

Table des matières

1	Centres-Étrangers 2021	2
2	France J1 2022	3
3	France J2 2022	5
4	Mayotte J2 2022	8
5	Polynésie J1 2022	10
6	Sujet 0 - 2022	11
7	France Sept 2021	14

1 Centres-Étrangers 2021

Notion abordée : structures de données : les piles.

Dans cet exercice, on considère une pile d'entiers positifs. On suppose que les quatre fonctions suivantes ont été programmées préalablement en langage Python :

- `empiler(P, e)` : ajoute l'élément `e` sur la pile `P` ;
- `depiler(P)` : enlève le sommet de la pile `P` et renvoie la valeur de ce sommet ;
- `est_vide(P)` : renvoie `True` si la pile `P` est vide et `False` sinon ;
- `creer_pile()` : renvoie une pile vide.

Dans cet exercice, seule l'utilisation de ces quatre fonctions sur la structure de données pile est autorisée.

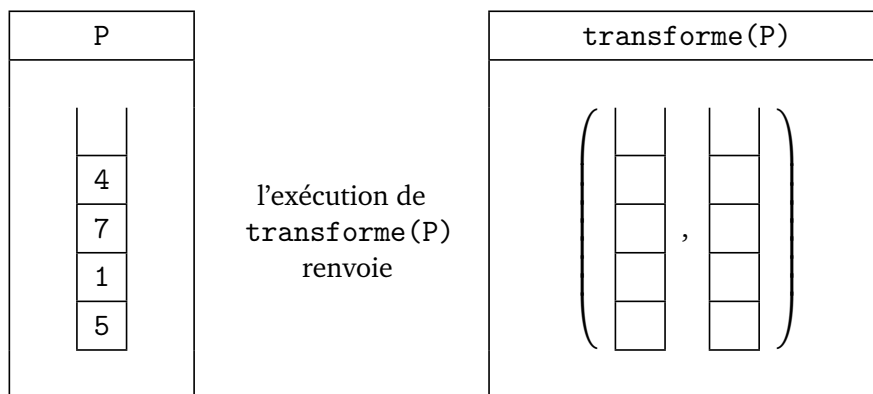
1. Recopier le schéma ci-dessous et le compléter sur votre copie en exécutant les appels de fonctions donnés. On écrira ce que renvoie la fonction utilisée dans chaque cas, et on indiquera `None` si la fonction ne renvoie aucune valeur.

	Etape 0 Pile d'origine P	Etape 1 <code>empiler(P,8)</code>	Etape 2 <code>depiler(P)</code>	Etape 3 <code>est_vide(P)</code>
	<div style="border: 1px solid black; padding: 5px; text-align: center;"> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto; text-align: center;">4</div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto; text-align: center;">7</div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto; text-align: center;">1</div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto; text-align: center;">5</div> </div>	<div style="border: 1px solid black; padding: 5px; text-align: center;"> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> </div>	<div style="border: 1px solid black; padding: 5px; text-align: center;"> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> </div>	<div style="border: 1px solid black; padding: 5px; text-align: center;"> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> <div style="border: 1px solid black; height: 20px; width: 20px; margin: 0 auto;"></div> </div>
Valeur renvoyée	

2. On propose la fonction ci-dessous, qui prend en argument une pile `P` et renvoie un couple de piles :

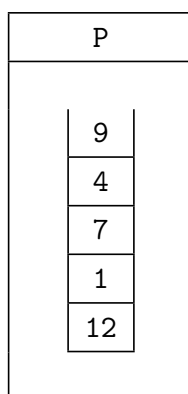
```
def transforme(P):
    Q = creer_pile()
    while not est_vide(P):
        v = depiler(P)
        empiler(Q,v)
    return (P,Q)
```

Recopier et compléter sur votre copie le document ci-dessous



3. Ecrire en langage Python une fonction `maximum` recevant une pile `P` comme argument et renvoyant la valeur maximale de cette pile. On ne s'interdit pas qu'après exécution de la fonction, la pile `P` soit vide.

On souhaite à présent connaître le nombre d'éléments d'une pile `P` à l'aide de la fonction `taille` recevant cette pile en paramètre. Par exemple, l'exécution de `taille(P)` avec la pile `P` ci-dessous renvoie l'entier 5.



