INFO-H-600 - Python
TP 2 - Fonctions and instruction flows

If an exercise is asking you to write a function, write it and test it with different interesting input values.

Ex. 1. What does the following program do? Explain it with simple values. Can you simplify it (replace it with an equivalent but simpler program) ?

```
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. What are the values of the variables at each instruction ?

```
a = 2
b = 3
c = 4
test1 = True
test2 = (b >= a) and (c >= b)
test3 = test1 or test2
arret = test3 and (not test2)
a += 1
b -= 1
c -= 2
test1 = True
test2 = (b >= a) and (c >= b)
test3 = test1 or test2
arret = arret or test2
```

Ex. 3. Some of the following programs are not correct. For each program, indicate what it prints and eventually which instructions may cause some errors.

1.

```
def my_test(a):
    print(a)

a = 5
my_test(a)
print(a)
```

2.

```
def my_test(a):
    a += 1
    print(a)

a = 5
my_test(a)
print(a)
```

3.

```
def my_test(b):
    a = 6
    print(b)

a = 5
my_test(a)
print(a)
```

4.

```
def my_test(a):
    a = a + 6
    c = a

a = 5
my_test(4)
print(a)
print(c)
```

5.

```
def my_test():
    a = 6
    print(a)

a = 5
my_test()
print(a)
```

6. (Bonus)

```
def my_test(a=8):
    print(a)

a = 5
my_test(9)
print(a)
```

Ex. 4. Write a function that checks whether an integer is a valid number for the Lotto (is between 1 and 42 included).

Ex. 5. Write a function that checks whether a year is leap. Leap years are the years divisible by 4 but not by 100 or the years divisible by 400.

Ex. 6. Write a function that receives the values of 3 dices and checks if it is possible to form 421 with these dices.

Ex. 7. Write two functions (`my_range_while(a,b)` and `my_range_for(a,b)`) which produce a list containing all the integers in the range $[a, b[$. One function should use a while loop and the other use a for loop.

Ex. 8. Write two functions (`my_range_while(a,b,step)` and `my_range_for(a,b,step)`) which produce a list containing all the integers in the range $[a, b[$ separeted by step.. One function should use a while loop and the other use a for loop.

```
print(my_range_while(3, 19, 3))    # prints [4, 7, 10, 13, 16]
```

Ex. 9. Write a function that produces a list containing the n first powers of 2.

```
print(powersOf2(4))    # prints [1, 2, 4, 8]
print(powersOf2(10))   # prints [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

Ex. 10. Write a function that produces a list containing the prime numbers inferior to a given upper-bound. Start by writing a function that checks whether a number is prime.

```
print(isPrime(11))          # True
print(prime_numbers(17))    # prints [2, 3, 5, 7, 11, 13]
print(prime_numbers(50))    # prints [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Ex. 11. Write a function that receives three numbers and returns a list composed of these three numbers in increasing order.

Ex. 12. Write a function that receives three numbers and returns a list composed of the two highest numbers (in any order).

Ex. 13. Given a second degree equation $ax^2 + bx + c = 0$, write a function receiving a, b and c and returning a list containing the possible solutions if any. Reminder $\Delta = b^2 - 4ac$.

- If Δ strictly positive, there are two solutions: $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ and $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$.
- If Δ is null, there is one solution $\frac{-b}{2a}$.
- If Δ is strictly negative, there is no real solution (your function should then return an empty list).

Ex. 14. An instant is composed of an hour, a minute and a second (ex: 5, 34, 22 = 5h34min22s). Write a funciton receiving an instant and returning a list containing the next instant (ex: `[5, 34, 23]`).

Ex. 15. Write a function that receives the coordinates of 4 points and check whether these points form a square. Your function should receive as parameters the coordinates of the four points in the following order: top left, top right, lower left and lower right.

Ex. 16. Write a funciton that generates a list of sublists in the following way:

