Yaoundé International Business School
developing innovative professionals

Software Development (python)
Academic year: 2023-2024
Level: 3

Course Instructor: TEKOH PALMA A.

Instruction:

- Carefully organize your work in a project folder and comment your code as much as possible
- The solution to each problem should be presented as a function that takes the necessary parameters and returns the expected results
- Problems that involve OOP should be designed and implemented using OOP concepts
- Each section should have a corresponding folder in your project organization
- Project folder should be named with the group name emailed to Tekohpalma@gmail.com
- Deadline: Saturday 10th Febuary 2024 – 1:00 am

SECTION A: BASIC PROGRAMMING (30pts)

1) Write a program that converts from degrees Fahrenheit F to degrees Celsius C using the following formula, and write the result to the screen:

$$C = \frac{(F-32)}{18}$$ (2pts)

2) Write a python function that returns the number of vowels in a string S

E.g. *Input S =" Good Morning"*
Output count = 4
Hint: your function should take a string ***S*** *as parameter and return number of vowels in the string* (2pts)

3) Write a program that computes the (two) roots of the quadratic equation:

$$ax^2 + bx + c = 0$$

where $a = 1.2$, $b = 2.3$ and $c = -3.4$.

$$x1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \qquad x2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$$

(2pts)

4) **Longest Monotonical run (4pts):** You are given the following definitions:

- A run of monotonically increasing numbers means that a number at position **k+1** in the sequence is greater than or equal to the number at position **k** in the sequence.

- A run of monotonically decreasing numbers means that a number at position k+1 in the sequence is less than or equal to the number at position k in the sequence.

- Implement a function that meets the specifications below.

```
def longest_run(L):
    """
    Assumes L is a list of integers containing at least 2 elements.
    Finds the longest run of numbers in L, where the longest run can
    either be monotonically increasing or monotonically decreasing.
    In case of a tie for the longest run, choose the longest run
    that occurs first.
    Does not modify the list.
    Returns the sum of the longest run.
    """
```

For example:

- If L = [10, 4, 3, 8, 3, 4, 5, 7, 7, 2] then the longest run of monotonically increasing numbers in L is [3, 4, 5, 7, 7] and the longest run of monotonically decreasing numbers in L is [10, 4, 3]. Your function should return the value 26 because the longest run of monotonically increasing integers is longer than the longest run of monotonically decreasing numbers.

- If L = [5, 4, 10] then the longest run of monotonically increasing numbers in L is [4, 10] and the longest run of monotonically decreasing numbers in L is [5, 4]. Your function should return the value 9 because the longest run of monotonically decreasing integers occurs before the longest run of monotonically increasing numbers.

651748248

5) **Converting numbers to Mandarin: (4pts)**

Numbers in Mandarin follow 3 simple rules.

- There are words for each of the digits from 0 to 10.
- For numbers 11-19, the number is pronounced as "ten digit", so for example, 16 would be pronounced (using Mandarin) as "ten six".

- For numbers between 20 and 99, the number is pronounced as “digit ten digit”, so for example, 37 would be pronounced (using Mandarin) as "three ten seven". If the digit is a zero, it is not included.

Here is a simple Python dictionary that captures the numbers between 0 and 10.
trans = {'0':'ling', '1':'yi', '2':'er', '3':'san', '4': 'si', '5':'wu', '6':'liu', '7':'qi', '8':'ba', '9':'jiu', '10': 'shi'}

We want to write a procedure that converts a number (between 0 and 99), written as a string, into the equivalent Mandarin.

Example Usage
- convert_to_mandarin('36') will return san shi liu
- convert_to_mandarin('20') will return er shi
- convert_to_mandarin('16') will return shi liu

6) **Tower of Hanoi** (**6pts**)**:** In the nineteenth century, a game called the Tower of Hanoi became popular in Europe. This game represents work that is under way in the temple of Brahma. At the creation of the universe, priests in the temple of Brahma were supposedly given three diamond needles, with one needle containing 64 golden disks. Each golden disk is slightly smaller than the disk below it. The priests’ task is to move all 64 disks from the first needle to the third needle. The rules for moving the disks are as follows:
   - Only one disk can be moved at a time.
   - The removed disk must be placed on one of the needles.
   - A larger disk cannot be placed on top of a smaller disk.

