CSCI 2202 Computer Modelling for Scientists Lab 1
Introduction to Programming with Turtle Geometry

Lab Goals:

- Gain familiarity with IDLE.
- Learn about assignments and objects.
- Learn about the looping constructs in Python

Part 1: Using IDLE

(1) Start IDLE.

(2) Consider throwing a ball up in the air. The vertical position of the ball y changes with time time according to:

$y(t) = v_0 t - \frac{1}{2}gt^2$ (ignoring air resistance)

Here v_0 is the *initial velocity*, g is the *acceleration due to gravity*, and t is the *time*. Given $v_0 = 5m/s$, $g = 9.81m/s^2$, we want to compute the position of the ball at $t = 0.6s$.

(3) Enter the following formula in your IDLE prompt and press Enter:

```
>>> 5 * 0.6 - 0.5 * 9.81 * 0.6**2
```

Note: $(0.6)^2$ **is written as 0.6**2**. This simply uses the *python* interpreter as a calculator. Now, if you wanted to find the y position of the ball at $t = 1s$, you would have to type in the formula once more.

However, there is an alternative: Use *variables* in the program (specifically v_0, t and g).

(4) From the IDLE menu click on: (`File` → `New File`) and type in the following:

```
# Create a variable for storing initial velocity in m/s
v_0 = 5

# A variable for storing the acceleration due to gravity in m/s^2
g = 9.81

# Create a variable for storing the desired time in seconds
t = 0.6

# Calculate the ball position in metres
y = v_0 * t - 0.5 * g * t**2
```

```
# Inform the user of the current ball position
print( "At t = " + str(t) + "the position of the ball is" + str(y))
```

then save the file as `ball1.py`. This is your first program. You can run your program from the menu: `Run` $\rightarrow$ `Run Module` or press F5 on your keyboard. Now, to find the position for any *other* value of t, change the line $t = 0.6$ so that there is a new value assigned.

The "#" above each assignment is a *comment*. It is there to remind you what some code (in this case a variable assignment) is doing. Everything after the "#", to the end of the line, is ignored by the interpreter. Comments should be used to make clear what the code is doing. Commenting is good programming practice.

In this program, the answer is *assigned* to a *variable* `y`. `print(...)` only prints String objects *i.e.* `str(..)`. The `y` variable is turned into a *string* object and *concatenated* with other strings using the `+` (the concatenation) operator for strings. One could simply use `print(str(y))`. The strings within quotes are*literals*. Such strings are printed as is. They are inserted to make the output more readable.

There are other (equivalent) ways to print:
`print( "At t = " , t , " the position of the ball is ", y)`. (Note the commas)
Ex. 1 For the next exercise, create a program `c2f.py` to convert temperatures from Celsius `(C)` degrees into corresponding Fahrenheit `(F)` degrees with the formula: $F = \frac{9}{5}C + 32$. The output of your program should be of form:

```
16 degrees Celsius is 60.8 degrees Fahrenheit.
```

Once the above program is running properly, you can instead prompt the user for a temperature after the program is running. To do this, replace the variable assignment `C = 16.0` line with:

```
C = float(input(‘‘Enter a temperature in degrees C: ’’))
```

... *contd.* $\rightarrow$

Part 2: Turtle Graphics:

- Python has a built-in module that supports *turtle graphics* called the "turtle" module. The module provides an environment that allows you to create and control the movements of a `turtle` object on a screen. The `turtle` has a pen that draws the path it is instructed to follow using *python* commands. We use the turtle module to draw simple geometric shapes. These exercises allows us to explore the fundamental constructs of programming in an entertaining way.
- Create a new Python file. Name it anything except `turtle.py`
- First we have to make sure Python knows we want to use the commands found in the Turtle module. To do this, we include the following line at the top of our file:

```
import turtle
```

- Next, we need to create a window and a `turtle` to place in the window. To do so, we use the following Python commands:

```
# Create a window and store it for later use
turtleWindow = turtle.Screen()

# Optional: Set the name of the turtle's window
turtleWindow.title(``Turtle Drawing Demo'')

# Create a turtle named mickey in the shape of a turtle
mickey = turtle.Turtle(shape=''turtle'')
```

- Time to draw some interesting things!
  The `turtle` module has (among other things), a collection of functions defined to control a 'turtle' object's motion. Some of these functions have *synonyms*: *e.g.* `forward( d )` & `fd( d )` to move the turtle a distance d, in the direction the turtle is facing.
  To have the turtle object (say) `mickey`, have one of these functions operated on it, write: `mickey.functionName(value)`. For example, `mickey.setposition(-200, 100)` moves *mickey* to the location $x = -200, y = 100$ on the screen.
  Once the *turtle* module is loaded, the following functions available to you:
  `forward( d )/fd( d )` moves turtle in the direction it's facing by `d` units.
  `backward( d )/bk( d )` will move your turtle backwards `d` units.
  `left( d )/ lt( d )` rotates turtle `d` degrees *counter-clockwise.*
  `right( d )/rt( d )` rotates turtle `d` degrees clockwise.
  `penup( )/pendown( )` tells turtle to stop/start drawing lines resp.
  By default, a newly created turtle will be in pendown() mode.
  `setposition( x, y )/ goto( x, y )` sends turtle to the position (x, y).

The two instructions shown below, draw a line and turn the turtle counter-clockwise by 90^o.

```
mickey.fd(100)
mickey.lt(90)
```

By default, the turtle starts at (0,0), at the centre of the window.

Moving East/North is to move along $+x/+y$ direction and to move West/South is to move along $-x/-y$ For example, `mickey.setposition(10,-40)` will make your turtle move East and South. (However, unless you insert `penup()` before the `setposition (...)` command, your turtle will to draw a line (from the its initial position to $(10,-40)$.
To end your turtle program, make sure to include `turtle.done()` as the last command in your file to let Python know you're done drawing with your turtle.

- Create a new file, type the commands in the order described above and run the program. Experiment by adding 4 or 5 more `forward(), back(), left() and right()` commands. Add the command `mickey.pencolor("blue")` before you move your turtle. What happens if you remove `shape=''turtle''` from the `turtle.Turtle()` function?

Deliverables: The code for each exercise should be in a separate file.

Put all your solutions in a folder titled: `Lab1_<name>_<B00number>`

Ex. 2 Once you have a grip on the above instructions, make a program to draw a square of size 100, each of whose sides are a different colour. Use *only* the commands shown above (you can cut-and-paste). Save it in a file called `square.py`. Show the result to the T.A. before proceeding.

Ex. 3 It is tedious to repeat the same commands. Change your program to use a loop for the repeated commands in *Ex. 1.*

Ex. 4 Make a program to draw a regular hexagon (a six-sided polygon).

Try working on the first problem for the 2^{nd} lab:

Ex. 1 (a) Start by modifying `polygon.py` from the last lab. The modified program should ask the user to input a value for the variable `numSides`.

(the Python command `numSides = int(input(''Enter the number of sides ''))` *does the job.* The `input` always reads the user input as a *String.* The `int` to the left of the `input` *casts* the input string into an integer.)