-dessus.

Elément de correction

```
def somme_ligne(M, i):
    n = len(M)
    s = 0
    for j in range(n):
        s += M[i][j]
    return s
def somme_colonne(M, j):
    n = len(M)
    s = 0
    for i in range(n):
        s += M[i][j]
    return s
def somme_diag1(M):
    n = len(M)
    s = 0
    for i in range(n):
        s += M[i][i]
    return s
def somme_diag2(M):
    n = len(M)
    s = 0
    for j in range(n):
        s += M[n - j - 1][j]
    return s
def carre_magique(C):
    n = len(C)
    ref = somme_ligne(C, 0)
    for i in range(1, n):
        if ref != somme_ligne(C, i):
            return False
    for j in range(n):
        if ref != somme_colonne(C, j):
            return False
    if somme_diag1(C) != ref or somme_diag2(C) != ref:
        return False
    return True
```

TP N°6 : Liste, tableau Numpy et classes

Ce sujet concerne le réseau routier marocain. Ce réseau peut être représenté par un dessin qui se compose de points et de traits continus reliant deux à deux certains de ces points : les points sont les villes, et les lignes sont les routes. On considère que toutes les routes sont à double sens.

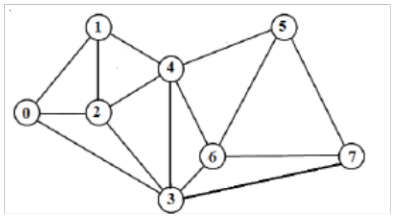
Première partie

Dans cette partie, nous supposerons que le réseau routier composé de n villes (avec n≥2). Les villes du réseau routier sont numérotées par des entiers allant de 0 à n-1.

A. Dans un premier lieu, nous représenterons une route du réseau routier entre deux villes i et j par une liste contenant leurs deux numéros dans un ordre quelconque, c.-à-d. par la liste [i,j] ou par la liste [j,i] indifféremment. Nous supposons qu'un réseau routier entre des villes est représenté par un tuple « reseau1 » à deux éléments où :

- reseau1[0] = liste des noms des villes appartenant au réseau (l'indice du nom d'une ville dans la liste des noms correspond au numéro qui lui est associé et qui sera utilisé dans la liste des routes, le nombre n correspond à la taille de la liste des noms)
- reseau1[1] = la liste non-ordonnée des routes déclarées entre les villes.

La figure ci-dessous donne l'exemple d'un réseau routier à 8 villes ayant 15 routes (chaque route entre deux villes est représentée par un trait entre elles). Ces routes sont représentées sous la forme de liste par le deuxième élément du tuple « reseau1 » .



reseau1 = (["ville1", "ville2","ville3","ville4","ville5","ville6","ville7","ville8"] , [[0,1], [4,3], [3,2], [2,0], [0,3], [2,1], [4,5], [5,7], [7,6], [6,4], [1,4], [6,5], [2,4], [6,3], [3,7]])

1. Ecrire une fonction « numeroVille(reseau1, nomVille) » qui renvoie le numéro de la ville, du reseau « reseau1 », dont le nom est fourni en paramètre.
2. Ecrire une fonction « existeRoute(reseau1,nomVille1, nomVille2) » qui renvoie True s'il existe, dans le réseau « reseau1 », une route entre les villes dont les noms sont fournis en paramètres et renvoie False sinon.
3. Ecrire une procédure « AjoutRouteReseau(nomVille1, nomVille2) » qui ajoute, à la liste des routes, une nouvelle route entre les villes fournies en paramètre (si cette route n'y figure pas déjà). Cette procédure lève l'exception «TraficException» (supposée déjà définie) si le nom de l'un de ces deux villes ne figure pas dans la liste des noms du villes du réseau « reseau1 ».

B. Nous supposons maintenant que les routes entre les villes sont représentées par une matrice symétrique (implémentée par une liste de liste) contenant des 0 et 1 (0 indique aucune route reliant les deux villes et 1 indique l'existence de route). Deux villes i et j sont considérées voisines, s'il existe une route entre la ville i et la ville j. Ainsi, le réseau de l'exemple de la figure ci-dessus est représenté sous la forme suivante :

```
reseau = ( ["name1","name2","name3","name4","name5","name6","name7","name8"]  ,
           [   [ 0,  1,  1,  1,  0,  0,  0,  0 ],
               [ 1,  0,  1,  0,  1,  0,  0,  0 ],
               [ 1,  1,  0,  1,  1,  0,  0,  0 ],
               [ 1,  0,  1,  0,  1,  0,  1,  1 ],
               [ 0,  1,  1,  1,  0,  1,  1,  0 ],
               [ 0,  0,  0,  0,  1,  0,  1,  1 ],
               [ 0,  0,  0,  1,  1,  1,  0,  1 ],
               [ 0,  0,  0,  1,  0,  1,  1,  0 ]   ]    )
```

1. Ecrire une fonction « creerReseau(listeNomsVilles) » qui crée, initialise et renvoie un réseau routier des villes de la liste « listeNomsVilles» n'ayant aucune route reliant ces villes.

2. Ecrire une fonction « listeVilles(reseau2, nomVille) » qui renvoie la liste des noms des villes voisines (reliées par une route) à la ville dont le nom est fourni en paramètre. Cette fonction renvoi la valeur none si la ville ne figure pas dans le réseau « reseau2 ».

3. Ecrire une fonction « dictDegresVille(reseau) » qui renvoie un dictionnaire comportant pour chaque ville (la clé du dictionnaire sera le nom de la ville), le degré de cette ville (ce nombre sera la valeur associée à la clé). Dans un réseau routier, le degré d'une ville est le nombre de villes qui lui sont voisines.

Deuxième partie

Dans cette partie, nous considérons que la modélisation des réseaux routier est effectuée en par les deux classes « Ville » et « Reseau ». Chaque ville, représenté par une instance de la classe « Ville », est défini par son nom, latitude et son longitude (les données latitude et longitude permet d'obtenir les coordonnées GPS et la position d'une ville sur une carte MAP). Un réseau routier est une instance de la classe « Reseau » qui est caractérisé par la liste de ses villes (formée par des instances de la classe « Ville ») et la liste des routes. Chacun de ces routes est représenté par une liste des deux instances de la classe « Ville » reliée par la route. La structure des classes « Ville » et « Reseau » est décrite par le diagramme suivant :

Ville
- name : String ; - latitude : float; - longitude : float
+ __init__(name: String, latitude: float, longitude: float) + get_Xxx(): …; + set_Xxx(…):…; + __str__(): String + __eq__(ville :Ville) : bool # opérateur d'égalité basé sur name …

Reseau
- listVille : list # liste d'instances de « Ville » - listRoutes : list # routes [[villeX [villeY], ….]
+ __init__() + get_Xxx(): … + addVille(name: String, latitude: float, longitude: float): void + getVilleByName(name :String) :Ville + addRoute(nameVille1: String, nameVille2: String) :void + getEstVoisin(nameVille1: String, nameVille2: String): bool + getListRoutes(nameVille: String): list # liste des Villes + getVilleMaxDegres():Ville …

1. Définir le constructeur de la classe « Ville»
2. Ecrire le code de la méthode «addRoute » permettant d'ajouter une nouvelle route à la liste « listRoutes ». Cette méthode vérifie en premier lieu la validité des noms villes fournies en paramètres. ces noms doivent exister dans la liste des villes, l'exception « AssertionError » est levé dans le cas contraire. Cette exception est également levée dans le cas où la route à ajouter figure déjà dans la liste des routes.
3. Définir la méthode « getVilleMaxDegres » permettant de retourner l'instance correspondante à la ville ayant le maximum des degrés (le degré d'une ville le nombre de ses voisinages c-à-d le nombre de villes qui lui sont reliées).