   The priests were told that once they had moved all the disks from the first needle to the third needle, the universe would come to an end. Our objective is to write a program in python that prints the sequence of moves needed to transfer the disks from the first needle to the third needle

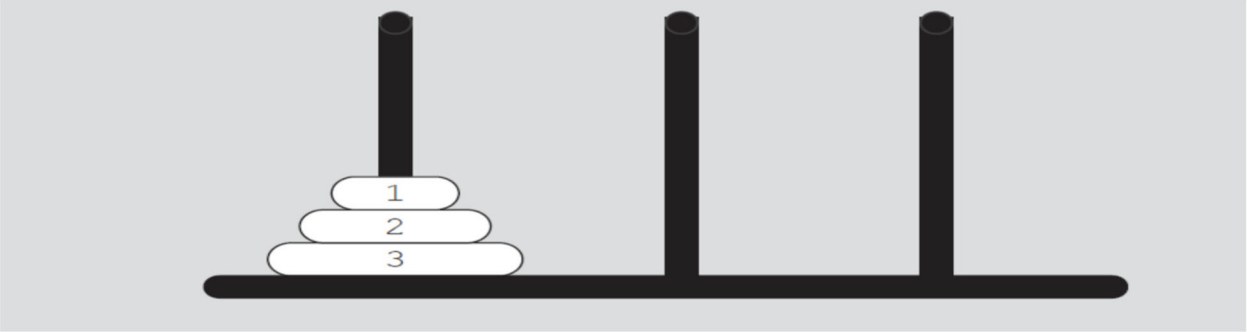

Figure 1: image of the tower of Hanoi with 3 disk and 3 pillars

7) N-Queens puzzle (10pts):

The N queens puzzle is the challenge of placing N non-attacking queens on an N×N chessboard. Write a program that solves the N queens problem.

- Usage: nqueens N
  - If the user called the program with the wrong number of arguments, print Usage: nqueens N, followed by a new line, and exit with the status 1
- where N must be an integer greater or equal to 4
  - If N is not an integer, print N must be a number, followed by a new line, and exit with the status 1
  - If N is smaller than 4, print N must be at least 4, followed by a new line, and exit with the status 1
- The program should print every possible solution to the problem
  - One solution per line
  - Format: see example
  - You don't have to print the solutions in a specific order
- You are only allowed to import the sys module

Read: Queen, Backtracking

```
palma@ubuntu:~/0x08. N Queens$ ./101-nqueens.py 4
[[0, 1], [1, 3], [2, 0], [3, 2]]
[[0, 2], [1, 0], [2, 3], [3, 1]]
palma@ubuntu:~/0x08. N Queens$ ./101-nqueens.py 6
[[0, 1], [1, 3], [2, 5], [3, 0], [4, 2], [5, 4]]
[[0, 2], [1, 5], [2, 1], [3, 4], [4, 0], [5, 3]]
[[0, 3], [1, 0], [2, 4], [3, 1], [4, 5], [5, 2]]
[[0, 4], [1, 2], [2, 0], [3, 5], [4, 3], [5, 1]]
palma@ubuntu:~/0x08. N Queens$
```

SECTION B: SOLVING REAL WORLD PROBLEMS WITH PYTHON (20pts)

Paying Off Credit Card Debt

Each month, a credit card statement will come with the option for you to pay a minimum amount of your charge, usually 2% of the balance due. However, the credit card company earns money by charging interest on the balance that you don't pay. So even if you pay credit card payments on time, interest is still accruing on the outstanding balance.

Say you've made a 5,000 FCFA purchase on a credit card with an 18% annual interest rate and a 2% minimum monthly payment rate. If you only pay the minimum monthly amount for a year, how much is the remaining balance?

You can think about this in the following way.

At the beginning of month **0** (when the credit card statement arrives), assume you owe an amount we will call $\mathbf{b_0}$ (*b* for *balance*; subscript *0* to indicate this is the balance at month 0). Any payment you make during that month is deducted from the balance. Let's call the payment you make in month 0, $\mathbf{p_0}$. Thus, your **unpaid balance** for month 0, $\mathbf{ub_0}$, is equal to $\mathbf{b_0 - p_0}$.

$$\mathbf{ub_0 = b_0 - p_0}$$

At the beginning of month 1, the credit card company will charge you interest on your unpaid balance. So if your annual interest rate is **r**, then at the beginning of month 1, your new balance is your previous unpaid balance $\mathbf{ub_0}$, **plus** the interest on this unpaid balance for the month. In algebra, this new balance would be

$$\boldsymbol{b_1 = ub_0 + \frac{r}{12.0} \cdot ub_0}$$

In month 1, we will make another payment, $\mathbf{p_1}$. That payment has to cover some of the interest costs, so it does not completely go towards paying off the original charge. The balance at the beginning of month 2, $\mathbf{b_2}$, can be calculated by first calculating the unpaid balance after paying $\mathbf{p_1}$, then by adding the interest accrued:

$$\mathbf{ub_1 = b_1 - p_1}$$

$$\boldsymbol{b_2 = ub_1 + \frac{r}{12.0} \cdot ub_1}$$

If you choose just to pay off the minimum monthly payment each month, you will see that the compound interest will dramatically reduce your ability to lower your debt.

