Sujet 0

Exercice 1

1. Le code commence par créer une pile vide Q puis dépile tous les éléments de P jusqu'à ce que cette pile soit vide. Chacun des éléments est immédiatement empilé dans Q.

   On obtient donc la pile :

   –
   8
   5
   2
   4
   –

2. On propose le code suivant :

```
def hauteur_pile (P): Q =
    creer_pile_vide()
    n = 0 while not (est_vide(P)): n
    = n + 1

        x = depiler(P) empiler(Q, x)

    while not(est_vide(Q)): x =
        depiler(Q) empiler(P, x)

    return n
```

3. Si l'on cherche le maximum à partir de l'élément i, on peut déjà s'assurer que la pile compte au moins i éléments. Cette vérification n'est toutefois pas mentionnée dans le sujet.

   Ensuite on parcourt tous les éléments jusqu'au i-ième en stockant la position du maximum.

```
def max_pile(P, i): assert i <= hauteur_pile(P), "La pile compte moins de i éléments"

    # Initialisation

    rang = 1 # L'indice de l'élément en cours de traitement rang_maxi = 1 # le rang du

    maximum en cours maxi = depiler(P) # au début, le maximum est le premier élément ...
    empiler(P, maxi) # ... que l'on rempile immédiatement

    Q = creer_pile_vide() # une pile vide pour stocker les éléments traités

    # On lit tous les éléments jusqu'au i-ième pour trouver le maximum

    while rang <= i: x =
        depiler(P)
    if x > maxi:
        maxi = x rang_maxi = rang
    empiler(Q, x) rang += 1

# On reconstitue la pile P while
not(est_vide(Q)):
    empiler(P, depiler(Q)) return rang_maxi
```

4. On va dans un premier temps s'assurer que la pile P contient au moins j éléments. Comme dans la question précédente, cette vérification n'est pas explicitement mentionnée dans le sujet.

   Pour le traitement, on dépile les j éléments dans une nouvelle pile Q. On vide alors Q dans une troisième pile R. Vider R dans P a alors pour effet de retourner les valeurs comme souhaité.

```
def retourner(P, j): assert j <= hauteur_pile(P), "La pile compte moins de j éléments"

    # Initialisation

    Q = creer_pile_vide() # une pile vide pour vider P R = creer_pile_vide()

    # une pile vide pour vider Q rang = 1 # le rang de l'élément en cours de
    traitement

    # On dépile les j premiers éléments dans Q
    while rang <= j: empiler(Q,
        depiler(P)) rang += 1

    # On vide Q dans R
    while not(est_vide(Q)): empiler(R,
        depiler(Q))

    # On vide R dans P
    while not(est_vide(R)):
        empiler(P, depiler(R))

    # La fonction ne renvoie rien (en réalité None en python)
    # On peut tout aussi bien se passer de retour # ce qui aura le
    même effet lors de l'exécution return None
```

5. On s'autorise à réutiliser les fonctions des questions précédentes. Le pseudo-code est le suivant :

   Fonction tri_crepes(P) :
       Vérifier que P est non-vide et renvoyer une erreur si ce n'est pas le cas h = hauteur_pile(P)

       Pour i allant de 0 à h-1 (exclus) :
           # inutile de retourner la dernière crêpe en fin de boucle car c'est la plus petite rang_maxi = maxi_pile(P,h-i)
           # on cherche le rang de l'élément maximal parmi les h-i premiers
           retourner(P,rang_maxi) # On retourne le haut de la pile jusqu'à l'élément maximal retourner(P,h-i) # On
           retourne toute la pile jusqu'en bas

   En python cela donne :

```
def tri_crepes(P) : assert not est_vide(P), "Il n'y a pas de crêpes à trier !" h =
    hauteur_pile(P)

    for i in range(0, h-1): rang_maxi =

        max_pile(P, h-i) retourner(P,

        rang_maxi) retourner(P, h-i)

    # La fonction ne renvoie rien (en réalité None en python)
    # On peut tout aussi bien se passer de retour # ce qui aura le
    même effet lors de l'exécution return None
```