Lab 1: Python Basic (Part A)

Section 1. What is Python?

Python is a very-high-level programming language. It supports common elementary data types such as numbers and characters, as well as high-level built-in data types such as sequences and dictionaries. It comes with many standard modules and third-party libraries. It is an interpreted language, which means you can run your Python statements or program directly without compilation. Python is freely available on Windows, Unix/Linux, and Mac OS. In short, Python is simple to learn yet very powerful and popular in data analysis and machine learning.

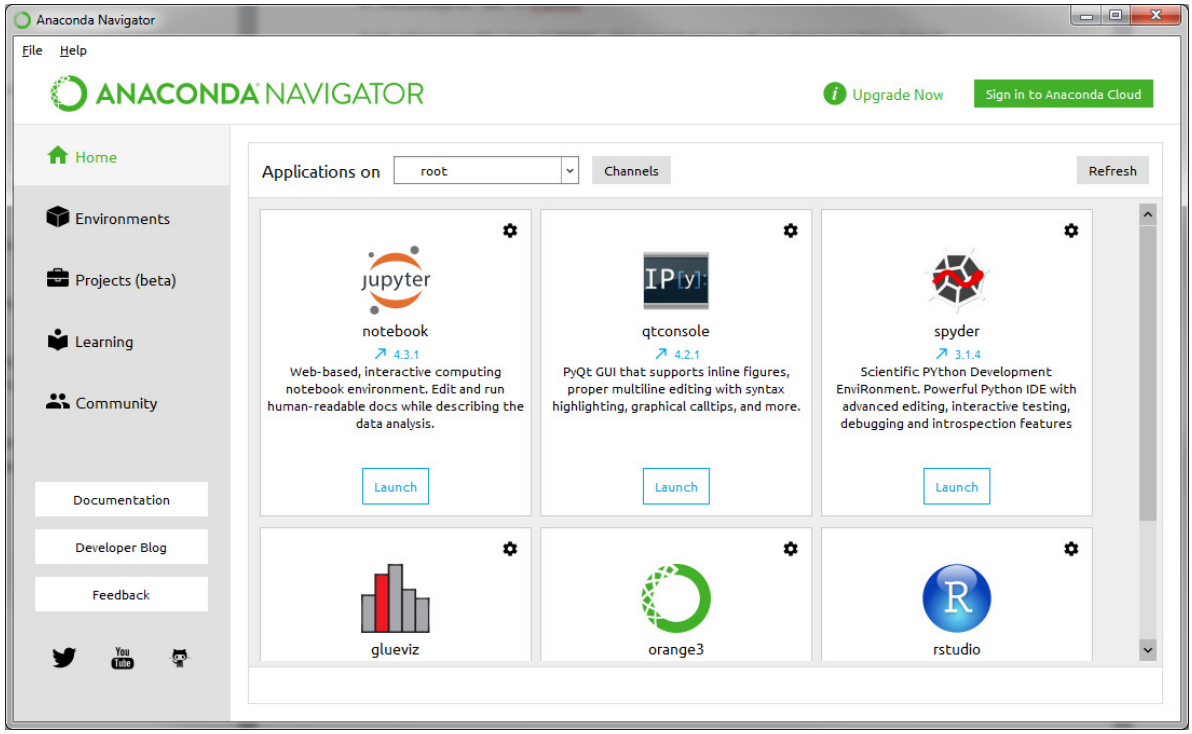
As an evolving programming language, Python has different versions. Python 2.x becomes legacy (yet popular), whilst Python 3.x is the current major version. This course will use **Python 3.6**.

This course will use **Anaconda** and **Spyder** as the Python development environment.

- **Anaconda** (https://anaconda.org/) is an open source software to manage Python development environments for Windows, Linux, and Mac OS. It provides easy management of a large collection of Python libraries.
- **Spyder** (https://pythonhosted.org/spyder/) is an open-source **integrated development environment** (IDE) for Python programming. You can use it to edit, test, and debug Python programs.

Now let's begin the journey of Python.

