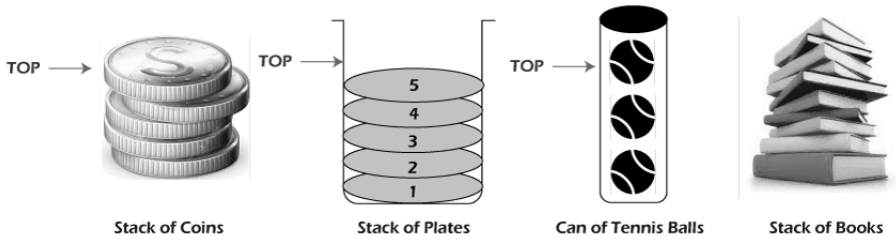
Les piles

Le concept de pile est basé sur celui d'une pile d'assiettes ou de crêpes : on ajoute des assiettes ou des crêpes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée.



Définition

En informatique, une pile (*stack*) est une structure de données (un type de variable) pour lesquelles on dispose des méthodes suivantes :

- `empile(elem)` : ajoute l'élément `elem` au-dessus de la pile (*push*);
- `depile()` : supprime et renvoie l'élément du dessus de la pile (*pop*).

Selon les langages, on dispose en général des méthodes supplémentaires suivantes :

- `est_vide()` : renvoie `True` si la pile est vide, `False` sinon (*is_empty*);
- `taille()` : renvoie le nombre d'éléments dans la pile (*size*).

Le langage Python ne propose pas de type « pile » au sens strict du terme. En revanche, le type `list` de Python est une bonne base pour créer une structure de données de pile. C'est ce qui est proposé dans le fichier pile.py, à télécharger pour réaliser les exercices suivants. pour répondre aux exercices, et il est interdit d'utiliser le type `list` de Python.

Exercice 1 – *(+)*

On considère le script ci-dessous.

```
p = Pile()
p.empile("n")
p.empile("s")
p.empile("i")
x = p.depile()
p.depile()
p.empile(x)
```

1. Donner l'état de la pile `p` à l'issue de ce script.

2. Parmi les modes FIFO (First In First Out) et LIFO (Last In First Out), lequel s'applique aux piles ?

Exercice 2 – *(+)*

Écrire une fonction `echange(p)` qui permute les deux éléments situés en haut de la pile `p` (on part du principe qu'elle contient au moins deux éléments). Faire les affichages permettant de montrer que le programme effectue ce qui est attendu. La modification de la pile a lieu « en place », c'est-à-dire qu'on n'a pas à renvoyer la pile car elle sera modifiée à l'intérieur et à l'extérieur de la fonction demandée.

Exercice 3 – *(++)*

Écrire une fonction `inverse(p)` qui renvoie une nouvelle pile constituée des mêmes éléments que `p`, mais dans l'ordre inverse.

Attention : la pile passée en paramètre doit être inchangée à la sortie de la fonction !

Exercice 4 – *(++)*

Écrire une fonction `copie(p)` qui renvoie une nouvelle pile dont les éléments sont ceux de `p`, dans le même ordre.
Indice : utiliser une pile auxiliaire.

Exercice 5 – *(++)*

1. Écrire une fonction `fond(p)` qui supprime et renvoie l'élément situé en bas de la pile `p`.
2. Écrire une fonction `rotation(p)` qui échange l'élément situé en haut de la pile avec celui situé en bas.

Exercice 6 – *(+++)*

Une expression est correctement parenthésée si à chaque signe de parenthésage ouvert correspond un signe de parenthésage fermé de même forme, et s'il n'y pas de « croisement » entre les correspondances.

Écrire une fonction `test_parenthesage` qui prend en argument une chaîne de caractères et renvoie `True` si celle-ci est correctement parenthésée, et `False` sinon.

On donne ci-dessous un jeu de tests pour détecter d'éventuels défauts de conception.

```
assert test_parenthesage("(a)(b)(((c)(d)))")==True
assert test_parenthesage("([b]){((c))[d]}")==True
assert test_parenthesage("(")==False
assert test_parenthesage("(a))")==False
assert test_parenthesage("((a))}")==False
assert test_parenthesage("[[a]")==False
assert test_parenthesage("[(a])")==False
assert test_parenthesage("{((a)})")==False
```

Exercice 7 – *(+++)*

On considère une pile `p` d'assiettes de couleur blanche ou verte, chacune étant identifiée par un numéro. On modélise une assiette par un tuple `(couleur, numero)`.

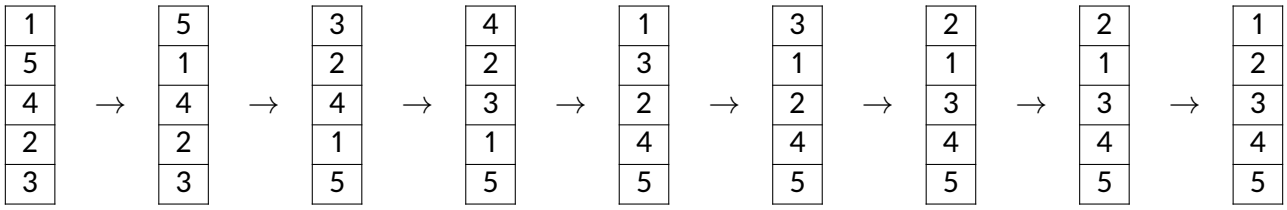
Concevoir une fonction `ranger` qui range une pile d'assiettes en plaçant les assiettes vertes au dessus des blanches, mais sans changer l'ordre relatif des numéros.

Exercice 8 – *(+++)*

On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile). On dispose uniquement d'une spatule que l'on peut insérer dans la pile de crêpes de façon à retourner l'ensemble des crêpes qui lui sont au-dessus. Le principe est le suivant :

- on recherche la plus grande crêpe ;
- on retourne la pile à partir de cette crêpe de façon à mettre cette plus grande crêpe tout en haut de la pile ;
- on retourne l'ensemble de la pile de façon à ce que cette plus grande crêpe se retrouve tout en bas ;
- la plus grande crêpe étant à sa place, on recommence le principe avec le reste de la pile.

On donne une illustration de ce principe sur un exemple :



1. Combien de retournements partiels de la pile sont nécessaires pour trier une pile de taille n ?
2. Programmer une fonction `tri(p)` qui trie en place une pile `p` en utilisant ce principe.

   Indice : décomposer le problème en sous-fonctions.