Let's look at an example. If you've got a 5,000FCFA balance on a credit card with 18% annual interest rate, and the minimum monthly payment is 2% of the current balance, we would have the following repayment schedule if you only pay the minimum payment each month:

Month	Balance	Minimum Payment	Unpaid Balance	Interest
0	5000.00	100 (= 5000 * 0.02)	4900 (= 5000 - 100)	73.50 (= 0.18/12.0 * 4900)
1	4973.50 (= 4900 + 73.50)	99.47 (= 4973.50 * 0.02)	4874.03 (= 4973.50 - 99.47)	73.11 (= 0.18/12.0 * 4874.03)
2	4947.14 (= 4874.03 + 73.11)	98.94 (= 4947.14 * 0.02)	4848.20 (= 4947.14 - 98.94)	72.72 (= 0.18/12.0 * 4848.20)

You can see that a lot of your payment is going to cover interest, and if you work this through month 12, you will see that after a year, you will have paid 1165.63 FCFA and yet you will still owe 4691.11 FCFA on what was originally a 5000.00 FCFA debt. Pretty depressing!

Problem 1: Paying the Minimum (10pts)

Write a program to calculate the credit card balance after one year if a person only pays the minimum monthly payment required by the credit card company each month.

The following variables contain values as described below:

1. `balance` - the outstanding balance on the credit card
2. `annualInterestRate` - annual interest rate as a decimal
3. `monthlyPaymentRate` - minimum monthly payment rate as a decimal

For each month, calculate statements on the monthly payment and remaining balance, and print to screen something of the format:

```
Month: 1
Minimum monthly payment: 96.0
Remaining balance: 4784.0
```

Be sure to print out no more than two decimal digits of accuracy - so print

```
Remaining balance: 813.41
```

instead of

```
Remaining balance: 813.4141998135
```

Finally, print out the total amount paid that year and the remaining balance at the end of the year in the format:

```
Total paid: 96.0
Remaining balance: 4784.0
```

A summary of the required math is found below:

Monthly interest rate= (Annual interest rate) / 12.0
Minimum monthly payment = (Minimum monthly payment rate) x (Previous balance)
Monthly unpaid balance = (Previous balance) - (Minimum monthly payment)
Updated balance each month = (Monthly unpaid balance) + (Monthly interest rate x Monthly unpaid balance)

Note: Depending on where you round in this problem, your answers may be off by a few cents in either direction. Do not worry if your solution is within +/- 0.05 of the correct answer.

Test Cases

Test Case 1:

balance = 4213
annualInterestRate = 0.2
monthlyPaymentRate = 0.04

Result Your Code Should Generate:

```
-------------------
Month: 1
Minimum monthly payment: 168.52
Remaining balance: 4111.89
Month: 2
Minimum monthly payment: 164.48
Remaining balance: 4013.2
Month: 3
Minimum monthly payment: 160.53
Remaining balance: 3916.89
Month: 4
Minimum monthly payment: 156.68
Remaining balance: 3822.88
Month: 5
Minimum monthly payment: 152.92
Remaining balance: 3731.13
Month: 6
Minimum monthly payment: 149.25
Remaining balance: 3641.58
Month: 7
Minimum monthly payment: 145.66
Remaining balance: 3554.19
Month: 8
Minimum monthly payment: 142.17
Remaining balance: 3468.89
Month: 9
Minimum monthly payment: 138.76
Remaining balance: 3385.63
Month: 10
Minimum monthly payment: 135.43
Remaining balance: 3304.38
Month: 11
Minimum monthly payment: 132.18
Remaining balance: 3225.07
Month: 12
Minimum monthly payment: 129.0
Remaining balance: 3147.67
Total paid: 1775.55
Remaining balance: 3147.67
```

Test Case 2:

balance = 4842
annualInterestRate = 0.2
monthlyPaymentRate = 0.04

Result Your Code Should Generate:

```
-------------------
Month: 1
Minimum monthly payment: 193.68
Remaining balance: 4725.79
Month: 2
Minimum monthly payment: 189.03
Remaining balance: 4612.37
Month: 3
Minimum monthly payment: 184.49
Remaining balance: 4501.68
Month: 4
Minimum monthly payment: 180.07
Remaining balance: 4393.64
Month: 5
Minimum monthly payment: 175.75
Remaining balance: 4288.19
Month: 6
Minimum monthly payment: 171.53
Remaining balance: 4185.27
Month: 7
Minimum monthly payment: 167.41
Remaining balance: 4084.83
Month: 8
Minimum monthly payment: 163.39
Remaining balance: 3986.79
Month: 9
Minimum monthly payment: 159.47
Remaining balance: 3891.11
Month: 10
Minimum monthly payment: 155.64
Remaining balance: 3797.72
Month: 11
Minimum monthly payment: 151.91
Remaining balance: 3706.57
Month: 12
Minimum monthly payment: 148.26
Remaining balance: 3617.62
Total paid: 2040.64
Remaining balance: 3617.62
```

Hints

Only two decimal digits of accuracy??

Round your answers to 2 decimal places.
How to think about this problem?