4. (a) Proposer une stratégie écrite en langage naturel et/ou expliquée à l'aide de schémas, qui permette de mettre en place une telle fonction.
- (b) Donner le code Python de cette fonction `taille(P)` (on pourra utiliser les cinq fonctions déjà programmées).

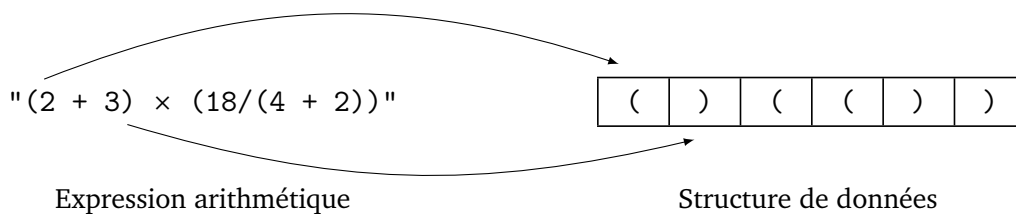
2 France J1 2022

Partie A : expression correctement parenthésée

On veut déterminer si une expression arithmétique est correctement parenthésée, c'est-à-dire que pour chaque parenthèse fermante `)` correspond une parenthèse précédemment ouverte `(`. Par exemple :

- L'expression arithmétique `"(2 + 3) × (18/(4 + 2))"` est correctement parenthésée.
- L'expression arithmétique `"(2 + 3) × (18/(4 + 2"` est non correctement parenthésée.

Pour simplifier les expressions arithmétiques, on enregistre, dans une structure de données, uniquement les parenthèses dans leur ordre d'apparition. On appelle *expression simplifiée* cette structure.



1. Indiquer si la phrase « les éléments sont maintenant retirés (pour être lus) de cette structure de données dans le même ordre qu'ils y ont été ajoutés lors de l'enregistrement » décrit le comportement d'une file ou d'une pile. Justifier.

Pour vérifier le parenthésage, on peut utiliser une variable contrôleur qui :

- est un nombre entier égal à 0 en début d'analyse de l'expression simplifiée ;
- augmente de 1 si l'on rencontre une parenthèse ouvrante "(" ;
- diminue de 1 si l'on rencontre une parenthèse fermante ")" .

Exemple : on considère l'expression simplifiée A : "(" (()) "

Lors de l'analyse de l'expression A, la variable contrôleur (initialement égal à 0) prend successivement pour valeur 1, 0, 1, 2, 1, 0. Le parenthésage est donc correct.

2. Ecrire, pour chacune des deux expressions simplifiées B et C suivantes, les valeurs successives prises par la variable contrôleur lors de leur analyse.

- Expression simplifiée B : "(" (()) "
- Expression simplifiée C : "(" ())) "

3. L'expression simplifiée B précédente est mal parenthésée (parenthèses fermantes manquantes) car la variable contrôleur est différent de zéro en fin d'analyse.

L'expression simplifiée C précédente est également mal parenthésée (parenthèse fermante sans parenthèse ouvrante) car la variable contrôleur prend une valeur négative pendant l'analyse.

Recopier et compléter uniquement les lignes 13 et 16 du code ci-dessous pour que la fonction `parenthesage_correct` réponde à sa description.

```

1 def parenthesage_correct(expression):
2     ''' fonction retournant True si l'expression arithmétique
3     simplifiée (str) est correctement parenthésée, False
4     sinon.
5     Condition: expression ne contient que des parenthèses
6     ouvrantes et fermantes '''
7     controleur = 0
8     for parenthese in expression: #pour chaque parenthèse
9         if parenthese == '(':
10             controleur = controleur + 1
11         else: # parenthese == ')'
12             controleur = controleur - 1
13             if controleur ... : # test 1 (à recopier et compléter)
14                 #parenthèse fermante sans parenthèse ouvrante
15                 return False
16         if controleur ... : # test 2 (à recopier et compléter)
17             return True #le parenthésage est correct
18     else:
19         return False #parenthèse(s) fermante(s) manquante(s)

```

Partie B : texte correctement balisé

On peut faire l'analogie entre le texte simplifié des fichiers HTML (uniquement constitué de balises ouvrantes `<nom>` et fermantes `</nom>`) et les expressions parenthésées :


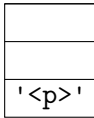
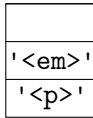
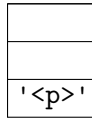
Par exemple, l'expression HTML simplifiée : `"<p></p>"` est correctement balisée.