Elément de correction

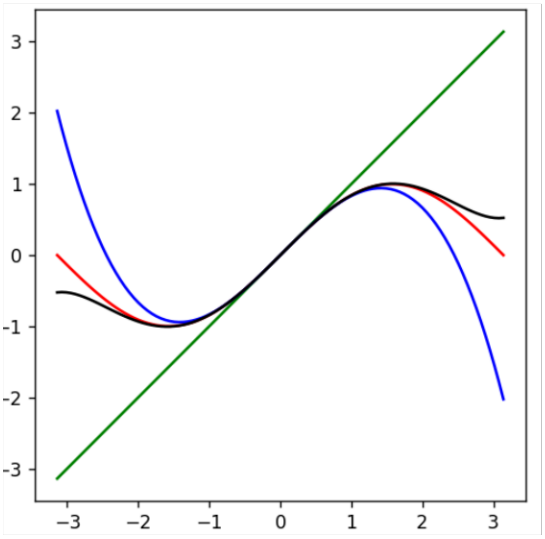
```
##################### Partie 1.A ####################
reseau1 = (["ville1", "ville2", "ville3", "ville4", "ville5", "ville6",
"ville7", "ville8"], [[0, 1], [4, 3], [3, 2], [2, 0], [0, 3], [2, 1], [4, 5],
[5, 7], [7, 6], [6, 4], [1, 4], [6, 5], [2, 4], [6, 3], [3, 7]])

class TraficException(Exception):
    def __init__(self, message):
        super().__init__(message)
def numeroVille(reseau1, nomVille) :
    return reseau1[0].index(nomVille) if nomVille in reseau1[0] else -1
def existeRoute(reseau1, nomVille1, nomVille2) :
    index1 = numeroVille(reseau1, nomVille1)
    index2 = numeroVille(reseau1, nomVille2)
    return  [index1, index2] in reseau1[1] or [index2, index1] in reseau1[1]
def AjoutRouteReseau(reseau1 ,nomVille1, nomVille2) :
    try :
        index1=numeroVille(reseau1,nomVille1)
        index2=numeroVille(reseau1,nomVille2)
        if [index1,index2] not in reseau1[1] and
          [index2,index1] not in reseau1[1] :reseau1[1].append([index1,index2])
        else :
            print(f"La route entre {nomVille1} et {nomVille2} existe déjà.")
    except ValueError :
        raise TraficException("Ouuups, Ville n existe Plus.....")

##################### Partie 1.B ####################
reseau = (
    ["name1", "name2", "name3", "name4", "name5", "name6", "name7", "name8"],
    [
        [0, 1, 1, 1, 0, 0, 0, 0],
        [1, 0, 1, 0, 1, 0, 0, 0],
        [1, 1, 0, 1, 1, 0, 0, 0],
        [1, 0, 1, 0, 1, 0, 1, 1],
        [0, 1, 1, 1, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 0, 1, 1],
        [0, 0, 0, 1, 1, 1, 0, 1],
        [0, 0, 0, 1, 0, 1, 1, 0]    ]  )
def creerReseau(listeNomsVilles):
    # return (listeNomsVilles, [[0  for i in range(len(listeNomsVilles))] for
j in range(len(listeNomsVilles))])
    return (listeNomsVilles, [[0] * len(listeNomsVilles)
                              for i in range(len(listeNomsVilles))])
def listeVilles(reseau2, nomVille) :
    if nomVille not in reseau2[0] :
        return None
    else :
        index  = reseau2[0].index(nomVille)
        liste  = [i for i, val in enumerate(reseau2[1][index]) if val == 1]
        listeV = [nom for nom in reseau2[0] if reseau2[0].index(nom) in liste]
    return listeV
def dictDegresVille(reseau) :
    return {reseau[0][i] : sum(reseau[1][i]) for i in range(len(reseau[0]))}
```

```
#################### Partie 2 ####################

class Ville :
    """ class qui represente l object Ville """
    def __init__(self, name, latitude, longitude):
        self._name      = name
        self._latitude  = latitude
        self._longitude = longitude
    def get_name(self):    return self._name
    def get_latitude(self):  return self._latitude
    def get_longitude(self):  return self._longitude
    def set_name(self, name):  self._name = name
    def set_latitude(self, latitude): self._latitude = latitude
    def set_longitude(self, longitude):  self._longitude = longitude
    def __str__(self):
        return "Name : {}, Latitude : {}, Longitude : {}".format
             (self._name, self._latitude, self._longitude)
    def __eq__(self, other):
        return isinstance(other, Ville) and other._name == self._name

class Reseau :
    """ class qui represente l object Reseau """
    def __init__(self, listeVille : list, listRoute : list):
        self._listeVille = listeVille
        self._listRouge  = listRoute
    def get_listeVille(self): return self._listeVille
    def getListRoutes(self): return self._listRouge
    def addVille(self, name : str, latitude : float, longitude : float):
        self._listeVille.append(Ville(name, latitude, longitude))
    def getVilleByName(self, name : str):
        for ville in self._listeVille :
            if ville.get_name() == name :
                return ville
        return None
    def addRoute(self, nameVille1, nameVille2):
        ville1 = self.getVilleByName(nameVille1)
        ville2 = self.getVilleByName(nameVille2)
        assert [ville1, ville2] not in self._listRouge and [ville2, ville1]
not in self._listRouge, "La route existe déjà dans le réseau."
        self._listRouge.append([ville1, ville2])
    def getEstVoisin(self, nameVille1: str, nameVille2: str) -> bool:
        ville1 = self.getVilleByName(nameVille1)
        ville2 = self.getVilleByName(nameVille2)
        return [ville1, ville2] in self._listRouge or [ville2, ville1] in
self._listRouge
    def getVilleMaxDegres(self):
        max_deg = -1
        ville_max = None
        for ville in self._listeVille:
            voisins = self.getListRoutes(ville.get_name())
            if len(voisins) > max_deg:
                max_deg = len(voisins)
                ville_max = ville
        return ville_max
```

TP N°7 : MathPlot et traçage de fonction

Exercice 1

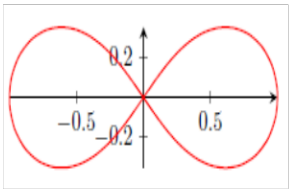
Représenter sur une même figure les représentations graphiques des fonctions $f1(x) = \sin(x)$, $f2(x) = x$, $f3(x) = \frac{x^3}{3!}$ et $f4(x) = \frac{x^3}{3!} + \frac{x^5}{5!}$ Sur l'intervalle $[-\pi, \pi]$



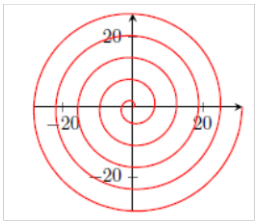
Exercice 2

Tracer la courbe de la fonction suivante :

$$La\ Lemniscate\ de\ Bernoulli \begin{cases} x(t) = \dfrac{\sin t}{1 + cos^2 t} \\ y(t) = \dfrac{\sin t \cos t}{1 + cos^2 t} \end{cases} \quad sur \quad [0, 2\pi]$$



$$La\ spirale\ d'Archimède \begin{cases} x(t) = t \cos(t) \\ y(t) = t \sin(t) \end{cases} \quad sur \quad [0, 10\ \pi]$$



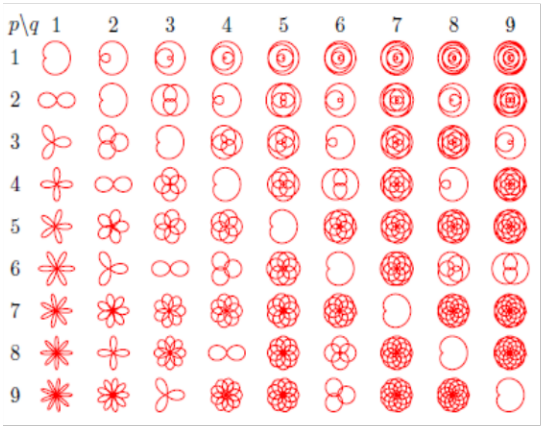
Exercice 2

Tracer les courbes des fonctions suivantes :

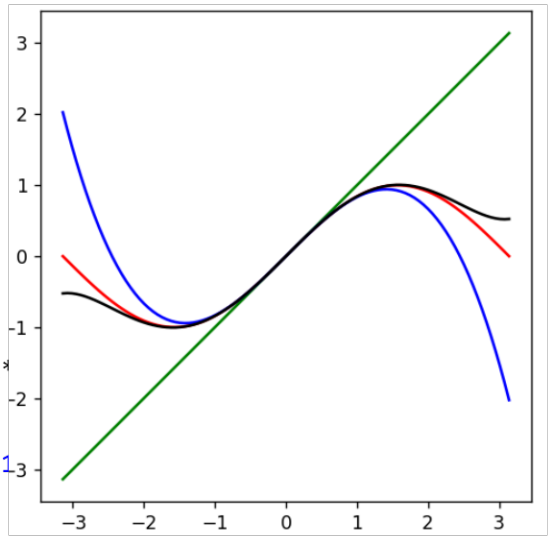
Les cyclo − harmoniques

$$\begin{cases} x(t) = \left(1 + cos\left(\frac{p}{q}t\right)\right)\cos(t) \\ y(t) = \left(1 + cos\left(\frac{p}{q}t\right)\right)\sin(t) \end{cases}$$

pour p et q entiers sur $[0, 2q\pi]$



Elément de correction



Exercice n°1

```
import matplotlib.pyplot as
import numpy as np

f1=lambda x:x
f2 =lambda x: x-(x**3)/6
f3= lambda x: x-(x**3)/6+(x*

x=np.linspace(-np.pi,np.pi,1
plt.plot(x,np.sin(x),'r')
plt.plot(x,f1(x),'g')
plt.plot(x,f2(x),'b')
plt.plot(x,f3(x),'k')

plt.axis("scaled")
plt.show()
```

