2 DS : LES FONCTIONS – LES COLLECTIONS D'OBJETS

- On attachera une importance à la concision, à la clarté, et à la précision de la rédaction
- Si, au cours du DS, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.
- Le devoir se compose de deux problèmes indépendants sur 6 pages.
- Les questions non traitées peuvent être admises pour traiter des questions ultérieures.

Problème 1. (20 pts)

ANALYSEUR LEXICAL D'UN LANGAGE

Présentation. Avant son exécution, tout code source d'un programme informatique doit être traduit en un autre code appelé code machine. Cette traduction est réalisée par un compilateur dont le rôle est de transformer le code source en une suite d'instructions élémentaires directement exécutables par le processeur.

Ainsi le rôle de l'analyseur lexical est d'identifier puis supprimer les caractères superflus du code source (commentaires, espaces, ..) et de reconnaître les mots clés, les identificateurs, les opérateurs qui sont définis par un ensemble de règles.

En plus, l'analyseur lexical signale les éventuelles erreurs de syntaxe et associe à chaque erreur le numéro de ligne dans laquelle elle intervient.

Dans ce problème, on se propose de mettre en œuvre un analyseur lexical. Il s'agit d'implémenter des fonctions en langage PYTHON pour un analyseur lexical d'un langage imaginaire qu'on appellera LIM.

***

Rappel. Les fonctions et méthodes prédéfinies qui peuvent être utilisées dans ce problème :

- La fonction ***len(seq)*** : renvoie le nombre d'éléments de la séquence seq.
- La fonction ***range(deb,fin,pas)*** : renvoie … avec le pas
- La méthode ***append(elem)*** ajoute l'élément elem à la fin d'une liste.
- La méthode ***values( )*** renvoie toutes les valeurs d'un dictionnaire.
- La méthode ***items( )*** renvoie tous les objets d'un dictionnaire.
- La fonction ***print(msg)*** affiche le message msg.

***

Gestion des commentaires. Un commentaire est une instruction qui n'est pas traduite par le compilateur. Pour notre langage LIM, un commentaire est une chaîne de caractères qui doit commencer obligatoirement par le caractère # (dièse).

Exemple. La chaîne de caractères com1='# du texte' est un commentaire. La chaîne de caractères com2='ceci est commentaire !' n'est pas un commentaire.

Question 1.	(1 pt) Ecrire une fonction ***comment(com)*** qui retourne True si la chaîne de caractères com est un commentaire du langage LIM et False sinon.

Gestion des espaces multiples. Dans une instruction, on a tendance à supprimer les espaces multiples qui se succèdent et de ne conserver qu'un seul espace.

Exemple. La chaîne de caractères inst= ' x = 21 ' ne doit pas être traitée par le processeur avant de supprimer les espaces superflus. On doit avoir le résultat suivant : inst= 'x = 21'.

Question 2.	(4 pts) Ecrire une fonction ***suppEspaces(inst)*** qui supprime les espaces multiples consécutifs (qui se suivent) entre les caractères de la chaîne ***inst*** en paramètre. S'il y'a plusieurs espaces au début ou à la fin, on ne conserve aucun.

Reconnaissance des mots clés. Tout langage de programmation définit un ensemble de mots clés qui sont des chaînes de caractères ayant des significations et des utilisations spécifiques pour ce langage.

Pour notre langage LIM, on suppose avoir défini :

- **MotsCles** : un tuple qui contient les chaînes de caractères qui représentent les mots clés du langage LIM. Pour le moment le tuple est : ***MotsCles = ('si','sinon','tantque','pour','definir')***.

On veut réaliser une fonction qui vérifie si une chaîne de caractères en paramètre est un mot clé du langage LIM ou non. Ça sera un mot clé si cet un élément du tuple MotCles.

Exemple. Pour une chaîne de caractères mot1='si' la fonction renvoie True, et pour mot2='entier' la fonction renvoie False.

Question 3.	(2 pts) Ecrire une fonction ***motCle(mot)*** qui retourne True si ***mot*** (le paramètre de la fonction) est un mot clé de LIM et False sinon.

Validité d'un identificateur. Un identificateur est une chaîne de caractères qui permet de définir et d'identifier les éléments d'un programme (les variables, les fonctions, ..). Il doit respecter un ensemble de règles particulières pour chaque langage.

