1 En autonomie sur les piles

1. Donnez les états successifs de la pile dans la séquence d'instructions suivante, ainsi que l'affiche produit.

```
pile = ApStack()
pile.push(1)
pile.push(7)
pile.push(5)
print(f"{pile.pop()}")
print(f"{pile.pop()}")
print(f"{pile.pop()}")
pile.is_empty()
```

2. Que fait la séquence d'instructions suivante ? Au besoin la modifier.

```
pile = ApStack()
pile.push(1)
pile.push(7)
pile.push(5)
while not pile.is_empty():
    print(f"{pile.top()}")
```

3. Indiquez ce que fait la fonction suivante.

```
def what_do_i_do(pile):
    res = 0
    while not pile.is_empty():
        res = res + pile.pop()
    pile.push(res)
```

2 Différentes représentations des éléments d'une pile

On veut réaliser une fonction `stack2chaine` qui représente tous les éléments d'une pile d'entiers sous la forme d'une chaîne de caractères, sans utiliser la méthode magique `__str__`, ni accéder directement à l'attribut de la classe ApStack. Une fois la chaîne de caractères construite, cette fonction ne doit pas avoir modifié la pile.

On va étudier deux représentations différentes qui seront illustrées en les appliquant sur la pile définie par :

```
st = ApStack()
for i in range(1, 5):
    st.push(i)
```

1. Dans un premier temps on demande une représentation verticale, un entier par ligne, dans l'ordre du sommet vers le bas de pile (voir exemple ci-dessous). La fonction doit parcourir la pile de façon itérative.

```
>>> print(stack_chaine1(st))
4
3
2
1
>>> st.top()
4
```

Le sommet de la pile `st` reste bien l'entier 4.

2. Dans un deuxième temps on demande une représentation horizontale, le bas de pile étant à gauche et le sommet à droite. La fonction doit parcourir la pile de façon récursive. La même pile que précédemment est affichée comme on le voit ci-dessous.

```
>>> print(stack_chaine2(st))
[1 2 3 4
>>> st.top()
4
```

3 Textes bien parenthésés

Dans de très nombreux contextes (compilation, pages HTML, XML, …), il est nécessaire de vérifier qu'un texte est *bien parenthésé*. Le bon parenthésage d'un texte signifie que

- toute parenthèse ouverte est fermée ;
- à toute parenthèse fermante correspond une parenthèse ouvrante ;
- et qu'à aucune position dans le texte le nombre de parenthèses fermantes n'est strictement supérieur à celui des parenthèses ouvrantes rencontrées auparavant.

À titre d'exemples, voici trois textes bien parenthésés (à gauche) et trois mal parenthésés (à droite).

bien	mal
`"(((()))))"`	`"(()(())"`
`"()()()"`	`"(())())"`
`""`	`"(()))(()"`

1. Vérifiez ces exemples.

Dans toute la suite on suppose que le texte est représenté par une chaîne de caractères, certains d'entre-eux étant des parenthèses ouvrantes `(` ou fermantes `)`.

2. Réalisez un prédicat qui prend un texte en paramètre et renvoie `True` si le texte est bien parenthésé, et `False` dans le cas contraire. L'algorithme doit utiliser une pile qui grandira lors de l'observation d'une parenthèse ouvrante et rapetissera lors de l'observation d'une parenthèse fermante.

4 Conversion en binaire

En utilisant une pile ne contenant que des 0 et des 1, programmez une fonction de conversion d'un nombre en binaire.

```
>>> to_bin(12)
'1100'
>>> to_bin(0)
'0'
```

5 Fusion de deux piles

1. Réalisez une fonction nommée `stack_merge` paramétrée par deux piles triées dans l'ordre croissant (du sommet vers le fond) et qui produit une liste triée dans l'ordre croissant des éléments contenus dans ces deux piles.

```
>>> st1 = ApStack()
>>> st1.push(11)
>>> st1.push(5)
>>> st1.push(1)
>>> st2 = ApStack()
>>> st2.push(9)
>>> st2.push(7)
>>> stack_merge(st1, st2)
[1, 5, 7, 9, 11]
```