```
print(triangle_list(2))    #prints [[1], [1, 2]]
print(triangle_list(4))    #prints [[1], [1, 2], [1, 2, 3], [1, 2, 3, 4]]
print(triangle_list(0))    #prints []
```

INFO-H-600 - Programmation
TP 2 - Fonctions and instruction flows
Corrections

Solution to the exercise 1:

If b is inferior or equal to 1, the program will show b. If this is not the case, if a is strictly higher than 0, the program will show a. If this is also not the case, which means that both b is strictly higher than 1 and q is inferior or equal to 0, then the program will show the sum of a and b.

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

Here is a simplified version:

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

Solution to the exercise 2:

instruction	a	b	c	test1	test2	test3	arret
`a = 2`	2						
`b = 3`	2	3					
`c = 4`	2	3	4				
`test1 = True`	2	3	4	`True`			
`test2 = (b >= a) and (c >= b)`	2	3	4	`True`	`True`		
`test3 = test1 or test2`	2	3	4	`True`	`True`	`True`	
`arret = test3 and (not test2)`	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`
`test1 = True`	3	2	2	`True`	`True`	`True`	`False`
`test2 = (b >= a) and (c >= b)`	3	2	2	`True`	`False`	`True`	`False`
`test3 = test1 or test2`	3	2	2	`True`	`False`	`True`	`False`
`arret = arret or test2`	3	2	2	`True`	`False`	`True`	`False`

Solution to the exercise 3:

1. The variable a inside my_test is proper to the function and is initialised to 5. Therefore both print instructions will print 5.
2. The variable a inside my_test is proper to the function and is initialised to 5 but then incremented to 6. The variable a outside my_test is initialised to 5 and is not changed! Therefore the print inside my_test prints 6 while the other print prints 5.
3. The variable b inside my_test is proper to the function and is initialised to 5. The variable a inside my_test is proper to the function and is initialised to 6. The variable a outside my_test is initialised to 5 and is not changed! Therefore both prints print 5.

4. The variable a inside my_test is proper to the function and is initialised to 5 but then changed to 10. The variable c inside my_test is proper to the function and is initialised to 10. The variable a outside my_test is initialised to 5 and is not changed by the function! The variable c does not exist outside my_test. The first print instruction will therefore print 5 and the second will create an error.

5. The variable a inside my_test is proper to the function and is initialised to 6. The variable a outside my_test is initialised to 5 and is not changed. Therefore the print instruction inside my_test prints 6 while the other print instruction prints 5.

6. The variable a inside my_test is proper to the function and is initialised to 9. The variable a outside my_test is initialised to 5 and is not changed. Therefore the print inside my_test prints 9 while the other print prints 5.

Solution to the exercise 4:

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

print(is_valid(-1)) # shows False
print(is_valid(1))  # shows True
print(is_valid(42)) # shows True
print(is_valid(98)) # shows False
```

Solution to the exercise 5:

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

print(is_leap(1998)) # shows False
print(is_leap(2000)) # shows True
print(is_leap(2011)) # shows False
print(is_leap(2012)) # shows True
```

Solution to the exercise 6:

```
def is_421(x,y,z):
    """Checks that the four dices can form 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))
```

Naive Solution :

```
def is_421(x,y,z):
    """Checks that the four dices can form 421."""
    if x == 4 and y == 2 and z == 1:
        return True
    elif x == 4 and y == 1 and z == 2:
        return True
    elif x == 2 and y == 4 and z == 1:
        return True
    elif x == 2 and y == 1 and z == 4:
        return True
    elif x == 1 and y == 4 and z == 2:
        return True
    elif x == 1 and y == 2 and z == 4:
        return True
    else:
        return False

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
```

Another solution with an identical flow of instructions:

```
def is_421(x,y,z):
    """Checks that the four dices can form 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)
```

A more effective solution (you can try to compare the flow of instructions):

```
def is_421(x,y,z):
    """Checks that the four dices can form 421."""
    if x == 4:
        if y == 2:
            if z == 1:
                return True    # order 4 2 1
        elif y == 1:
            if z == 2:
                return True    # order 4 1 2
    elif x == 2:
        if y == 4:
            if z == 1:
                return True    # order 2 4 1
        elif y == 1:
            if z == 4:
                return True    # order 2 1 4
    elif x == 1:
        if y == 4:
            if z == 2:
                return True    # order 1 4 2
        elif y == 2:
            if z == 4:
                return True    # order 1 2 4
    return False
```

Solution based on the fact that all dices are between 1 and 6:

```
def is_421(x,y,z):
    """Checks that the four dices can form 421."""
    return x+y+z==7 and (x == 4 or y == 4 or z == 4)
```

Solution to the exercise 7:

Solution with while:

```
def my_range_while(a,b):
        liste = []
        while (a < b):
                liste.append(a)
                a += 1

        return liste
```

First solution with for:

```
def my_range_for(a,b):
        liste = []
        for i in range(b-a):
                liste.append(a + i)
        return liste
```

Second solution with for:

