Exo 1 :

```
p=creer_pile() # on sauvegarde la pile dans une variable
empiler(p,'A')
empiler(p,'A')
empiler(p,'B')
empiler(p,'C')
empiler(p,'D')
empiler(p,'B')
depiler(p)
depiler(p)
sommet(p)      # renvoie C
depiler(p)
est_vide(p)    # on verifie si p est vide
```

Exo 2 :

- un historique entraîne généralement l'utilisation d'une pile
- un listing entraîne généralement l'utilisation d'un tableau
- cela dépend du fonctionnement de l'élève mais vraissemblablement un tableau
- les points de vie d'un personnage utilise plutôt une pile

Exo 3 : Si on considère que la variable `pileA` est la pile contenant les actions faites jusqu'à présent, on peut proposer :

```
pileA = creer_pile()

def action(pA,action):
    empiler(pA, action)

def UNDO(p):
    depiler(p)
```

```
pileA = creer_pile()
pileB = creer_pile()
def faireAction(pA,action):
    # nb : on pourrait aussi vider pileB
    empiler(pA, action)
def UNDO(pA,pB):
    empiler(pB, depiler(pA))
def REDO(pA,pB):
    empiler(pA, depiler(pB))
```

1 Utilisation papier d'une pile

Exo 4 : Bien que l'attendu était des algorithmes, ici les fonctions ont directement été écrites en langage python.

1. dépile k éléments d'une pile (si elle contient moins de k éléments, alors elle sera vidée)

```
def depileA(p,k):
    for i in range(0, min(k,taille(p))):
        depiler(p)
    return p
```

2. inverse toute la pile.

```
def inverseB(p):
    q=creer_pile()
    while taille(p)>0 :
        empiler(q,depiler(p))
    return q
```

3. inverse le sommet et le pénultième élément de la pile,

```
def inverseC(p):
    if taille(p)<=1 : return p
    sommet=depiler(p)
    psommet=depiler(p)
    empiler(p,sommet)
    empiler(p,psommet)
    return p
```

4. inverse le sommet et le premier élément de la pile,

```
def inverseD(p):
    if taille(p)<=1 : return p
    q=creer_pile()
    sommet=depiler(p)
    while taille(p)>1 :
        empiler(q,depiler(p))
    base=depiler(p)
    empiler(p,sommet)
    while taille(q)>0 :
        empiler(p,depiler(q))
    empiler(p,base)
    return p
```

Exo 5 :

```
def verif_parenthese(tab):
    p=creer_pile()
    for e in tab:
        if e=='(' :
            empiler(p,1)
        if e==')' :
            if taille(p)==0:
                print("Mal parenthese")
                return
            depiler(p)
    if taille(p) > 0:
        print("Mal parenthese")
        return
    print("Bien parenthese")
```

→ vérification (ci-dessous, seuls les deux premiers sont bien parenthèsés).

```
verif_parenthese([4,'*','(',1,'+',2,')'])     # list("4*(1+2)")
verif_parenthese(list("(4)*((1)+2)"))
verif_parenthese(['(',4,'*',1,'+',2,'('])     # list("(4*1+2(")
verif_parenthese(list("4*)1+2)"))
verif_parenthese(list("4*)1+2("))
verif_parenthese(list("(4))1+2("))
```

⚠ list("12+3") renvoie ['1','2','+','3']. On ne l'utilisera qu'avec des chiffres et non des nombres.

2 Réalisation d'une pile

Exo 6 : On peut proposer

```
def creer_pileF(t):
    p  = (t+1)*[None] # la pile est un tableau de taille determinee
    p[0]= 0           # la 1ere case possede l'indice du sommet de la pile
    return p
def depiler(p):
    if p[0]==0 : return  # si la pile est vide, on ne renvoie rien
    p[0]-=1
    return p[p[0]+1]
def empiler(p, e):
    p[0]+=1
    p[p[0]]=e
def taille(p):
    return p[0]
def est_vide(p):
    return p[0]==0
def sommet(p):
    if p[0]==0: return   # si la pile est vide, on ne renvoie rien
    return p[p[0]]
```

3 Utilisation d'une pile

cf. fichier CPGE-Info-Piles-TP-corrige.py