INF1 – P21

Examen Final

Durée : 1 heure 30

.

Document autorisé : une feuille de notes A4 recto-verso
Calculatrices, téléphones, traducteurs et ordinateurs interdits.

Attention : chaque partie doit être rédigée sur une copie séparée

N.B. : on s'attachera à fournir tout commentaire utile et à écrire de façon claire et lisible.

1ère partie : vente à distance *(7 points)*

Un site internet de vente à distance gère les commandes et le stock à l'aide des classes suivantes :

- La classe `Produit` permet de représenter un produit par sa référence (`ref`), son prix unitaire (`prix`), et la quantité de ce produit en stock (`stock`) :

```
class Produit :
    def __init__(self, ref, stock, prix) :
        self.ref = ref
        self.stock = stock
        self.prix = prix

    def toString(self) :
        return f'{self.ref}({self.stock}) - prix : {self.prix}'
```

  N.B. : on considérera que la référence d'un produit est son nom tout simplement.

- La classe `Stock` permet de gérer la liste des produits :

```
class Stock :
    def __init__(self) :
        self.produits = []
    def afficheStock(self) :
        for produit in self.produits :
            print(produit.toString())
```

- La classe `ItemCommande` représente un item de commande, c'est-à-dire le nom d'un produit et la quantité désirée :

```
class ItemCommande :
    def __init__(self, ref, nb) :
        self.ref = ref
        self.nb = nb
```

Une commande est simplement une liste constituée d'un ou plusieurs items de commande.

Voici un exemple d'utilisation :

```
stock = Stock()
stock.ajouteListeProduits([Produit('stylo bleu', 1500, 1.1),
                           Produit('gomme', 200, 0.55),
                           Produit('cahier A4', 350, 2.3)
                           Produit('petit agenda', 589, 13)])
commande = [ItemCommande('gomme', 100),
            ItemCommande('cahier A4', 300),
            ItemCommande('petit agenda', 200)]
if stock.commandePossible(commande) :
    print('Montant :', stock.montantCommande(commande))
    stock.fournitCommande(commande)
    stock.afficheStock()
```

On vous demande d'écrire les méthodes de la classe Stock suivantes :

1. Écrire la méthode `ajouteListeProduits(self, listeProduits)` qui permet d'ajouter une liste de produits à la liste `self.produits`.
2. Écrire la méthode `trouveProduit(self, ref)` qui retourne un produit à partir de sa référence. On supposera que la référence `ref` est bien celle d'un des produits du stock.
3. Écrire la méthode booléenne `commandePossible(self, commande)` qui détermine si une commande est possible ou non en fonction des stocks disponibles. On supposera que les références demandées sont correctes, que les quantités sont positives et que deux items ne correspondent pas à la même référence.
4. Écrire la méthode `montantCommande(self, commande)` qui retourne le montant d'une commande.
5. Écrire la méthode `fournitCommande(self, commande)` qui modifie le stock après la commande.

N.B. les questions sont indépendantes entre elles. La méthode `trouveProduit` de la question 2 pourra éventuellement être utilisée dans les autres questions.

```
class Produit :
    def __init__(self, ref, stock, prix) :
        self.ref = ref
        self.stock = stock
        self.prix = prix

    def toString(self) :
        return f'{self.ref}({self.stock}) - prix : {self.prix}'

class Stock :
    def __init__(self) :
        self.produits = []
    def afficheStock(self) :
        for produit in self.produits :
            print(produit.toString())

# ajoute une liste de produits
```

```
    def ajouteListeProduits (self, listeProduits) :
        for produit in listeProduits :
            self.produits.append(produit)


# en supposant que le produit existe)
    def trouveProduit(self, ref) :
        i = 0
        trouve = False
        while i < len(self.produits) and not trouve :
            p = self.produits[i]
            if ref == p.ref :
                trouve = True
                pr = p
            i += 1
        return pr

# On est certain que 2 items n'ont pas la mêmme référence

    def commandePossible(self, commande) :
        possible = True
        k = 0
        while k < len(commande) and possible :
            item = commande[k]
            produit = self.trouveProduit(item.ref)
            if item.nb > produit.stock :
                possible = False
            k += 1
        return possible

# donne le montant d'une commande
    def montantCommande(self, commande) :
        total = 0
        for item in commande :
            produit = self.trouveProduit(item.ref)
            total += item.nb * produit.prix
        return total

# modifie le stock après commande
    def fournitCommande(self, commande) :
        if self.commandePossible(commande) :
            for item in commande :
                produit = self.trouveProduit(item.ref)
                produit.stock -= item.nb
            print('Le stock a été modifié')
        else :
            print("La commande n'est pas possible")

class ItemCommande :
    def __init__(self, ref, nb) :
        self.ref = ref
        self.nb = nb
```

---------- ***{ prendre une nouvelle copie }*** ----------

2ème partie : récursivité *(6 points)*

1. On donne la définition récursive suivante du PGCD de deux nombres :
   « Le PGCD de deux nombres entiers est égal au PGCD du plus petit et du reste de la division du plus grand par le plus petit. Le PGCD est alors le dernier reste non nul. »

   Ainsi, partant des nombres 50 et 46, on obtient successivement 46 et 4, puis 4 et 2, puis 2 et 0. Le pgcd est donc 2.

   Écrire une fonction récursive retournant le pgcd de deux nombres suivant cette règle.

```
def pgcd(a, b) :
    print(a, b)
    if b == 0 :
        return a
    if a >= b :
        return pgcd(b, a % b)
    else :
        return pgcd(a, b % a)
```

2. Une liste l_1 est une sous-liste d'une liste l si tous les éléments de l_1 sont des éléments de l et si l'ordre des éléments de l_1 est respecté dans l. Ainsi :

   [2, 10, 4, 8] est une sous-liste de [9, 2, 11, 10, 7, 3, 4, 8, 25]
   [2, 10, 4, 8] n'est pas une sous-liste de [9, 10, 7, 3, 4, 8, 25]
   Écrire une fonction récursive indiquant si une liste l_1 est une sous-liste d'une liste l.