Un identificateur du langage LIM est valide s'il respecte les 4 conditions suivantes :

- Sa longueur est strictement inférieure à 80.
- Ne contient aucun espace.
- Ne commence pas par un chiffre ('0','1','2','3','4','5','6','7','8','9').
- Ne doit pas être un mot clé du langage LIM.

Question 4.	(5 pts) Ecrire une fonction ***identificateur(iden)*** qui retourne True si son paramètre (la chaîne de caractères ***iden***) est un identificateur valide de LIM et False sinon.

Analyse d'une instruction. Une instruction du langage LIM est correcte si elle vérifie l'une des conditions suivante :

- L'instruction est un commentaire du langage LIM.
- L'instruction commence par un identificateur valide suivi d'un espace suivi d'une chaîne de caractères. On doit supprimer les espaces superflus.
- L'instruction commence par un mot clé du langage LIM suivi par un espace, suivi d'une chaîne de caractères et se termine par le caractère ':' (deux points).

Exemple. La chaîne de caractères inst1='si x<y :' est une instruction valide.

L'instruction inst2='5=7' n'est pas une instruction valide.

L'instruction inst3='alpha = 5.55' est une instruction valide.

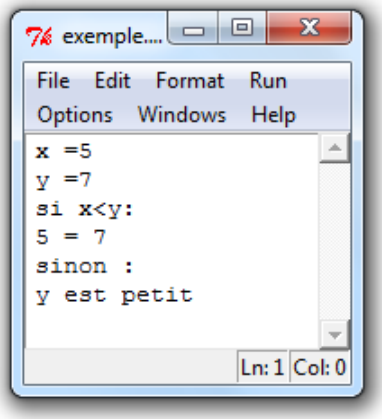
Question 5. (4 pts) Ecrire une fonction ***analyserInstruction(inst)*** qui retourne True si la chaîne de caractères ***inst*** en paramètre correspond à une instruction correcte de LIM et False sinon.

Analyse d'un code source. On suppose maintenant que les instructions d'un fichier source sont mises dans une liste. On définit ainsi une liste ListeInst :

- **ListeInst** = ['x =5','y =7','si x<y:','5 = 7','sinon :','y est petit']

Qui sera l'équivalent d'un fichier source illustré à coté.

Au lieu de compiler et d'analyser le fichier, on va analyser la liste **ListeInst.**



Question 6. (4 pts) Ecrire une fonction ***analyserSource(L)*** qui analyse toutes les instructions la liste ***L*** en paramètre. Cette fonction affiche sur l'écran les indices de tous les éléments de la liste ***L*** qui correspondent à des instructions incorrectes. Elle affiche « succes de compilation » dans le cas où toutes les instructions sont correctes.

Exemple. Après l'appel de la fonction : ***analyserSource(ListeInst)*** on aura sur l'écran l'affichage du numéro 3 qui est l'indice de l'instruction '5 = 7' qui n'est pas correcte.

Problème 2. (20 pts)

DÉTERMINATION DES ITINÉRAIRES

Présentation. Dans le monde actuel, il est très utile de pouvoir localiser géographiquement des personnes ou des objets aussi bien pour des besoins privés que professionnels.

Au niveau professionnel, la géo-localisation est très utilisée notamment pour la navigation routière mais aussi dans plusieurs autres domaines (Transport, logistique, énérgie, ..) favorisant ainsi un gain de productivité et une sécurité accrue.

Des systèmes de géo-localisation – tels que GPS – permettent le repérage de la position géographique exacte de leurs usagers, ainsi que le calcul d'itinéraires.

C'est dans cette optique que s'inscrit le problème suivant concernant des algorithmes de détermination de chemins et d'itinéraires entre les points d'un plan.

