Une des problématique en informatique est le stockage des données : comment les stocker pour pouvoir y accéder/mes traiter de manière optimale ? Cela dépend du traitement que je dois leur appliquer...

Vous avez pour l'instant beaucoup manipulé les tableaux. C'est un type de structure de données possible. Dans un tableau, on peut accéder de la même manière à n'importe quelle case du tableau en en spécifiant l'indice. Cela sous entend que les données sont ordonnées. Ce n'est pas adapté à toutes les situations.

Il existe d'autres types de structures de données (dictionnaire, pile, file ...) . Chacune est caractérisée par les opérations qu'elle permet (ou non) de faire et leur coût. Au programme d'IPT, on trouve la structure de pile.

I Structure de Pile

«Étudiant face à son travail» (allégorie)

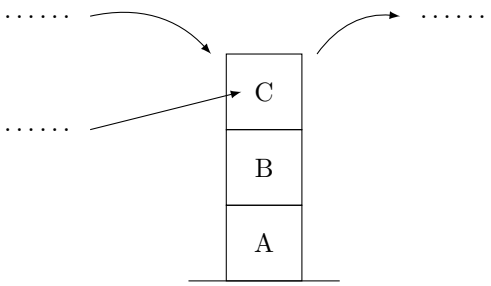
La structure de pile correspond exactement à l'image associée d'une pile de feuilles, d'assiettes... On peut :

- ajouter des choses au sommet de la pile (par exemple ajouter une assiette sale sur le haut de la pile)
- enlever des choses sur le sommet (par exemple enlever une assiette sale sur le sommet)

mais on ne peut pas regarder/retirer un élément au milieu de la pile (si on prend une assiette au milieu de la pile, cette dernière s'effondrera...)

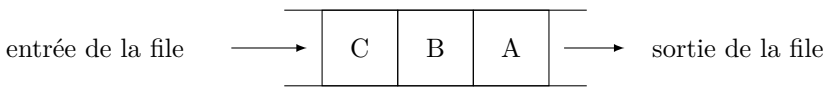
Bref, **on ne peut accéder qu'au dernier élément ajouté, appelé le sommet de la pile.**

oral : ? Dans la pile vide, on a d'abord stocké A, puis B (on dit qu'on a empilé B), puis C.
Dans la configuration actuelle, on ne peut qu'accéder qu'à C, le sommet de la pile. On peut soit empiler une nouvelle valeur sur C, soit retirer C de la pile (on dit dépiler C).
Pour accéder à l'élément A, on devra d'abord dépiler C puis dépiler B.



On parle de « **dernier arrivé, premier sorti** » (Last In, First Out)

Remarque : Il existe une autre structure de données fondamentale, les files (par analogie avec les files d'attente), où les éléments sortent dans leur ordre d'arrivée. On parle de « **premier arrivé, premier sorti** » (First In, First Out) mais elles ne sont pas au programme.



II Opérations caractérisant la structure de pile

- `creer_pile( )`, éventuellement `creer_pile(n)` renvoie une nouvelle pile initialement vide, éventuellement de capacité maximale `n`
- `depiler(p)` dépile et renvoie le sommet de la pile `p` (se dit `pop` en anglais)
- `empiler(p,e)` empile l'élément/valeur `e` sur le sommet de la pile `p` (se dit `push` en anglais)

Exo 1 : dérouler

```
p=creer_pile( )
empiler(p,10)
empiler(p,5)
empiler(p,7)
depiler(p)
```

Afin d'accroître la facilité d'utilisation de la pile, on peut y ajouter d'autres fonctions qui ne modifient pas la pile

- `taille(p)` renvoie le nombre d'élément de la pile
- `est_vide(p)` renvoie `True` si la pile est vide, `False` sinon
- `sommet(p)` permet d'accéder à la valeur du sommet de la pile sans la dépiler

III Quand rencontre-t-on des piles ?

A chaque fois que l'on souhaite que la dernière information arrivée soit la première traitée.

Pour les avertis : les piles sont des structures fondamentales car dans les langages compilés, elles servent à gérer les paramètres d'appels des procédures et des fonctions, leurs variables locales, leurs points de retour.

Pour tous, pour stocker l'historique des actions effectuées dans un logiciel, on utilise une pile. Lorsque vous annulez une opération (avec Ctrl Z), c'est bien en partant de la dernière utilisée. Et cette pile a une capacité maximale ! Votre historique de navigation vous affiche les sites visités en partant du plus récemment visité !

Par contre si on connaît a priori le nombre d'éléments à stocker ou que l'accès à un élément quelconque est nécessaire, on préfèrera un tableau !

Exo 2 : Quelle structure de données choisir pour ces tâches ?