On ne tiendra pas compte dans cette partie des balises ne comportant pas de fermeture comme `
` ou ``.

Afin de vérifier qu'une expression HTML simplifiée est correctement balisée, on peut utiliser une pile (initialement vide) selon l'algorithme suivant. On parcourt successivement chaque balise de l'expression :

- lorsque l'on rencontre une balise ouvrante, on l'empile ;
- lorsque l'on rencontre une balise fermante :
 - si la pile est vide, alors l'analyse s'arrête : le balisage est incorrect ,
 - sinon, on dépile et on vérifie que les deux balises (la balise fermante rencontrée et la balise ouvrante dépilée) correspondent (c'est-à-dire ont le même nom) si ce n'est pas le cas, l'analyse s'arrête (balisage incorrect).

Exemple : état de la pile lors du déroulement de cet algorithme pour l'expression simplifiée `'<p></p>'` qui n'est pas correctement balisée.

<code>'<p></p>'</code> pile 	<code>'<p></p>'</code> ↑ pile balise ouvrante on l'empile 	<code>'<p></p>'</code> ↑ pile balise ouvrante on l'empile 	<code>'<p></p>'</code> ↑ pile balise fermante on dépile les balises <code>''</code> et <code>'</p>'</code> ne correspondent pas 
--	---	---	---

Etat de la pile lors du déroulement de l'algorithme

4. Cette question traite de l'état de la pile lors du déroulement de l'algorithme.

- Représenter la pile à chaque étape du déroulement de cet algorithme pour l'expression `"<p></p>"` (balisage correct).
- Indiquer quelle condition simple (sur le contenu de la pile) permet alors de dire que le balisage est correct lorsque toute l'expression HTML simplifiée a été entièrement parcourue, sans que l'analyse ne s'arrête.

5. Une expression HTML correctement balisée contient 12 balises.

Indiquer le nombre d'éléments que pourrait contenir au maximum la pile lors de son analyse.

3 France J2 2022

Cet exercice porte sur les structures de données.

La poussette est un jeu de cartes en solitaire. Cet exercice propose une version simplifiée de ce jeu basée sur des nombres.

On considère une pile constituée de nombres entiers tirés aléatoirement. Le jeu consiste à réduire la pile suivant la règle suivante : quand la pile contient du haut vers le bas un triplet dont les termes du haut et du bas sont de même parité, on supprime l'élément central.

Par exemple :

- ▷ Si la pile contient du haut vers le bas, le triplet 1 0 3, on supprime le 0.
- ▷ Si la pile contient du haut vers le bas, le triplet 1 0 8, la pile reste inchangée.

On parcourt la pile ainsi de haut en bas et on procède aux réductions. Arrivé en bas de la pile, on recommence la réduction en repartant du sommet de la pile jusqu'à ce que la pile ne soit plus réductible. Une partie est « gagnante » lorsque la pile finale est réduite à deux éléments exactement.

Voici un exemple détaillé de déroulement d'une partie.

Premier parcours de la pile				Parcours suivant				Dernier parcours			
7	7										
5	5	7		7							
4	4	5	7	5	7	7		7			
3	3	3	5	3				3			
8	8	8	3								
9	9	9	9	9	9	9		9		7	
6	6	6	6	6	6	6		6		9	
										6	6

Q1. (a) Donner les différentes étapes de réduction de la pile suivante :

4
9
8
7
4
2

(b) Parmi les piles proposées ci-dessous, donner celle qui est gagnante.

5
4
5
4
2
1

Pile A

4
5
4
9
2
0

Pile B

3
4
8
7
6
1

Pile C

L'interface d'une pile est proposée ci-dessous. On utilisera uniquement les fonctions figurant dans le tableau suivant :

Structure de données abstraite : Pile

- ▷ `creer_pile_vide()` renvoie une pile vide
- ▷ `est_vide(p)` renvoie `True` si `p` est vide, `False` sinon
- ▷ `empiler(p, element)` ajoute `element` au sommet de `p`
- ▷ `depiler(p)` retire l'élément au sommet de `p` et le renvoie
- ▷ `sommet(p)` renvoie l'élément au sommet de `p` sans le retirer de `p`
- ▷ `taille(p)` renvoie le nombre d'éléments de `p`

Q2. La fonction `reduire_triplet_au_sommet` permet de supprimer l'élément central des trois premiers éléments en partant du haut de la pile, si l'élément du bas et du haut sont de même parité. Les éléments dépilés et non supprimés sont replacés dans le bon ordre dans la pile.

