1 Introduction

Si on cherche en Python une structure une structure adaptée pour stocker une liste d'objets, le **tableau** (on parle ici de la classe *list* de Python) est particulièrement adapté. Cette structure permet avec ses méthodes **append** et **pop** d'ajouter ou supprimer un élément. Néanmoins, les tableaux en Python ne correspondent pas exactement à la définition « informatique » d'une liste. De plus, si nous passons à un autre langage, il n'y a pas forcément d'implémentation de ce genre de structure.

L'objectif de ce thème est donc de :

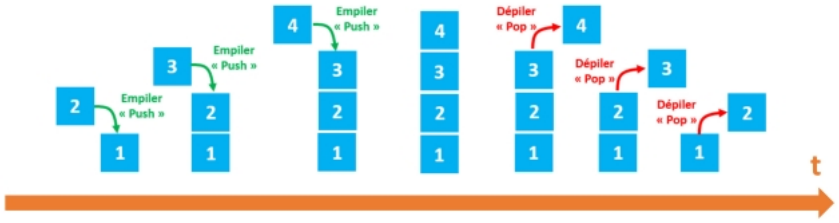
- définir d'un point de vue théorique la notion de **pile**, **file** et **liste** ;
- proposer, avec un langage objet, une classe qui répond à chaque définition ;
- développer, avec la récursivité ou des boucles, les méthodes attendues pour ces objets, notamment avec le notion de liste chainées.

2 Les piles (« stacks » en anglais)

Définition d'une pile

On appelle PILE une **structure de données abstraite fondée sur le principe « dernier arrivé, premier sorti »**. Elles sont dites de type **LIFO** (Last In, First Out). C'est le principe même de la pile d'assiette : c'est la dernière assiette posée sur la pile d'assiettes sales qui sera la première lavée.

Voici un schéma :



L'insertion d'un élément dans la pile est appelée « Empiler » et la suppression d'un élément de la pile est appelée « Dépiler ». Dans la pile, nous gardons toujours trace du dernier élément présent dans la liste avec un pointeur appelé top. D'une manière générale, voici les **fonctions primitives** définies pour la structure de données **Pile**.

Structure de données abstraite : Pile

une Pile utilise : des éléments à empiler, des Booléens
les opérations sur une piles sont :

- creer_pile_vide : $\emptyset \rightarrow Pile$
  Cette fonction renvoie une pile vide. Par exemple `maPile = creer_pile_vide()` crée la pile vide maPile.
- est_vide : $Pile \rightarrow booléen$
  `est_vide(maPile)` renvoie True si maPile est vide, et False sinon.
- empiler : $Pile, élément \rightarrow Rien$
  `empiler(maPile, élément)` ajoute élément au sommet de la pile.
- depiler : $Pile \rightarrow élément$
  `depiler(maPile)` renvoie élément qui est au sommet de la pile **en le retirant de la pile**
- lire_sommet : $Pile \rightarrow élément$
  `lire_sommet(maPile)` renvoie élément qui est au sommet de la pile **sans le retirer de la pile**

De nombreuses applications s'appuient sur l'utilisation d'une pile. En voici quelques-unes :

- dans un navigateur web, une pile sert à mémoriser les pages web visitées ; l'adresse de chaque nouvelle page visitée est empilée et l'utilisateur dépile l'adresse de la page précédente en cliquant sur le bouton « Afficher la page précédente »
- l'évaluation des expressions mathématiques en notation post-fixée (ou polonaise inverse) utilise une pile ;
- la fonction « Annuler la frappe » (Undo en anglais) d'un traitement de texte mémorise les modification apportées au texte dans une pile ;
- la récursivité (une fonction qui fait appel à elle-même) utilise également une pile ;
- les empilements de conteneurs sur un bateau ou un jeu de carte dans lequel on pioche peuvent être modélisés par des piles...

Remarque : en informatique, un dépassement de pile ou débordement de pile (en anglais, « stack overflow ») est un bug causé par un processus qui, lors de l'écriture dans une pile, écrit à l'extérieur de l'espace alloué à la pile, écrasant ainsi des informations nécessaires au processus. Stack Overflow est aussi connu comme étant un site web proposant des questions et réponses sous la forme d'un forum d'entre-aide sur un large choix de thèmes concernant la programmation informatique. Il y a de forte chance que vous trouviez la réponse à un problème informatique même ardu sur Stack Overflow! Enfin on ne peut pas dans une pile, comme dans une file, prendre une assiette à n'importe quel rang dans la pile (on risquerait de tout faire tomber et de casser toutes les assiettes), ni ajouter une assiette n'importe où dans la pile.

