Exercice 1 : Inversion pile

En utilisant les fonctions *empile*( ), *Pile_vide*() et *desempile*( ) du cours, proposer une fonction *inverser_pile*( ) qui prend pour argument une pile *p* et qui renvoie une autre pile dont les éléments sont les éléments de *p* dans l'ordre inverse, *p* n'est pas modifiée.

```
def empile(p,e):
    p.append(e)


def desempile(p):
    N=len(p)
    dernier=p[N-1]
    p.pop()
    return dernier

def PileVide(p):
    N=len(p)
    if N ==0 :
        return True
    else :
        return False

def inverser_pile(p):
    inter_p=[]
    inverse_p=[]
    while PileVide(p) != True:
        val =desempile(p)
        empile(inter_p,val)
        empile(inverse_p,val)
    while PileVide(inter_p) != True:
        empile(p,desempile(inter_p))
    return inverse_p
print(inverser_pile([0,1,2,3,4]))
```

.

Exercice 2 : Dépiler plusieurs éléments

Dans cet exercice, vous devez utiliser les fonctions $empile(\ ), desempile(\ ), PileVide(\ )$ pour élaborer les fonctions demandées :

Ecrire une fonction $depileK(\ )$ qui prend pour argument une pile p et un entier K et qui désempile p des K derniers éléments de p s'ils existent ou qui désempile entièrement p si $len(p) < K$. La fonction retourne la pile dépilée (en partie ou en totalité).

```
#1e méthode : on teste à chaque fois si p est vide
def depileK(p,K):
    for i in range(K):
        if not PileVide(p):
            desempile(p)
        else:
            return p
    return p
#2e méthode : on teste si K entraîne une liste vide
def desempileK(p,K):
    if len(p)>K:
        for i in range(K):
            desempile(p)
    else :
        for i in range(len(p)):
            desempile(p)
    return p
#3e méthode : idem avec une boucle while
def depileK(p,K):
    if len(p)<=K:
        return []
    else :
        i=K
        while i>0:
            desempile(p)
            i=i-1
        return p
#4e méthode : idem à la 1e méthode mais avec while
def desempileL2(p,K):
    i=0
    while i<K and not(PileVide(p)):
        desempile(p)
        i=i+1
    return p
print(desempileL2([0,1,2,3,4,5],7))
```

Exercice 3 : Dépiler jusqu'à un élément E

Dans cet exercice, vous devez utiliser les fonctions $empile(\ ), desempile(\ ), PileVide(\ )$ pour élaborer la fonction demandée :

Ecrire une fonction $depileE(\ )$ qui prend pour argument une pile p et un élément E et qui désempile p tant que l'élément E n'est pas rencontré ou que p n'est pas vide. La fonction retourne la pile dépilée avec E inclus.

```
def empile(p,e):
    p.append(e)
    return p

def desempile(p):
    N=len(p)
    dernier=p[N-1]
    p.pop()
    return dernier
def PileVide(p):
    N=len(p)
    if N ==0 :
        return True
    else :
        return False
def desempileE(p,E):
    if PileVide(p) != True:
        elt=p[-1]
        while not PileVide(p) and elt !=E:
            desempile(p)
            elt=p[-1]
    return p
print(desempileE([0,1,2,3,4,5,6,7,8,9],7))

def desempileE2(p,E):
    while not PileVide(p):
        elt=desempile(p)
        if elt==E:
            empile(p,E)#pour remettre l'élément
            return p
    return p
```

Exercice 4 : Permutation

Dans cet exercice, vous devez utiliser les fonctions *empile*( ), *desempile*( ) (éventuellement *PileVide*()) pour élaborer la fonction demandée :

