INFO-H-100 - Programmation
TP 3 - Tests et fonctions

Lorsque l'exercice demande d'écrire une fonction, écrivez-la fonction demandée et testez-la avec plusieurs valeurs pertinentes.

Ex. 1. Qu'affiche ce programme ? Expliquez avec quelques valeurs représentatives. Pouvez-vous le simplifier (le remplacer par un programme équivalent mais plus simple) ?

```
def f(a, b):
    if a > 0:
        if b > 1:
            print(a)
        else:
            print(b)
    else:
        if b > 1:
            print(a + b)
        else:
            print(b)
```

Ex. 2. Quelles sont les valeurs des variables au fur et à mesure de l'exécution de ces instructions ?

```
a = 2
b = 3
c = 4
test_1 = True
test_2 = (b >= a) and (c >= b)
test_3 = test_1 or test_2
stop = test_3 and (not test_2)
a += 1
b -= 1
c -= 2
test_1 = True
test_2 = (b >= a) and (c >= b)
test_3 = test_1 or test_2
stop = stop or test_2
```

Ex. 3. Écrire une fonction qui vérifie si un nombre entier est valide pour le Lotto, c'est-à-dire qu'il est compris entre 1 et 42 inclus.

Ex. 4. Écrire une fonction qui vérifie qu'une année est bissextile. Sont bissextiles les années divisibles par 4 mais non divisibles par 100 ou les années divisibles par 400.

Ex. 5. Écrire une fonction qui vérifie qu'un point est dans un rectangle horizontal. Un point est représenté par un tuple `(x,y)` et un rectangle est représenté par deux points : le coin supérieur gauche et le coin inférieur droit. Nous supposons que les x croissent vers la droite et que les y croissent vers le haut.

Ex. 6. Écrire une fonction qui reçoit 4 points et qui vérifie si ces 4 points forment un carré horizontal. Ces 4 points étant respectivement le coin supérieur gauche, le point supérieur droit, le coin inférieur gauche et le coin inférieur droit.

Ex. 7. Écrire une fonction qui reçoit les valeurs de trois dés à six faces et qui vérifie que ces dés forment un 421.

Ex. 8. Écrire une fonction qui reçoit trois nombres et qui renvoie un tuple composé de ces trois nombres dans l'ordre croissant.

Ex. 9. Écrire une fonction qui reçoit trois nombres et qui renvoie les deux plus grands.

Ex. 10. Soit l'équation du second degré $ax^2 + bx + c = 0$. Écrivez une fonction qui reçoit a, b et c et qui un tuple des solutions réelles (s'il y en a) de cette équation. Pour rappel, $\Delta = b^2 - 4ac$.
- Si Δ est strictement positif, l'équation admet deux solutions $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ et $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$.
- Si Δ est nul, l'équation admet une solution $\frac{-b}{2a}$.

– Si Δ est négatif, l'équation n'admet pas de solution réelle.
  Un tuple vide s'écrit `()`, alors qu'un tuple avec un seul élément s'écrit, par exemple, `(5, )`

Ex. 11. On représente un instant comme un triplet (heure, minutes, secondes). Écrire un fonction qui reçoit un instant et qui renvoie l'instant de la seconde suivante. On considère que l'instant suivant de 23 heures 59 minutes et 59 secondes est 0 heure 0 minute et 0 seconde.

Ex. 12. On représente un instant comme un triplet (heure, minutes, secondes). Écrire une fonction qui additionne deux instants. On considère que l'instant suivant de 23 heures 59 minutes et 59 secondes est 0 heure 0 minute et 0 seconde.

INFO-H-100 - Programmation
TP 3 - Tests et fonctions
Corrections

Solution de l'exercice 1:

Si b est inférieur ou égal à 1, ce programme affiche b. Sinon, si a est strictement supérieur à 0, ce programme affiche a. Sinon, c'est-à-dire que b est strictement supérieur à 1 et a est inférieur ou égal à 0, ce programme affiche la somme de a et de b.

```
f(10, 0)  #prints 0
f(10, 1)  #prints 1
f(1, 2)   #prints 1
f(10, 3)  #prints 10
f(0, 10)  #prints 10
f(-10, 3) #prints -7
```

Voici une version simplifiée :

```
def f(a, b):
    if b <= 1:
        print(b)
    elif a > 0:
        print(a)
    else:
        print(a + b)
```