2.1 Quelques exercices théoriques

Voici résumé sous la forme d'un tableau les opérations que l'on peut réaliser sur un objet de type Pile :

Action sur la pile :	**Méthode de la classe Pile** :
Créer une pile vide appelée maPile	maPile = creer_pile_vide()
La pile est-elle vide ?	est_vide(maPile)
Empiler un nouvel élément sur la pile	empiler(maPile,élément)
Dépiler un élément de la pile	depiler(maPile)
Lire la valeur au sommet de la pile	lire_sommet(maPile)

Dans les exercices suivants, vous n'utiliserez que les méthodes liées au piles et les bases de programmation impérative : affectation, lecture, écriture, tests, boucles...

Exercice 1.

Indiquer quelles seront les instructions pour créer un pile appelée maPile, avec dans l'ordre les éléments 12, 14, 8, 7, 19, 22, le sommet de la pile étant 22.

Exercice 2.

On reprend maPile. Quelle instruction affichera le sommet de la pile ?

Exercice 3.

On reprend maPile. Quelle instruction supprimera l'élément correspondant au sommet de la pile ?

Exercice 4.

On reprend maPile. On souhaite insérer l'élément 20 entre les éléments 8 et 7 en conservant tous les autres éléments de la pile : comment doit-on procéder ?

2.2 Implémenter une pile avec un tableau dynamique Python

Il est possible d'utiliser les tableaux dynamiques Python, class *list*, pour implémenter simplement à la volée des piles.

Exercice 5.

1. Quelle instruction en Python permet de créer une pile vide appelée maPile ?
2. Quelle instruction Python permet d'ajouter un élément à maPile ?
3. Quelle instruction Python permet de lire le sommet de maPile ?
4. Quelle instruction permet de dépiler maPile ?
5. Quelle instruction permet d'empiler un nouvel élément à maPile ?
6. Programmer les cinq fonctions de bases sur les piles avec des fonctions en Python. Enregistrer pour pouvoir utiliser ces fonctions par la suite...

```
# les cinq méthodes de base sur les piles en Python, à compléter

def creer_pile_vide():
        return

def est_vide(pile):
        return

def empiler(pile, element):

def depiler(pile):
        return

def lire_sommet(pile):
        return

maPile = creer_pile_vide()
print(est_vide(maPile))
empiler(maPile, 5)
empiler(maPile, 8)
print(est_vide(maPile))
x = depiler(maPile)
print(x)
print(lire_sommet(maPile))
```

Exercice 6.

On reprend les fonctions définies ci-dessus. Écrire une fonction inverse_pile(pile), qui prend comme argument une pile, et qui renvoie une pile contenant les mêmes éléments mais dans l'ordre inverse, en utilisant exclusivement les fonctions déjà définies sur les piles.

```
def inverse_pile(pile):
        ...
        return pileInverse

maPileInverse = inverse_pile(maPile) # où maPile est une pile donnée
```

Exercice 7.

On reprend les fonctions définies ci-dessus. Écrire une fonction trie_pile(pile), qui prend comme argument une pile, et qui renvoie une pile contenant les mêmes éléments mais dans l'ordre croissant, en utilisant exclusivement les fonctions de base sur les piles ?

(Remarque : il s'agit en fait du tri par insertion avec une pile...)

```
def trie_pile(pile):
        ...
        return pileTriee

maPileTriee = trie_pile(maPile) # où maPile est une pile donnée
```

2.3 Utiliser la programmation objet pour implémenter des piles

Dans la partie précédente, nous avons vu comment implanter une pile avec le paradigme fonctionnel. Il est possible d'implanter des piles en utilisant le **paradigme objet**. En Python on définit une classe (**class**) avec son constructeur, ses valeurs et ses méthodes. Le code ci-dessous donne le début de ce travail.

```
class Pile :
    def __init__(self):
        self.valeurs = []

    def __str__(self):
        ch = ""
        for x in self.valeurs :
            ch = "|\t" + str(x) + "\t|" + "\n" + ch
        ch = "\nEtat de la pile:\n" + ch
        return ch

maPile = Pile()
```