Exercice n°2

```
import numpy as np
import matplotlib.pyplot as plt

def style(ax,xlim, ylim):
   ax.spines['left'].set_position('zero') ;
   ax.spines['right'].set_color('none')
   ax.spines['bottom'].set_position('zero')
   ax.spines['top'].set_color('none')
   ax.grid(False);
   ax.set_xlim(-xlim, xlim); ax.set_ylim(-ylim, ylim)
   ax.text(xlim + 0.1, 0.0, r'x(t)',horizontalalignment='center', fontsize=10)
   ax.text(0.0, ylim+0.01, r'y(t)', horizontalalignment='center', fontsize=10)

t1 = np.linspace(0, 2 * np.pi, 1000)
# Lemniscate de Bernoulli
x1 = lambda t:np.sin(t) / (1 + np.cos(t) ** 2)
f1 = lambda t:np.sin(t) * np.cos(t) / (1 + np.cos(t) ** 2)
t2 = np.linspace(0, 10 * np.pi, 5000)
# Spirale d'Archimède
x2 = lambda t:t*np.cos(t)
f2 = lambda t:t*np.sin(t)
```

```
ax= plt.subplot(2, 1, 1)
ax.plot(x1(t1), f1(t1), 'r')
style(ax,1.2,0.37);
ax.text(0.0,0.44, r'$x(t)=\frac{sin(t)}{1+cos(t)^2} , y(t)=\frac{ sin(t)cos(t)}
        {1+cos(t)^2}/ t ∈[0,2π]$',horizontalalignment='center',fontsize=10)
ax= plt.subplot(2, 1, 2)
ax.plot(x2(t2), f2(t2), 'b') ; style(ax,35,35)
ax.text(0.0,42, r'$x(t)=tcos(t) , y(t)=tsin(t)  / t ∈[0,10π]$',
horizontalalignment='center', fontsize=10)
plt.show()
```

Exercice n°3

```
import matplotlib.pyplot as plt
import numpy as np
def style(ax):
    ax.spines['left'].set_position('zero')
    ax.spines['bottom'].set_position('zero')
    ax.spines['left'].set_visible(False) ; ax.spines['right'].set_visible(False)
    ax.spines['bottom'].set_visible(False) ; ax.spines['top'].set_visible(False)
    ax.get_xaxis().set_visible(False) ; ax.get_yaxis().set_visible(False)

n=10
ax = plt.subplot(n, n, 1) ; style(ax)
ax.text(0.5,0.5,r'$\frac{p}{q}$',horizontalalignment = 'right',fontsize = 15)
for p in range (1,n):
    ax = plt.subplot(n, n, p+1) ;  style(ax)
    ax.text(0.5, 0.5,  p, horizontalalignment='right', fontsize=12)
    ax = plt.subplot(n, n, p*n+1) ;  style(ax)
    ax.text(0.5, 0.5, p, horizontalalignment='right', fontsize=12)
for p in range (1,n):
    for q in range (1,n):
        t = np.linspace(0, 2 * q * np.pi, 400)
        # Cyclo-harmonique
        tmp = 1 + np.cos(p / q * t)
        x = tmp * np.cos(t) ;   y = tmp * np.sin(t)
        pos=p*n+q+1
        ax= plt.subplot(n, n, pos) ;  style(ax)
        plt.plot(x, y,"r")
plt.show()
plt.savefig('harmonique.png')
```

TP N°8 : Statistique et visualisation avec Pondas

L'objectif du projet est d'illustrer comment prétraiter et fusionner des ensembles de données pour calculer les mesures nécessaires et les préparer à une analyse. Nous allons utiliser l'ensemble de données COVID19, qui comprend les données relatives au nombre cumulé de cas confirmés, par jour, dans chaque pays. En outre, nous avons également un ensemble de données différent qui se compose de divers facteurs de vie. Nous allons fusionner les deux données et essayer d'établir une relation entre elles.

1. En utilisant la méthode read_csv, importer l'ensemble de données (Utiliser lecture csv de pandas pour importer les données depuis le fichier « covid19.csv ») et visualiser l'entête des données (utiliser la méthode head).
2. En utilisant la méthode drop de pandas, Supprimer les étiquettes 'Lat' et 'Long' (considérer inutiles pour ce TP, utiliser) et remplacer l'ensemble de données actuel.
3. En utilisant la méthode groupby, effectuer une agrégation des lignes par pays et un cumule de cas confirmés par jour.
4. Réaliser une visualisation, sur un graphe à courbe, des données relatives à certains pays spécifiques. Reprendre cette visualisation pour les pays dont le cumule total de cas confirmés dépasse 10e6
5. Afficher un Histogramme de l'évolution des cas confirmés de la Chine et visualiser son taux d'infection (taux d'accroissement ou dérivée) en utilisant la méthode « diff() »
6. Etablir le taux d'infection maximal pour tous les pays. Créer un nouveau dataFrame « Corona » avec uniquement la colonne de ces taux classés en ordre décroissant. Utiliser la fonction crosstab(variable, "freq") pour déterminer les effectifs associés aux 10 premiers taux d'infection maximal et tracer le diagramme circulaire(type « pie ») correspondants.
7. Créer un dataFrame « worldHR » à partir d'un 2ème ensemble de données "worldHapReport.csv" qui comporte un certain nombre d'indicateurs

économiques, politiques et sociaux permettant de mesurer si les gens sont satisfaits de leurs vies quotidiennes ou non. Importation du jeu de données.

8. Supprimer les colonnes : 'Overall rank','Score','Generosity' et 'Perceptions of corruption'. Nomer les index de la trame de données 'Pays ou région' (utiliser la méthode set_index)
9. Fusionnez les jeux de données du dataFrame « Corona » avec le jeu de donnée « worldHR » (utiliser la méthode join)
10. Etablir la Matrice de corrélation pour représente la corrélation entre toutes les deux colonnes de notre ensemble de données
11. Utiliser l'outil de Data Visualization Seaborn pour Tracer un graphique boxplot (boîte à moustaches) afin de comparer les médianes associées aux indicateurs 'GDP per capita' et log('max infection rate').

Elément de correction

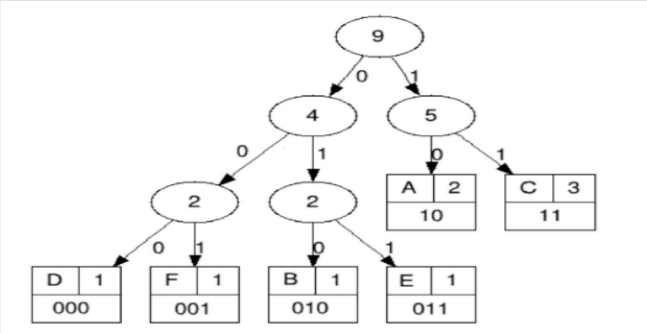
```
# 1 Import Covid19 Data
covid_data = pd.read_csv('covid19.csv')
# print(covid_data.shape)
# print(covid_data.head(1))
# print(covid_data.columns)

# 2
covid_data.drop(['Lat', 'Long'], axis=1, inplace=True)
# print(covid_data.head(1))

# 3.
covid_data_agg = covid_data.groupby('Country/Region').sum()
print(covid_data_agg.head(4))

# 4.
covid_data_agg.loc['China'][1:].plot()
covid_data_agg.loc['Morocco'][1:].plot()
covid_data_agg.loc['France'][1:].plot()
plt.legend()
plt.show()
#print(covid_data_agg[covid_data_agg.columns[1:]].sum(axis=1))
covid_data_agg.drop(['Province/State'], axis=1, inplace=True)
#print(covid_data_agg[covid_data_agg.sum(axis=1) > 1000000])

#for country in covid_data_agg[covid_data_agg.sum(axis=1)>1000000].head(5).index:
#     covid_data_agg.loc[country].plot()
#
# plt.legend()
# plt.show()

# 5
#       son taux d'infection (taux d'accroissement ou dérivée) en utilisant la
méthode « diff()»

# Appliquer la méthode diff() permet de calculer la différence entre les éléments
adjacents
# dans chaque colonne
# print(covid_data_agg.loc['China'])
# print(covid_data_agg.loc['China'].diff())
#
# covid_data_agg.loc['China'].diff().plot()
# plt.show()


# 6.

countries= covid_data_agg.index
corona =pd.DataFrame(index=countries, columns=['max infection rate'],
      data=[covid_data_agg.loc[country].diff().max() for country in countries] )
corona.sort_values(by='max infection rate', ascending=False,inplace=True)
# print(corona.head(10))
# t = pd.crosstab(corona['max infection rate'].head(10), "freq",normalize=True )
```

```
# t.plot.pie(subplots=True)
#
# plt.legend(labels=corona.index, loc='upper right')
# plt.show()


# 7 .

# Load Happiness Report Data
worldHR = pd.read_csv('worldHapReport.csv')
print(worldHR.head())
print(worldHR.shape)


# 8

# List of useless columns
dropCol = ['Overall rank', 'Score', 'Generosity', 'Perceptions of corruption']


worldHR.drop(dropCol, inplace=True, axis=1)
print(worldHR.shape)
print(worldHR.head())
worldHR.set_index('Country or region', inplace=True)
print(worldHR.head())

# 9.
#print(pd.concat ([ worldHR, corona],axis=1, join ="inner"))

data = corona.join(worldHR, how='inner')
print(data.columns)
#print(data)

#10.
print(data.corr())
print(data['GDP per capita'].corr(data['max infection rate']))
# Plot a correlation matrix and viualise it
sns.heatmap(data.corr(), annot=True)
plt.show()

# 11.
data['max infection rate']=np.log(data['max infection rate'])
sns.scatterplot(data= data, x='GDP per capita', y='max infection rate')
plt.show()

# Make a regression plot
sns.regplot(data= data, x='GDP per capita', y='max infection rate')
plt.show()
```

