Université de Carthage Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul Département Mathématique-Informatique	IPEIN المعهد التحضيري للدراسات الهندسية بنابل Institut Préparatoire aux Etudes d'Ingénieurs de Nabeul **Devoir Surveillé 1 d'informatique**	Année : 2022/2023 Filière :SM/SP/ST Niveau d'étude : 2ème année Semestre : 1 Nombre de pages : Date : *jeudi 3 novembre 2022* Durée : 1h:30

Exercice 1 : 9 points

Une file est une structure de données de type FIFO (First In First Out) : le premier entré est le premier sorti.
On suppose qu'on a les fonctions suivantes dans un fichier « op_file.py »:

- CreerFile() permettant de créer une file vide
- Enfiler(F,elt) permettant d'ajouter un élément « elt » en queue de la file
- Defiler(F) permettant de défiler et retourner la tête de la file
- EstVide(F) à résultat booléen, permettant de vérifier si la file est vide

Une pile est une structure de données de type LIFO (Last In First Out) : le dernier entré est le premier sorti.
On suppose qu'on a les fonctions suivantes dans un fichier « op_pile.py » :

- creer_pile() : création et renvoi d'une pile vide
- est_vide(P) : renvoie « True » si la pile P est vide, « False » sinon.
- sommet(P) : renvoie l'élément sommet de la pile P.
- depiler(P) : supprime le sommet de la pile P, et renvoie la valeur de ce sommet
- empiler(P,elt) : ajoute l'élément « elt » au sommet de la pile P.
- **taille(P)** : renvoie le nombre d'éléments appartenant à la pile P.
- **copier(P)** : renvoie une nouvelle pile contenant les mêmes éléments que la pile P.

Question 1 :

Créer une fonction **calculer_position(P, element)** ayant pour paramètres une pile **P** et la valeur d'un **élément** appartenant à P. Cette fonction renvoie la **position** de cet élément dans **P** telle que

la position **est compté en partant du sommet de P (**La position du sommet est **1**) :

Exemple: si **P** est la pile = | **6** | 3 | **9** | 5 |

calculer_position (P, **6**) == 1

calculer_position (P, **9**) == 3

Après appel de cette fonction, la pile P devra avoir retrouvé son état d'origine.

Question 2 :

2 .1. Créer une fonction **max_pile(P, n)** ayant pour paramètres une pile **P** et un entier **n**

Cette fonction renvoie l'élément **maximum** parmi les « **n** derniers éléments empilés en haut de la pile P ».

Exemple : si **P** est la pile de la **question 1** : **max_pile**(P, **4**) == 9
max_pile(P, **2**) == 6

Après l'appel, la pile P devra avoir retrouvé son état d'origine (Ne pas oublier d'exploiter la fonction **copier(P)**

La fonction doit aussi vérifier que **n** ne dépasse pas **taille(P)** (vous pouvez utiliser par exemple l'instruction **assert**)

2.2. En utilisant les deux fonctions précédentes**,** créer une fonction **position_max_pile(P, n)** qui appelle la fonction **max_pile(P, n),** puis retourne **la position** de l'élément **maximum** obtenu**.**

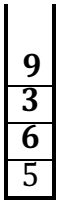
Question 3 :

Créer une fonction **inverser(P , m)** ayant pour paramètres une pile **P** et un entier **m**.

Cette fonction **inverse l'ordre** des « **m** derniers éléments empilés en haut de la pile P »

(Pour inverser l'ordre, on pourra utiliser une **file** temporaire **ou bien deux piles** temporaires).

Exemple : si **P** est la pile de la question. **1**,
après l'appel de **inverser(P, 3),** l'état de la pile **P sera :**



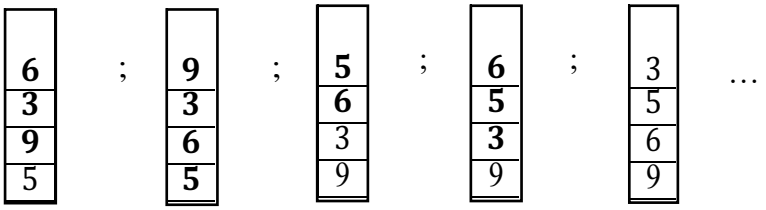
Question 4 : L'objectif de cette question est de trier une pile en utilisant les deux fonctions précédentes.

Le tri d'une pile consiste à réordonner ses éléments du plus **grand** (placé en **bas** de la pile) au plus **petit** (placé en **haut** de la pile).

On écrira d'abord une version itérative puis une autre version récursive du **principe de tri suivant** :

- On recherche la **position** du plus grand élément,
- On **inverse** la pile **à partir de cette position** de façon à mettre **la plus grande valeur** tout en haut de la pile,
- On **inverse l'ensemble de la pile** de façon à ce que **cette plus grande valeur** se retrouve tout en **bas**,
- La plus grande valeur étant à sa place, on **recommence** le même principe (récursivement) sur **le reste** de la pile :

Exemple :



Pour manipuler la pile, on utilisera **uniquement** les 3 fonctions : **position_max_pile (**question **2.2), inverser (**question **3),** ainsi que la fonction **taille (page 1)**

- Écrire une fonction **itérative trierPile(P)** ayant pour paramètre une pile P. Cette fonction trie la pile P selon la méthode ci-dessus. (le code de cette fonction est simple et ne dépasse pas 5 ou 6 lignes).

Question 5 :

Écrire une fonction récursive **trierPile_rec (P, n)** ayant pour paramètres une pile **P** et un entier **n.** Cette fonction trie les « **n** éléments en haut de la pile P » en appliquant (récursivement) le principe de tri de la question précédente.

