Autres exercices pour pratiquer ITI1520 (Final 2017)

1. Quelle est la meilleure description de la fonction suivante ?

```
def maFonc(n) :
    for i in range (2, n):
        if n % i == 0:
            return False
        i = i + 1
    return True
```

(a) Retourne True si son paramètre n est premier et False sinon
(b) Retourne False si son paramètre n est premier et True sinon
(c) Retourne True si son paramètre n est pair et False sinon
(d) Retourne False si son paramètre n est pair et True sinon
(e) Aucune de ces réponses

2. Qu'affiche le programme suivant?

```
import math
def taille_format(x):
        unités = ['b', 'Kb', ',Mb', 'Gb', 'Tb']
        u = int(min(math.floor(math.log(x, 1000)), 4))
        # math.floor(x) Retourne  le plus grand entier inférieur ou égal à x.
        # math.log(x, 1000) retourne le logarithme de x à la base 1000, calculée comme log(x)/log(1000).
        return str(x/(1000**u))+unités[u]

print(taille_format (300047))
```

(a) 300.047Kb (b) 300.0Kb (c) 3.0Kb (d) 3b (e) 347.Kb

3. Qu'affiche le programme suivant ?

```
def maFonction(lst, debut =0, fin =-1):
    if fin == -1:
        fin = len(lst)
    mp, i = 0, 0
    while i < len(lst):
        if i >= debut and i <= fin and lst[i] > mp:
            mp = lst[i]
        i = i + 1
    return mp

serie = [9, 3, 6, 1, 7, 5, 4, 8, 2]
print(maFonction(serie, 2))
```

(a) erreur (b) 8 (c) 9
(d) 1 (e) Aucune de ces réponses

4. Quelle est la meilleure description de la fonction suivante ?

```
def maFonction(lst):
    '''(list of str)->str
    Précondition: lst n'est pas vide
    '''
    ch = ""
    i = len(lst)
    while i > 0 :
        i = i - 1
        ch = ch + lst[i]
    return ch
```

(a) Renvoie une chaine de caractères formée par les éléments de la liste *lst* dans l'ordre inverse
(b) Renvoie une chaine de caractères formée par les éléments de la liste *lst*
(c) Renvoie une liste de caractères formée par les éléments de la liste *lst* dans l'ordre inverse
(d) Renvoie une liste de caractères formée par les éléments de la liste *lst*
(e) Aucune de ces réponses

5. Quelle est la meilleure description de la fonction suivante ?

```
def maFonction(ch, car, deb=0):
    '''(str, str, int)->int
    Précondition: ch contient au moins un caractère
    '''

    i = deb
    while i < len(ch):
        if ch[i] == car:
            return i
        i = i + 1
    return -1
```

(a) renvoie le caractère car dans la chaîne ch s'il s'y trouve et -1 sinon.
(b) renvoie la première position du caractère car dans la chaîne ch s'il s'y trouve et -1 sinon. S'il y'en a plusieurs elle retourne sa première position.
(c) renvoie le nombre des occurrences du caractère car dans une chaîne s'il s'y trouve et -1 sinon.

(d) Erreur
(e) Aucune de ces réponses

6. Quelle est la meilleure description de la fonction suivante ?

```
def test(ch, car):
    '''(str, str, int)->int
    Précondition: ch contient au moins un caractère
    '''
     i, nc = 0, 0
    while i < len(ch):
        if ch[i] == car:
            nc = nc + 1
        i = i + 1
     return nc
```

(a) Renvoie le nombre des occurrences d'un caractère donné dans une chaîne s'il s'y trouve
(b) Renvoie l'indice du caractère car dans la chaîne ch s'il s'y trouve
(c) Renvoie le caractère car dans la chaîne ch s'il s'y trouve
(d) Erreur
(e) Aucune de ces réponses

7. Qu'affiche le programme suivant ?

```
t1 = [31, 28, 31, 30, 31, 30]
t2 = ['Janvier','Février','Mars','Avril','Mai','Juin']
c, d = 1, 0
while d < 6 :
   t2[c:c] = [t1[d]]
   c, d = c+2, d+1
print(t2)
```

(a) ['Janvier', 31, 'Février', 28, 'Mars', 31, 'Avril', 30, 'Mai', 31, 'Juin', 30]
(b) ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', *31, 28, 31, 30, 31, 30*]
(c) [*31, 28, 31, 30, 31, 30, 'Janvier','Février','Mars','Avril','Mai','Juin']*
(d) Erreur
(e) Aucune de ces réponses

8. Pour la fonction suivante, combien d'opérations sont exécutées dans la fonction, approximativement, quand n devient large (n est le nombre d'éléments dans la liste lst)?

```
def maFonction(lst):
    '''(list de int)->None
      Précondition: ls une matrice carrée qui n'est pas vide
    '''

    for i in range(len(lst)):
        for j in  range(len(lst)):
            if i==j:
                lst[i][j] = 0
# Afficher les indices des éléments restés à 1 (on ignore l'élément 0) :
    for i in  range(len(lst)):
        if lst[i]:
            print(i, end ="")
```

(a) $O(\log_2 n)$ (b) $O(n)$ (c) $O(n \log_2 n)$ (d) $O(n^2)$ (e) $O(n^3)$

9. Que renvoie la fonction suivante ?

```
def maFonction():
'''None->(list of int)
    '''

    lst = [1]*10
    for i in range(2,len(lst)):
        for j in range(i*2, len(lst), i):
            lst[j] = 0
    return lst
```

