```
# ======================================================== #
# UNE IMPLEMENTATION ORIENTEE OBJETS DE LA NOTION DE PILE #
# ======================================================== #

class stack:
    """Une classe permettant d'effectuer diverses manipulations sur les piles.
    Version du 04 octobre 2016.
    Rappelons que, fondamentalement, nous utilisons la structure de liste
    pour définir notre notion de pile ! Notre classe comporte ainsi un seul
    attribut (appelé "liste") et correspondant à la liste représentant la
    pile.
    """

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

    # On représente ...
    # Rappelons que la méthode __repr__ doit renvoyer une chaîne de caractères.

    def __repr__(self):
        if self.liste != []:
            # Le sommet est ... en haut !!!
            S = 'Sommet'
            for i in range(len(self.liste)):
                S += '\n*** ' + str(self.liste[i]) + ' ***'
        else:
            S = 'La pile est vide !'
        return(S)

    """On définit maintenant les méthodes correspondant aux manipulationx de
    base (cf. le cahier des charges) des piles.
    """

    # La pile est-elle vide ?

    def isempty(self):
        return(self.liste == [])

    # Ajout d'un élément au sommet

    def push(self,e):
        self.liste.insert(0,e)

    # Suppression du sommet courant

    def pop(self):
        del self.liste[0]

    # Obtention du sommet. Celui-ci n'est pas détruit.

    def peek(self):
        return(self.liste[0])

    """On définit maintenant les méthodes plus complexes demandées dans le TD.
    """

    # Copie de la pile

    def copy(self):
        s2 = stack()
        while not self.isempty():
            s2.push(self.peek())
            self.pop()
        s_copy = stack()
        while not s2.isempty():
            self.push(s2.peek())
            s_copy.push(s2.peek())
```

```
            s2.pop()
        del s2
        return(s_copy)

    # Inversion de la pile

    def reverse(self):
        s2 = self.copy()
        rs = stack()
        while not s2.isempty():
            rs.push(s2.peek())
            s2.pop()
        del s2
        return(rs)

    # Permutation circulaire (acte 1)

    def circperm(self,n):
        # REMARQUE : ce code NE TIENT PAS compte du fait que n peut être
        # supérieur à la taille de la pile...
        # self est modifiée en place.

        # Pile auxiliaire
        s2 = stack()
        # Boucle principale
        for i in range(n):
            # On place le sommet de s dans x et on dépile le reste de self dans s2
            x = self.peek()
            self.pop()
            while not self.isempty():
                s2.push(self.peek())
                self.pop()
            # self est vide, on y place x
            self.push(x)
            # on dépile s2 dans self
            while not s2.isempty():
                self.push(s2.peek())
                s2.pop()
        # Destruction de la pile auxiliaire
        del s2

    # Permutation circulaire (acte 2)

    def circperm2(self,k):
        # Pile auxiliaire
        s2 = stack()
        # On récupère le sommet de self
        x = self.peek()
        self.pop()
        # On dépile k-1 éléments de self dans s2
        for i in range(k-1):
            s2.push(self.peek())
            self.pop()
        # On replace x x dans self
        self.push(x)
        # On dépile s2 dans self (on peut aussi utiliser une boucle while...)
        for i in range(k-1):
            self.push(s2.peek())
            s2.pop()
        # Destruction de la pile auxiliaire
        del s2

# Pour tester...

s=stack()
s.push(2)
s.push(5)
```

```
s.push(10)
s.push(50)
s.push(500)
s.push(25000)
s.push(55000)
s_copy = s.copy()
print("     Pile d\'origine...")
print(s)
print("     Copie...")
print(s_copy)
s_rev = s.reverse()
print("     Pile d\'origine...")
print(s)
print("     Pile inversée...")
print(s_rev)
print("     Permutation circulaire (acte 1)...")
print(s)
s.circperm(2)
print(s)
print("     Permutation circulaire (acte 2)...")
print(s_copy)
s_copy.circperm2(3)
print(s_copy)
```