

Exercices - Listes, Piles, Files

Exercice 1 Listes

Dans cette exercice, nous allons reprendre l'interface de gestion des listes vue dans le cours.

1. Voici une série d'instructions.

```
L = listeVide()
ajoutEnTete(10,L)
ajoutEnTete(9,L)
ajoutEnTete(7,L)
t = supprEntete(L)
```

Représenter à chaque étape l'état de la liste sous la forme d'une liste chaîné en indiquant la tête de liste.

2. Représenter la liste L2 créée de la manière suivante :

```
L2 = listeVide()
L2 = cons(5, cons(4, cons(3, cons(2, cons(1, cons(0,L2)))))
```

3. Que retourne l'instruction `car(L2)` ?
4. Que retourne l'instruction `cdr(L2)` ?
5. Que retourne l'instruction `car(cdr(L2))` ?
6. Que retourne l'instruction `cdr(cdr(cdr(L2)))` ?
7. Que retourne l'instruction `car(cdr(cdr(L2)))` ?
8. Que retourne l'instruction `cdr(cdr(cdr(cdr(cdr(cdr(L2)))))` ?

Exercice 2 Piles

1. Soit une pile P initialement vide. On exécute les instructions suivantes :

```
push(P,4)
push(P,7)
a = pop(P)
b = taille(P)
c = pop(P)
push(P,3)
push(P,2)
d = taille(P)
```

Représenter le contenu de la pile P et donner la valeur de a, la valeur de b, la valeur de c et la valeur de d.

2. On implémente maintenant une pile à l'aide d'une liste python.

- (a) Expliquer l'instruction `pile = []`.
- (b) Expliquer l'instruction `pile.append(5)`.
- (c) Expliquer l'instruction `pile.append(10)`.
- (d) Expliquer l'instruction `pile.pop()`.
- (e) Donner l'état de la pile `pile` après l'exécution du programme ci-contre.

```
pile = []
tab = [5,8,6,1,3,7]
pile.append(5)
pile.append(10)
pile.append(8)
pile.append(15)
for i in tab:
    if i > 5:
        pile.pop()
```

Exercice 3 Files

1. Soit une file F initialement vide. On exécute les instructions suivantes :

```
enqueue (F, 6)
enqueue (F, 3)
a = dequeue (F)
enqueue (F, 9)
b = taille (F)
enqueue (F, 17)
c = dequeue (F)
enqueue (F, 2)
d = taille (F)
```

Représenter le contenu de la file F et donner la valeur de a, la valeur de b, la valeur de c et la valeur de d.

2. On implémente maintenant une file à l'aide d'une liste python.

- (a) Expliquer l'instruction `file = []`.
- (b) Expliquer l'instruction `file.append(5)`.
- (c) Expliquer l'instruction `file.append(10)`.
- (d) Expliquer l'instruction `file.pop(0)`.
- (e) Donner l'état de la file `file` après l'exécution du programme ci-contre.

```
file = []
tab = [2,78,6,89,3,17]
file.append(5)
file.append(10)
file.append(8)
file.append(15)
for i in tab:
    if i > 50:
        file.pop(0)
```

Exercice 4

Dans cet exercice nous avons à notre disposition 4 fonctions :

- `pileVide()` qui permet de créer une pile vide
- `estVide(P)` qui renvoie True si la pile P est vide et False dans le cas contraire
- `empiler(P, val)` qui ajoute la valeur val à la pile P
- `depiler(P)` qui renvoie l'élément au sommet de la pile P et qui supprime cet élément de la pile.

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

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

```
Q = pileVide ()
while not estVide ( P ):
    empiler ( Q , depiler ( P ))
```

4	← Sommet (Top)
2	
5	
8	

Pile P

2. (a) 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 feuille le programme Python suivant implémentant la fonction `hauteur_pile` en remplaçant les ??? par les bonnes instructions.

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

- (b) 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.