(a) [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
(b) [1, 1, 1, 1, 0, 1, 0, 1, 0, 0]
(c) [1, 1, 1, 1, 0, 1, 0, 1, 0, 1]
(d) [1, 1, 0, 1, 0, 1, 0, 1, 0, 0]
(e) Erreur
(f) Aucune de ces réponses

10. Considérant la classe `Point` suivante :

```
class Point(object):
   def __init__(self, xcoord=0, ycoord=0):
      self.x = xcoord
      self.y = ycoord

   def affiche_point(self):
      return "Coord. horiz:"+str(self.x)+ " ; Coord. vert:"+str(self.y)
```

Qu'afficherait le programme principal suivant :

```
p8= Point(5)
print(p8.affiche_point())
```

(a) Coord. horiz:5 ; Coord. vert:0
(b) Coord. horiz:5 ; Coord. vert:5
(c) Coord. horiz:0 ; Coord. vert:0
(d) Erreur
(e) Aucune de ces réponses

11. Si on veut ajouter une méthode *distance* à la classe *Point (dans la partie #CODE MANQUANT) qui permettrait de calculer la distance entre deux points* du système de coordonnées *(2 objets de type* `Point`*).* Quelle est la meilleure definition de cette méthode *distance:*

```
class Point(object):
   def __init__(self, xcoord=0, ycoord=0):
      self.x = xcoord
      self.y = ycoord

   def affiche_point(self):
      return "Coord. horiz:"+str(self.x)+ " ; Coord. vert:"+str(self.y)

 # CODE MANQUANT

 (I)
    def distance(self,autre):
          dx =self.x - autre.x
          dy =self.y - autre.y
          return (dx*dx + dy*dy)**0.5
```

(II)
```
def distance(self1,self2):
    dx =self1.x – self2.x
    dy =self1.y – self2.y
    return (dx*dx + dy*dy)**0.5
```

(III)
```
def distance(self,autre):
    dx =self. xcoord - autre. xcoord
    dy =self. xcoord - autre. xcoord
    return (dx*dx + dy*dy)**0.5
```

(a) I
(b) II
(c) III
(d) Aucune de ces réponses

12. Considérant la classe suivante qui utilise la classe *Point* de la question antérieure.

```
class Pointset(object):
    def __init__(self, points):
    '''(Pointset, list des objecs de type Point) -> None
    Initialise le Pointset avec la liste points.'''
    self.points = points
```

Lesquelles des lignes suivantes peuvent être utilisées pour créer un objet de classe `Pointset`?

(I)
```
points=[Point(1,2), Point(2,1), Point(0,0)]
nouveau_pointset=Pointset(points)
```
(II)
```
nouveau_pointset=Pointset( Point(1,2), Point(2,1), Point(0,0) )
```
(III)
```
nouveau_pointset=Pointset([Point(0,0)])
```

(a) I (b) II (c) III (d) I et II (e) I et III

13. La diagonale d'une matrice carrée commence dans le coin gauche supérieur jusqu'au coin droit inférieur. Par exemple, pour cette matrice carrée:
1 3 5
2 4 5
4 0 8

représentée par la liste 2D [[1, 3, 5], [2, 4, 5], [4, 0, 8]], les valeurs de la diagonale sont 1, 4 et 8.
Considérez la fonction suivante, avec une partie de code qui manque:

```
def calcule_diagonale_et_non_diagonale(L):
    '''(list) -> tuple de (list, list)
    Precondition: Les listes sont des listes 2D des entiers.
    Retourne un tuple avec 2 éléments. Le premier élément est une liste avec des
    valeurs de la diagonale de la matrice carrée L et le deuxième élément est
    une liste avec les autres valeurs de L.
    >>> calcule_diagonale_et_non_diagonale([[1, 3, 5], [2, 4, 5], [4, 0, 8]])
    ([1, 4, 8], [3, 5, 2, 5, 4, 0])
    '''
    diagonale = []
    non_diagonale = []
    for r in range(len(L)):
        for c in range(len(L)):
            # CODE MANQUANT
    return (diagonale, non_diagonale)
```

Lesquels des fragments de code suivants sont corrects pour compléter cette fonction?

(I)

```
if r == c:
    diagonale.append(L[r][c])
else:
    non_diagonale.append(L[r][c])
```

(II)

```
if r == c:
    diagonale.append(L[r][c])
non_diagonale.append(L[r][c])
```

(III)

```
if r == c:
    diagonale.append(L[r][c])
if r != c:
    non_diagonale.append(L[r][c])
```

(a) I seulement
(b) II seulement
(c) III seulement
(d) I et II
(e) I et III

14) Considérer une autre version de la fonction *maSomme(x,n)* qui calcule n*x pour tout entier n ≥0 (Q26 de Pratiquer_Final précédent):

```
def maSomme2(x,n):
    '''Return x*n, fonctionne seulement pour les entiers non négatives de n'''

    if n <= 0: return 0
    res = maSomme2(x, n//2)
    res = res+res
    if n % 2 == 1: #  n  est impair, on veut avoir 2*(n//2)*x+x
        res = res+x
    return res
```

Si nous appelons cette fonction avec n = 128, alors :
(a) cette fonction effectue 8 appels de fonctions récursives et 8 additions
(b) cette fonction effectue 8 appels de fonctions récursives et 128 additions
(c) cette fonction effectue 128 appels de fonctions récursives et 8 additions
(d) cette fonction effectue 128 appels de fonctions récursives et 128 additions

15. Dans la question précédente, combiens d'arguments (ou paramètres actuels) y-a-t'il dans l'appel *maSomme2*?

(a) 0 (b)1 (c) 2 (d) 128 (e) Aucune réponse n'est valide