Implémentations de piles

1 Piles à capacité finie

```
class Pile:
    "classe pile à capacité finie"

    def __init__(self, N=50):
        """initialisation d'une pile vide de capacité finie N"""
        self.T = [None for i in range(N)]
        self.taille = 0
        self.capacite = N

    def est_vide(self):
        """teste si la pile est vide"""
        return self.taille == 0

    def est_pleine(self):
        """teste si la pile est pleine"""
        return self.taille == self.capacite

    def depile(self):
        assert not self.est_vide(), 'Pile vide'
        n = self.taille
        self.taille -= 1
        return self.T[n - 1]

    def empile(self, x):
        "empile x sur la pile"
        assert not self.est_pleine(), 'Pile pleine'
        self.taille += 1
        self.T[self.taille - 1] = x

    def sommet(self):
        "renvoie le sommet de la pile"
        assert self.taille == 0, 'Pile vide'
        return self.T[self.taille - 1]

    def __repr__(self):
        "Pour l'affichage"
        a_afficher = ['T                  T']
        if self.taille != 0:
            for e in self.T[self.taille - 1::-1]:
                chaine = str(e)
                if len(chaine) > 15:
                    chaine = chaine[:10] + '[...]'
                a_afficher.append('| {:^15} |'.format(chaine))
        a_afficher.append('\_________________/')
        return "\n".join(a_afficher)
```

2 Piles non bornées

```
class Pile:
    "classe pile non bornée"

    def __init__(self):
        """initialisation d'une pile vide"""
        self.L = []

    def est_vide(self):
        """teste si la pile est vide"""
        return self.L == []

    def depile(self):
        assert not self.est_vide(), 'Pile vide'
        return self.L.pop()

    def empile(self, x):
        "empile x sur la pile"
        self.L.append(x)

    def sommet(self):
        "renvoie le sommet de la pile"
        assert not self.est_vide(), 'Pile vide'
        return self.L[-1]

    def __repr__(self):
        """Pour l'affichage"""
        a_afficher = ['T                T']
        for e in self.L[::-1]:
            chaine = str(e)
            if len(chaine) > 15:
                chaine = chaine[:10] + '[...]'
            a_afficher.append('| {:^15} |'.format(chaine))
        a_afficher.append('\_________________/')
        return "\n".join(a_afficher)
```