TP N°9 : Liste, tableau Numpy et classes

Le codage de Huffman utilise un code à longueur variable pour représenter un symbole de la source (par exemple un caractère dans un fichier). Le code est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source, un code court étant associé aux symboles de source les plus fréquents. Les étapes de construction d'un tel code sont les suivantes :

- La première étape du codage de Huffman consiste àcréer un dictionnaire contenant la liste des caractères présents dans le texte, associé à leur fréquence dans ce texte. Exemple "AABCDCCEF" donnera{'A':2,'B':1,'C':3,'D':1,'E':1,'E':1}
- La deuxième étape consiste à construire un arbre de Huffman qui permet ensuite de coder chaque caractère.



Chaque caractère constitue une des feuilles de l'arbre à laquelle on associe un poids valant son nombre d'occurance. Puis l'arbre est crée suivant un principe simple : on associé à chaque fois les deux les deux noeux de plus faibles poids pour créer un nœud dont le poids équivalent à la somme des poids de ses fils jusqu'à n'en avoir plus qu'un, la racine. On associe ensuite par exeemple le code 0 à la banche de gauche et le code 1 à la branche de droite. Pour obtenir le code binaire de chaque caractère, on on descend sur l'arbre (voir l'exemple de la figure ci-dessu) de la racine jusqu'aux feuilles en ajoutant à chaque fois , un code 0 ou un code 1 selon la branche suivie.

Les feuilles de l'arbre de Huffman(leaf) sont codées sous la forme d'un tuple avec comme premier élément le caractère et comme deuxième élément le poids. Pour l'arbre donnée en exemple en figure ci-dessus, on aura 6 touples ('A',2), ('B',1), ('C',3), ('D',1), ('E',1) et ('F',1)

Pour notre texte "AABCDCCEF", le code du caractère "B" est "010". Ainsi ce texte de 9 caractères ASCII (72bits) sera codé en binaire "10 10 010 11 000 11 11 011 001" (22bits).

A. Première partie : Approche structurée

1. Dictionnaire des fréquences

Dans cette première partie, on veut trier les caractères par leurs fréquence d'apparition dans un texte . On considère que ces caractères sont placés dans un dictionnaire

a. Écrire la fonction « makeDictFreq » qui reçoit une chaine de caractères et qui retourne le dictionnaire des fréquences de ses caractères.

b. Écrire la fonction « sortDictFreq » qui reçoit en paramètres un dictionnaire des fréquences et construit et retourne une liste triée (par un ordre croissant des fréquences) des couples (caractère, fréquence). Utiliser un agorithme de tri par insertion.

c. Écrire une procedure « insertListFreq » qui reçoit en paramètres une liste triée des des couples (caractère, fréquence) et un tuple (caractère, fréquence) et qui insert ce dernier dans une position adéquate pour que la liste reste triée.

2. Construire l'arbre de Huffman

Dans cette deuxième partie, on veut construire l'arbre de Huffman à partir de la liste triée des couples (caractère fréquence). On représente un arbre de Huffman de la manière suivante :

- Les noeuds internes sont des triplets (poids,fils-gauche,fils-droit)
- Les feuilles sont des couples (caractère,poids).
- Le poids est un entier qui représente la fréquence d'un caractère dans une feuille, ou la somme des fréquences de tous les caractères regroupés par un noeud interne.

Par exemple, l'arbre de l'exemple ci-dessus est représenté par la liste :

[9, (4, (2, ("d", 1), ("F", 1)), (2, ("B", 1), ("E", 1))), (5, ("A", 2), ("F", 3))

a. Écrire la fonction booléenne feuille qui reçoit un tuple et qui teste si ce tuple représente une feuille.

b. Écrire la fonction « poids » qui calcule le poids d'un arbre (ou partie d'un arbre reçu en parametre sous forme d'un tuple qu'il s'agisse d'un nœud interne ou d'une feuille).

c. Écrire la fonction « merge », qui fusionne deux arbres de Huffman. On considère que le 1^{ier} et le 2ème parametres represente respectivement le fils de gauche et le file de droite de l'arbre fusionné. Ainsi, la fonction fusion retourne un tuple de la forme (poids,fils-gauche,fils-droit)

d. Écrire la fonction arbre-codage qui, étant donnée la liste des fréquences triée, construit l'arbre de Huffman. On procédera par insertions successives de la fusion des deux premiers arbres dans le reste de la liste.

3. Code de Huffman

a. Écrire la fonction construitCode étant donné un arbre de Huffman(reçu en parametre), construit et retourne un dictionnaire des codes associés à chaque caractère.

b. Écrire la fonction codage qui, étant donnés une phrase à décoder et l'arbre de Huffman, construit et retourne la chaine (de de 0 et de 1) de codage de Huffman correspondant.

c. Écrire la fonction decodage qui, étant donnés un arbre de Huffman et une chaine de codage (de de 0 et de 1), décode et routourne la phrase initiale qui a été codée.

B. deuxième partie : Aproche orientée objet

1. Dictionnaire des fréquences

Dans cette partie, nous allons aborder le problème de codage de Hoffman avec une implémentation orientée objet. Nous considérons les classes « Node » et « HuffmanCode » suivantes :

```
class Node:
   def __init__(self, char, freq):
     self.char = char
     self.freq = freq
     self.left = None
     self.right = None
   def __lt__(self, other):
     if(other==None):return False
     if not isinstance(other,Node):
         return False
     return self.freq < other.freq
   def __str__(self):
     if self.char== None:
       return "("+str(self.freq)+")"
     else:
       return "("+str(self.char)+","
            + str(self.freq) + ")"
Exemple
huff=HuffmanCode("AABCDCCEF")
huff.mergeNodes()
huff.makeCodes()
```

```
class HuffmanCode:
   def __init__(self, text):
     self.heap=
       HuffmanCode.makeNode(text)
     self.codes = {}
     self.reverseCodes = {}
   @staticmethod
   def makeDictFreq (text):
      ...
   @staticmethod
   def makeNode(text):
      ...
   def insertNode (self, node):
      ...
   def mergeNodes(self):
      ...
   def makeCodes(self):
      ...
   def getEncodedText(self,text):
      ...
   def decodeText(self,
                  encoded_text):
      ...
```

En se basant sur l'implémentation réalisée dans la partie précédente, écrie les différentes méthodes de la classe « HuffmanCode ».

- La méthode « makeNode » qui reçoit en paramètres une chaine de caractères, fait appel à la méthode « makeDictFreq » pour élaborer un dictionnaire des fréquences des caractères de la chane. La méthode « makeNode » construit une liste des instances de la classe « Node » puis retourne cette liste après l'avoir trié (par un ordre croissant des fréquences).

- La méthode « mergeNodes » construit l'arbre de huffman par fusion succssive des neuds (instance de la classe « Node ») de la liste « heap ». Il faut utiliser 2 fois la primitive pop(0) (neuds ayant la fréquence les plus faibles) pour retirer 2 neuds de la liste « heap » et insérer le neud résultant de la fusion dans même liste « heap » .

- La méthode « makeCodes » assure un parcout en profondeur de l'arbre de huffman (la liste « heap ») pour déterminer le code huffman associé à chaque caractère. Ces codes (et les code inverses) sont placés dans le dictionnaire « codes » (les code inverse càd la lettre corespondant à chaque code sont placé dans le dictionnaire « reverseCodes ».

