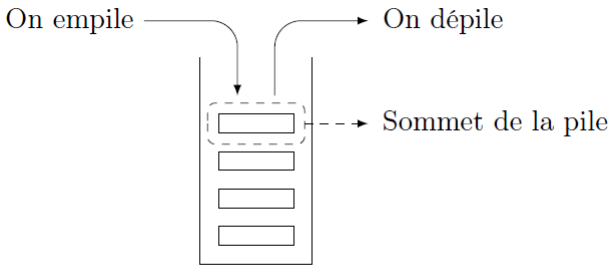
Math93.com

Notion de Piles et Files - NSI

Première partie

Notions de piles

Une pile est une structure de données abstraite fondée sur le principe « dernier arrivé, premier sorti »



Une pile

Exercice 1. Les idées de base

On munit la structure de données Pile de quatre fonctions primitives définies dans le tableau ci-dessous. :

- creer_pile_vide : ∅ ⟶ Pile
  creer_pile_vide() : renvoie une pile vide
- est_vide : Pile ⟶ Booléen
  est_vide(pile) : renvoie True si pile est vide, False sinon
- empiler : Pile, Élément ⟶ Rien
  empiler(pile, element) : ajoute element au sommet de la pile
- depiler : Pile ⟶ Élément
  depiler(pile) : renvoie l'élément au sommet de la pile en le retirant de la pile

Question 1

On suppose dans cette question que le contenu de la pile P est le suivant (les éléments étant empilés par le haut).

4
2
5
8

Quel sera le contenu de la pile Q après exécution de la suite d'instructions suivante ?

```
Q = creer_pile_vide ()
while  not  est_vide(P):
    empiler(Q, depiler(P))
```

Question 2

1. On appelle hauteur d'une pile le nombre d'éléments qu'elle contient. La fonction **hauteur_pile** prend en paramètre une pile **P** et renvoie sa hauteur. Après appel de cette fonction, la pile **P** doit avoir retrouvé son état d'origine.

   Exemple : si **P** est la pile de la question 1 : **hauteur_pile(P) = 4**.

   Recopier et compléter sur votre copie le programme Python suivant implémentant la fonction **hauteur_pile** en remplaçant les `???` par les bonnes instructions.

```
def hauteur_pile(P):
    Q = creer_pile_vide ()
    n = 0
    while  not(est_vide(P)):
        ???
        x = depiler(P)
        empiler(Q,x)
    while not(est_vide(Q)):
        ???
        empiler(P, x)
    return ???
```

2. Créer une fonction **max_pile** ayant pour paramètres une pile **P** et un entier **i**. Cette fonction renvoie la position **j** de l'élément maximum parmi les **i** derniers éléments empilés de la pile **P**. Après appel de cette fonction, la pile **P** devra avoir retrouvé son état d'origine. La position du sommet de la pile est **1**.

Question 3

Créer une fonction **retourner** ayant pour paramètres une pile **P** et un entier **j**. Cette fonction inverse l'ordre des **j** derniers éléments empilés et ne renvoie rien. On pourra utiliser deux piles auxiliaires.

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

5
2
4
8

Question 4

L'objectif de cette question est de trier une pile de crêpes.
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.*

Exemple :

6		9		5		6		3	
3	$\Longrightarrow$	3	$\Longrightarrow$	6	$\Longrightarrow$	5	$\Longrightarrow$	5	FIN
9		6		3		3		6	
5		5		9		9		9	

Créer la fonction **tri_crepes** ayant pour paramètre une pile **P**. Cette fonction trie la pile **P** selon la méthode du tri crêpes et ne renvoie rien. On utilisera les fonctions créées dans les questions précédentes.

Exemple : Si la pile **P** est

7
14
12
5
8

, après l'appel de **tri_crepes(P)**, la pile P devient

5
7
8
12
14

Exercice 2. Avec des classes

On crée une classe Pile qui modélise la structure d'une pile d'entiers. Le constructeur de la classe initialise une pile vide. La définition de cette classe sans l'implémentation de ses méthodes est donnée ci-dessous.

```
class Pile:
    def __init__ (self):
        """Initialise la pile comme une pile vide."""

    def est_vide(self):
        """Renvoie True si la liste est vide, False sinon."""

    def empiler(self, e):
        """Ajoute l'élément e sur le sommet de la pile, ne renvoie rien."""
    def depiler(self):
        """Retire l'élément au sommet de la pile et le renvoie."""

    def nb_elements(self):
        """Renvoie le nombre d'éléments de la pile. """

    def afficher(self):
        """Affiche de gauche à droite les éléments de la pile, du
        fond de la pile vers son sommet. Le sommet est alors
        l'élément affiché le plus à droite. Les éléments sont
        séparés par une virgule. Si la pile est vide la méthode
        affiche "pile vide"."""
```

Seules les méthodes de la classe ci-dessus doivent être utilisées pour manipuler les objets Pile.

1. (a) Écrire une suite d'instructions permettant de créer une instance de la classe *Pile* affectée à une variable *pile1* contenant les éléments 7, 5 et 2 insérés dans cet ordre.
   Ainsi, à l'issue de ces instructions, l'instruction *pile1.afficher()* produit l'affichage : *7, 5, 2.*

   (b) Donner l'affichage produit après l'exécution des instructions suivantes.

```
element1 = pile1.depiler()
pile1.empiler(5)
pile1.empiler(element1)
pile1.afficher()
```

2. On donne la fonction mystere suivante :

```
def mystere(pile, element):
    pile2 = Pile()
    nb_elements = pile.nb_elements()
    for i in range(nb_elements):
        elem = pile.depiler()
        pile2.empiler(elem)
        if elem == element:
            return pile2
    return pile2
```

   (a) Dans chacun des quatre cas suivants, quel est l'affichage obtenu dans la console ?

- Cas n^o 1

```
>>>pile.afficher()
7, 5, 2, 3
>>>mystere(pile, 2).afficher()
```

- Cas n^o 2

```
>>>pile.afficher()
7, 5, 2, 3
>>>mystere(pile, 9).afficher()
```

- Cas n^o 3

```
>>>pile.afficher()
7, 5, 2, 3
>>>mystere(pile, 3).afficher()
```

- Cas n^o 4

```
>>>pile.est_vide()
True
>>>mystere(pile, 3).afficher()
```

(b) Expliquer ce que permet d'obtenir la fonction mystere.

3. Écrire une fonction **etendre(pile1, pile2)** qui prend en arguments deux objets *Pile* appelés *pile1* et *pile2* et qui modifie *pile1* en lui ajoutant les éléments de *pile2* rangés dans l'ordre inverse. Cette fonction ne renvoie rien. On donne ci-dessous les résultats attendus pour certaines instructions.

```
>>>pile1.afficher()
7, 5, 2, 3
>>>pile2.afficher()
1, 3, 4
>>>etendre(pile1, pile2)
>>>pile1.afficher()
7, 5, 2, 3, 4, 3, 1
>>>pile2.est_vide()
True
```

4. Écrire une fonction **supprime_toutes_occurences(pile, element)** qui prend en arguments un objet *Pile* appelé *pile* et un élément *element* et supprime tous les éléments *element* de *pile*. On donne ci-dessous les résultats attendus pour certaines instructions.

```
>>>pile.afficher()
7, 5, 2, 3, 5
>>>supprime_toutes_occurences(pile, 5)
>>>pile.afficher()
7, 2, 3
```

5. Programmer toutes les méthodes de la classe pile en utilisant une liste vide comme constructeur d'une instance de **Pile**.