```
def my_range_for(a,b):
        liste = []
        for i in range(a, b):
                liste.append(i)
        return liste
```

Last solution without while nor for:

```
def my_range(a,b):
        return list(range(a, b))
```

Solution to the exercise 8:

Solution with while:

```
def my_range_step_while(a,b,step):
        liste = []
        while (a < b):
                liste.append(a)
                a += step

        return liste
```

Solution with for:

```
def my_range_step_while(a,b,step):
        liste = []
        for i in range(a, b, step):
                liste.append(i)

        return liste
```

Solution to the exercise 9:

```
def powersOf2(n):
    li = []
    power = 1
    for i in range(n):
        li.append(power)
        power = power*2

    return li

print(powersOf2(4))   #affiche [1, 2, 4, 8]
print(powersOf2(10))  #affiche [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

Solution to the exercise 10:

```
def isPrime(nombre):
    if(number < 2):
        return False
    for i in range(2, number):
        if (number % i == 0):
            return False
    return True

def prime_numbers(max):
    primes = []
    for i in range(max):
        if (isPrime(i)):
            primes.append(i)

    return primes

print(prime_numbers(17))   #affiche [2, 3, 5, 7, 11, 13]
print(prime_numbers(50))   #affiche [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Solution to the exercise 11:

```
def in_order(x,y,z):
"""Returns a list composed of the three integers in increasing order."""
    if x < y:
        if y < z:
            return [x,y,z]
        elif x < z:
            return [x,z,y]
        else:
            return [z,x,y]
    else:
        if x < z:
            return [y,x,z]
        elif y < z:
            return [y,z,x]
        else:
            return [z,y,x]

print(in_order(1,2,3))  # prints [1,2,3]
print(in_order(1,1,1))  # prints [1,1,1]
print(in_order(3,2,1))  # prints [1,2,3]
print(in_order(1,3,2))  # prints [1,2,3]
print(in_order(3,1,1))  # prints [1,1,3]
```

Solution to the exercise 12:

```
def two_maxs(a,b,c):
    if a < b and a < c: # a is the minimum
        return [b,c]
    elif b < c          # b is the minimum
        return [a,c]
    else:               # c is the minimum
        return [a,b]

print(two_maxs(1,2,3)) # prints [2,3]
print(two_maxs(2,2,3)) # prints [2,3]
print(two_maxs(1,2,2)) # prints [2,2]
print(two_maxs(3,2,1)) # prints [3,2]
print(two_maxs(1,3,2)) # prints [3,2]
```

Solution to the exercise 13:

```
import math

def sols_second_degree(a,b,c):
    d = b**2 - 4*a*c
    if d > 0:
        ds = math.sqrt(d)
        return [(-b+ds)/2/a, (-b-ds)/2/a]
    elif d == 0:
        return [-b/2/a]
    else:
        return []

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

Solution to the exercise 14:

```
def next_instant(h, m, s):
    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_instant(23,59,59)) # prints [0, 0, 0]
print(next_instant(10,10,59)) # prints [10, 11, 0]
print(next_instant(10,59,59)) # prints [11, 0, 0]
print(next_instant(0,0,0))    # prints [0, 0, 1]
```

Solution to the exercise 15:

```
    def is_square(x1, y1, x2, y2, x3, y3, x4, y4):
    """Checks that the four points form a square.

    The points are supposed to follow the following layout:

                1        2

                3        4
    """
    return(x1 == x3 and y1 == y2 and x2 == x4 and y3 == y4 and x2-x1 == y1-y3)

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 to the exercise 16:

```
def triangle_list(n):
    triangle = []
    for i in range(n):
        sub_list = []
        for j in range(i+1):
            sub_list.append(j+1)
        triangle.append(sub_list)

    return triangle

print(triangle_list(2))   # prints [[1], [1, 2]]
print(triangle_list(4))   # prints [[1], [1, 2], [1, 2, 3], [1, 2, 3, 4]]
print(triangle_list(0))   # prints []
```

Another solution :

```
def triangle_list(n):
    master_list = []
    sub_liste = []
    for i in range(n):
        sub_list.append(i+1)
        master_list.append(sub_list[:])
    return master_list

print(triangle_list(2))  # prints [[1], [1, 2]]
print(triangle_list(4))  # prints [[1], [1, 2], [1, 2, 3], [1, 2, 3, 4]]
print(triangle_list(0))  # prints []
```