Recopier et compléter sur la copie le code de la fonction `reduire_triplet_au_sommet` prenant une pile `p` en paramètre et qui la modifie en place. Cette fonction ne renvoie donc rien.

```

1 def reduire_triplet_au_sommet(p):
2     a = depiler(p)
3     b = depiler(p)
4     c = sommet(p)
5     if a % 2 != ... :
6         empiler(p, ...)
7     empiler(p, ...)

```

Q3. On se propose maintenant d'écrire une fonction `parcourir_pile_en_reduisant` qui parcourt la pile du haut vers le bas en procédant aux réductions pour chaque triplet rencontré quand cela est possible.

- (a) Donner la taille minimale que doit avoir une pile pour être réductible.
 (b) Recopier et compléter sur la copie :

```

9 def parcourir_pile_en_reduisant(p):
10     q = creer_pile_vide()
11     while taille(p) >= ... :
12         reduire_triplet_au_sommet(p)
13         e = depiler(p)
14         empiler(q, e)
15     while not est_vide(q):
16         ...
17         ...
18     return p

```

Q4. Partant d'une pile d'entiers `p`, on propose ici d'implémenter une fonction récursive `jouer` renvoyant la pile `p` entièrement simplifiée. Une fois la pile parcourue de haut en bas et réduite, on procède à nouveau à sa réduction à condition que cela soit possible. Ainsi :

- ▷ Si la pile `p` n'a pas subi de réduction, on la renvoie.
- ▷ Sinon on appelle à nouveau la fonction `jouer`, prenant en paramètre la pile réduite.

Recopier et compléter sur la copie le code ci-dessous :

```

20 def jouer(p):
21     taille_p = taille(p)
22     q = parcourir_pile_en_reduisant(p)
23     if ... :
24         return p
25     else:
26         return jouer(...)

```

4 Mayotte J2 2022

La notation polonaise inverse (NPI) permet d'écrire des expressions de calculs numériques sans utiliser de parenthèse. Cette manière de présenter les calculs a été utilisée dans des calculatrices de bureau dès la fin des années 1960. La NPI est une forme d'écriture d'expressions algébriques qui se distingue par la position relative que prennent les nombres et leurs opérations.

Par exemple :

Notation classique	Notation NPI
$3 + 9$	$3\ 9\ +$
$8 \times (3 + 5)$	$8\ 3\ 5\ +\ \times$
$(17 + 5) \times 4$	$17\ 5\ +\ 4\ \times$

L'expression est lue et évaluée de la gauche vers la droite en mettant à jour une pile.

- ★ Les nombres sont empilés dans l'ordre de la lecture.
- Dès la lecture d'un opérateur (+, −, ×, /), les deux nombres au sommet de la pile sont dépilés et remplacés par le résultat de l'opération effectuée avec ces deux nombres. Ce résultat est ensuite empilé au sommet de la pile.

A la fin de la lecture, la valeur au sommet est renvoyée.

Exemple : l'expression $7\ 3\ 25\ +\ \times$ qui correspond au calcul $7 \times (3 + 25)$ s'évalue à 196 comme le montrent les états successifs de la pile créée, nommée p :

- ★ On empile la valeur 7.
- ★ On empile la valeur 3.
- ★ On empile la valeur 25.
- ★ On remplace les deux nombres du sommet de la pile (25 et 3) par leur somme 28.
- ★ On remplace les deux nombres du sommet de la pile (28 et 7) par leur produit 196.