Pour trier **toute la pile**, il suffit d'utiliser cette fonction de la façon suivante : **trierPile_rec (P, taille(P))**

Question 6 : Écrire une fonction <u>récursive</u> **est_triée(P)** qui retourne **True** si les éléments de la pile **P** sont triés dans l'ordre définit dans la question précédente, et retourne **False** sinon. L'idée est de dépiler le sommet et le comparer avec l'élément suivant de P, puis selon le résultat de cette comparaison, rappeler la même fonction (récursivement) sur le reste de la Pile.

Exercice 2 : 6 points

<u>Partie 1 (Réalisation d'une Pile sous la forme d'un dictionnaire)</u> :

La création d'une pile vide est réalisée par la fonction :

```
def creer_pile():
    P = dict()
    return P
```

Le but est de représenter une Pile par un dictionnaire **P où :**

- chaque **clé :** est un réel indiquant **l'instant d'empilement** d'un élément dans la Pile (cet instant du temps nous permettra de gérer le principe **« Dernier Entré Premier Sorti** » de la Pile)
- chaque **valeur : P[clé]** représente un **élément** actuellement **empilé** dans la Pile.

- chaque appel à **empiler(P, element)**: permet d'empiler l'élément passée en paramètre, avec l'instant d'empilement obtenu en invoquant la fonction **perf_counter( )** du module **time** (cette fonction ne prend aucun paramètre et retourne un réel) ;
- **Question 1 :** écrire la fonction **empiler(P, element)** décrite ci-dessus

Pour vous aider, on donne la fonction **depiler** qui supprime l'élément le plus récent de la Pile et le retourne

```
def depiler(P):
    assert (type(P) == dict)
    assert len(P) > 0 , "Erreur Pile Vide"
    cléMax = max( P.keys() )
    return P.pop(cléMax)
```

- **Question 2 :** écrire la fonction **`taille(P)`** : permet de retourner la taille de la Pile;
- **Question 3 :** écrire la fonction **`estvide(P)`** : retourne **`True`** si la Pile est vide, **`False`** sinon ;
- **Question 4 :** écrire la fonction **`sommet(P)`** : retourne le sommet de la Pile (l'élément le plus récent) et déclenche une exception si la pile est vide.

<u>Partie 2 (Appliquer une structure de pile pour déterminer les positions de parenthèses)</u> :

En utilisant une Pile, écrire une fonction **positionsParenthèses(chaine)** permettant de retourne les **positions** des parenthèses d'une **chaine** correctement parenthésée, c'est-à dire, pour chaque parenthèse fermante "**)**" **correspond** une parenthèse **précédemment** ouverte "**(**".

Par exemple, pour **chaine** = "2*(1+(3-1))", la fonction retournera la liste de tuples : **[(6,10) , (3,11)]**

L'idée est de parcourir la chaine par indice, tel que :

- dès que l'on rencontre une parenthèse ouvrante, on empile son **indice + 1**,
- dès que l'on rencontre une parenthèse fermante, on dépile pour récupérer la position du début de parenthésage…

Exercice 3 (révision notions de base): 5 points

Nous allons nous intéresser au problème suivant : étant donné un ensemble de points du plan, indentifier le couple de points les plus proches au sens de la distance euclidienne (utile dans les transports, aériens …)

Nous représenterons chaque **Point** par un `tuple` de flottants (x, y) représentant ses coordonnées.
L'ensemble des Points sera stocké dans une structure de type `set` .
Dans le module **math,** la fonction **dist(p, q)** retourne la distance euclidienne entre deux Points **p** et **q.**

Question 1 : Ecrire une fonction **plus_proches_voisins(ensPts)** qui prend pour argument l'ensemble (set) de Points et retourne un couple de Points situés à une distance minimale l'un de l'autre.
(il suffit de faire une recherche exhaustive, complexité $\mathbf{O(n^2)}$ avec **n** = len(**ensPts)**)

Question 2 :

cities.txt

File Edit View Language

```
1 Sousse:10.63699:35.82539
2 Douz:9.038601:33.456535
3 Sidi Bouzid:9.48494:35.03823
4 Sfax:10.766163:34.747847
5 Gabes:10.097522:33.888077
6 Tozeur:8.122933:33.918534
7
```

Écrire une fonction ***loadCities*** permettant de récupérer les coordonnées de villes à partir d'un fichier texte où chaque ligne est de la forme : « ***nom_ville*** **: abscisse : ordonnée** ». Cette fonction prend en paramètres le nom du fichier et renvoie un dictionnaire **D*cities*** contenant les données des villes récupérées à partir du fichier :
Le dictionnaire **D*cities*** a le format suivant :

- chaque **clé** est un **Point** représentant les coordonnées d'une ville;
- chaque **valeur** est une chaine (**str**) représentant le nom d'une ville.

Question 3 :
Écrire une fonction **plus_proches_villes(D*cities*)** qui prend pour argument le dictionnaire **D*cities*** construit précédemment, et affiche les **noms** des deux villes situées à une distance minimale l'une de l'autre.
(Ne pas oublier d'utiliser la fonction **plus_proches_voisins**)

ANNEXE - Quelques fonctions Python

f=open (nomF, 'r') permet d'ouvrir le fichier nomF en mode lecture.
f.close( ) permet de fermer un fichier.
f.readlines() : permet de lire et retourner le contenu de toutes les lignes d'un fichier dans une liste.

source.**split**(motif) retourne une liste formée par des chaînes de caractères résultantes du découpage de la chaîne source autour de la chaîne motif.