To help you get started, here is a rough outline of the stages you should probably follow in writing your code:

- For each month:
    - Compute the monthly payment, based on the previous month's balance.
    - Update the outstanding balance by removing the payment, then charging interest on the result.
    - Output the month, the minimum monthly payment and the remaining balance.
    - Keep track of the total amount of paid over all the past months so far.
- Print out the result statement with the total amount paid and the remaining balance.

Use these ideas to guide the creation of your code.

Problem 2: Paying Debt Off In A Year (10pts)

Now write a program that calculates the minimum **fixed** monthly payment needed in order pay off a credit card balance within 12 months. By a fixed monthly payment, we mean a single number which does not change each month, but instead is a constant amount that will be paid each month.

In this problem, we will *not* be dealing with a minimum monthly payment rate.

The following variables contain values as described below:

1. `balance` - the outstanding balance on the credit card
2. `annualInterestRate` - annual interest rate as a decimal

The program should print out one line: the lowest monthly payment that will pay off all debt in under 1 year, for example:

```
Lowest Payment: 180
```

Assume that the interest is compounded monthly according to the balance at the end of the month (after the payment for that month is made). The monthly payment must be a

multiple of 10FCFA and is the same for all months. Notice that it is possible for the balance to become negative using this payment scheme, which is okay. A summary of the required math is found below:

Monthly interest rate = (Annual interest rate) / 12.0
Monthly unpaid balance = (Previous balance) - (Minimum fixed monthly payment)
Updated balance each month = (Monthly unpaid balance) + (Monthly interest rate x Monthly unpaid balance)

Test Cases

Test Case 1:

balance = 3329
annualInterestRate = 0.2

Result Your Code Should Generate:

Lowest Payment: 310

Test Case 2:

balance = 4773
annualInterestRate = 0.2

Result Your Code Should Generate:

Lowest Payment: 440

Test Case 3:

balance = 3926
annualInterestRate = 0.2

Result Your Code Should Generate:

Lowest Payment: 360

Hints

Hint: How to think about this problem?

- Start with 10FCFA payments per month and calculate whether the balance will be paid off in a year this way (be sure to take into account the interest accrued each month).
- If 10FCFA monthly payments are insufficient to pay off the debt within a year, increase the monthly payment by 10FCFA and repeat.

Hint: A way of structuring your code

- If you are struggling with how to structure your code, think about the following:
    - Given an initial balance, what code would compute the balance at the end of the year?
    - Now imagine that we try our initial balance with a monthly payment of 10FCFA. If there is a balance remaining at the end of the year, how could we write code that would reset the balance to the initial balance, increase the payment by 10FCFA, and try again (using the same code!) to compute the balance at the end of the year, to see if this new payment value is large enough.
    - I'm still confused! A good way to implement this problem will be to use a loop structure. You may want to refresh your understanding of **while** loops. Think hard about how the program will know when it has found a good minimum monthly payment value - when a good value is found, the loop can terminate.
- Be careful - you don't want to overwrite the original value of `balance`. You'll need to save that value somehow for later reference!

SECTION C: OBJECT ORIENTED PROGRAMMING WITH PYTHON (50pts)

Tasks

0. Simple rectangle (5.5pts)

Write an empty class Rectangle that defines a rectangle:

- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 0-main.py
#!/usr/bin/python3
Rectangle = __import__('0-rectangle').Rectangle

my_rectangle = Rectangle()
print(type(my_rectangle))
print(my_rectangle.__dict__)

guillaume@ubuntu:~/0x08$ ./0-main.py
<class '0-rectangle.Rectangle'>
{}
guillaume@ubuntu:~/0x08$
```

- File: `0-rectangle.py`

1. Real definition of a rectangle (5.5pts)

Write a class Rectangle that defines a rectangle by: (based on 0-rectangle.py)

- Private instance attribute: width:
  - property def width(self): to retrieve it
  - property setter def width(self, value): to set it:
    - width must be an integer, otherwise raise a TypeError exception with the message width must be an integer
    - if width is less than 0, raise a ValueError exception with the message width must be >= 0
- Private instance attribute: height:
  - property def height(self): to retrieve it
  - property setter def height(self, value): to set it:

        - height must be an integer, otherwise raise a TypeError exception with the message height must be an integer
        - if height is less than 0, raise a ValueError exception with the message height must be >= 0
- Instantiation with optional width and height: def __init__(self, width=0, height=0):
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 1-main.py
#!/usr/bin/python3
Rectangle = __import__('1-rectangle').Rectangle

my_rectangle = Rectangle(2, 4)
print(my_rectangle.__dict__)

my_rectangle.width = 10
my_rectangle.height = 3
print(my_rectangle.__dict__)

guillaume@ubuntu:~/0x08$ ./1-main.py
{'_Rectangle__height': 4, '_Rectangle__width': 2}
{'_Rectangle__height': 3, '_Rectangle__width': 10}
guillaume@ubuntu:~/0x08$
```

