MP* - 2020/2021, Lycée Blaise Pascal

TP d'informatique Python 1

Les piles

1 Récupération d'un module de gestion des piles

1. Connectez-vous au réseau du lycée.

- L'identifiant (*login*) est : *Les 7 premières lettres du nom + initiale du prénom* (sans espace ; exemple pour Albert Einstein : `einsteia`) ;
- Le mot de passe : *année+mois+jour* de naissance (exemple : Albert Einstein étant né le 14 mars 1879, son mot de passe est 18790314).

Remarque. En cas de nom composé, « l'espace » est remplacé par un « tiret bas » (tiret « 8 » ou *underscore*). Exemple, pour Pierre de Fermat : `de_fermp`

2. Dans le dossier partagé de la classe trouvez les fichiers `pile.py` et `ODS.txt`.

3. Recopiez les fichiers `pile.py` et `ODS.txt` dans un répertoire de la partition `K:` (clic droit puis « `enregistrez sous...` »)

4. Avec Pyzo, créez un fichier `TD01.py` et sauvegardez-le **dans le même répertoire** que les deux fichiers précédents.

5. Dans le fichier `TD01.py`, commencez votre programme par la ligne suivante :

```
from pile import *
```

Vous disposez maintenant d'un nouveau type d'objet, la classe `Pile`.

- Pour créer une pile (initialement vide) nommée `p`, on écrit `p = Pile()`
- Pour lire la hauteur de la pile `p`, on écrit `p.hauteur`
- Pour empiler un élément `x` sur la pile `p`, on écrit `p.empile(x)`
- Pour dépiler l'élément du haut de la pile `p` et le stocker dans une variable `a`, on utilise la syntaxe `a = p.depile()`
- Pour regarder (sans dépiler) l'élément du haut de la pile `p` et le stocker on écrit `a = p.sommet()`
- Pour tester si la pile `p` est vide, on écrit `if p.est_vide():`

Testez ces commandes avec le programme ci-dessous. Avant de l'exécuter, prévoyez la réponse puis vérifiez que votre prévision était correcte.

```
from pile import *
q = Pile()
for k in range(10):
    q.empile(k)
print(q.sommet())
print(q.hauteur)
for k in range(q.hauteur):
    print(q.depile())
print(q.est_vide())
q.depile()
```

2 Exercices d'échauffement

Pour chaque exercice, après avoir écrit le programme, on précisera la complexité de l'algorithme proposé en fonction de la hauteur n de la pile.

1. Créez une fonction `affiche(p)` qui prend en argument une pile `p` et qui renvoie une liste contenant les éléments de `p` (de celui du haut en tête jusqu'à celui du bas en dernier). A la fin de la fonction `affiche(p)`, il faut que la pile `p` soit inchangée. Vous pourrez utiliser cette fonction dans les exercices suivants pour vérifier que vos programmes sont corrects.

2. Écrivez une fonction `echange(p)` qui prend en argument une pile `p` (supposée de hauteur supérieure ou égale à 2) et qui échange les deux éléments du haut de la pile.

3. Écrivez une fonction `deuxieme(p)` qui prend en argument une pile `p` (supposée de hauteur supérieure ou égale à 2) et qui renvoie le deuxième élément de la pile en partant du haut. Après l'exécution de cette fonction, la pile ne doit pas être modifiée.

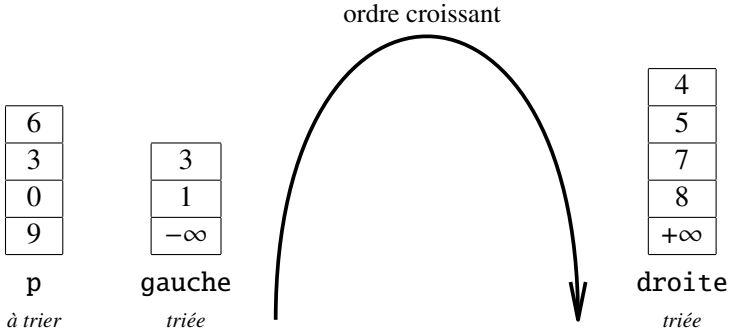
4. Écrivez une fonction `auFond(p)` qui prend en argument une pile `p` (supposée non-vide) et met l'élément du haut de la pile tout en bas (en laissant les autres dans l'ordre de départ).

