Alternative correction DS 1 INFORMATIQUE

2ème année – Novembre 2022

Exercice 1 :

Question 1 : **calculer_position(P, element)** renvoie la position d'un élément **appartenant** à P — **1,5 pt**

```
from op_pile import *        0.25pts  importation du module pile

def calculer_position(P, element):
    # Initialisation
    rang = 1    0.25pts  #La position de l'élément en cours de traitement
    Pcopie  =  copier(P)                          0.25pts
    while  element  != sommet(Pcopie):            0.25pts
                depiler(Pcopie)                   0.25pts
                rang += 1                         0.25pts
    return rang
```

Question 2 : — **1,75 pt**

```
2.1. def max_pile( P, n ) :
        assert n <= taille(P)      0.25pts #on peut aussi exiger n > 0
        # copier la pile
        Pc  =  copier(P)           0.25pts
        max = depiler(Pc)          0.25pts # au début, le maximum est le sommet

        while n-1 > 0 :            0.25pts
              elt = depiler(Pc)    0.25pts
              if elt > max: max = elt    0.25pts
              n -= 1                     0.25pts
        return max
2.2. def position_max_pile( P, n ) : (Bonus 0.50pts : hors-barème)
        return calculer_position(P, max_pile( P, n ))
```

Question 3 : — **1,75 pt**

```
def inverser(P , m ) :
        # assert m <= taille(P) (Bonus 0.25pts : hors-barème)
        from op_file import *          0.25pts
        F = creerFile()          0.25pts

        while m > 0 :                  0.25pts
             Enfiler(F,depiler(P))     0.25pts
             m -= 1                    0.25pts

        while not EstVide(F):          0.25pts
              empiler(P, Defiler(F))   0.25pts
```

Question 4 : — 1,25 pt

```
def trierPile(P) :
    n = taille(P)                          0.25pts
    for i in range(n-1):                   0.25pts #ou bien boucle inversée
        pos_max = position_max_pile(P, n-i)    0.25pts
        #if(pos_max != n-i) : (Bonus 0.25pts : optimisation hors-barème)
        inverser(P, pos_max)               0.25pts
        inverser(P, n-i)                   0.25pts
```

Question 5 : — 1,50 pt

```
def trierPile_rec(P, n) :
    if (n > 1) :                                  0.25pts
        pos_max = position_max_pile(P, n)         0.25pts
        #if(pos_max != n) : (Bonus 0.25pts : optimisation hors-barème)
        inverser(P, pos_max)                      0.25pts
        inverser(P, n)                            0.25pts
        trierPile_rec(P, n-1)                     0.50pts
```

Question 6 : — 1,25 pt

```
def  est_triée(P):
    if (taille(P) < 2) : return True              0.25pts
    elt = depiler(P)                              0.25pts
    if (elt > sommet(P)): is_sorted = False       0.25pts
    else :
        is_sorted = est_triée(P)   0.25pts
    empiler(P, elt)                0.25pts
    return is_sorted
```

Exercice 2:

Partie 1 :

```
def empiler(P, element):
    from time import perf_counter          0.25pts
    instant_courant = perf_counter()       0.50pts
    P[instant_courant] = element           0.75pts
```

1,50 pt

```
def taille(P):
    return len(P)
```

0,50pt

```
def estvide(P):
    return taille(P) == 0     #ou bien  len(P) == 0
```

0,50pt

```
def sommet(P) :
    assert taille(P) > 0 , "Pile Vide"   0.25pts
    cléMax = max( P.keys() )      0.50pts
    return P[ cléMax ]            0.75pts
```

1,50 pt

Partie 2 :

```
def positionsParenthèses(chaine) :
    lst = []                                                    0.25pt
    pile = creer_pile()                                         0.25pt

    for i in range(len(chaine)):                                0.25pt

        if  chaine[i] == '(':                                   0.25pt

                empiler(pile, i+1)                              0.25pt

        elif  chaine[i] == ')':                                 0.25pt

                lst.append( (depiler(pile), i+1) )              0.50pt

    return lst
```

2 pt

Exercice 3 :

```
def plus_proches_voisins(ensPts) :

    from math import dist  (Bonus 0.25pts hors-barème)

    # initialisation :          0.25 pts

    p_min =  ensPts.pop()

    q_min =  ensPts.pop()

    ensPts.add(p_min)

    ensPts.add(q_min)

            # Fin de l'initialisation

    for p in ensPts:            0.25pts

        for q in ensPts:        0.25pts

            # notation de la condition du if :      0.50pts

            if (p != q and dist(p_min, q_min) > dist(p,q)):

                            (p_min, q_min) = (p,q)    0.25pts

    return (p_min, q_min)      0.25pts
```

1,75 pt

def loadCities(nomFichier) : f = open(nomFichier , 'r') **0.25pts** listVilles = f.readlines() **0.25pts** f.close() **0.25pts** **Dcities** = dict() **0.25pts** for ligne in listVilles: **0.25pts** ville = ligne.strip().split(":") **0.25pts** **point = (float(**ville[1]**), float(**ville[2]**))** **0.50pts** **Dcities[point] =** ville[0] **0.25pts** return **Dcities**	**Total: 2,25pt**
def plus_proches_villes(Dcities): **ens = set(Dcities.keys()) #ou** boucle **ens.add(…)** **0.50pt** **(p_min, q_min) = plus_proches_voisins(**ens**)** 0.25pt **print(Dcities[p_min], Dcities[q_min])** 0.25pt **# on accepte aussi :** **# return (Dcities[p_min], Dcities[q_min])**	**1 pt**