2. Votre fonction a-t-elle un effet de bord ?

6 En autonomie sur les files

1. Donnez les états successifs de la file dans la séquence d'instructions suivante, ainsi que l'affiche produit.

```
file = ApQueue()
file.enqueue(1)
file.enqueue(7)
file.enqueue(5)
print(f"{file.dequeue()}")
print(f"{file.dequeue()}")
```

```
print(f"{file.dequeue()}")
file.is_empty()
```

3. Indiquez ce que fait la fonction suivante.

```
def what_do_i_do(file):
    res = ''
    while not file.is_empty():
        res = res + file.dequeue()
    return res
```

7 Parcours itératifs d'arbres binaires

On définit `exemple_td` par :

```
exemple_td = AB("A", AB("B", AB("C", AB(), AB()), AB("D", AB(), AB())),
                     AB("E", AB(), AB("F", AB(), AB())))
```

1. Dessinez l'arbre binaire `exemple_td`.

On définit la fonction suivante :

```
def parcoursl(arbre: ArbreBinaire) -> list[A]:
    """Renvoie la liste des étiquettes de `arbre`
    (l'ordre est celui d'un parcours en largeur)
    Précondition : aucune
    """
    res = []
    file = ApQueue()
    file.enqueue(arbre)
    while not file.is_empty():
        a_traiter = file.dequeue()
        if not a_traiter.est_vide():
            res.append(a_traiter.etiquette())
            file.enqueue(a_traiter.gauche())
            file.enqueue(a_traiter.droit())
    return res
```

2. Donnez la valeur des variables locales `file` et `res` à la fin de chaque itération lors de l'appel `parcoursl(exemple_td)`. De quel parcours s'agit-il ?

On définit maintenant :

```
def parcoursp(arbre: ArbreBinaire) -> list[A]:
    """Renvoie la liste des étiquettes de `arbre`
    (l'ordre est celui d'un parcours en profondeur préfixe)
    Précondition : aucune
    """
    res = []
    pile = ApStack()
    pile.push(arbre)
    while not pile.is_empty():
        a_traiter = pile.pop()
        if not a_traiter.est_vide():
            res.append(a_traiter.etiquette())
            pile.push(a_traiter.droit())
            pile.push(a_traiter.gauche())
    return res
```

3. Donnez la valeur des variables locales `pile` et `res` à la fin de chaque itération lors de l'appel `parcoursp(exemple_td)`. De quel parcours s'agit-il ?

On définit ensuite :

```
def parcoursi(arbre: ArbreBinaire) -> list[A]:
    """Renvoie la liste des étiquettes de `arbre`
```

```
    (l'ordre est celui d'un parcours en profondeur infixe)
    Précondition : aucune
    """
    res = []
    pile = ApStack()
    courant = arbre
    while not pile.is_empty() or not courant.est_vide():
        if not courant.est_vide():
            pile.push(courant)
            courant = courant.gauche()
        else:
            courant = pile.pop()
            res.append(courant.etiquette())
            courant = courant.droit()
    return res
```

4. Donnez la valeur des variables locales `pile`, `res` et `courant` à la fin de chaque itération lors de l'appel `parcoursi(exemple_td)`. De quel parcours s'agit-il ?

On définit enfin :

```
def parcourss(arbre: ArbreBinaire) -> list[A]:
    """Renvoie la liste des étiquettes de `arbre`
    (l'ordre est celui d'un parcours en profondeur suffixe)
    Précondition : aucune
    """
    res = []
    pile = ApStack()
    courant = arbre
    dernier_traite = ArbreBinaire()
    while not pile.is_empty() or not courant.est_vide():
        if not courant.est_vide():
            pile.push(courant)
            courant = courant.gauche()
        else:
            a_traiter = pile.pop()
            droit = a_traiter.droit()
            if not droit.est_vide() and dernier_traite != droit:
                pile.push(a_traiter)
                courant = droit
            else:
                res.append(a_traiter.etiquette())
                dernier_traite = a_traiter
    return res
```

5. Donnez la valeur des variables locales `pile`, `res`, `courant` et `dernier_traite` à la fin de chaque itération lors de l'appel `parcourss(exemple_td)`. De quel parcours s'agit-il ?

8 Implanter Une Pile avec deux Files

6. Comment implanter une structure de pile en utilisant dans les méthodes uniquement deux files et une variable temporaire ?

9 Implanter une File avec deux Piles.

7. Comment implanter un structure de file en utilisant dans les méthodes uniquement deux piles et une variable temporaire ?