***

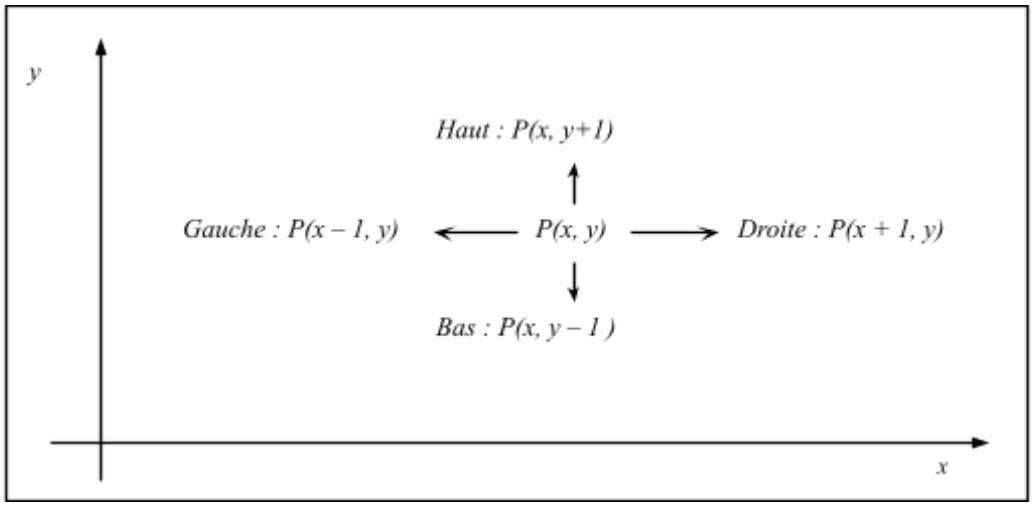
Notations.

- Soit $\mathcal{P}$ un plan, on notera P(x, y), un point de coordonnées *x* et *y* dance ce plan représenté par un tuple *(x, y)*.
- Soit ***N*** une variable de type ***int*** strictement positive à laquelle on donnera la valeur 10 ***(N = 10)*** dans tout le problème. On notera $\mathcal{P}$ (N), l'ensemble des points du plan $\mathcal{P}$ ayant des coordonnées *x* et *y* entières positives telles que (0<=*x*< N) et (0<=*y*< N).

- Soient A et B deux points distincts de $\mathcal{P}$ (N). on appelle **chemin de A vers B**, une liste C, non vide et ordonnée de points distincts (tous différents) de $\mathcal{P}$ (N) tel que :
  $C = [(x_A, y_A), (x_1, y_1), .., (x_i, y_i), .., (x_B, y_B)]$

Chaque point *P(x, y)* de ce chemin est relié au point suivant *P(s,t)* par l'une des 4 relations suivantes :

a. **Gauche** : *P(s, t) est à gauche de P(x, y). c'est-à-dire :* ***s == x – 1*** *et* ***t == y***
b. **Droite** : *P(s, t) est à droite de P(x, y). c'est-à-dire :* ***s == x + 1*** *et* ***t == y***
c. **Haut** : *P(s, t) est au dessus de P(x, y). c'est-à-dire :* ***s == x*** *et* ***t == y + 1***
d. **Bas :** *P(s, t) est en dessous de P(x, y). c'est-à-dire :* ***s == x*** *et* ***t == y – 1***



Exemples de chemins. Soient ***A (2, 3)*** et ***B (5, 5)*** deux points de $\mathcal{P}$ (10) :

C1 = [(2, 3), (3, 3), (4, 3), (4, 4), (4, 5), (5, 5)]

C2 = [(2, 3), (2, 4), (3, 4), (3, 5), (4, 5), (5, 5)]

Les listes C1 et C2 représentent 2 chemins différents de A vers B.

***

Construction de chemins.

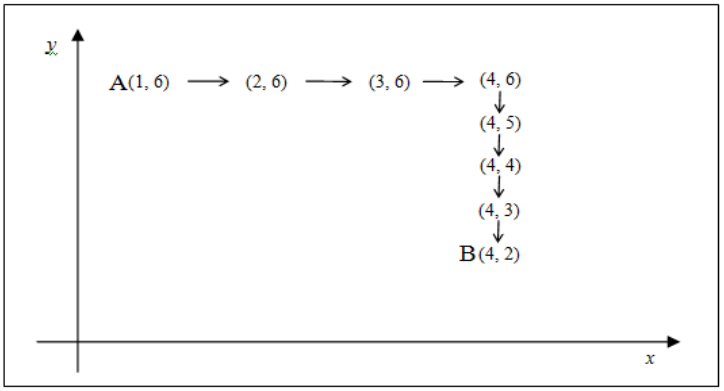
- Soit ***Max*** une variable de type ***int*** strictement positive qui contient le nombre maximal de points que peut contenir un chemin entre 2 points quelconques.