Solution de l'exercice 2:

instruction	a	b	c	test_1	test_2	test_3	stop
a = 2	2						
b = 3	2	3					
c = 4	2	3	4				
test_1 = True	2	3	4	True			
test_2 = (b >= a) and (c >= b)	2	3	4	True	True		
test_3 = test_1 or test_2	2	3	4	True	True	True	
stop = test_3 and (not test_2)	2	3	4	True	True	True	False
a += 1	3	3	4	True	True	True	False
b -= 1	3	2	4	True	True	True	False
c -= 2	3	2	2	True	True	True	False
test_1 = True	3	2	2	True	True	True	False
test_2 = (b >= a) and (c >= b)	3	2	2	True	False	True	False
test_3 = test_1 or test_2	3	2	2	True	False	True	False
stop = stop or test_2	3	2	2	True	False	True	False

Solution de l'exercice 3:

```
def is_valid_for_lotto(number):
    return 1 <= number <= 42
```

```
print(is_valid_for_lotto(-1)) #prints False
print(is_valid_for_lotto(1))  #prints True
print(is_valid_for_lotto(42)) #prints True
print(is_valid_for_lotto(98)) #prints False
```

Solution de l'exercice 4:

```
def is_leap_year(year):
    return (year % 400 == 0) or (year % 100 != 0 and year % 4 == 0)

print(is_leap_year(1998)) #prints False
print(is_leap_year(2000)) #prints True
print(is_leap_year(2011)) #prints False
print(is_leap_year(2012)) #prints True
```

Solution de l'exercice 5:

```
def is_in_rectangle(point, upper_left, lower_right):
    """ vérifie si le point (tuple (x, y)) est dans le rectangle
        défini par ses coins supérieur gauche et inférieur droit"""
    (x, y) = point
    (x_1, y_1) = upper_left
    (x_2, y_2) = lower_right
    return (x >= x_1 and x <= x_2 and y >= y_2 and y <= y_1)

print(is_in_rectangle((1, 1), (0, 2), (3, 0)))  #prints True
print(is_in_rectangle((0, 0), (0, 2), (3, 0)))  #prints True
print(is_in_rectangle((-1, 0), (0, 2), (3, 0))) #prints False
```

Solution de l'exercice 6:

```
def is_square(upper_l, upper_r, lower_l, lower_r):
    """vérifie que le quadrilatere dont les 4 sommets sont upper_l, upper_r,
       lower_l et lower_r, respectivement les coins supérieur gauche, supérieur droit,
       inférieur gauche et inférieur droit, forme un carré horizontal"""
    (x_1, y_1) = upper_l
    (x_2, y_2) = upper_r
    (x_3, y_3) = lower_l
    (x_4, y_4) = lower_r
    return (x_1 == x_3 and y_1 == y_2 and x_2 == x_4 and y_3 == y_4 and (x_2 - x_1) == (y_1 - y_3))

print(is_square((0, 1), (1, 1), (0, 0), (1, 0))) #prints True
print(is_square((0, 1), (2, 1), (0, 0), (2, 0))) #prints False
print(is_square((0, 1), (1, 1), (0, 0), (2, 0))) #prints False
```

Solution de l'exercice 7:

```
def is_421(x, y, z):
    """vérifie que les trois entiers x,y et z forment un 421"""
    return ((x == 4 or y == 4 or z == 4) and
            (x == 2 or y == 2 or z == 2) and
            (x == 1 or y == 1 or z == 1))
```

Solution naïve :

```
def is_421(x, y, z):
    """vérifie que les trois entiers x,y et z forment un 421"""
    flag = False
    if x == 4 and y == 2 and z == 1:
        flag = True
    elif x == 4 and y == 1 and z == 2:
        flag = True
    elif x == 2 and y == 4 and z == 1:
        flag = True
    elif x == 2 and y == 1 and z == 4:
        flag = True
    elif x == 1 and y == 4 and z == 2:
        flag = True
    elif x == 1 and y == 2 and z == 4:
        flag = True
    return flag

print(is_421(4, 2, 1))    #prints True
print(is_421(4, 1, 2))    #prints True
print(is_421(2, 4, 1))    #prints True
print(is_421(2, 1, 4))    #prints True
print(is_421(1, 4, 2))    #prints True
print(is_421(1, 2, 4))    #prints True
```

```
print(is_421(4, 2, 2))    #prints False
print(is_421(-1, 2.0, 4)) #prints False
```

Autre solution identique au niveau du flot d'exécution :

```
def is_421(x, y, z):
  """vérifie que les trois entiers x,y et z forment un 421"""
  return (x == 4 and y == 2 and z == 1 or
          x == 4 and y == 1 and z == 2 or
          x == 2 and y == 4 and z == 1 or
          x == 2 and y == 1 and z == 4 or
          x == 1 and y == 4 and z == 2 or
          x == 1 and y == 2 and z == 4)
```