1. Recopier le code ci-dessus dans un fichier Python dont le nom reprendra le titre du TP.
2. En programmation orientée objet (POO), on instancie un objet maPile vide appartenant à la classe Pile de la manière suivante : `maPile = Pile()`

   Une fois l'objet instancié on peut lui appliquer les méthodes de la classe à laquelle il appartient de la manière suivante :

   `monObjet.methodeClasse()`

   En Python, quelle est de nom de la méthode « constructeur » qui permet de créer un objet d'une classe choisie ?
3. Pour notre **class** Pile, quelle est le type Python qui est utilisé pour stocker les valeurs de la pile ? Que vaut-il quand le constructeur est appelé ?
4. A quoi sert la méthode spéciale **__str__** ? Qu'est-elle sensée renvoyer ?
5. Connaissez d'autres méthodes spéciales dans la POO en Python ?
6. A la suite du ficher ou à la console, après avoir exécuté le script, saisir `print(maPile)`. Vous devez constater qu'à ce stade, l'état de la pile est vide. Analyser le code de la fonction appelée avec **print**.
7. Implanter pour la classe `Pile` la méthode `empiler` qui prend comme argument une valeur et qui ne renvoie rien. Vérifier que les instructions suivantes fonctionnent.

   ```
   maPile.empiler(8)
   maPile.empiler(5)
   maPile.empiler(10)
   print(maPile)
   ```

8. Implanter pour la classe `Pile` la méthode `est_vide()` qui ne prend pas d'argument et renvoie `True` si la pile est vide, et `False` sinon. Vérifier que les instructions suivantes fonctionnent.

   ```
   print(maPile.est_vide())
   print(maPile)
   ```

9. Implanter pour la classe `Pile` la méthode `depiler()` qui ne prend pas d'argument, renvoie l'élément au sommet de la pile et le supprime de la pile. Vérifier que les instructions suivantes fonctionnent.

```
print(maPile.depiler())
print(maPile)
```

10. Implanter pour la classe `Pile` la méthode `lire_sommet()` qui ne prend pas d'argument et renvoie l'élément au sommet de la pile sans changer la pile. Vérifier que les instructions suivantes fonctionnent.

```
print(maPile.lire_sommet())
print(maPile)
```

11. Implanter pour la classe `Pile` la méthode `hauteur_pile()` qui ne prend pas d'argument et renvoie le nombre d'éléments dans la pile. Vérifier que les instructions suivantes fonctionnent.

```
print(maPile.hauteur_pile())
print(maPile)
```

12. Implanter pour la classe `Pile` la méthode `vider()` qui ne prend pas d'argument et vider le contenu de la pile. Vérifier que les instructions suivantes fonctionnent.

```
print(maPile.vider())
print(maPile)
```

13. Imaginer et implanter d'autres méthodes qui pourraient être intéressantes pour les piles... et tester ces méthodes.

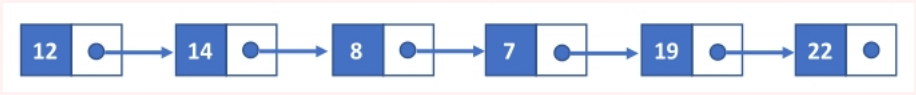
2.4 Utiliser la notion de liste chainée pour implanter une pile

Pour implanter une pile on peut :

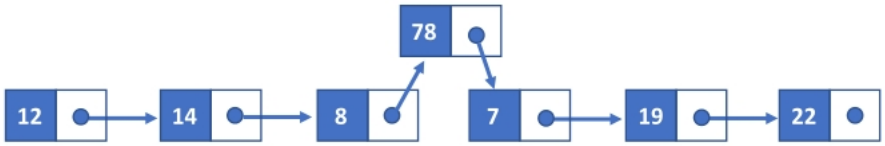
- utiliser la notion de tableau, que l'on retrouve dans tous les langages. En Python, le type `list` , encore appelé « liste Python « ou « Tableau Python » est adapté ;
- utiliser la notion de liste chainée, avec la programmation objet. Cette méthode est très efficace, rapide, facilement portable et évolutive.

Définition d'une liste chaînée

Dans une liste chaînée, à chaque élément de la liste on associe 2 cases mémoires : la première case contient l'élément et la deuxième contient l'adresse mémoire de l'élément suivant.