Elément de correction

```
#A. Premiere partie : Approche structuree
#1.Dictionnaire des frequences
#a. Fonction makeDictFreq
def makedictfreq(texte):
    # Initialise un dictionnaire vide pour stocker les fréquences des caractères
    dic = {}
    # Parcours chaque caractère dans le texte
    for char in texte:
        # Vérifie si le caractère est déjà dans le dictionnaire
        if char in dic:
            # Incrémente la fréquence du caractère s'il est déjà présent
            dic[char] += 1
        else:
            # Ajoute le caractère au dictionnaire avec une fréquence de 1
            dic[char] = 1
    # Retourne le dictionnaire des fréquences des caractères
    return dic
print("Dictionnaire de frequences des caractere du chaine ")
chainedecarateres = "ENGINEERING"
resultat = makedictfreq(chainedecarateres)
print(resultat)

#b. Fonction sortDictFreq
def sortDictFreq(dict_frequence):
    # Convertir le dictionnaire en une liste de tuples (clé, valeur)
    items = list(dict_frequence.items())
    # Parcourir la liste à partir du deuxième élément (index 1)
    for i in range(1, len(items)):
        k = i  # Indice de la clé actuelle
        cle, valeur = items[k]  # Récupérer le tuple (clé, valeur) à l'indice k
        while valeur < items[k - 1][1] and k > 0:
            items[k] = items[k - 1]
            k -= 1
            items[k] = (cle, valeur)
    # Renvoyer la liste triée
    return items

print("Liste triee")
dict_frequence= {'A': 2, 'B': 1,'C': 3,'D': 1,'E': 1,'F': 1}
liste_triee = sortDictFreq(dict_frequence)
print(liste_triee)

#c. Fonction insertListFreq
def insertListFreq(liste_triee, nouveau_tuple):
    # Initialise l'indice i à la fin de la liste triée
    i = len(liste_triee) - 1
    # Parcourt la liste triée en ordre décroissant
    # jusqu'à trouver la bonne position pour le nouveau tuple
    while i >= 0 and nouveau_tuple[1] < liste_triee[i][1]:
        i -= 1
    # Insère le nouveau tuple à la position trouvée
    liste_triee.insert(i + 1, nouveau_tuple)
```

```
print("Pour que la liste soit adequate")
liste_triee = [('B', 1), ('D', 1), ('E', 1), ('F', 1), ('A', 2), ('C', 3)]
nouveau_tuple = ('Z', 3)
insertListFreq(liste_triee, nouveau_tuple)
print(liste_triee)

#2.Construire l'arbre de Huffman
#a
def feuille(arbre):
    # Vérifie si l'arbre est une feuille en s'assurant qu'il s'agit d'un tuple,
    # que sa longueur est égale à 2, et que le deuxième élément est un entier
    return isinstance(arbre, tuple) and len(arbre) == 2 and isinstance(arbre[1],
int)

#b
def poids(arbre):
    # Si l'arbre est une feuille, retourne le poids de la feuille
    if feuille(arbre):
        return arbre[1]  # Retourne le poids de la feuille
    else:
      return arbre[0]#Si ce n'est pas une feuille,retourne le poids du nœud interne

#c
def merge(n1, n2):
    # Fusionne deux nœuds en créant un nouveau nœud avec le poids combiné,
    # et les deux nœuds d'origine comme sous-arbres.
    return (poids(n1) + poids(n2), n1, n2)

#d
def arbre_codage(liste):
    # Continue la boucle tant que la liste a plus d'un élément
    while len(liste) > 1:
        # Retire les deux premiers nœuds de la liste
        n1 = liste.pop(0)
        n2 = liste.pop(0)
        # Fusionne les deux nœuds pour former un nouveau nœud
        n = merge(n1, n2)
        # Ajoute le nouveau nœud à la liste
        liste.append(n) #Ajoute le nouvel arbre résultant de la fusion des 2 arbres
        # Trie la liste en fonction du poids de chaque nœud
        liste.sort(key=poids)  # Trie la liste après chaque fusion
    # Retourne la racine de l'arbre de codage final
    return liste[0]

#lors du tri, la liste sera ordonnée en fonction des poids des arbres.
# Les arbres avec les poids les plus bas seront en tête de liste

#3. Code de Huffman
#a
def construitCode(arbre, code='', dic={}):
    # Construit un dictionnaire de codes binaires à partir de l'arbre de codage
    if feuille(arbre):
    #Si c'est une feuille, ajoute le code binaire correspondant au dictionnaire
        dic[arbre[0]] = code
```

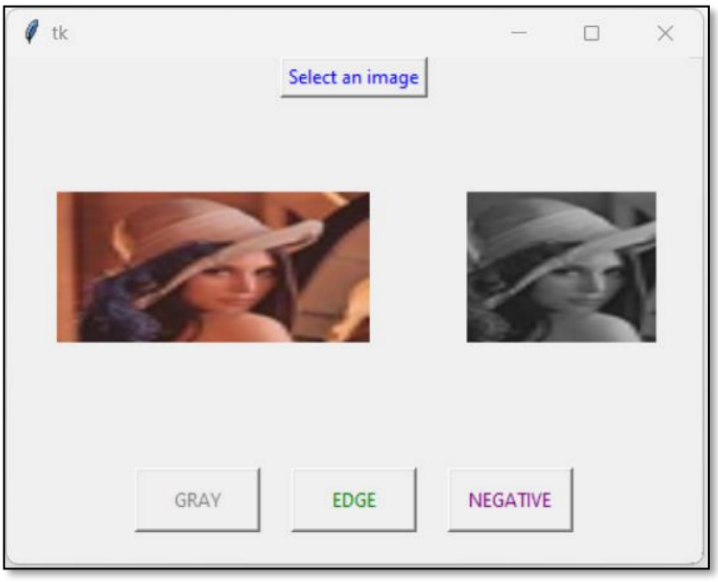
```
        else:
            # récursivement explore les sous-arbres gauche et droit
            # '0' pour le sous-arbre gauche et '1' pour le sous-arbre droit
            construitCode(arbre[1], code + '0', dic)
            construitCode(arbre[2], code + '1', dic)
        # Retourne le dictionnaire de codes binaires construit
        return dic

    #b
    def codage(phrase, arbre):
        # Construit le dictionnaire de codes binaires à partir de l'arbre de codage
        dic = construitCode(arbre)
        # Initialise une variable pour stocker le code binaire résultant
        code = ""
        # Parcours chaque caractère dans la phrase d'origine
        for char in phrase:
              code += dic[char]
        # Retourne le code binaire résultant
        return code

    #c
    def decodage(arbre, code):
        # Initialise une chaîne vide pour stocker la phrase décodée
        phrase_decodee = ''
        # Initialise une variable pour suivre la position dans l'arbre de codage
        arbre_courant = arbre
        # Parcours chaque bit dans le code binaire
        for bit in code:
            if bit == '0':
                # Déplacement vers le fils gauche de l'arbre
                arbre_courant = arbre_courant[1]
            else:
                # Déplacement vers le fils droit de l'arbre
                arbre_courant = arbre_courant[2]
            # Vérifie si l'arbre courant est une feuille
            if feuille(arbre_courant):
                # Ajoute le caractère correspondant à la phrase décodée
                phrase_decodee += arbre_courant[1]
                # Réinitialise l'arbre courant pour le prochain bit
                arbre_courant = arbre
        # Retourne la phrase décodée
        return phrase_decodee
```

TP N°10 : IHM et traitement d'images

Énoncé

Une image est définie par un tableau bidimensionnel (matrice) composée d'un nombre fini d'éléments, dont chacun est appelé pixel et possède une valeur particulière. Cette valeur est l'intensité du pixel, elle peut être une couleur RGB. Cette dernière est déterminée par la combinaison des intensités rouge, verte et bleue stockées dans chaque plan de couleur à l'emplacement du pixel. Un pixel dont les composantes de couleur sont (0, 0, 0) est affiché en noir, et un pixel dont les composantes de couleur sont (1, 1, 1), c'est-à-dire (255, 255, 255), est affiché en blanc.

L'objectif de ce TP est la lecture et l'affichage d'une image couleur RVB et sa conversion en niveaux de gris, en négatif et en images de bord. On considère l'interface graphique suivante :



Le bouton de sélection d'image est utilisé pour sélectionner une image (une boîte de dialogue du fichier est utilisée pour sélectionner l'image d'entrée souhaitée sur le disque dur). Après avoir sélectionné une image, quatre types d'opérations sont effectuées en cliquant sur un bouton pour chaque opération. Le bouton RVB est utilisé pour afficher l'image couleur RVB, le bouton gris est utilisé pour afficher l'image grise de l'image sélectionnée, le bouton bord est utilisé pour afficher le bord de l'image sélectionnée, et le bouton négatif est utilisé pour afficher le négatif de l'image sélectionnée.