Une autre solution plus rapide (analysez le flot d'éxécution) :

```
def is_421(x, y, z):
    """vérifie que les trois entiers x,y et z forment un 421"""
    flag = False
    if x == 4:
        if y == 2:
            if z == 1:
                flag = True   #ordre 4 2 1
        elif y == 1:
            if z == 2:
                flag = True   #ordre 4 1 2
    elif x == 2:
        if y == 4:
            if z == 1:
                flag = True   #ordre 2 4 1
        elif y == 1:
            if z == 4:
                flag = True   #ordre 2 1 4
    elif x == 1:
        if y == 4:
            if z == 2:
                flag = True   #ordre 1 4 2
        elif y == 2:
            if z == 4:
                flag = True   #ordre 1 2 4
    return flag
```

Une solution plus élégante est de trier ces trois nombres avant de tester leurs valeurs comme fait dans l'exercice 8 : .

```
def is_421(x, y, z):
    """vérifie que les trois entiers x,y et z forment un 421"""
    (x, y, z) = sort_3_items(x, y, z)
    return x == 1 and y == 2 and z == 4
```

Solution basée sur le fait que les dés ont 6 faces de 1 à 6 :

```
def is_421(x, y, z):
    """vérifie que les trois entiers x,y et z forment un 421"""
    return (x + y + z == 7) and (x == 4 or y == 4 or z == 4)
```

Solution de l'exercice 8:

```
def sort_3_items(x, y, z):
    if x < y:
        if y < z:
            res = (x, y, z)
        elif x < z:
            res = (x, z, y)
        else:
            res = (z, x, y)
    else:
        if x < z:
            res = (y, x, z)
        elif y < z:
            res = (y, z, x)
        else:
            res = (z, y, x)
    return res

print(sort_3_items(1, 2, 3))  #prints (1, 2, 3)
print(sort_3_items(1, 1, 1))  #prints (1, 1, 1)
print(sort_3_items(3, 2, 1))  #prints (1, 2, 3)
print(sort_3_items(1, 3, 2))  #prints (1, 2, 3)
print(sort_3_items(3, 1, 1))  #prints (1, 1, 3)
```

Solution de l'exercice 9:

```
def two_max(a, b, c):
    if a < b and a < c: #a est le minimum
        maximum = (b,c)
    elif b < c:         #b est le minimum
        maximum = (a,c)
    else:               #c est le minimum
        maximum = (a,b)
    return maximum

print(two_max(1, 2, 3)) #prints (2, 3)
print(two_max(2, 2, 3)) #prints (2, 3)
print(two_max(1, 2, 2)) #prints (2, 2)
print(two_max(3, 2, 1)) #prints (3, 2)
print(two_max(1, 3, 2)) #prints (3, 2)
```

Explications : trouver le minimum plutôt que d'essayer de déterminer les deux nombres les plus grands.

Solution de l'exercice 10:

```
import math

def quad_equation_sol(a, b, c):
    d = b ** 2 - 4 * a * c
    if d > 0:
        ds = math.sqrt(d)
        print((-b + ds) / 2 / a, (-b - ds) / 2 / a)
    elif d == 0:
        print(-b / 2.0 / a)

print(quad_equation_sol(1, 2, 3)) x2 + 2x + 3 : delta < 0 -> ()
print(quad_equation_sol(1, -2, 1)) #x2 - 2x + 1 : delta = 0 -> (1.0,)
print(quad_equation_sol(1,-3,2)) #x2 - 3x + 2 : delta > 0 -> (2.0, 1.0)
```

Solution de l'exercice 11:

```
def next_moment(moment):
    (h, m, s) = moment
    s += 1
    if(s == 60):
        s = 0
        m += 1
    if(m == 60):
        m = 0
        h += 1
    if(h == 24):
        h = 0
    return (h, m, s)


print(next_moment((23, 59, 59))) #prints (0, 0, 0)
print(next_moment((10, 10, 59))) #prints (10, 11, 0)
print(next_moment((10, 59, 59))) #prints (11, 0, 0)
print(next_moment((0, 0, 0)))    #prints (0, 0, 1)
```

Solution de l'exercice 12:

```
def sum_moments(moment_1, moment_2):
    (h_1, m_1, s_1) = moment_1
    (h_2, m_2, s_2) = moment_2
    s = s_1 + s_2
    m = m_1 + m_2
    h = h_1 + h_2
    if(s >= 60):
        m += s // 60
        s = s % 60
    if(m >= 60):
        h += m // 60
        m = m % 60
    if(h >= 24):
        h = h % 24
    return (h, m, s)

print(sum_moments((10, 10, 10), (10, 10, 10))) #prints(20, 20, 20)
print(sum_moments((10, 10, 10), (10, 10, 50))) #prints(20, 21, 0)
print(sum_moments((10, 10, 10), (10, 50, 50))) #prints(21, 1, 0)
print(sum_moments((10, 10, 10), (13, 49, 50))) #prints(0, 0, 0)
```