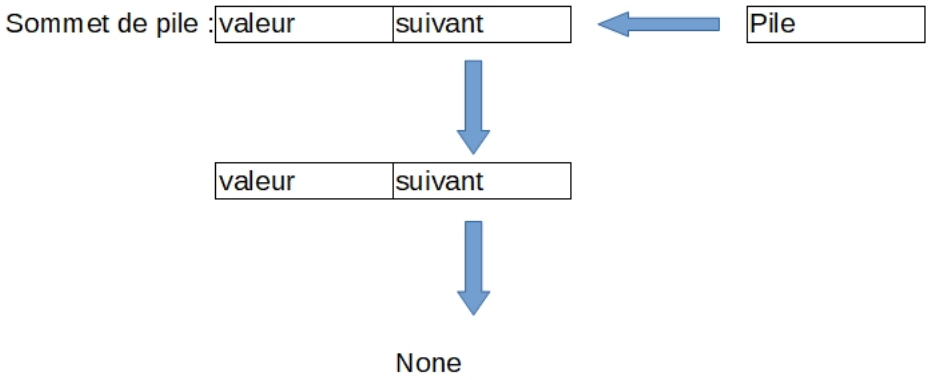


Avec ce type de structure, il est aisé d'insérer un nouvel élément :



Le principe de l'implantation d'une pile avec la notion de liste chainée :

- on crée un objet **Cellule**, qui contient deux attributs : **valeur** et **suivant**. Chaque élément de la pile est donc une **Cellule**. Comme son nom l'indique, **valeur** contient la valeur, et **suivant** est un pointeur qui désigne la **Cellule** suivante ;
- on crée un objet **Liste**, qui est simplement un pointeur vers la **Cellule** au sommet de la pile si la pile n'est pas vide, ou bien **None** sinon.



Le code ci-dessous explique la démarche : on crée une `class` appelée `Cellule` avec ses deux attributs et `valeur` et `suivant`, puis on définit une `class` de type `Pile` avec un seul attribut `sommet` qui est en fait un pointeur vers la cellule qui est en haut de pile. Recopier et coller le code initial dans un éditeur Python. Comme vous pouvez le remarquer, il est souvent utile de connaitre le nombre d'éléments présents dans une pile, c'est pour cela qu'on a ajouté l'attribut `taille` qui contiendra cette valeur.

```
class Cellule:

    def __init__(self, valeur, suivant = None):
        self.valeur = valeur
        self.suivant = suivant


class Pile :

    def __init__(self):
        self.taille = 0
        self.sommet = None


    def __str__(self):
        ch = "\nEtat de la pile :\n"
        pointeur = self.sommet
        while pointeur != None :
            ch += "|\t" + str(pointeur.valeur) + "\t|" + "\n"
            pointeur = pointeur.suivant
        return ch

# on crée une pile vide maPile
maPile = Pile()
```

1. Analyser le constructeur de la classe `Cellule`. Quels sont les arguments? Que signifie `suivant = None`?
2. Analyser le constructeur de la classe `Pile`. Quels sont les paramètres pris? Comment sont initialisés les attributs?
3. Exécuter le code et vérifier le résultat obtenu à la console avec `> > > print(maPile)` . Quelle fonction de l'objet est appelée quand on utilise `print` de cette manière?
4. Implanter pour la classe `Pile` la fonction `empiler`. Vérifier ensuite que cette méthode fonctionne bien en empilant plusieurs valeurs, et en appelant `print`.

```
def empiler(self, valeur) :
        self.sommet = Cellule(...)
        self.taille ...
```

5. Implanter la méthode `depiler` et vérifier son fonctionnement.

```
def depiler(self):
        if self.taille > 0 :
            valeur = ...
            self.sommet =
            self.taille ...
            return ...
```

6. Enfin implanter les méthodes `lire_sommet(self)` et `est_vide(self)` , puis tester leur fonctionnement.

```
def est_vide(self):
        return self.taille == 0

def lire_sommet(self):
        return self.sommet.valeur
```

7. Reprendre et analyser la méthode spéciale `str`.

Quelques explications sur la structure récursive de la méthode.

Quand on saisit **maPile = Pile()** , l'attribut **sommet** est alors par défaut None.

Quand on empile le premier objet, 8 par exemple, alors
maPile = Cellule(8, None)

Quand on empile un second objet, par exemple 5, alors
maPile = Cellule(5, Cellule(8, None))
et ainsi de suite...