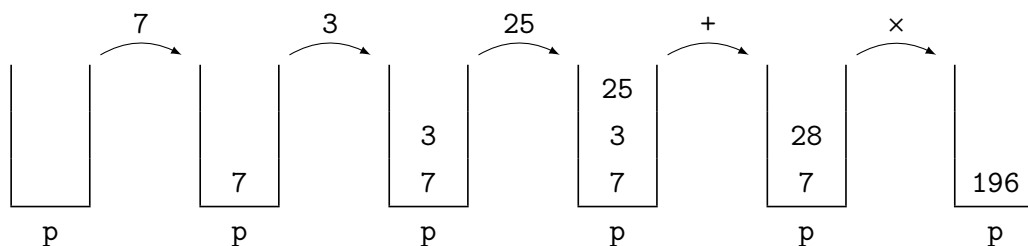
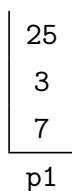


Schéma descriptif des différentes étapes d'exécution

1. En vous inspirant de l'exemple ci-dessus, dessiner le schéma descriptif de ce que donne l'évaluation par la NPI de l'expression $12 \ 4 \ 5 \ \times \ +$.
2. On dispose de la pile suivante nommée p1 :



On rappelle ci-dessous les primitives de la structure de pile (LIFO : Last In First out) :

Fonction	Description
pile_vide()	Crée et renvoie une nouvelle pile vide.
empiler(p, e)	Place l'élément e au sommet de la pile p.
depiler(p)	Supprime et renvoie l'élément se trouvant au sommet de la pile p.
est_vide(p)	Renvoie un booléen indiquant si la pile p est vide ou non.

On dispose aussi de la fonction suivante, qui prend en paramètre une pile p :

```
def top(p) :
    x = depiler(p)
    empiler(p, x)
    return x
```

On exécute la ligne suivante `temp = top(p1)` :

- (a) Quelle valeur contient la variable `temp` après cette exécution ?
 - (b) Représenter la pile p1 après cette exécution.
3. En utilisant uniquement les quatre primitives d'une pile, écrire en langage Python la fonction `addition` qui prend en paramètre une pile p d'au moins deux éléments et qui remplace les deux nombres du sommet de p par leur somme.
Remarque : cette fonction ne renvoie rien, mais la pile p est modifiée.
 4. On considère que l'on dispose également d'une fonction `multiplication` qui prend en paramètre une pile p d'au moins deux éléments et qui remplace les deux nombres du sommet de p par leur produit (on ne demande pas d'écrire cette fonction).
Recopier et compléter, en n'utilisant que les primitives d'une pile et les deux fonctions `addition` et `multiplication`, la suite d'instructions (ci-dessous) qui réalise le calcul $(3 + 5) \times 7$ dont l'écriture en NPI est : $3 \ 5 \ + \ 7 \ \times$

```
p = pile_vide()
empiler(p,3)
# à compléter
```

5 Polynésie J1 2022

La classe Pile utilisée dans cet exercice est implémentée en utilisant des listes Python et propose quatre éléments d'interface :

- Un constructeur qui permet de créer une pile vide, représentée par [] ;
- La méthode `est_vide` qui renvoie `True` si l'objet est une pile ne contenant aucun élément, et `False` sinon ;
- La méthode `empiler` qui prend un objet quelconque en paramètre et ajoute cet objet au sommet de la pile. Dans la représentation de la pile dans la console, cet objet apparaît à droite des autres éléments de la pile ;
- La méthode `depiler` qui renvoie l'objet présent au sommet de la pile et le retire de la pile.

Exemples :

```
>>> mapile = Pile()
>>> mapile.empiler(2)
>>> mapile
[2]
>>> mapile.empiler(3)
>>> mapile.empiler(50)
>>> mapile
[2, 3, 50]
>>> mapile.depiler()
50
>>> mapile
[2, 3]
```

La méthode `est_triee` ci-dessous renvoie `True` si, en dépilant tous les éléments, ils sont traités dans l'ordre croissant, et `False` sinon.

```
1 def est_triee(self):
2     if not self.est_vide() :
3         e1 = self.depiler()
4         while not self.est_vide():
5             e2 = self.depiler()
6             if e1 ... e2 :
7                 return False
8             e1 = ...
9     return True
```

1. Recopier sur la copie les lignes 6 et 8 en complétant les points de suspension.

On crée dans la console la pile A représentée par [1, 2, 3, 4].

2. (a) Donner la valeur renvoyée par l'appel `A.est_triee()`.
(b) Donner le contenu de la pile A après l'exécution de cette instruction.

