QBUS6850: Tutorial 1 – Getting Started

Objectives

- To configure and become familiar with Anaconda and Spyder;
- To learn fundamental Python programming concepts;

1. Anaconda Navigator and Spyder

First open Anaconda Navigator then click the “Launch” button for Spyder Spyder will open to the following screen.

We assume students have certain very basic knowledge or skills of Python. However tutorial 1 will help students review some important programming concepts related to this course.

If students are familiar with all these, students are encouraged to complete assign tasks straightaway.

2. Python Scripts

So far you’ve used the interactive console. The console is great for quick testing and checking variables but for more complex tasks it would be better to store your code so that you can use or run it later.

Let’s create a Python script. A Python script is just a text file that contains Python code, which Python will read and execute line by line.

In Spyder go to File > New File. This will create a new empty Python script in the Script Editor (left pane of Spyder).

In the file add some Python code at the end. For example, the Hello World print function.

```
my_string = "Hello World!"
print(my_string)
```

Then click the green Run (play icon). The output will be displayed on the console. From now on we will work with Python Scripts.

3. List

Lists are a way to store an ordered collection of variables or objects. Let’s create a list and store it in the variable my_list. Type the following code in Spyder Console window

```
my_list = [1, 2, 4, 8, 16]
```

```
print(my_list)
```

Write down what happened here.

List items can be accessed by their index. Python uses zero-indexing which means the first item starts at 0. Print the value of the first item:

```
print(my_list[0])
```

To add items to the list you can use the append function

```
my_list.append(32)
```

To find out how many items are in the list use the len function

```
print(len(my_list))
```

To create an empty list, you can use the list constructor function then append items later

```
my_new_list=list()
my_new_list.append(10)
my_new_list.append(12)
print(my_list)
```

4. Conditionals

Most of time, we want to execute a different piece of code based on the current state or value of a variable? This is where the conditional statements if, elif and else come into play. These statements allow us to branch the execution path. Copy the following into your script file, then run your script file.

```
#Accept input, tehn convert input string to integer
input_number = int(input(“Enter a number:  ”))
if input_number > 3
   print(“number is greater than 3”)
elif input_number == 3
   print(“number is equal to 3”)
else
   print(“number is less than 3”)
```

Note: The input function waits to the user to enter a string that ends in a return or new line character. The int function converts the input string to an integer value.

5. Loops

A fundamental component of computer programming is iteration. In other words the ability to do repeated tasks.

In Python we do this using loops. There are two types of loops:

- For loop
- While loop

Let's start with an example of the For loop by iterating over our list of numbers that we created earlier and

```
for number in my_list:
    print(number)
```

You can also use list indexing to do the same thing. Try this

```
for i in range(len(my_list)):
    print(my_list[i])
```

While loops are a little more complicated. They continue iterating until a conditional statement becomes false. Try the following

```
Count = 0
while (count < len(my_list)):
    print(my_list[count])
    count = count + 1
```

Note: Be careful with loops (particularly while loops) as you may accidentally create an infinite loop which Python can't escape from. You will need to quit Spyder to stop

6. Functions

Functions in Python (and other programming languages) are based on functions from mathematics. They encapsulate a piece of repeated functionality and they allow optional input and produce optional output.

Functions are a great way to divide your code into reusable building blocks, which makes your code more readable and may save you time and effort.

Try this example

```
def add(a,b):
    return a + b

c = add (10,5)
print(c)
```

The function called "add" has a two parameters which we called "a" and "b". Inside the function Python will replace each occurrence of the parameters with the values that we specified. You need know how to define a function in Assignment 1.

Note: def keyword means "define"

7. String Formatting

A common task you may encounter is outputting strings that are nicely formatted. You can use the format function of string objects to help you.

The format function scans the input string for curly braces “{ }”. Each curly brace has a number. This number is an index into the parameter list of the format function. In this example {0} corresponds to “name” and {1} corresponds to age.

```
Name = input(“What is your name?  “)
Year = int(input(“Enter your birth year:  ”))
Age = 2017 - Year

fancy_string = “Hi {0}, you are {1} years
                old.”.format(Name, Age)

Print(fancy_string)
```

Note:
- The . (dot) syntax calls a function belonging to that object. In this example we are calling the “format” function belonging to the string object.
- The format function replaces occurrences of numbered curly braces with corresponding parameter values.

8. Classes and Objects (Advanced topics)

An object is a container that represents a “thing”. Inside the container there are:
- Functions
- Attributes

Imagine we want to represent a person in our program. We might define a “person” class. People have attributes like age, height, weight, name etc. They also “do” things, in other words they have functionality. Some function examples might be: eat, sleep, walk and study.

Lets look at a more practical example: a customer’s bank account. We need to keep track of the balance and which customer it belongs to. We also need to be able to withdraw and deposit money.

```
class Customer(object):
    #A bank customer
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def __str__(self):
        return “Account for {0}, balance of
```

```
                {1}”.format(self.name, self.balance)

    def withdraw(self, amount):
        self.balance = self.balance – amount
        return self.balance

    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance
```

Note: The functions that start and end with double underscores (__) are special functions in Python. The _init_ function (initialize) is called the class constructor. It gets called when you create the object. The _str_ function (string) is called when printing the object.

Create an instance of the customer class by calling the constructor and specifying a name (use your own name) and a starting balance

```
new_customer = Customer(“Steve”, 1000.00)
```

You can then access the attributes of the object by using the dot notation.

```
print(new_customer.balance)
print(new_customer.name)
```

We have also explicitly define the string function so you can try

```
Print(new_customer)
```

Accessing the functions of an object is much the same

```
print(new_customer..withdraw(100))
print(new_customer.balance)
```

More info at https://docs.python.org/3/tutorial/classes.html