Simon Fraser University
School of Computing Science

Lab Week 8

(No submission required for this lab.)

Objective

In this week's lab, our objective is to understand the following concepts and practise creating/using them in Python programs:

1. Practise creating user-defined functions
2. Explore the Turtle Graphics module (in preparation for our Assignment 4)

We shall continue practising creating user-defined functions and Python programs that incorporate user-defined functions in our next lab.

Notes

1. For this lab, you are free to work on our own or with a partner.
2. This is a long lab. If you do not have enough time to finish these lab exercises within our lab session, no problem! You can finish the lab on our own outside the lab session either
   a. by coming back to CSIL and using a CSIL workstation in a lab room where there is no scheduled lab taking place, or
   b. by using our own computer at home or on campus or elsewhere.
3. Let's make sure we show our work to one of the TA's in the lab to ensure we understand the concepts (see Objectives above) exercised in this lab.

Lab Prep

Once we have logged into a CSIL Windows workstation, let's access our **U: drive/CMPT120** and create a directory called **Lab5**. Let's store any files we create as part of this lab into this directory.

Exercise 1 – User-defined functions

In all the exercises of this lab, feel free to use the Python visualizer (see its link on the Home page of our course web site under the Code and Examples column of our Lecture 18) to view the execution flow as it executes our functions (as we have done in class).

1. Read Section 3.5 from our online textbook and do the exercises described in the last two paragraphs of this section.

2. Let's have a look at the function called `printString( )` defined below:

   Function definition:

```
def printString():
     for index in 'Spam':
          print(index, end="")
```

   a) Copy the function definition above into a Python program and call this function from the main part of the program to make sure it works.

   b) Write a more general version of `printString( )`, i.e., a function that takes a string as a parameter. In doing so, we are using the **generalisation** guideline. What is being generalised? How are we generalising it?

      Then, create another function called `printTwice()` that prints the string twice by calling our new and more generalised version of `printString()`. **The body of this function** `printTwice()` **must only have two statements** + the `return` statement.

c) Define a new function called `printFour()` that takes a string as a parameter and prints this string four times. **The body of this function** `printFour()` **must only have two statements** (+ the `return` statement), **not four** (+ the `return`statement).

d) Can we generalise `printFour()` by creating a function (we will need to rename it) that prints a string **x** times?

3. Exploration –> Creating Python programs that are composed of functions.

Consider the following problem statement:

Problem statement: Develop a Python program to compute the area of various shapes: triangle, circle, rectangle, square, ellipse.

In our lecture Friday, we shall go over the design and implementation of its solution. For now, let's have a look at the first version of the solution called **`AreaCalculator_1.py`** posted on the Home page of our course web site under the Code and Examples column of our Lecture 18. Let's read its header comment block. Does what it says makes sense to us? Let's have a look at the code of this Python program. Can we understand what it does?

Is there a syntax error in **`AreaCalculator_1.py`**? Does it execute? Can we come up with a few test cases?

Exercise 2 – Explore the Turtle Graphics module

We had a demo in class illustrating what this Turtle thing was. However, since we are not covering the Turtle module in class (aside from the demo), one of the goals of this lab is to further introduce us to it and allowing us to explore it. Getting to know the Turtle module in this lab will prepare us for our next course activities.

As part of our exploration, let us have a look at this web site:

- https://docs.python.org/3.5/library/turtle.html

Since Turtle is a Python module, let's first start by looking at Python modules.

Importing modules and using their built-in functions:

1. In our programs, when we need to call (make use of) built-in functions that are located in a module, we must first import this module. We do so by using an `import` statement, for example, `import turtle`, which we place below the header comment block and above our function definitions.

   Placing our `import` statements in this location allows us to use the built-in functions of the module(s) everywhere in our program: from our `import`statement all the way down to the end of our program, in other words, in the "# Main" part of our program, (also called "the top level" of our program) and inside any functions we have defined below our `import` statement(s).

   In a nutshell, as soon as a function is defined (or imported as part of a module), it can be used in the rest of our program.