Image en niveaux de gris :

Les niveaux de gris sont une gamme de nuances monochromatiques allant du noir au blanc. Par conséquent, une image en niveaux de gris ne contient que des nuances de gris et aucune couleur. La principale caractéristique des images en niveaux de gris est l'égalité des niveaux de couleur rouge, vert et bleu. La fonction cv2.cvtColor ( ) de OpenCV-Python est utilisée pour convertir une image couleur RVB en image en niveaux de gris. Une image RVB est convertie en images grises à l'aide de la méthode suivante : $Y = 0,299R + 0,587G + 0,114B$

Image négative :

L'image négative est un type d'image qui peut être formé en soustrayant la valeur RVB de 255. La fonction cv2.bitwise_not ( ) est utilisée pour convertir l'image couleur RGB en image négative en couleur. Une image couleur RVB (c'est-à-dire une image positive) est convertie en une image négative en utilisant les méthodes suivantes

$R_n = 255 – R_p G_n = 255 - G_p B_n = 255 – B_p$ où p et n sont respectivement positifs et négatifs.

Image de bord :

La détection des contours comprend une variété de méthodes mathématiques qui visent à identifier les points d'une image numérique où la luminosité de l'image change brusquement ou présente des discontinuités. Les points où la luminosité de l'image change brusquement sont généralement organisés en un ensemble de segments de lignes courbes appelés bords. Les bords sont souvent associés aux limites des objets dans une image. Dans ce codage, la méthode de détection des bords de Canny est utilisée pour convertir une image RVB en image de bord en utilisant cv2.canny ( ).

Élément de correction

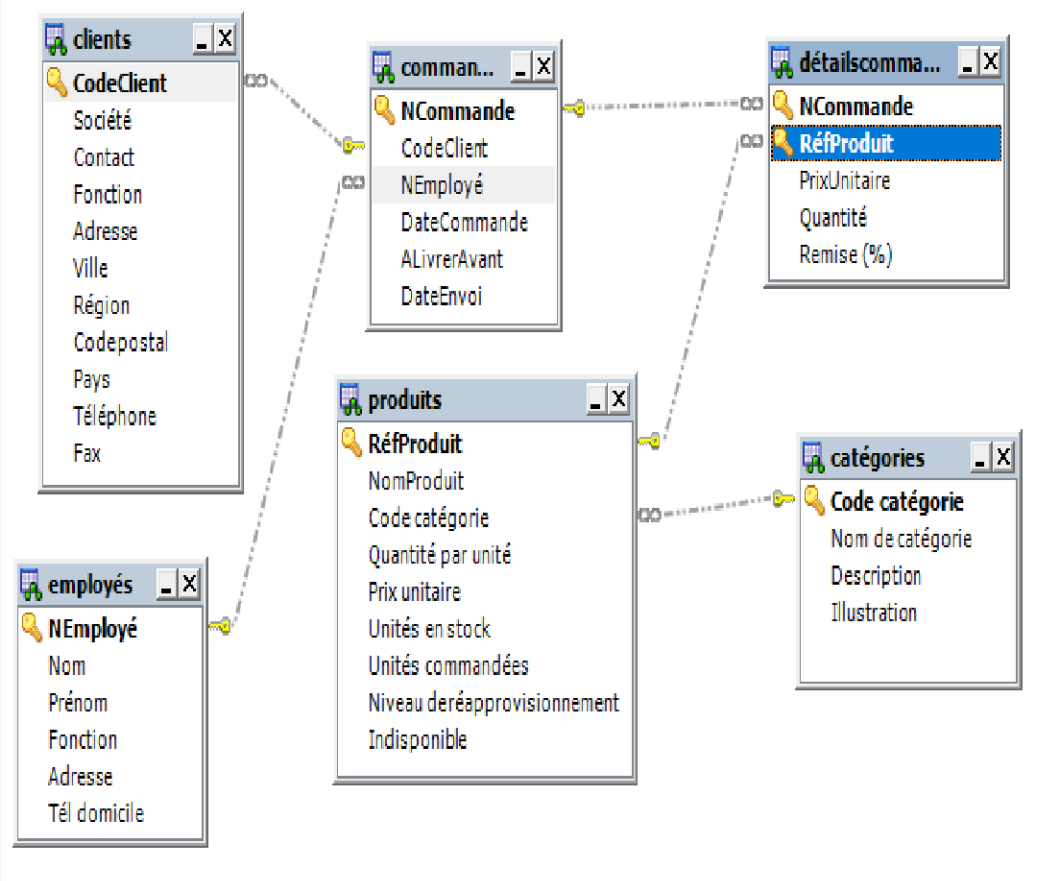
```
# importer les paquets nécessaires
from tkinter import *
from PIL import Image
from PIL import ImageTk
from tkinter import filedialog
import cv2
from tkinter import filedialog
from tkinter.filedialog import askopenfilename
def start():
    global image1,path
    global panel
    path = filedialog.askopenfilename()
    image1 = cv2.imread(path)
    image1 = cv2.resize(image1,(200,100))
    image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
    image = Image.fromarray(image1)
    image = ImageTk.PhotoImage(image)
    if panel is None:
        panel = Label(image=image)
        panel.image = image
        panel.pack(side="left", padx=30, pady=60)
    else:
        panel.configure(image=image)
        panel.image = image
def gray_image():
    global panelB
    gray = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    gray = Image.fromarray(gray)
    gray = ImageTk.PhotoImage(gray)
    if panelB is None:
        panelB = Label(image=gray)
        panelB.image = gray
        panelB.pack(side="left", padx=30, pady=60)
    else:
        panelB.configure(image=gray)
        panelB.image = gray
def edge_image():
    global panelB
    gray = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    edge = cv2.Canny(gray, 50, 100)
    edge = Image.fromarray(edge)
    edge = ImageTk.PhotoImage(edge)
    if panelB is None:
        panelB = Label(image=edge)
        panelB.image = edge
        panelB.pack(side="left", padx=30, pady=60)
```

```
        else:
            panelB.configure(image=edge)
            panelB.image = edge
    def negative_image():
        global panelB
        image = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
        negative = cv2.bitwise_not(image)
        negative = Image.fromarray(negative)
        negative = ImageTk.PhotoImage(negative)
        if panelB is None:
            panelB = Label(image=negative)
            panelB.image = negative
            panelB.pack(side="left", padx=30, pady=60)
        else:
            panelB.configure(image=negative)
            panelB.image = negative
    root = Tk()
    root.geometry('600x400')
    fm = Frame(root, width=300, height=200)
    fm.pack(side=BOTTOM, expand=NO, fill=NONE)
    panelB = None
    panel = None
    btn = Button(root, text="Select an image",
    command=start,activebackground='red',fg="blue")
    btn.pack(side="top")
    btn2 = Button(fm, text="GRAY",
    command=gray_image,activebackground='red',height=2, width=10,fg="gray")
    btn2.pack(side=LEFT, padx="10", pady="20")
    btn3 = Button(fm, text="EDGE",
    command=edge_image,activebackground='red',height=2, width=10,fg="green")
    btn3.pack(side=LEFT, padx="10", pady="20")
    btn4 = Button(fm, text="NEGATIVE",
    command=negative_image,activebackground='red',height=2,
    width=10,fg="purple")
    btn4.pack(side=LEFT, padx="10", pady="20")
    root.mainloop()
```

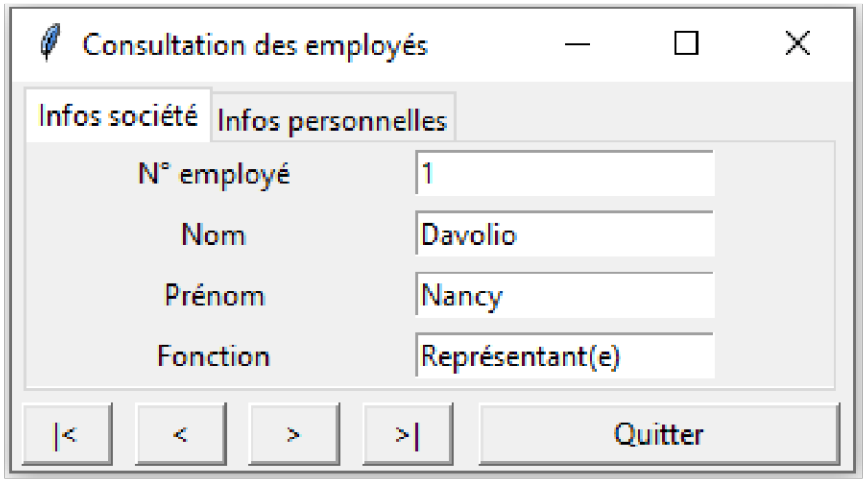
TP N°10 : IHM et accès aux bases de données