On souhaite maintenant écrire le code d'une méthode `depileMax` d'une pile non vide ne contenant que des nombres entiers et renvoyant le plus grand élément de cette pile en le retirant de la pile. Après l'exécution de `p.depileMax()`, le nombre d'éléments de la pile `p` diminue donc de 1.

```

1 def depileMax(self):
2     assert not self.est_vide(), "Pile vide"
3     q = Pile()
4     maxi = self.depiler()
5     while not self.est_vide() :
6         elt = self.depiler()
7         if maxi < elt :
8             q.empiler(maxi)
9             maxi = ...
10        else :
11            ...
12    while not q.est_vide():
13        self.empiler(q.depiler())
14    return maxi

```

3. Recopier sur la copie les lignes 9 et 11 en complétant les points de suspension.

On crée la pile B représentée par [9, -7, 8, 12, 4] et on effectue l'appel B.depileMax().

4. (a) Donner le contenu des piles B et q à la fin de chaque itération de la boucle while de la ligne 5.
- (b) Donner le contenu des piles B et q avant l'exécution de la ligne 14.
- (c) Donner un exemple de pile qui montre que l'ordre des éléments restants n'est pas préservé après l'exécution de depileMax.

On donne le code de la méthode traiter :

```

1 def traiter(self):
2     q = Pile()
3     while not self.est_vide():
4         q.empiler(self.depileMax())
5     while not q.est_vide():
6         self.empiler(q.depiler())

```

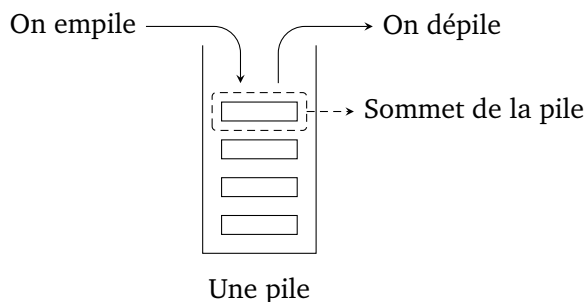
5. (a) Donner les contenus successifs des piles B et q
- avant la ligne 3,
 - avant la ligne 5,
 - à la fin de l'exécution de la fonction traiter
- lorsque la fonction traiter est appliquée sur la pile B contenant [1, 6, 4, 3, 7, 2].
- (b) Expliquer le traitement effectué par cette méthode.

6 Sujet 0 - 2022

Exercice 1

Cet exercice porte sur la notion de pile et sur la programmation de base en python.

On rappelle qu'une pile est une structure de données abstraite fondée sur le principe « dernier arrivé, premier sorti » :



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

Structure de données abstraite : Pile
Utilise : Éléments, Booléen
Opérations :
▷ <code>creer_pile_vide</code> : $\emptyset \rightarrow \text{Pile}$ <code>creer_pile_vide()</code> renvoie une pile vide.
▷ <code>est_vide</code> : $\text{Pile} \rightarrow \text{Booléen}$ <code>est_vide(pile)</code> renvoie True si pile est vide, False sinon.
▷ <code>empiler</code> : $\text{Pile}, \text{Élément} \rightarrow \text{Rien}$ <code>empiler(pile, element)</code> ajoute <code>element</code> au sommet de la pile.
▷ <code>depiler</code> : $\text{Pile} \rightarrow \text{Élément}$ <code>depiler(pile)</code> renvoie l'élément au sommet de la pile en le retirant de la pile.

Q1. 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 ?

```

1 Q = creer_pile_vide()
2 while not est_vide(P):
3     empiler(Q, depiler(P))

```

Q2. (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 Q1. alors `hauteur_pile(P)=4`.
 Recopier et compléter sur votre copie le programme python implémentant la fonction `hauteur_pile` en remplaçant les ??? par les bonnes instructions.

```

1 def hauteur_pile(P):
2     Q = creer_pile_vide()
3     n = 0
4     while not(est_vide(P)):
5         # ???
6         x = depiler(P)