Ecrire une fonction *permut*( ) qui prend pour argument une pile p et qui permute le dernier élément avec l'avant dernier s'ils existent. La fonction retourne la pile ainsi modifiée.

```
def empile(p,e):
    p.append(e)
    return p

def desempile(p):
    N=len(p)
    dernier=p[N-1]
    p.pop()
    return dernier
def PileVide(p):
    N=len(p)
    if N ==0 :
        return True
    else :
        return False
def permut1(p):
    if len(p)>1:
        val1=desempile(p)
        val2=desempile(p)
        empile(p,val1)
        empile(p,val2)
    return p
print(permut1([0,1,2,3,4]))



def permut2(p):
    if not(PileVide(p)) :
        sommet = desempile(p)
    if not(PileVide(p)) :#a noter que if est nécessaire
(elif ignoré si le 1e est exécuté)
        nouveau_sommet=desempile(p)
        empile(p,sommet)
        empile(p,nouveau_sommet)
    return p

print(permut2([0,1,2,3,4]))
```

Exercice 5 : Permutation circulaire

Dans cet exercice, vous devez utiliser les fonctions $empile(\ ), desempile(\ ), PileVide(\ )$ pour élaborer les fonctions demandées :

Ecrire une fonction $permut\_circ(p,k)$ qui réalise une permutation circulaire sur les k derniers éléments d'une pile p.

```
def permute_circ(p,k):
    p_inter=[]
    if len(p)>=k:#assure que la permutation aboutisse
        der=desempile(p)
        for i in range(k-1):
            empile(p_inter,desempile(p))
        empile(p,der)
        while PileVide(p_inter)!=True:
            empile(p,desempile(p_inter))
    return p
```

```
def permutcirc_a(p,k):
    if not(PileVide(p)) :
        sommet=desempile(p)
    i=1
    inter_p=[]
    while i<k and not(PileVide(p)):
        empile(inter_p,desempile(p))
        i=i+1
    empile(p,sommet)
    while  not(PileVide(inter_p)):
        empile(p,desempile(inter_p))
    return p
print(permutcirc_a([0,1,2,3,4],3))

def permutcirc_b(p,k):
    if not(PileVide(p)) :
        sommet=desempile(p)
    inter_p=[]
    while k>1 and not(PileVide(p)):
        empile(inter_p,desempile(p))
        k=k-1
    empile(p,sommet)
    while  not(PileVide(inter_p)):
        empile(p,desempile(inter_p))
    return p
print(permutcirc_b([0,1,2,3,4],3))
def permut_circ(p,k):
    if PileVide(p)!= True :
        sommet=desempile(p)
    i=k-1
    L_inter=[]
    while i>0 and not PileVide(p) :
        der=desempile(p)
        empile(L_inter,der)
        i=i-1
    empile(p,sommet)
    while not PileVide(L_inter):
        der=desempile(L_inter)
        empile(p,der)
    return p
```

Exercice 6 : Test de la mise en parenthèse d'une expression mathématique

Dans cet exercice, vous devez utiliser les fonctions $empile(\ ), desempile(\ ), PileVide(\ )$ pour élaborer la fonction demandée :

Ecrire un algorithme utilisant une pile et permettant de savoir si le nombre de parenthèses ouvrantes est égale au nombre de parenthèses fermantes. L'expression mathématique à tester est donnée dans une liste de chaîne de caractères. Par exemple, l'expression $3 \times (4 + 2)$ sera donnée sous la forme ["3","*","(","4","+","2",")"]. L'algorithme renvoie *True*, si la mise en paranthèse est correcte, *False*, sinon.