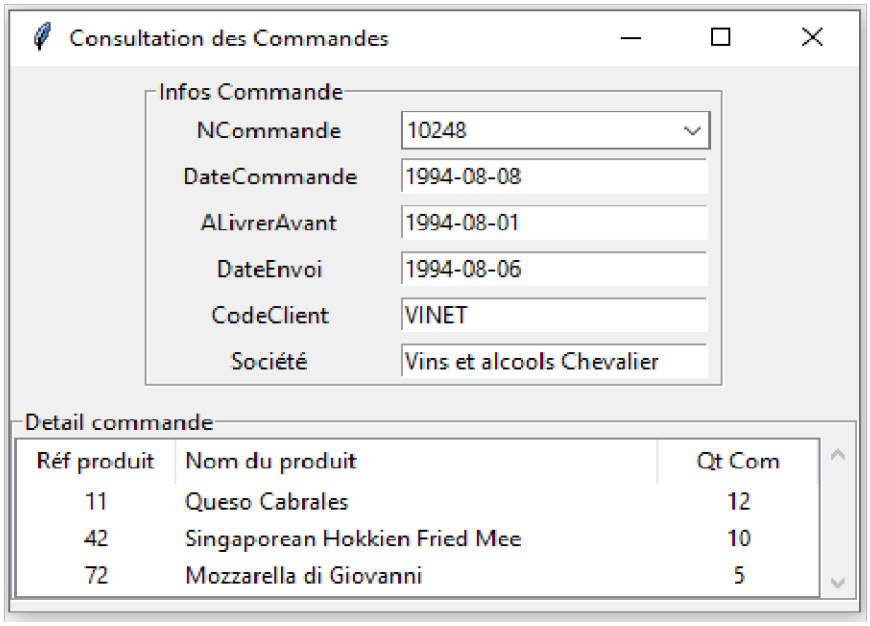
Énoncé

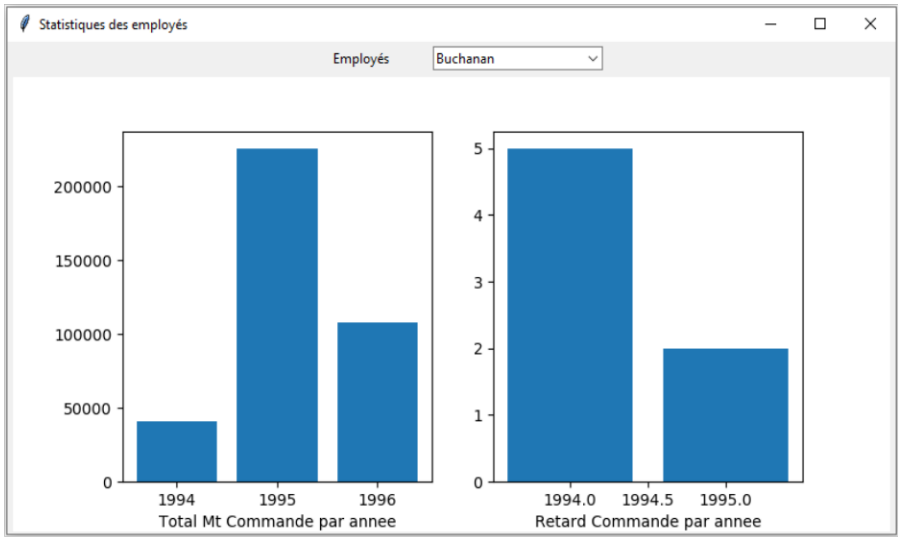
Le but de ce TP est de créer des interfaces graphiques pour une application de gestion commerciale. Les informations relatives aux ventes sont stockées dans une base de données MYSQL « dbcomptoire». Le schéma de cette base est donné par la figure ci-dessous



1. Définir la classe « Connexion » permettant d'obtenir une connexion à la base de données « GestionPersonnes ».
2. On considère les interfaces graphiques suivantes. Définir la classe permettant d'assurer les fonctionnalités de chaque interface.


Consultation des employés
Infos société
Infos personnelles
N° employé
1
Nom
Davolio
Prénom
Nancy
Fonction
Représentant(e)
|<
<
>
>|
Quitter


Consultation des Commandes
Infos Commande
NCommande
10248
DateCommande
1994-08-08
ALivrerAvant
1994-08-01
DateEnvoi
1994-08-06
CodeClient
VINET
Société
Vins et alcools Chevalier
Detail commande
Réf produit
Nom du produit
Qt Com
11
Queso Cabrales
12
42
Singaporean Hokkien Fried Mee
10
72
Mozzarella di Giovanni
5


Statistiques des employés
Employés
Buchanan
200000
150000
100000
50000
0
1994
1995
1996
Total Mt Commande par annee
5
4
3
2
1
0
1994.0
1994.5
1995.0
Retard Commande par annee

Élément de correction

1

```
from pymysql import *
class Connexion(object):
   def __init__(self):
      self.dbconection = connect(host='localhost',
                    port=3306, user='root', db='dbcomptoire'  )
      self.dbcursor = self.dbconection.cursor()
   def commit_db(self):
      self.dbconection.commit()
   def close_db(self):
      self.dbcursor.close(); self.dbconection.close()
```

2

Interface consultation des employés

```
from tkinter import Frame, Tk, ttk, DISABLED, NORMAL
from tkinter.ttk import *
from dao import Connexion
class Interface(Frame):
    def initComponents(self, window):
        Frame.__init__(self, window)
        window.geometry("385x215")
        self.tabControl = ttk.Notebook(window)
        self.tab1 = ttk.Frame(self.tabControl)
        self.tab2 = ttk.Frame(self.tabControl)
        self.tabControl.add(self.tab1, text='Infos société')
        self.tabControl.add(self.tab2, text='Info personnelles')
        self.tabControl.pack(expand=1, fill="both")
        # La premiere tab de notre interface
        Label(self.tab1, text='N° Employé :').grid(column=0,
                    row=0, padx=10, pady=10)
        self.nemp = Entry(self.tab1, width=35)
        self.nemp.grid(column=1, row=0, padx=10, pady=10)
        Label(self.tab1, text='Nom :').grid(column=0, row=1,
                   padx=10, pady=10)
        self.nom = Entry(self.tab1, width=35)
        self.nom.grid(column=1, row=1, padx=10, pady=10)
        Label(self.tab1, text='Prénom :').grid(column=0, row=2,
                  padx=10, pady=10)
        self.prnom = Entry(self.tab1, width=35)
        self.prnom.grid(column=1, row=2, padx=10, pady=10)
        Label(self.tab1, text='Fonction :').grid(column=0,
                    row=3, padx=10, pady=10)
        self.fonction = Entry(self.tab1, width=35)
        self.fonction.grid(column=1, row=3, padx=10, pady=10)
```

```
        self.firstButton = Button(window, text='|<', command='')
        self.firstButton.pack(side="left", expand=1)
        self.previousButton=Button(window, text='<', command='')
        self.previousButton.pack(side="left", expand=1)
        self.nextButton = Button(window, text='>', command='')
        self.nextButton.pack(side="left", expand=1)
        self.lastButton = Button(window, text='>|', command='')
        self.lastButton.pack(side="left", expand=1)
        style = Style()
        style.configure('W.TButton', font=('calibri', 10,
                'bold', 'underline'), foreground='red')
        self.cancelButton = Button(window, text='Quitter',
                 style='W.TButton', command=window.destroy)
        self.cancelButton.pack(side="left", expand=1)

        # La deuxieme tab de notre interface consultation
        Label(self.tab2, text='N° Employé :').grid(column=0,
               row=0, padx=10, pady=10)
        self.nemptab2 = Entry(self.tab2, width=35)
        self.nemptab2.grid(column=1, row=0, padx=10, pady=10)
        Label(self.tab2, text='Adresse Personnelle
                :').grid(column=0, row=1, padx=10, pady=10)
        self.address = Entry(self.tab2, width=35)
        self.address.grid(column=1, row=1, padx=10, pady=10)
        Label(self.tab2, text='Tel Domicile :').grid(column=0,
                   row=2, padx=10, pady=10)
        self.tel = Entry(self.tab2, width=35)
        self.tel.grid(column=1, row=2, padx=10, pady=10)

    def __init__(self, window):
        self.initComponents(window)
        self.list_emp= Connexion.Connexion().get_all_employees()
        self.pos=0;
        self.affiche()
        self.firstButton.bind("<Button>",
                              lambda x:self.move_first())
        self.lastButton.bind("<Button>",
                             lambda x:self.move_last())
        self.nextButton.bind("<Button>",
                             lambda x:   self.next())
        self.previousButton.bind("<Button>",
                                 lambda x:   self.previous())
```

```
    def affiche(self):
        self.nemp.delete(0, 'end')
        self.nemp.insert(0, self.list_emp[self.pos][0])
        self.nom.delete(0, 'end')
        self.nom.insert(0, self.list_emp[self.pos][1])
        self.prnom.delete(0, 'end')
        self.prnom.insert(0, self.list_emp[self.pos][2])
        self.fonction.delete(0, 'end')
        self.fonction.insert(0, self.list_emp[self.pos][3])
        self.nemptab2.delete(0, 'end')
        self.nemptab2.insert(0, self.list_emp[self.pos][0])
        self.address.delete(0, 'end')
        self.address.insert(0, self.list_emp[self.pos][4])
        self.tel.delete(0, 'end')
        self.tel.insert(0, self.list_emp[self.pos][5])
        if self.pos == 0:
            self.previousButton['state'] = DISABLED
        else:
            if self.pos == len(self.list_emp) - 1:
                self.nextButton['state'] = DISABLED
            else:
                self.previousButton['state'] = NORMAL
                self.nextButton['state'] = NORMAL

    def move_first(self):
        self.pos=0
        self.affiche()
    def move_last(self):
        self.pos = len(self.list_emp)-1
        self.affiche()
    def next(self):
        if self.pos != len(self.list_emp)-1:
            self.pos+=1
            self.affiche()
    def previous(self):
        if self.pos != 0:
            self.pos -= 1
            self.affiche()

window = Tk()
window.title('Consultation des employés')
interface = Interface(window)
interface.mainloop()
```