- File: `1-rectangle.py`

2. Area and Perimeter (5.5pts)

Write a class Rectangle that defines a rectangle by: (based on 1-rectangle.py)

- Private instance attribute: width:
    - property def width(self): to retrieve it
    - property setter def width(self, value): to set it:
        - width must be an integer, otherwise raise a TypeError exception with the message width must be an integer
        - if width is less than 0, raise a ValueError exception with the message width must be >= 0
- Private instance attribute: height:
    - property def height(self): to retrieve it
    - property setter def height(self, value): to set it:

        - `height` must be an integer, otherwise raise a `TypeError` exception with the message `height must be an integer`
        - if `height` is less than `0`, raise a `ValueError` exception with the message `height must be >= 0`
- Instantiation with optional `width` and `height`: `def __init__(self, width=0, height=0):`
- Public instance method: `def area(self):` that returns the rectangle area
- Public instance method: `def perimeter(self):` that returns the rectangle perimeter:
    - if `width` or `height` is equal to `0`, perimeter is equal to `0`
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 2-main.py
#!/usr/bin/python3
Rectangle = __import__('2-rectangle').Rectangle


my_rectangle = Rectangle(2, 4)
print("Area: {} - Perimeter: {}".format(my_rectangle.area(),
my_rectangle.perimeter()))


print("--")


my_rectangle.width = 10
my_rectangle.height = 3
print("Area: {} - Perimeter: {}".format(my_rectangle.area(),
my_rectangle.perimeter()))


guillaume@ubuntu:~/0x08$ ./2-main.py
Area: 8 - Perimeter: 12
--
Area: 30 - Perimeter: 26
guillaume@ubuntu:~/0x08$
```

- File: `2-rectangle.py`

3. String representation (5.5pts)

Write a class `Rectangle` that defines a rectangle by: (based on `2-rectangle.py`)

- Private instance attribute: `width`:
    - property `def width(self):` to retrieve it

    - property setter `def width(self, value):` to set it:
        - `width` must be an integer, otherwise raise a `TypeError` exception with the message `width must be an integer`
        - if `width` is less than `0`, raise a `ValueError` exception with the message `width must be >= 0`
- Private instance attribute: `height`:
    - property `def height(self):` to retrieve it
    - property setter `def height(self, value):` to set it:
        - `height` must be an integer, otherwise raise a `TypeError` exception with the message `height must be an integer`
        - if `height` is less than `0`, raise a `ValueError` exception with the message `height must be >= 0`
- Instantiation with optional `width` and `height`: `def __init__(self, width=0, height=0):`
- Public instance method: `def area(self):` that returns the rectangle area
- Public instance method: `def perimeter(self):` that returns the rectangle perimeter:
    - if `width` or `height` is equal to `0`, perimeter has to be equal to `0`
- `print()` and `str()` should print the rectangle with the character `#`: (see example below)
    - if `width` or `height` is equal to 0, return an empty string
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 3-main.py

#!/usr/bin/python3

Rectangle = __import__('3-rectangle').Rectangle


my_rectangle = Rectangle(2, 4)

print("Area: {} - Perimeter: {}".format(my_rectangle.area(),
my_rectangle.perimeter()))


print(str(my_rectangle))

print(repr(my_rectangle))


print("--")


my_rectangle.width = 10

my_rectangle.height = 3

print(my_rectangle)

print(repr(my_rectangle))
```

```
guillaume@ubuntu:~/0x08$ ./3-main.py
Area: 8 - Perimeter: 12
##
##
##
##
<3-rectangle.Rectangle object at 0x7f92a75a2eb8>
--
##########
##########
##########
<3-rectangle.Rectangle object at 0x7f92a75a2eb8>
guillaume@ubuntu:~/0x08$
```

- File: `3-rectangle.py`

4. Eval is magic (5.5pts)

Write a class `Rectangle` that defines a rectangle by: (based on `3-rectangle.py`)

- Private instance attribute: `width`:
  - property `def width(self):` to retrieve it
  - property setter `def width(self, value):` to set it:
    - `width` must be an integer, otherwise raise a `TypeError` exception with the message `width must be an integer`
    - if `width` is less than `0`, raise a `ValueError` exception with the message `width must be >= 0`
- Private instance attribute: `height`:
  - property `def height(self):` to retrieve it
  - property setter `def height(self, value):` to set it:
    - `height` must be an integer, otherwise raise a `TypeError` exception with the message `height must be an integer`
    - if `height` is less than `0`, raise a `ValueError` exception with the message `height must be >= 0`
- Instantiation with optional `width` and `height`: `def __init__(self, width=0, height=0):`
- Public instance method: `def area(self):` that returns the rectangle area
- Public instance method: `def perimeter(self):` that returns the rectangle perimeter:
  - if `width` or `height` is equal to `0`, perimeter has to be equal to `0`
- `print()` and `str()` should print the rectangle with the character `#`: (see example below)