```

```

7     empiler(Q, x)
8     while not(est_vide(Q)):
9         # ???
10        empiler(P, x)
11    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 Q1. alors `max_pile(P, 2) = 1`.

- Q3. 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 Q1 alors après l'appel de `retourner(P, 3)`, l'état de la pile `P` sera :

5
2
4
8

- Q4. L'objectif de cette question est de trier une pile de crêpes.

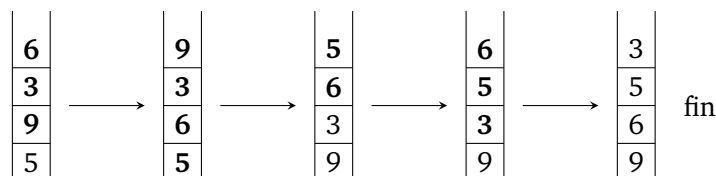
On modélise une pile de crêpes par une pile d'entiers représentant le diamètre de chaque crêpe. On souhaite réordonner les crêpes de la plus grande (placée en bas de la pile) à la plus petite (placée en haut de la pile).

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

7
14
12
5
8

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

7
14
12
5
8

7 France Sept 2021

Une entreprise fabrique des yaourts qui peuvent être soit nature (sans arôme), soit aromatisés (fraise, abricot ou vanille).

Pour pouvoir traiter informatiquement les spécificités de ce produit, on va donc créer une classe Yaourt qui possèdera un certain nombre d'attributs :

- son genre : nature ou aromatisé ;
- son arôme : fraise, abricot, vanille ou aucun ;
- sa date de durabilité minimale (DDM) exprimée par un entier compris entre 1 et 365 (on ne gère pas les années bissextiles). Par exemple, si la DDM est égale à 15, la date de durabilité minimale est le 15 janvier.

On va créer également des méthodes permettant d'interagir avec l'objet Yaourt pour attribuer un arôme ou récupérer un genre par exemple. On peut représenter cette classe par le tableau de spécifications ci-dessous :

Yaourt	
ATTRIBUTS :	METHODES :
★ genre	★ Construire(arome,duree)
★ arome	★ Obtenir_Arome()
★ duree	★ Obtenir_Genre()
	★ Obtenir_Duree()
	★ Attribuer_Arome(arome)
	★ Attribuer_Duree(duree)
	★ Attribuer_Genre(arome)

1. La classe Yaourt est déclarée en Python à l'aide du mot-clé class :

```
class Yaourt:
    """Classe définissant un yaourt caractérisé par :
        - son arome
        - son genre
        - sa durée de durabilité minimale"""
```

L'annexe 1 de l'exercice 4 donne le code existant et l'endroit des codes à produire dans les questions suivantes.

(a) Quelles sont les assertions à prévoir pour vérifier que l'arôme et la durée correspondent bien à des valeurs acceptables. Il faudra aussi expliciter les commentaires qui seront renvoyés. Pour rappel :

- L'arôme doit prendre comme valeur 'fraise', 'abricot', 'vanille' ou 'aucun'.
- Sa date de durabilité minimale (DDM) est une valeur entière comprise entre 1 et 365.

(b) Pour créer un yaourt, on exécutera la commande suivante :

```
Mon_Yaourt = Yaourt('fraise',24)
```

Quelle valeur sera affectée à l'attribut `genre` associé à `Mon_Yaourt` ?

- (c) Ecrire en Python une méthode `GetArome(self)`, renvoyant l'arôme du yaourt créé.
2. On appelle *mutateur* une méthode permettant de modifier un ou plusieurs attributs d'un objet. Sur votre copie, écrire en Python le mutateur `SetArome(self,arome)` permettant de modifier l'arôme du yaourt.
On veillera à garder une cohérence entre l'arôme et le genre.
3. On veut créer une pile contenant le stock de yaourts. Pour cela il faut tout d'abord créer une pile vide :

```
def creer_pile():
    pile = []
    return pile
```

- (a) Créer une fonction `empiler(p,Yaourt)` qui renvoie la pile `p` après avoir ajouté un objet de type `Yaourt` à la fin.
- (b) Créer une fonction `depiler(p)` qui renvoie l'objet à dépiler.
- (c) Créer une fonction `estVide(p)` qui renvoie `True` si la pile `p` est vide et `False` sinon.
- (d) Qu'affiche le bloc de commandes suivantes ?

```
mon_Yaourt1 = Yaourt('aucun',18)
mon_Yaourt2 = Yaourt('fraise',24)
ma_pile = creer_pile()
empiler(ma_pile, mon_Yaourt1)
empiler(ma_pile, mon_Yaourt 2)
print(depiler(ma_pile).GetDuree())
print(estVide(ma_pile))
```