TD 3 : AP

1. Donnez les états successifs de la pile dans la séquence d'instructions suivante.

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

[] -> [1] -> [1,7] -> [1,7,5] -> [1,7] -> [1] -> []

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

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

pop

-> vide la pile en affichant l'élément dépilé.

3. Indiquez ce que fait la fonction suivante.

```
def what_do_i_do(st):
    n = 0
    while not st.is_empty():
        n = n + st.pop()
    st.push(n)
```

-> fait la somme des éléments de la pile en la vidant. et stocke le résultat dans la pile.

Expressions postfixées

1. Écrivez sous forme postfixée les expressions arithmétiques suivantes. Dans cette notation, les opérateurs sont écrits après les opérandes.
   1. $(a + b) \times c$;
   2. $\frac{a}{b} \times c - \frac{e}{d \times e} - (f - g)$.
2. Indiquez les états successifs de la pile lors de l'évaluation de ces deux expressions.

1. b, a, +, c, x

2. b, a, /, c, x, d, e, x, c, /, -, g, f, -, -.

b. x, c, +, a, b -> c, +, a, b ; x -> +, a, b ; xc -> (+) xc -> (a+b) x c.

3 Inversion des éléments d'une liste

On suppose donnée une liste ℓ. Comment à l'aide d'une pile, inverser l'ordre des éléments de ℓ ?

-> Pile().

-> Liste

-> tant que liste non vide alors
retirer premier élément et l'empiler.

-> tant que pile non vide alors
dépiler Pile et l'ajouter à liste.

4 Impression des éléments d'une pile

On veut réaliser une procédure `stack_print` qui imprime (sur la sortie standard) tous les éléments d'une pile d'entiers. Une fois son travail d'impression réalisé, cette procédure de type ne doit pas avoir modifié la pile.

On va étudier deux affichages différents qui seront illustrés 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 un affichage vertical, un entier par ligne, dans l'ordre du sommet vers le bas de pile. Par exemple

```
>>> stack_print1(st)
4
3
2
1
```

```
def stack_print(st) -> None:
    aux = ApStack()
    while not st.is_empty():
        tmp = st.pop
        print(tmp)
        aux.push(tmp)
    while not aux.is_empty():
        stack.push(aux.pop()).
```

```
def stack_print_2(st) -> None:
    aux = ApStack()
    while not st.is_empty():
        aux.push(st.pop)
    while not aux.is_empty():
        tmp = aux.pop()
        print(tmp, end=' ')
        st.push(tmp)
```

```
>>> st.top()
4
```

le sommet de la pile `st` est l'entier 4.

2. Dans un deuxième temps on demande un affichage horizontal, le bas de pile étant à gauche et le sommet à droite. La même pile que précédemment est imprimée comme on le voit ci-dessous.

```
>>> stack_print2(st)
[1 2 3 4
>>> st.top()
4
```

2. Dans cette question on suppose que les seules parenthèses sont les parenthèses (et).

Réalisez un prédicat qui prend un texte en paramètre et renvoie~:

```
def bon_parenthese(txt)->bool:
    aux = ApStack().
    for par in txt:
        match par:
            case '(':
                aux.push(par)
            case:
                aux.pop()
    return aux.is_empty().
```

3. On suppose dans cette question que le texte peut contenir trois types de parenthèses~: `()`, `[]` et `{}`.

Reprogrammez le prédicat précédent.

```
def bon_parenthese_2(txt)->bool:
    aux = {}
    for par in txt:
        if est_ouvrante(par):
            if par in aux:
                aux[par].push(par)
            else:
                aux[par] = ApStack()
                aux[par].push(par).
        else:                                  * if aux[get_ouvrante(par)].is_empty(): return False.
            aux[get_ouvrante(par)].pop()
    return all(st.is_empty for st in aux.values())
```

6 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'
```

```
def to_bin(n:int)->str:
    st=ApStack()
    Res=""
    while n !=0:
        st.push(n%2)
        n=n//2
    while not st.is_empty():
        Res+=str(st.pop())
    Return Res.
```

7 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 ?
3. Reprendre la première question en supposant les deux piles triées dans l'ordre croissant du fond vers le sommet. La liste à construire doit toujours être triée dans l'ordre croissant.

```
def stack_merge(st1:ApStack,st2:ApStack)->ApStack:
    Res=ApStack()
    while not st1.is_empty OR not st2.is_empty():
        if st1.top()<st2.top():
            Res.push(st1.pop())
        else:
            Res.push(st2.pop())
    while not st1.is_empty():
        Res.push(st1.pop())
    while not st2.is_empty():
        Res.push(st2.pop())
    return Res.
```