    - if width or height is equal to 0, return an empty string
- repr() should return a string representation of the rectangle to be able to recreate a new instance by using eval() (see example below)
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 4-main.py
#!/usr/bin/python3
Rectangle = __import__('4-rectangle').Rectangle

my_rectangle = Rectangle(2, 4)
print(str(my_rectangle))
print("--")
print(my_rectangle)
print("--")
print(repr(my_rectangle))
print("--")
print(hex(id(my_rectangle)))
print("--")

# create new instance based on representation
new_rectangle = eval(repr(my_rectangle))
print(str(new_rectangle))
print("--")
print(new_rectangle)
print("--")
print(repr(new_rectangle))
print("--")
print(hex(id(new_rectangle)))
print("--")

print(new_rectangle is my_rectangle)
print(type(new_rectangle) is type(my_rectangle))

guillaume@ubuntu:~/0x08$ ./4-main.py
##
##
```

```
##
##
--
##
##
##
##
--
Rectangle(2, 4)
--
0x7f09ebf7cc88
--
##
##
##
##
--
##
##
##
##
--
Rectangle(2, 4)
--
0x7f09ebf7ccc0
--
False
True
guillaume@ubuntu:~/0x08$
```

File: `4-rectangle.py`

5. Detect instance deletion (5.5pts)

Write a class `Rectangle` that defines a rectangle by: (based on `4-rectangle.py`)

- Private instance attribute: `width`:

    - property `def width(self):` to retrieve it
    - property setter `def width(self, value):` to set it:
        - `width` must be an integer, otherwise raise a `TypeError` exception with the message `width must be an integer`
        - if `width` is less than `0`, raise a `ValueError` exception with the message `width must be >= 0`
- Private instance attribute: `height`:
    - property `def height(self):` to retrieve it
    - property setter `def height(self, value):` to set it:
        - `height` must be an integer, otherwise raise a `TypeError` exception with the message `height must be an integer`
        - if `height` is less than `0`, raise a `ValueError` exception with the message `height must be >= 0`
- Instantiation with optional `width` and `height`: `def __init__(self, width=0, height=0):`
- Public instance method: `def area(self):` that returns the rectangle area
- Public instance method: `def perimeter(self):` that returns the rectangle perimeter:
    - if `width` or `height` is equal to `0`, perimeter has to be equal to `0`
- `print()` and `str()` should print the rectangle with the character `#`:
    - if `width` or `height` is equal to 0, return an empty string
- `repr()` should return a string representation of the rectangle to be able to recreate a new instance by using `eval()`
- Print the message `Bye rectangle...` (`...` being 3 dots not ellipsis) when an instance of `Rectangle` is deleted
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 5-main.py

#!/usr/bin/python3

Rectangle = __import__('5-rectangle').Rectangle


my_rectangle = Rectangle(2, 4)

print("Area: {} - Perimeter: {}".format(my_rectangle.area(),
my_rectangle.perimeter()))


del my_rectangle


try:

    print(my_rectangle)

except Exception as e:

    print("[{}] {}".format(e.__class__.__name__, e))
```

```
guillaume@ubuntu:~/0x08$ ./5-main.py
Area: 8 - Perimeter: 12
Bye rectangle...
[NameError] name 'my_rectangle' is not defined
guillaume@ubuntu:~/0x08$
```

- File: `5-rectangle.py`

6. How many instances (5.5pts)

Write a class Rectangle that defines a rectangle by: (based on 5-rectangle.py)

- Private instance attribute: width:
  - property def width(self): to retrieve it
  - property setter def width(self, value): to set it:
    - width must be an integer, otherwise raise a TypeError exception with the message width must be an integer
    - if width is less than 0, raise a ValueError exception with the message width must be >= 0
- Private instance attribute: height:
  - property def height(self): to retrieve it
  - property setter def height(self, value): to set it:
    - height must be an integer, otherwise raise a TypeError exception with the message height must be an integer
    - if height is less than 0, raise a ValueError exception with the message height must be >= 0
- Public class attribute number_of_instances:
  - Initialized to 0
  - Incremented during each new instance instantiation
  - Decremented during each instance deletion
- Instantiation with optional width and height: def __init__(self, width=0, height=0):
- Public instance method: def area(self): that returns the rectangle area
- Public instance method: def perimeter(self): that returns the rectangle perimeter:
  - if width or height is equal to 0, perimeter has to be equal to 0
- print() and str() should print the rectangle with the character #:
  - if width or height is equal to 0, return an empty string
- repr() should return a string representation of the rectangle to be able to recreate a new instance by using eval()
- Print the message Bye rectangle... (... being 3 dots not ellipsis) when an instance of Rectangle is deleted
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 6-main.py
#!/usr/bin/python3
Rectangle = __import__('6-rectangle').Rectangle