2. So far, we have used the `import <moduleName>` syntax to import modules. Here are other syntaxes we can use:

   - `import <moduleName> as <short name given to module>`
   - `from <moduleName> import <functionName>`

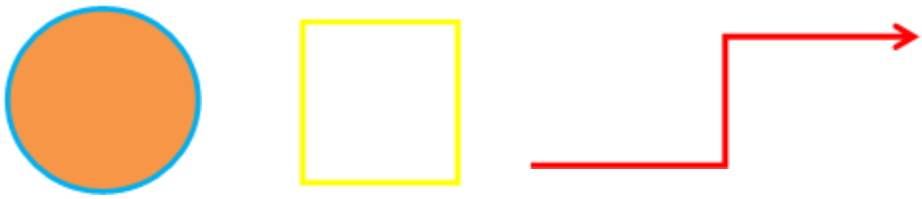
   We can therefore import the Turtle Graphics module in a variety of ways:

   - `import turtle`
     and call its functions as follows: `turtle.forward(100)`
   - `import turtle as t`
     and call its functions as follows: `t.forward(100)`
     recommended when using the Turtle module – can you see why?
   - `from random import randint`
     and call its functions as follows: `number = randint(1,10)`

Exploration Exercises:

1. Download, test and play with the **Turtle_Lab_5.py** program file.

2. Create a Python program that draws the following shapes using the built-in Turtle module functions.



About colours:

There are several options for manipulating colours. Assuming we included the statement `import turtle as t` in our Python program:

a. We can state the colour as a string such as `"blue"`, `"red"`, etc. in any of the functions requiring a colour as an argument such as the function `pencolor( )`: `t.pencolor("red")`

b. We can also state colours as a 3 integer (decimal) value, where the first value indicates the amount of red (R), the second value indicates the amount of green (G), and the third, the amount of blue (B). We indicate each amount by using an integer ranging from 0 (the colour is not present) to 255 (the colour is fully present). If we want to use this colour coding scheme, we first must include the statement `t.colormode(255)` in our program.

Here is an example:

```
t.colormode(255)
red = 34     # Number between 0 and 255
green = 200  # Number between 0 and 255
blue = 27    # Number between 0 and 255
col = (red,green,blue)
t.pencolor(col)
```

See **Turtle_Lab_5.py** for another example.

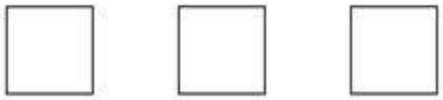
The following web page is very useful when we want to describe a colour by its RGB (red/green/blue) components:

- colour codes: http://www.rapidtables.com/web/color/RGB_Color.htm

Looking at this web page, can we figure out which colour we obtain when using the RGB values in the above example.

3. Let's add some code to the main part of our Python program such that it now draws a black square. Let's make sure we use a `for` loop to draw a black square.

   Now, let's be bad ☺ and copy/paste this code twice such that our program produces the black squares illustrated below.



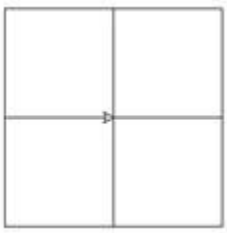
   We will have to add some code to move our turtle such that it draws the black squares in the three locations illustrated above.

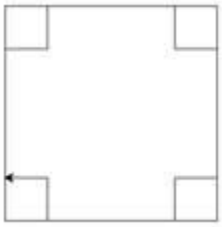
4. Let's now create a function by **encapsulating** (this word is defined in Chapter 4 of our online textbook) the code that draws one square into a function. Let's make our function such that it does not take any parameters (yet) and it does not return anything. Let's modify the main part of our program such that it calls this function three times (still producing the three black squares illustrated above.

5. Let's modify our function such that it now takes a parameter, i.e., a colour. And as above, let's call this function three times to create the shapes below, each with its particular colour.

6. Can we generalize our square function further by adding the length of the sides as a second parameter?

7. Let's now add some more code to the Python program we have created so far. This new code and functions must draw the shapes illustrated below in any colour.



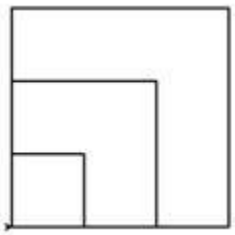


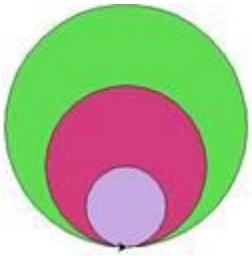
Let's see if we can write the code without using `for` loops first, then let's rewrite the code using `for` loops. Can we see the benefit of using `for` loops?

8. Let's add a final code fragment to the Python program we have created so far. This new code fragment must draw the following shapes in any colour using `for` loops and functions.

Hint: Can we reuse our function `square(colour, side)`.

Also, let's implement a `circle(…)` function with the radius of the circle and its colour as parameters.





We should now be "well equipped" to do our Assignment 4. ☺

Have fun!