Interface consultation des commandes

```
from tkinter import *
from tkinter import ttk
from Connexion import *

class Interface(Frame):
    champs=["DateCommande","ALivrerAvant","DateEnvoi","CodeClient",
            "Société"]
    def __init__(self, fenetre, **kwargs):
        self.db = Connexion()
        Frame.__init__(self, fenetre)
        self.pack(fill=BOTH)
        frm1 = Frame(self, pady=8)
        frm1.pack()
        label=Label(frm1,text="NCommande").grid(row=0,
             column=0,pady=8)
        self.db.dbcursor.execute('SELECT NCommande FROM commandes')
        results = self.db.dbcursor.fetchall()
        self.NCommande = IntVar()
        self.combo = ttk.Combobox(frm1, values=results,
             textvariable=self.NCommande )
        self.combo.grid(row=0, column=1,
                 pady=8);self.combo.current(0)
        self.widget=dict()
        for i in range(len (Interface.champs)):
            c=Interface.champs[i]
            Label(frm1, text=c, width= 20).grid(row=i+1, column=0,
                  padx=5, pady=2,sticky=NW)
            self.widget[c] = Entry(frm1)
            self.widget[c].grid(row=i+1, column=1, padx=5, pady=2)
        frm2 = Frame(self);   frm2.pack()
        self.tree = ttk.Treeview(frm2, columns=  (1, 2, 3),
                height=3, show="headings")
        self.tree.place(x=50, y=50, width=300)
        self.tree.column(1, width=50);
        self.tree.column(2, width=100)
        self.tree.column(3, width=150);
        self.tree.heading(1, text="Réf produit");
        self.tree.heading(2, text="Nom du produit")
        self.tree.heading(3, text="Qt Com");
        sbar = ttk.Scrollbar(frm2,
                orient='vertical',command=self.tree.yview)
```

```
            sbar.pack(side=RIGHT, fill=Y)
            self.tree.pack(side=LEFT, expand=YES, fill=BOTH)
            self.tree.configure(yscroll=sbar.set)
            self.combo.bind("<<ComboboxSelected>>", self.comboAction)
        def comboAction (self, event):
            req="select NCommande, DateCommande, ALivrerAvant,
                  DateEnvoi, cd.CodeClient, Société from" \
                " commandes cd join clients c on c.CodeClient= " \
                " cd.CodeClient where NCommande = "+ str(self.NCommande)
            self.db.dbcursor.execute(req)
            results = self.db.dbcursor.fetchone()
            for i in range(len(Interface.champs)):
                c=Interface.champs[i]
                self.widget[c].delete(0, END)
                self.widget[c].insert(0, self.results[self.pos][i])

fenetre = Tk()
fenetre.title("Consultation des employés")
interface = Interface(fenetre)
interface.mainloop()
```

Interface des statistiques des employés

```
from tkinter import *
from tkinter.ttk import *
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from numpy import array
from dao import Connexion

class Interface(Frame):
    def __init__(self, fenetre):
        Frame.__init__(self, fenetre)
        window.geometry("485x355")
        self.pack(fill=BOTH)
        frm1 = Frame(self);  frm1.pack()
        Label(frm1, text='Employé :').grid(column=0,row=0)
        # list_values = dbconn.get_employees()
        self.nomemp = StringVar()
        self.empname=Combobox(master=frm1,textvariable=
            self.nomemp,values = list_values, state='readonly')
        self.empname.grid(column=1, row=0)
        self.empname.bind("<<ComboboxSelected>>",lambda x:
                    self.plot())
        self.fig = Figure(figsize=(8, 8), dpi=100)
```

```
            self.fig.clear()
        def plot(self):
            name=self.nomemp.get()
            res = array(dbconn.get_total_mt_cmd(name))
            annee = res.transpose()[0];  mt = res.transpose()[1]
            plot1 = self.fig.add_subplot(121)
            plot1.clear()
            plot1.bar(annee,mt)
            plot1.text(0.0,-1.5,'Total Mt Commande par année',
                    horizontalalignment='center', fontsize=10)
            self.canvas = FigureCanvasTkAgg( self.fig, master=self)
            # placing the canvas on the Tkinter window
            self.canvas.get_tk_widget().pack(fill=BOTH,expand=True)
            self.canvas.draw_idle()
            res = array(dbconn.get_retard_cmnd(name))
            print(res)
            anneeRetard = res.transpose()[0]
            retard = res.transpose()[1]
            plot2 =  self.fig.add_subplot(122)
            plot2.clear()
            # plotting the graph
            plot2.bar(anneeRetard,retard)
            self.canvas = FigureCanvasTkAgg( self.fig, master=self)
            self.canvas.draw_idle()
            #placing the canvas on the Tkinter window
            self.canvas.get_tk_widget().pack(fill=BOTH, expand=True)

window = Tk()
dbconn = Connexion.Connexion()
window.title('Statistiques des employés')
interface = Interface(window)
interface.mainloop()
```

Exemple de projets de module

Plusieurs plateformes open source rendent le codage de réseaux de neurone facile (à titre d'exemple TensorFlow, Theano, CNTK ou Keras …). A partir de données réelles le réseau peut "apprendre" pour ensuite être capable de traiter de nouvelles données.

1ière partie du travail (commun)

La 1ère partie du travail à réaliser consiste à concevoir un réseau de neurone de convolution afin de classifier une image. Les 1ière tests se feront sur la base de données MNIST qui comporte des images manuscrites de chiffres. Une fois que le réseau a appris, on lui soumet une image d'un chiffre et il doit produire en sortie le chiffre reconnu.

2ème partie du travail (spécifique)

L'objectif de cette partie est d'exploiter la reconnaissance d'images de la base de données MNIST et élargir cette expérience pour d'autre finalités :

- **Sujet 1.** Extraction et détection d'objet dans une image(«Object detection»)
- **Sujet 2.** Détection des piétons dans une image ou vidéo (« Pedestrian detection »)
- **Sujet 3.** Détection des plaques d'immatriculation d'une voiture
- **Sujet 4.** Système de détection d'incendie (« Fire Detection »)
- **Sujet 5.** Reconnaissance des visages (Face recognition)
- **Sujet 6.** Reconnaissance du port du mask (« Face Mask Detection »)
- **Sujet 7.** Reconnaissance du texte (agent conversationnel chatbot)
- **Sujet 8.** Reconnaissance de la voix (agent conversationnel voicebot)
- **Sujet 9.** Reconnaissance des émotions humaines
- **Sujet 10.** Système de surveillance visuelle
- **Sujet 11.** Détection et Suivi d'un ballon de basketball dans une vidéo
- **Sujet 12.** Détection et Suivi d'une balle de tennis dans une vidéo
- **Sujet 13.** Agent conversationnel chatbot basé sur l'API openAI
- **Sujet 14.** Détection et construction de la trajectoire d'un ballon de football

NB. Chaque projet se conclut par un rapport et une soutenance. Le rapport doit souligner la contribution des étudiants, illustrer les résultats numériques obtenus et la façon dont ils ont été obtenus. La soutenance doit reprendre les points principaux, décrire brièvement la façon dont le travail a été réalisé.