5. Écrivez une fonction `inverse(p)` qui prend en argument une pile `p` et qui renverse l'ordre des éléments de cette pile (le haut en bas et le bas en haut). On souhaite que la pile `p` elle-même soit modifiée et non pas obtenir le résultat dans une autre pile.

3 Tri avec des piles

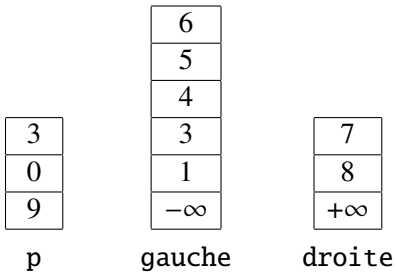
On dispose d'une pile `p` remplie de nombres (entiers ou flottants) qui ne sont pas rangés dans l'ordre croissant. On souhaite trier les éléments pour qu'à la fin, la pile `p` contienne les éléments dans l'ordre croissant (le plus grand en haut de la pile). L'idée est la suivante : on dispose de deux piles nommées `gauche` et `droite`. Initialement, on met dans la pile `gauche` le « nombre » $-\infty$ (nous verrons dans un instant comment faire cela) et dans la pile droite, nous mettons $+\infty$. Les différentes étapes se poursuivent de sorte qu'à chaque étape, la pile `gauche` est dans l'ordre croissant, la pile `droite` dans l'ordre décroissant et que tous les éléments de `droite` soient plus grands que ceux de `gauche`.

Une étape intermédiaire :



Pour traiter un nouvel élément de la pile `p` (ici le nombre 6), on dépile l'une des deux piles `gauche`/`droite` dans l'autre jusqu'à ce que le nouvel élément soit à sa place entre les sommets des deux piles puis on l'empile à gauche (ou à droite, c'est sans importance). Par exemple, pour traiter l'étape précédente, on dépile d'abord 4 et 5 de `droite` vers `gauche` puis on peut empiler 6. L'étape suivante sera :

L'étape suivante :



On poursuit de même jusqu'à avoir vidé la pile `p`. Alors les éléments sont rangés dans l'ordre en partant du bas de `gauche` jusqu'au bas de `droite`. Il reste à remettre (correctement) les éléments dans `p`.

Dans le module `numpy` se trouve la variable `inf` qui vaut $+\infty$ (et se comporte comme tel dans les opérations et les comparaisons).

```
from numpy import inf
print(4<inf)
print(4<-inf)
```

6. Programmez le tri décrit ci-dessus.

7. Précisez l'intérêt d'avoir placé $-\infty$ et $+\infty$ au fond des piles `gauche` et `droite`.

8. Quelle est la complexité de ce tri par rapport à la hauteur n de la pile `p` ?

9. Testez ce tri sur des exemples. Pour générer une pile de hauteur 100 remplie d'entiers tirés aléatoirement entre 0 et 1000, vous pourrez utiliser la fonction `rd.randint(a,b)` (après l'importation `import numpy.random as rd`) qui tire aléatoirement un entier entre `a` inclus et `b` exclu.

4 Le mot le plus long

Dans le jeu télévisé *Des chiffres et des lettres*, l'une des épreuves consiste à l'aide d'un tirage de neuf lettres, à proposer le mot le plus long possible qui s'écrit en utilisant les lettres proposées. Par exemple, pour le tirage `MHIQOUEJS`, le mot le plus long fait 8 lettres et c'est `OHMIQUES`. On se propose d'écrire un programme qui trouve tous les mots les plus longs possibles.

En important le module `pile`, vous avez également importé une variable nommée `dico` qui est une liste de tous les mots du dictionnaire `ODS` (officiel du scrabble). Ces mots sont écrits en majuscules sans accents.

10. Créez une fonction `trouveEntier(mot)` qui renvoie `True` si le `mot` proposé se trouve dans la liste `dico` et `False` sinon. Le dictionnaire étant rangé dans l'ordre, il ne faut surtout pas faire un parcours complet du dictionnaire dans son intégralité car il y a beaucoup plus rapide : la recherche dichotomique. Rappel : on se donne deux entiers `gauche` et `droite` valant initialement `0` et `len(dico)-1` ; à chaque étape on cherche si le mot se trouve dans le dictionnaire entre les positions `gauche` et `droite` (incluses). Pour cela, on calcule la position `milieu` $= \left\lfloor \frac{\texttt{gauche} + \texttt{droite}}{2} \right\rfloor$ et on compare (avec le comparateur usuel <) notre `mot` et la donnée du dictionnaire en position `milieu`. S'ils sont égaux, on a trouvé le `mot` dans le dictionnaire ; sinon on modifie `gauche` ou `droite` afin de conserver l'intervalle d'indices qui peut contenir notre `mot`. Lorsque l'intervalle `[gauche ; droite]` est vide, c'est la preuve que le `mot` n'est pas présent dans le dictionnaire.