Exemple : si `P` est la pile de la question 1 : `max_pile(P, 2) = 1`

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, après l'appel de `retourner(P, 3)`, l'état de la pile `P` sera :

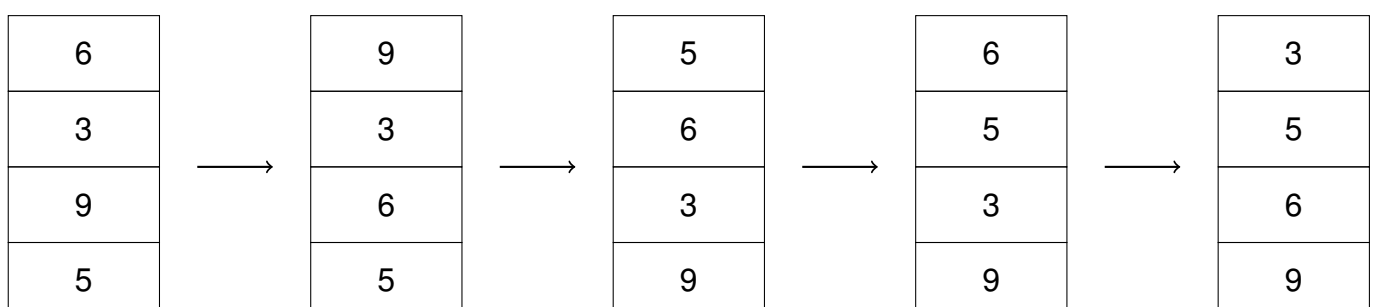
5	← Sommet (Top)
2	
4	
8	

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.

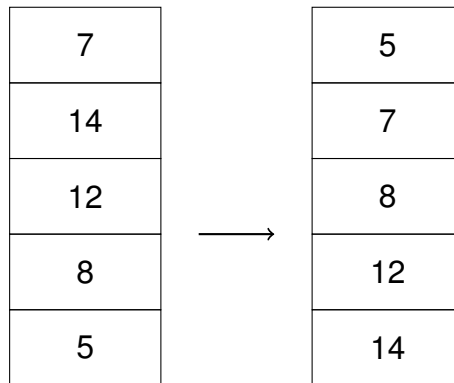
Exemple :



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



Exercice 5 Extrait BAC 2022 sujet 6

Un supermarché met en place un système de passage automatique en caisse. Un client scanne les articles à l'aide d'un scanner de code-barres au fur et à mesure qu'il les ajoute dans son panier. Les articles s'enregistrent alors dans une structure de données. La structure de données utilisée est une file définie par la classe `Panier`, avec les primitives habituelles sur la structure de file. Pour faciliter la lecture, le code de la classe `Panier` n'est pas écrit.

```
class Panier():
    def __init__(self):
        """Initialise la file comme une file vide."""

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

    def enfiler(self, e):
        """Ajoute l'element e en derniere position de la file, ne renvoie rien."""

    def defiler(self):
        """Retire le premier element de la file et le renvoie."""
```

Le panier d'un client sera représenté par une file contenant les articles scannés. Les articles sont représentés par des tuples (`code_barre`, `designation`, `prix`, `horaire_scan`) où

- `code_barre` est un nombre entier identifiant l'article ;
- `designation` est une chaine de caractères qui pourra être affichée sur le ticket de caisse ;
- `prix` est un nombre décimal donnant le prix d'une unité de cet article ;
- `horaire_scan` est un nombre entier de secondes permettant de connaître l'heure où l'article a été scanné.

1. On souhaite ajouter un article dont le tuple est le suivant (31002, "café noir", 1.50, 50525).

Ecrire le code utilisant une des quatre méthodes ci-dessus permettant d'ajouter l'article à l'objet de classe `Panier` appelé `panier1`.

2. On souhaite définir une méthode `remplir(panier_temp)` dans la classe `Panier` permettant de remplir la file avec tout le contenu d'un autre panier `panier_temp` qui est un objet de type `Panier`.

Recopier et compléter le code de la méthode `remplir` en remplaçant les pointillés par la primitive de file qui convient.