Question 1.	(3 pts) Ecrire une fonction ***initChemin( )*** qui initialise une liste de tuples par les valeurs *(-1, -1)* et la retourne comme résultat. Les points *P (-1, -1)* n'appartient pas à $\mathcal{P}$(N) mais permettent juste d'initialiser la liste.

Chemin horizontal puis vertical. Soient A et B deux points de $\mathcal{P}$ (N). On appelle ***CheminHV*** de A vers B le chemin construit de telle sorte à se déplacer à partir de A horizontalement (soit à gauche, soit à droite) jusqu'à arriver au point ayant la même abscisse que B, puis se déplacer verticalement (soit en haut, soit en bas) jusqu'à arriver au point B.

Exemple. N = 10. Soient A= (1, 6) et B= (4, 2) de $\mathcal{P}$(10), alors ***CheminHV*** de A vers B est :

C = [(1, 6), (2, 6), (3, 6), (4, 6), (4, 5), (4, 4), (4, 3), (4, 2)]



Question 2. (3 pts) Ecrire une fonction ***cheminHV(A, B)*** qui crée et retourne la liste ***HVAB*** de tuples représentant le cheminHV de A vers B. Dans ce cas ***HVAB*[0]** contiendra le tuple (x_A, y_A). ***HVAB*[-1]** contiendra le tuple (x_B, y_B).

Distance d'un chemin et chemin minimal. Soit C un chemin reliant deux points A et B. on appelle distance du chemin C, le nombre de points du chemin C différents du point A.

Exemple. Soient A= (2, 6) et B= (4, 3) deux points et C = [(2, 6), (2, 5), (2, 4), (2, 3), (3, 3), (4, 3)] un chemin de A vers B. La distance de C est de **5**.

On considère maintenant plusieurs chemins différents et tous menant de A vers B. On représente ces chemins par un dictionnaire ***DictChemins*** où les clés seront des chaînes de caractères : C_1, C_2, .. C_N et les valeurs sont les listes représentant ces chemins.

Exemple. Pour les deux points A= (2,6) et B= (4, 3), ***DictChemins*** aura la représentation suivante :

DictChemins = { 'C1' : [(2, 6), (3, 6), (4, 6), (4, 5), (4, 4), (4, 3)] ,

'C2' : [(2, 6), (2, 5), (2, 4), (2, 3), (2, 2), (3, 2), (3, 3), (4,3)] ,

'C3' : [(2, 6), (2, 5), (2, 4), (3, 4), (3, 3), (4, 3)] ,

'C4' : [(2, 6), (1, 6), (0, 6), (0, 5), (0, 4), (0, 3), (1, 3), (2, 3), (3, 3), (4, 3)] , }

Avec C1 et C3 de distance 5, C3 de distance 7 et C4 de distance 9.

D'autre part la distance minimale reliant les 2 points A et B (du dictionnare ***DictChemins***) est 5.

Le chemin minimal est donc C1 = [2, 6), (3, 6), (4, 6), (4, 5), (4, 4), (4, 3)].

Si on a plusieurs chemins minimaux, on prend le premier chemin minimal trouvé pendant le parcours du dictionnaire.

Question 3. (2 pts) Ecrire une fonction ***distanceChemin(cle)*** qui retourne la distance d'un chemin identifié par sa clé passée en paramètre.

Question 4. (3 pts) Ecrire une fonction ***distanceMinimale(DictChm)*** qui retourne la distance minimale de plusieurs chemins du dictionnaire ***DictChm*** passé en paramètre.

Question 5. (2 pts) Ecrire une fonction ***cheminMinimal(DictChm)*** qui retourne le chemin minimal des chemins du dictionnaire ***DictChm*** passé en paramètre.

Question 6. (1 pt) Ecrire une fonction ***cheminRetour(C)*** qui retourne le chemin de retour qui est le chemin inverse de C. Si C est un chemin de A vers B, la fonction retourne le chemin de B vers A.

* fin de l'épreuve *