my_rectangle_1 = Rectangle(2, 4)
my_rectangle_2 = Rectangle(2, 4)
print("{:d} instances of Rectangle".format(Rectangle.number_of_instances))
del my_rectangle_1
print("{:d} instances of Rectangle".format(Rectangle.number_of_instances))
del my_rectangle_2
print("{:d} instances of Rectangle".format(Rectangle.number_of_instances))


guillaume@ubuntu:~/0x08$ ./6-main.py
2 instances of Rectangle
Bye rectangle...
1 instances of Rectangle
Bye rectangle...
0 instances of Rectangle
guillaume@ubuntu:~/0x08$
```

- File: `6-rectangle.py`

7. Change representation (5.5pts)

Write a class Rectangle that defines a rectangle by: (based on 6-rectangle.py)

- Private instance attribute: width:
  - property def width(self): to retrieve it
  - property setter def width(self, value): to set it:
    - width must be an integer, otherwise raise a TypeError exception with the message width must be an integer
    - if width is less than 0, raise a ValueError exception with the message width must be >= 0
- Private instance attribute: height:
  - property def height(self): to retrieve it
  - property setter def height(self, value): to set it:
    - height must be an integer, otherwise raise a TypeError exception with the message height must be an integer

        - if height is less than 0, raise a ValueError exception with the message height must be >= 0
- Public class attribute number_of_instances:
    - Initialized to 0
    - Incremented during each new instance instantiation
    - Decremented during each instance deletion
- Public class attribute print_symbol:
    - Initialized to #
    - Used as symbol for string representation
    - Can be any type
- Instantiation with optional width and height: def __init__(self, width=0, height=0):
- Public instance method: def area(self): that returns the rectangle area
- Public instance method: def perimeter(self): that returns the rectangle perimeter:
    - if width or height is equal to 0, perimeter has to be equal to 0
- print() and str() should print the rectangle with the character(s) stored in print_symbol:
    - if width or height is equal to 0, return an empty string
- repr() should return a string representation of the rectangle to be able to recreate a new instance by using eval()
- Print the message Bye rectangle... (... being 3 dots not ellipsis) when an instance of Rectangle is deleted
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 7-main.py
#!/usr/bin/python3
Rectangle = __import__('7-rectangle').Rectangle


my_rectangle_1 = Rectangle(8, 4)
print(my_rectangle_1)
print("--")
my_rectangle_1.print_symbol = "&"
print(my_rectangle_1)
print("--")


my_rectangle_2 = Rectangle(2, 1)
print(my_rectangle_2)
print("--")
Rectangle.print_symbol = "C"
print(my_rectangle_2)
```

```
print("--")


my_rectangle_3 = Rectangle(7, 3)
print(my_rectangle_3)


print("--")


my_rectangle_3.print_symbol = ["C", "is", "fun!"]
print(my_rectangle_3)


print("--")


guillaume@ubuntu:~/0x08$ ./7-main.py
########
########
########
########
--
&&&&&&&&
&&&&&&&&
&&&&&&&&
&&&&&&&&
--
##
--
CC
--
CCCCCCC
CCCCCCC
CCCCCCC
--
['C', 'is', 'fun!']['C', 'is', 'fun!']['C', 'is', 'fun!']['C', 'is', 'fun!']['C',
'is', 'fun!']['C', 'is', 'fun!']['C', 'is', 'fun!']
['C', 'is', 'fun!']['C', 'is', 'fun!']['C', 'is', 'fun!']['C', 'is', 'fun!']['C',
'is', 'fun!']['C', 'is', 'fun!']['C', 'is', 'fun!']
```

```
['C', 'is', 'fun!']['C', 'is', 'fun!']['C', 'is', 'fun!']['C', 'is', 'fun!']['C',
'is', 'fun!']['C', 'is', 'fun!']['C', 'is', 'fun!']

--

Bye rectangle...

Bye rectangle...

Bye rectangle...

guillaume@ubuntu:~/0x08$
```

- File: `7-rectangle.py`

8. Compare rectangles (5.5pts)

Write a class Rectangle that defines a rectangle by: (based on 7-rectangle.py)

- Private instance attribute: width:
  - property def width(self): to retrieve it
  - property setter def width(self, value): to set it:
    - width must be an integer, otherwise raise a TypeError exception with the message width must be an integer
    - if width is less than 0, raise a ValueError exception with the message width must be >= 0
- Private instance attribute: height:
  - property def height(self): to retrieve it
  - property setter def height(self, value): to set it:
    - height must be an integer, otherwise raise a TypeError exception with the message height must be an integer
    - if height is less than 0, raise a ValueError exception with the message height must be >= 0
- Public class attribute number_of_instances:
  - Initialized to 0
  - Incremented during each new instance instantiation
  - Decremented during each instance deletion
- Public class attribute print_symbol:
  - Initialized to #
  - Used as symbol for string representation
  - Can be any type
- Instantiation with optional width and height: def __init__(self, width=0, height=0):
- Public instance method: def area(self): that returns the rectangle area
- Public instance method: def perimeter(self): that returns the rectangle perimeter:
  - if width or height is equal to 0, perimeter has to be equal to 0
- print() and str() should print the rectangle with the character #:
  - if width or height is equal to 0, return an empty string