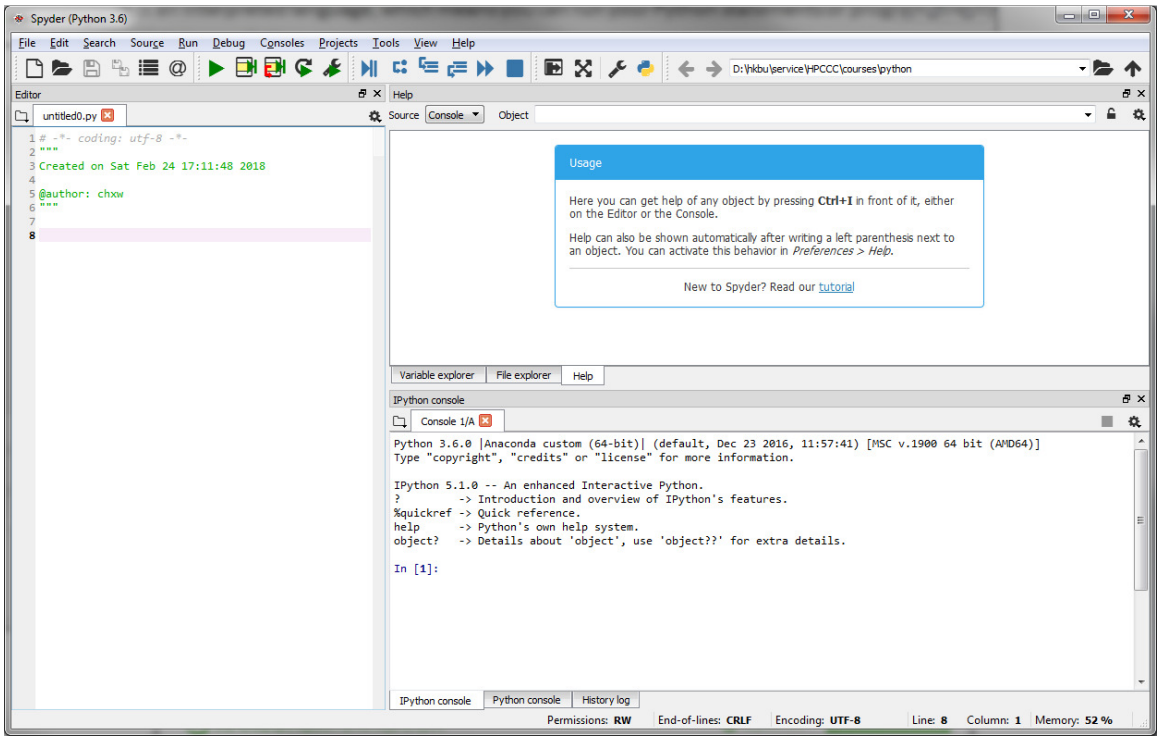
Step 1. Open Anaconda Navigator. You should see a window like the following one:



There are five items on the left panel: Home, Environments, Projects (beta), Learning, and Community. Let's introduce some of them:

- Home: you can launch different IDEs to develop your Python programs, including Spyder.
- Environments: you can manage different development environments for your projects. E.g., the default root environment is using Python 3.6. If you also need to use Python 2.7 for another project, you can create a new environment that uses Python 2.7.
- Learning: You can find a plenty of learning materials and documents about Python.

Step 2. Go back to "Home" and launch Spyder. You should see the Spyder window similar to the following one:



The main Spyder window contains three panes:

- **Editor** pane (left side), where you can edit your Python programs
- The right top pane is shared by "**Variable explorer**"/"**File explorer**"/"**Help**"
- **IPython console** pane (right bottom), the interactive Python interpreter to run your Python statements

Below are some important facts and concepts that you should know:

- There is a **working directory** textbox on the right top, which helps you to locate your python files or data. Use it together with the "File explorer" pane.
- In the "**File explorer**", if you double-click a Python file (or other data files with a supported format like .txt, .csv, etc.), it will be opened and shown in the **Editor pane**.
- **IPython** is the most popular interactive Python interpreter that can improve your working productivity. In the IPython console, you can do the following tasks:
    - Execute any Python statement by typing it and pressing the Enter-key;

- Execute a Python program (.py file) using the **%run** command;
- Execute a code segment from the clipboard by Copy-and-Paste;
- Interact with the operating system through commands like %**pwd**, %**ls, %cd, %mkdir, etc.**;
- IPython supports **tab completion** and **command history**.

Step 3. The concept of "cell" in Spyder Editor

In Spyder editor, a Python program can be divided into many cells. Each **cell** begins with a line of "**#%%**". Click anywhere in a cell, then press "**Ctrl + Enter**". The Python statements in the cell will be executed in IPython console – this is a shortcut of "Copy-and-Paste". This feature can save your time of typing codes in IPython console.

Before we go ahead, let's introduce a few important concepts in Python language:

- ***Comments***: Any text preceded by the sharp sign # is a comment, and will be ignored by the Python interpreter. So, the line of "**#%%**" is just a special comment.
- ***Indentation*****:** Python uses whitespaces to organize its codes. A **colon** ":" denotes the start of an indented code block, after which all the statements must be indented by the same amount until the end of the code block. In practice, we recommend **4** whitespaces or a tab space as the indentation.

Comparison between Python and other programming languages:

Python language	Another language that uses braces { } to organize the code blocks:

# ␣ represents a whitespace 	
for x in array:
␣␣␣␣if x < pivot:
␣␣␣␣␣␣␣␣less.append(x)
␣␣␣␣else:
␣␣␣␣␣␣␣␣greater.append(x)
	for x in array {
	  if x < pivot {
	  less.append(x)
	  } else {
	  greater.append(x)
	  }
	}

- ***Function*****:** A function is a block of statements that perform a particular action. Once you define a function, you can reuse it by calling the function name. Python supports **built-in functions** and **user-defined functions**.