L'idée est donc d'empiler à chaque parenthèse ouvrante rencontrée et de dépiler si une parenthèse fermante est rencontrée (en lisant l'expression de gauche à droite). Si l'écriture est bonne la pile est vide.

```
def parenthese_1(chaine):#problème si il y a plus de ) que de (
    pile=[]
    for i in chaine:
        val=i
        if val =="(":
            empile(pile,val)
        elif val ==")":
            desempile(pile)
    if PileVide(pile)==True:
        print("bon nombre de parenthèse")
    else :
        print("pas bon")
parenthese_1("3((*4*'*(3))")

def parenthese_2(chaine):
    pile=[]
    for i in chaine:
        val=i
        if val =="(":
            empile(pile,val)
        elif val ==")" :
            if not PileVide(pile):
                desempile(pile)
            else :
                return print("pb de parenthèse")#on stoppe la fonction
    if PileVide(pile)==True:
        print("bon nombre de parenthèse")
    else :
        print("pas bon")
parenthese_2("3((*4*'*(3))))")
```

Exercice 7 : Calculette polonaise

Dans cet exercice, vous devez utiliser les fonctions $empile(\ ), desempile(\ ), PileVide(\ )$ pour élaborer la fonction demandée :

L'écriture polonaise inversée ne nécessite pas de parenthèse, elle a été autrefois utilisée dans certaines calculatrices.

Principe :

- Quand on rencontre un nombre, on ne fait rien
- Quand on rencontre un opérateur (+,-,*,/), on l'applique aux deux précédents nombres

$$7 + 6 \rightarrow 7\ 6 +$$

$$(10 + 5) * 3 \rightarrow 10\ 5 +\ 3 *$$

$$10 + 2 * 3 \rightarrow 10\ 2\ 3 *\ +$$

$$(2 + 8) * (6 + 11) \rightarrow 8\ 2 +\ 6\ 11 +\ *$$

Ecrire une fonction :

- qui prend en argument une liste contenant les éléments d'un calcul en écriture polonaise inversée. Par exemple $[2,3,"+",4,"*"]$ pour $(2 + 3) * 4$
- qui utilise une pile initialement vide (une liste vide).
- qui, pour chaque élément lue de la liste de gauche à droite :
  - → si l'élément est un nombre, alors ce nombre est rajouter à la pile avec la fonction empile( )
  - → si l'élément est une opération (*,/,+,-,), alors la pile est dépilée deux fois pour réaliser cette opération
- A la fin la pile ne contient plus qu'un seul élément, c'est le résultat du calcul

```
def polonaise(chaine):
    pile=[]
    for i in chaine:
        if i=="*":
            a=desempile(pile)
            b=desempile(pile)
            empile(pile,a*b)
        elif i=="/":
            a=desempile(pile)
            b=desempile(pile)
            empile(pile,b/a)
        elif i=="+":
            a=desempile(pile)
            b=desempile(pile)
            empile(pile,a+b)
        elif i=="-":
            a=desempile(pile)
            b=desempile(pile)
            empile(pile,b-a)
        else :
            empile(pile,i)
    return pile
```

Exercice 8 :

Dans cet exercice, vous devez utiliser les fonctions $empile(\ ), desempile(\ ), PileVide(\ )$ pour élaborer la fonction demandée :

1) On considère une pile p (ici représentée par une liste de nombres $[x_1, x_2, , \ldots, x_n]$) triée de manière décroissante. Proposer une fonction *insert* qui prend en argument une pile triée et un nombre x et qui insert x dans cette pile pour en conserver son tri décroissant.
2) Ecrire une fonction *tri* qui insert les éléments de la liste à trier dans une pile à l'aide de la fonction *insertion*. Lorsque tous les éléments ont été insérés, ils sont retirés de la pile et placés un à un à la bonne place dans la liste qui est alors triée dans l'ordre décroissant.

```
def insertion(p,x):
    """on insert récursivement x dans p décroissant"""
    if PileVide(p) or x<p[len(p)-1]:
        p.append(x)
    else :
        y=desempile(p)
        insertion(p,x)
        empile(p,y)
p=[10,9,8,7,6,4,3,2,1,0]
insertion(p,5)
print(p)


def tri(L):
    p=[]
    for i in range(len(L)):
        x=desempile(L)
        empile(p,x)
    while PileVide(p)==False:
        L.append(p.pop())
L=[10,9,8,7,6,5,4,3,2,1,0]
tri(L)
print(L)
```