- repr() should return a string representation of the rectangle to be able to recreate a new instance by using eval()
- Print the message Bye rectangle... (... being 3 dots not ellipsis) when an instance of Rectangle is deleted
- Static method def bigger_or_equal(rect_1, rect_2): that returns the biggest rectangle based on the area
    - rect_1 must be an instance of Rectangle, otherwise raise a TypeError exception with the message rect_1 must be an instance of Rectangle
    - rect_2 must be an instance of Rectangle, otherwise raise a TypeError exception with the message rect_2 must be an instance of Rectangle
    - Returns rect_1 if both have the same area value
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 8-main.py
#!/usr/bin/python3
Rectangle = __import__('8-rectangle').Rectangle

my_rectangle_1 = Rectangle(8, 4)
my_rectangle_2 = Rectangle(2, 3)

if my_rectangle_1 is Rectangle.bigger_or_equal(my_rectangle_1, my_rectangle_2):
    print("my_rectangle_1 is bigger or equal to my_rectangle_2")
else:
    print("my_rectangle_2 is bigger than my_rectangle_1")



my_rectangle_2.width = 10
my_rectangle_2.height = 5
if my_rectangle_1 is Rectangle.bigger_or_equal(my_rectangle_1, my_rectangle_2):
    print("my_rectangle_1 is bigger or equal to my_rectangle_2")
else:
    print("my_rectangle_2 is bigger than my_rectangle_1")

guillaume@ubuntu:~/0x08$ ./8-main.py
my_rectangle_1 is bigger or equal to my_rectangle_2
my_rectangle_2 is bigger than my_rectangle_1
```

```
Bye rectangle...
Bye rectangle...
guillaume@ubuntu:~/0x08$
```

- File: `8-rectangle.py`

9. A square is a rectangle (5.5pts)

Write a class `Rectangle` that defines a rectangle by: (based on `8-rectangle.py`)

- Private instance attribute: `width`:
  - property `def width(self):` to retrieve it
  - property setter `def width(self, value):` to set it:
    - `width` must be an integer, otherwise raise a `TypeError` exception with the message `width must be an integer`
    - if `width` is less than `0`, raise a `ValueError` exception with the message `width must be >= 0`
- Private instance attribute: `height`:
  - property `def height(self):` to retrieve it
  - property setter `def height(self, value):` to set it:
    - `height` must be an integer, otherwise raise a `TypeError` exception with the message `height must be an integer`
    - if `height` is less than `0`, raise a `ValueError` exception with the message `height must be >= 0`
- Public class attribute `number_of_instances`:
  - Initialized to `0`
  - Incremented during each new instance instantiation
  - Decremented during each instance deletion
- Public class attribute `print_symbol`:
  - Initialized to `#`
  - Used as symbol for string representation
  - Can be any type
- Instantiation with optional `width` and `height`: `def __init__(self, width=0, height=0):`
- Public instance method: `def area(self):` that returns the rectangle area
- Public instance method: `def perimeter(self):` that returns the rectangle perimeter:
  - if `width` or `height` is equal to `0`, perimeter has to be equal to `0`
- `print()` and `str()` should print the rectangle with the character `#`:
  - if `width` or `height` is equal to 0, return an empty string
- `repr()` should return a string representation of the rectangle to be able to recreate a new instance by using `eval()`
- Print the message `Bye rectangle...` (`...` being 3 dots not ellipsis) when an instance of `Rectangle` is deleted

- Static method `def bigger_or_equal(rect_1, rect_2):` that returns the biggest rectangle based on the area
    - `rect_1` must be an instance of `Rectangle`, otherwise raise a `TypeError` exception with the message `rect_1 must be an instance of Rectangle`
    - `rect_2` must be an instance of `Rectangle`, otherwise raise a `TypeError` exception with the message `rect_2 must be an instance of Rectangle`
    - Returns `rect_1` if both have the same area value
- Class method `def square(cls, size=0):` that returns a new Rectangle instance with `width == height == size`
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x08$ cat 9-main.py
#!/usr/bin/python3
Rectangle = __import__('9-rectangle').Rectangle


my_square = Rectangle.square(5)
print("Area: {} - Perimeter: {}".format(my_square.area(), my_square.perimeter()))
print(my_square)


guillaume@ubuntu:~/0x08$ ./9-main.py
Area: 25 - Perimeter: 20
#####
#####
#####
#####
#####
Bye rectangle...
guillaume@ubuntu:~/0x08$
```

Good Luck