TP 1 : Piles – Corrigé

Echauffement

1.
```
def pop2(p):
    x=pop(p)  #dernier
    y=pop(p)  #avant-dernier
    push(x,p) #on remet le dernier
    return y  #on renvoie l'avant-dernier
```

2.
```
def echange_parite(p):
    x=pop(p)
    y=pop(p)
    if x%2==y%2:  #même parité : on échange
        push(x,p)
        push(y,p)
    else:         #sinon : on remet dans le même ordre
        push(y,p)
        push(x,p)
```

Exercice 1 : Mots bien parenthésés

```
def parentheses(s):
    p=creer_pile()           #stocke les position des (
    for i in range(len(s)):
        if s[i]=='(':        #on empile la position de ()
            p.append(i)
        elif s[i]==')':      #quand on rencontre )
            if est_vide(p):  #si la pile est vide :
                return False #il y a trop de )
            else:
                j=pop(p)     #sinon on récupère la position de
                print(j,i)   #la ( qui correpond et on renvoie le couple
    return est_vide(p)       #si la pile est vide au final
                             #c'est bien parenthésé
```

Exercice 2 : Mélange de cartes

```
def melange(p1,p2):
    p3=creer_pile()
    #tant qu'il reste des choses dans les 2 piles
    while not est_vide(p1) and not est_vide(p2):
        #on choisit au hasard entre les 2
        n=rd.randint(1,2)
        #on prend dans celle qui est choisie
        #pour mettre dans la nouvelle pile
        if n==1:
            push(pop(p1),p3)
        else:
            push(pop(p2),p3)
    #une des 2 piles est vide
    #on vide ce qui reste
    while not est_vide(p1):
        push(pop(p1),p3)
    while not est_vide(p2):
        push(pop(p2),p3)
    return p3
```

Exercice 3 : Notation polonaise inverse

On parcourt la liste. Quand on rencontre un nombre, on l'empile. Quand on rencontre une opération, on dépile les deux derniers nombres, on leur applique l'opération et on empile le résultat.

A la fin du parcours, la pile contient uniquement la valeur du résultat

```
def eval_NPI(L):
    p=creer_pile()
    for e in L:
        if e=='+':
            x=pop(p)
            y=pop(p)
            push(x+y,p)
        elif e=='*':
            x=pop(p)
            y=pop(p)
            push(x*y,p)
        else:
            push(e,p)
    return pop(p)
```

Exercice 4 : Exploration d'un labyrinthe

1. On vérifie si les cases qui entourent (i, j) existent bien (*i.e.* on vérifie qu'on ne sort pas du tableau), et on rajoute à la liste toutes les positions des cases voisines qui ne sont pas des murs.

```
def casesVoisines(laby,i,j):
    n=len(laby)
    L=[]
    if i>0:
        if laby[i-1][j]!=0:
            L.append((i-1,j))
    if j>0:
        if laby[i][j-1]!=0:
            L.append((i,j-1))
    if i<n-1:
        if laby[i+1][j]!=0:
            L.append((i+1,j))
    if j<n-1:
        if laby[i][j+1]!=0:
            L.append((i,j+1))
    return L
```

2. *Parcours du labyrinthe*

```
def parcours(laby):
    n=len(laby)
    #initialisation : pile vide
    p=creer_pile()
    #on marque la case d'entrée et on l'empile
    laby[0][1]=2
    push((0,1),p)
    #itération
    while not est_vide(p):
        #on dépile une position marquée
        i,j=pop(p)
        #on cherche ses voisins
        L=casesVoisines(laby,i,j)
        #on marque tous les voisins non marqués
        for x in L:
            a,b=x
            if laby[a][b]!=2:
                laby[a][b]=2
                push((a,b),p)
    #on vérifie que la case de sortie est marquée
    #si c'est le cas, on peut sortir du labyrinthe
    return laby[n-1][n-2]==2
```

Exercice 5 : s'il vous reste du temps

1.
```
def enlever(x,p):
    p1=creer_pile()
    while not est_vide(p):
        a=pop(p)
        if a!=x:
            push(a,p1)
    while not est_vide(p1):
        push(pop(p1),p)
```

2.
```
def topbottom(p):
    p1=creer_pile()
    top=pop(p)
    while not est_vide(p):
        push(pop(p),p1)
    bottom=pop(p1)
    push(top,p)
    while not est_vide(p1):
        push(pop(p1),p)
    push(bottom,p)
```