11. Créez une fonction `trouveDebut(mot)` qui est une variante de la fonction précédente : elle renvoie `True` lorsque `mot` est le début d'un vrai mot du dictionnaire. On précise que `texte.startswith(debut)` renvoie `True` lorsque la chaîne de caractères `texte` commence par la chaîne de caractères `debut`.

Pour programmer le jeu *Le mot le plus long*, l'idée est la suivante. On crée une pile contenant des couples `(debut, restantes)` où `debut` est le début d'un vrai mot du dictionnaire écrit à l'aide de certaines lettres parmi celles proposées au départ et `restantes` est la liste des lettres de départ qui n'ont pas été utilisées pour écrire ce début de mot. Au départ, la pile contient le couple `("", toutes les lettres)`. Pour l'exemple qui a donné `OHMIQUES`, la pile contient initialement le couple `("", "MHIQOUEJS")`. A chaque étape, on dépile un couple, on crée toutes les successions `debut+lettre` où `lettre` est l'un des caractères de `restantes` et, si `debut+lettre` est encore un début de mot du dictionnaire, on le réempile (avec la liste des lettres restantes correspondante). Sur l'exemple proposé, l'une des étapes sera donc :

- dépilement de `("M", "HIQOUEJS")`
- création de `"MH"` → on ne fait rien car ce n'est pas le début d'un mot
- création de `"MI"` → c'est le début d'un mot donc on empile `("MI", "HQOUEJS")`
- création de `"MQ"`, `"MO"`, etc

Lorsque `debut+lettre` est un mot *complet* du dictionnaire, on l'écrit dans une liste `resultats` : c'est l'un des mots que l'on peut former avec les lettres disponibles. Lorsque la pile est vide, on a épuisé toutes les possibilités pour créer des mots à partir des lettres proposées et il suffit alors de rechercher, dans la liste `resultats`, les mots les plus longs pour les afficher.

12. Programmez un script qui demande à l'utilisateur un tirage de 9 lettres puis écrit tous les mots les plus longs à partir de ce tirage.

5 Le compte est bon

Si vous avez été suffisamment rapides, vous pouvez également programmer *Le compte est bon*. Cette fois on demande à l'utilisateur de donner 6 nombres. Normalement, ces nombres doivent appartenir à `{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100}`. On demande également à l'utilisateur un nombre à 3 chiffres que l'on cherche à atteindre : c'est notre `but`. On cherche alors à créer le nombre `but` en utilisant (au plus une fois) les nombres proposés et en les combinant avec les 4 opérations (on ne doit pas utiliser de valeurs négatives ou nulles ni de valeurs non entières). Le résultat consiste à afficher toutes les successions d'opérations menant, à partir des 6 nombres proposés, au `but` demandé. L'idée est similaire à celle du *mot le plus long*. On crée une pile sur laquelle on empile des couples `(listeNombres, texte)` où `listeNombres` est une liste des nombres créés à partir des nombres de départ et de certaines opérations tandis que le `texte` décrit ces opérations. Par exemple, partant des nombres `1, 2, 3, 4, 5, 25`, la pile contient initialement le couple `([1, 2, 3, 4, 5, 25], "")` car avec aucune opération, on dispose des 6 nombres de départ. A une certaine étape, on trouvera `([3, 3, 4, 5, 25], "1+2=3\n")` puis, par exemple `([3, 5, 12, 25], "1+2=3\n3*4=12\n")`. NB : `"\n"` est un saut de ligne.

A chaque étape, on dépile un couple sur la pile, on regarde tous les nombres qu'on peut créer avec deux des nombres de la `listeNombres` et on réempile de nouveaux couples `(listeNombres, texte)`. Si on a créé le nombre recherché, il faut alors afficher le `texte` car c'est la liste des calculs qui mènent au résultat demandé.