- répertoire téléphonique
- journal d'appel
- liste des devoirs à faire pour un élève de prépa

Exo 3 : On dispose d'une structure de pile à laquelle sont associées les seules opérations `creer_pile()`, `empiler(p,e)`, `sommet(p)`, `depiler(p)` et `est_vide()`.
Le but est de construire les opérations ou fonctions suivantes à l'aide de ces éléments

1. Une fonction `depilerK(p,k)` qui dépile k éléments de la pile p (si elle contient moins de k éléments, elle sera vidée sans lever d'erreur)
2. une fonction `depilerjusque(p,e)` qui dépile la pile p jusqu'à trouver l'élément e ou que la pile soit vide
3. `inverse(p)` inverse toute la pile
4. `permutesommet(p)` qui inverse le sommet et le pénultième élément(l'élément sous le sommet)

IV Simuler une pile en Python avec un tableau

A notre niveau, en Python, le plus simple pour implémenter une structure de pile est d'utiliser les tableaux/listes. Il s'agit de coder les opérations caractéristiques des piles en utilisant les méthodes liées aux listes. Puis dans la suite du TP, on s'interdira les opérations sur les listes quand on manipulera les piles. (On parle d'encapsulation car on pourrait changer la manière dont les piles sont implémentées sans changer le code utilisant les piles).

```
1  def creer_pile():
2      return ...
3
4  def sommet(p):
5      return ...
6
7  def est_vide(p):
8
9
10
11
```

```
12 def empiler(p,e):
13
14
15
16 def depiler(p):
17
18
19
20 .
```

Remarque : Les méthodes `push` et `pop` existent déjà pour les listes en Python...

Dans certains langages, l'instanciation d'un tableau nécessite de connaître au préalable sa taille. Bien que la structure de pile théorique soit de capacité infinie, on est alors amené à gérer des piles bornées (ayant une capacité maximale).

TP

Vous commencerez par implémenter les fonctions caractéristiques des piles ci-avant.

Exo 4 : Écrire une fonction `taille(p)` renvoyant le nombre d'éléments de la pile (sans utiliser `len`, mais uniquement les fonctions caractéristiques des piles !)

Exo 5 : Les langages informatiques possèdent une grammaire stricte(en Python il y a : après le else, toute parenthèse ouverte est fermée...), qu'il faut respecter afin que l'ordinateur comprenne le code écrit. La première étape est donc toujours une vérification que le code fourni est syntaxiquement correct : c'est le « parsage ». Il en va de même en mathématiques avec le parenthésage.
$(2 \times 5 + 3)$ est une expression correcte.
$(2 \times (5 + 3)$ ou $)2 \times (5 + 3$ sont des expressions incorrectes.

1) Écrire une fonction `test_parentheses(expression)` qui prend en argument une expression ne contenant que des parenthèses comme `'(2*(5+3)'` et vérifie si son parenthésage est correct.
   indication : On n'a pas spécialement besoin d'une pile ici
2) Écrire une fonction `test_parenthesescrochets(expression)` qui prend en argument une expression comme `'[2*(5+3)-1]*3'` et vérifie si son parenthésage est correct.
   indication : On a ici besoin d'une pile, cette pile pourra contenir les symboles '(' et '['
3) Écrire, en utilisant une pile, une fonction `parentheses(expression)` prenant en argument une expression contenant des parenthèses (pas de crochets) et renvoyant la position des parenthèses ouvrantes et fermantes correspondantes.
   par exemple,
   `parentheses('2*(2+5*(1+5)-1)')` renverra `[(2,14),(7,11)]`.

Exo 6 : A partir des fonctions de base, implémenter une fonction `melange` qui, étant donné deux piles, mélange leurs éléments dans une nouvelle pile de sorte que : tant qu'aucune pile n'est vide, on retire l'élément sommet d'une des piles choisie aléatoirement et on l'empile sur la nouvelle pile.

Exo 7 : La notation polonaise inverse (du polonais Jan Lukasiewicz 1878-1956) consiste à écrire chaque opérande puis l'opérateur. On parle aussi de notation « post-fixe ». Ainsi :
$5 - 2$ s'écrit $[5,2,'-']$
$1 - 2 \times 3$ s'écrit $[1,2,3,\times,'-']$
$(1 + 2) \times 3$ s'écrit sans besoin de recourir à des parenthèses $[1,2,'+',3,\times]$
Si on considère l'entrée de l'expression mathématique comme un tableau, on peut alors évaluer cette expression avec une pile. On lit le tableau en empilant les nombres lus puis, dés qu'on atteint un opérateur, on dépile les deux derniers éléments, on effectue l'opération et on empile le résultat. On ne se focalisera (au moins dans un premier temps) que sur les opérateurs $+$ et $\times$.

1) Evaluer les expressions post-fixes suivantes :
   $A = 5\ 3\ +$ $B = 5\ 3\ 4\ +\ \times$ $C = 5\ 3\ +\ 4\ \times$
2) Transformer les expressions suivantes en expressions post-fixe :
   $D = 14 + 6 \times 7$ $E = (5 \times 2 - 3) \times 3 + 4$
3) Proposer une fonction `eval_polo(p)` qui à partir d'un tableau correspondant aux entrées d'une expression en notation polonaise inverse, évalue l'expression.