Ce TP se base sur la classe `pile` définie ci–dessous.

```
class pile:

    def __init__(self):
        self.lst=[]

    def empty(self):
        return self.lst==[]

    def push(self,x):
        self.lst.append(x)

    def pop(self):
        if self.empty():
            raise ValueError("vide")
        return self.lst.pop()
```

## – I –

1. Écrire une méthode d'entête `depth(self)` renvoyant la profondeur (*i.e* le nombre d'éléments) d'une pile donnée.
2. Écrire une méthode d'entête `display(self)` affichant une pile sous forme... de pile. Plus précisément, un appel de `a.print()` doit afficher les éléments de `a` les uns au–dessus des autres, avec le dernier ajouté au sommet.
3. (a) Écrire une méthode d'entête `pileup(self,L)` prenant en argument une liste `L` et ajoutant ses éléments sur le dessus de la pile, en commençant par le premier.
   (b) Modifier la méthode `pileup` pour qu'elle renvoie une erreur de type `TypeError` si `L` n'est pas une liste. On rappelle que si `a` est un objet `python`, un appel à `type(a)` renvoie le type de `a` (essayer sur des exemples).
4. Écrire une méthode d'entête `clone(self)` créant une copie d'une pile, de façon à ce qui si `a` est une pile, exécuter `b=a.clone()` créé une nouvelle pile `b` identique à `a`.

## – II –

1. Écrire une fonction d'entête `equals(a,b)` déterminant si deux piles `a` et `b` sont égales.
2. Écrire une fonction d'entête `lookfor(x,a)` déterminant si un élément `x` appartient à une pile `a` **sans modifier** `a`.
3. Écrire une fonction d'entête `popsecond(a)` dépilant (en le renvoyant) le deuxième élément de la pile `a`.
4. Écrire une fonction d'entête `bottomtop(a)` échangeant le sommet et la base de la pile `a`.
5. Réécrire les méthodes `depth` et `display` de façon "purement pilesque".