   Indication : pour écrire la fonction on pourra considérer le premier élément des deux listes et appeler récursivement sur la fin d'au moins une des deux listes. On obtient un succès si l'on a réussi à vider l1 avant l.

```
def sousListe(sous, liste) :
    if sous == [] :
        return True
    if liste == [] :
        return False
    if sous[0] == liste[0] :
        return sousListe(sous[1:], liste[1:])
    else :
        return sousListe(sous, liste[1:])
```

---------- ***{ prendre une nouvelle copie }*** ----------

3ème partie : fichiers de joueurs *(7 points)*

Le sélectionneur Didier Deschamps a fait appel à 26 joueurs pour composer l'équipe de France qui participera à l'Euro2021 de football.
Il dispose d'un fichier texte contenant le nom et le prénom de ses joueurs. Chaque ligne de ce fichier contient le nom et le prénom séparés par une virgule (pas d'espace avant ou après la virgule), de la manière suivante :

…
Benzema,Karim
...
Mbappe,Kylian
…
Toutefois, le sélectionneur souhaite avoir plus d'informations sur chaque joueur. Pour cela, il envoie son fichier à la fédération française de football, et demande qu'on y ajoute, pour chaque joueur : la date de naissance et la liste des clubs où le joueur a évolué en parcours professionnel, chaque club étant suivi du nombre de buts marqués par le joueur dans ce club.

Chaque ligne du fichier final aura la structure suivante :
…
Benzema,Karim,19/12/1987,Real Madrid,279,Olympique Lyonnais,81,…
...

1) Donnez une structure de données de type dictionnaire qui peut contenir toutes les données relatives à un joueur, souhaitées par le sélectionneur. Utilisez les données relatives au joueur cité comme exemple.

```
{"nom": "Benzema", "prenom" : "Karim",
 "ddn" : {"jour":19, "mois":12, "annee":1982},
 "clubs":[{"club":'Real Madrid',"nb_buts":279},
          {"club":'Olympique Lyonais',"nb_buts":81}]}
```

2) Écrire une fonction `creer_joueur(nom, prenom)`, qui prend en entrée le nom et prénom d'un joueur, demande à l'utilisateur les autres informations, et retourne le dictionnaire correspondant au joueur.

```
def creer_joueur(nom, prenom):
     print("Entrez la date de naissance du joueur : ",nom,", ",
prenom)
     ddn_jour = int(input("jour : "))
     ddn_mois = int(input("mois : "))
     ddn_annee = int(input("année : "))
     #lecture de la liste de clubs
     print("Entrez la liste de clubs où le joueur a évolué en
parcours professionnel :")
     liste_clubs = []
     continuer = 'O'
     while continuer.upper() == 'O' :
          club = input("Club : ")
          nb_buts = int(input("Nombre de buts marqués : "))
          liste_clubs.append({"club" : club, "nb_buts" :
nb_buts})
          continuer = input("Continuer (O/N) ?")
     un_joueur = {"nom": nom, "prenom" : prenom, "ddn" :
{"jour": ddn_jour, "mois": ddn_mois, "annee" : ddn_annee},
"clubs": liste_clubs}
     return un_joueur
```

3) Écrire une fonction `creer_liste_joueurs(nom_fichier)`, qui prend en entrée le nom du fichier d'origine du sélectionneur et retourne la liste des joueurs. Cette fonction fera appel à `creer_joueur(nom, prenom)` pour créer les joueurs.

```
def extrait_nom_prenom(ligne):
     i = 0
     chaine = ""
     #extrait le nom
     while ligne[i] != ',' :
          chaine = chaine + ligne[i]
          i+=1
     #extrait le prenom (sans le "\n" de fin ligne)
     nom_prenom = [chaine, ligne[i+1 : len(ligne)-1]]
     return(nom_prenom)

def creer_liste_joueur(fich_selec) :
     fp = open(fich_selec, "r") #ouvre le fichier d'origine du
sélectionneur en lecture
     liste_joueurs_selec = []
     ligne = fp.readline()
     while ligne != "" :
          nom_prenom = extrait_nom_prenom(ligne)
          nom = nom_prenom[0]
          prenom = nom_prenom[1]
          liste_joueurs_selec.append(creer_joueur(nom, prenom))
          ligne = fp.readline()
     fp.close()
     return liste_joueurs_selec
```

N.B. : pour extrait_nom_prenom on peut aussi utiliser la méthode index des chaines de caractères. La méthode split pourrait également être utilisée.

4) Écrire une fonction `fich_select_final(liste_joueurs, nom_fichier)`, qui prend en entrée la liste de joueurs et crée le fichier final de Didier Deschamps.

```
def fich_select_final(liste_joueurs, nom_fichier) :
    fp = open(nom_fichier,"w")
    for i in range(0,len(liste_joueurs)) :
        joueur = liste_joueurs[i]
        une_ligne = joueur["nom"] + "," + joueur["prenom"] +
                    "," + str(joueur["ddn"]["jour"]) + "/" +
                    str(joueur["ddn"]["mois"]) + "/" +
                    str(joueur["ddn"]["annee"]) + ","
        for j in range(0,len(joueur["clubs"])) :
            une_ligne += joueur["clubs"][j]["club"] + "," +
                         str(joueur["clubs"][j]["nb_buts"])
        fp.write(une_ligne + "\n")
    fp.close()
```