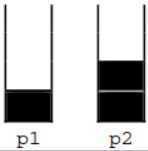
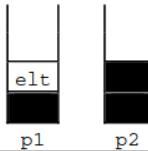
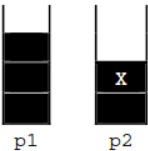
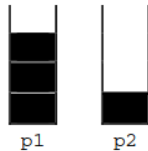
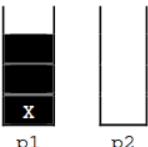
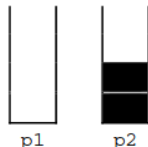
```
def remplir(self, panier_temp):
    while not panier_temp. ....:
        article = panier_temp. ...
        self. ...(article)
```

3. Pour que le client puisse connaître à tout moment le montant de son panier, on souhaite ajouter une méthode `prix_total()` à la classe `Panier` qui renvoie la somme des prix de tous les articles présents dans le panier. Ecrire le code de la méthode `prix_total`. Attention, après l'appel de cette méthode, le panier devra toujours contenir ses articles.
4. Le magasin souhaite connaître pour chaque client la durée des achats. Cette durée sera obtenue en faisant la différence entre le champ `horaire_scan` du dernier article scanné et le champ `horaire_scan` du premier article scanné dans le panier du client. Un panier vide renverra une durée égale à zéro. On pourra accepter que le panier soit vide après l'appel de cette méthode. Ecrire une méthode `duree_courses` de la classe `Panier` qui renvoie cette durée.

Exercice 6 Extrait BAC 2021 sujet 3

Une méthode simple pour gérer l'ordonnancement des processus est d'exécuter les processus en une seule fois et dans leur ordre d'arrivée.

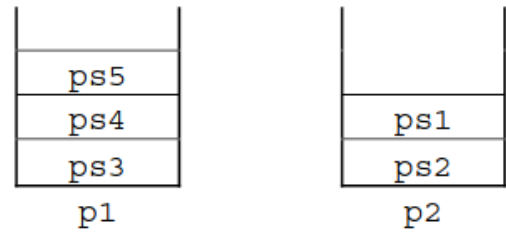
1. Parmi les propositions suivantes, quelle est la structure de données la plus appropriée pour mettre en œuvre le mode FIFO (First In First Out) ?
 - (a) liste
 - (b) dictionnaire
 - (c) pile
 - (d) file
2. On choisit de stocker les données des processus en attente à l'aide d'une liste Python `lst`. On dispose déjà d'une fonction `retirer(lst)` qui renvoie l'élément `lst[0]` puis le supprime de la liste `lst`.
Écrire en Python le code d'une fonction `ajouter(lst, proc)` qui ajoute à la fin de la liste `lst` le nouveau processus en attente `proc`.
3. On choisit maintenant d'implémenter une file `file` à l'aide d'un couple `(p1,p2)` où `p1` et `p2` sont des piles.
Ainsi `file[0]` et `file[1]` sont respectivement les piles `p1` et `p2`.
Pour enfiler un nouvel élément `elt` dans `file`, on l'empile dans `p1`.
Pour défiler `file`, deux cas se présentent.
 - La pile `p2` n'est pas vide : on dépile `p2`.
 - La pile `p2` est vide : on dépile les éléments de `p1` en les empilant dans `p2` jusqu'à ce que `p1` soit vide, puis on dépile `p2`.

	État de la file avant	État de la file après
<code>enfiler(file,elt)</code>	 <p>p1 p2</p>	 <p>p1 p2</p>
<code>defiler(file)</code> cas où <code>p2</code> n'est pas vide	 <p>p1 p2</p>	 <p>p1 p2</p>
<code>defiler(file)</code> cas où <code>p2</code> est vide	 <p>p1 p2</p>	 <p>p1 p2</p>

On considère la situation représentée ci-contre :

On exécute la séquence d'instructions suivante :

```
enfiler(file, ps6)
defiler(file)
defiler(file)
defiler(file)
enfiler(file, ps7)
```



Représenter le contenu final des deux piles à la suite de ces instructions.

4. On dispose des fonctions :

- `empiler(p, elt)` qui empile l'élément `elt` dans la pile `p`,
- `depiler(p)` qui renvoie le sommet de la pile `p` si `p` n'est pas vide et le supprime,
- `pile_vide(p)` qui renvoie `True` si la pile `p` est vide, `False` si la pile `p` n'est pas vide.

- Écrire en Python une fonction `est_vide(f)` qui prend en argument un couple de piles `f` et qui renvoie `True` si la file représentée par `f` est vide, `False` sinon.
- Écrire en Python une fonction `enfiler(f, elt)` qui prend en arguments un couple de piles `f` et un élément `elt` et qui ajoute `elt` en queue de la file représentée par `f`.
- Écrire en Python une fonction `defiler(f)` qui prend en argument un couple de piles `f` et qui renvoie l'élément en tête de la file représentée par `f` en le retirant.