Example 1.1: Running a user-defined function **hello**() in IPython:

```
#%%
def hello():
    print("Hello, Python!")
```

In this example, we define a Python function named **hello()**, which simply calls a built-in library function **print()** to display a **string** "Hello, Python!" on the console. You can use "**Ctrl + s**" to save your Python file.

Use your mouse to click anywhere in the cell of **hello()**, then press "**Ctrl + Enter**". You will see that the content of **hello()** has been copied into the IPython console. Now click in the IPython console.

In the IPython console, type "**hello()**". You will see that the **hello()** function is being executed: the message "Hello, Python!" is shown in the console. Try to type "**hello**" in the IPython console and see what will happen.

```
In [1]: def hello():
   ...:     print("Hello, Python!")

In [2]: hello()
Hello, Python!
```

Trial-and-Error Exercise 1.1:

In the IPython console, what will happen if you input "hello" instead of "hello()"?

Trial-and-Error Exercise 1.2:

Type in the following function in the Editor and try to execute it in the IPython console by "Ctrl + Enter", what will happen? Notice that there are 4 whitespaces in the first print() statement but 3 whitespaces in the second print() statement.

```
#%%
def hello():
    print("Hello, Python!")
   print("again")
```

Example 1.2: Running a user-defined function **inputName**() in IPython:

```
#%%
def inputName():
    name = input("What's your name? ")
    print(name)
```

In this example, we define a function named **inputName()**, which performs two actions: (1) it calls a built-in library function **input()** to prompt the user to type in some content after a **string** "What's your name?", and assign the content to a variable **name**; (2) it then calls **print**() function to show the content of variable **name** on the console. Notice that you don't need to specify the **data type** of variable name, which is different from many other programming languages.

Run function inputName() in the IPython console as follows:

1. Click anywhere in the cell of inputName(). Press "Ctrl + Enter".
2. In the IPython console, type **inputName()**.
3. You can also try to type **inputName** (without typing the parenthesis) in the IPython console and see what will happen.

Trial-and-Error Exercise 1.3:

Revise the function of **inputName()** by replacing the **print(name)** statement with **print("name")**. Run the function **inputName()** in IPython console and see what will happen.

Discussion: in the Python source code of function **inputName()**, what's the difference between **name** and **"name"**?

Section 2. Python Basics

In this section, we are going to learn some fundamental programming concepts in Python.

Constants, Variables, and Data Types

A computer program usually needs to use some data that cannot be changed (or **immutable**). These data are called **constants**. In Python, **literal constants** include numbers and strings. E.g.,

- Integer constants: 1, 3, 100, -20
- Floating point constants: 3.14, 155.9, 1.72E-3 (i.e., the scientific notation of 1.72×10^{-3})
- Complex constants: 2 + 3j, 10 - 6.5j
- String constants: 'hello', "Good morning!", "big data analysis", "What's your name?". More about string will be discussed in Lab 4.

More often, computer programs are designed to process data; hence the value of the data could be changed. **Variables** are used to store such **mutable** information. A variable is identified by an **identifier**. In Python, the identifiers begin with a letter or underscore "_" and can include digits. Identifiers in Python are **case-sensitive** (e.g., variable *abc* is different from variable *ABC*).

Python keywords: The following Python keywords have special meaning and shouldn't be used as variable names:

```
'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'exec', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or', 'pass', 'print', 'raise', 'return', 'try', 'while', 'with', 'yield'
```

There are different types of information such as text, numbers, sound, image, video, etc. To handle different types of information, computer programs use different **data types**. We will introduce a few commonly used Python data types in this lab.

Numeric types: to store a single numerical value

int: integers, no decimal point or fractional part

```
num1 = 5
num2 = 3
num3 = num1 + num2
```

float: floating point numbers are real numbers with a decimal point and fractional part

```
float1 = 3.14
```

```
float2 = 5.22
float3 = float1 * float2
```

Remark: In Python, * means multiplication, which will be discussed in Lab 2.

complex: complex numbers, using 'j' or 'J' to represent the root of -1.

```
complex1 = (2 + 3j)
complex2 = complex(3,4)
```

bool: True (1) or False (0), a subtype of integers. Be careful of case-sensitivity.

```
bool1 = False
bool2 = 1 < 2
```

Example 2.1: Type the following statements in your IPython console. The built-in function **type()** can return the type of an object.

```
type(100)
type(3.14)
type(2 + 3j)
type(True)
type(False)
```

Trial-and-Error Exercise 2.1:

Input **type(true)** in the IPython console, what will happen?

Example 2.2: Type the following statements in your IPython console and understand how to assign a value to a variable.

```
#%%
x = 100;
print(x);

x = 200;
print(x);

y = x + 10;
print(y);

z = x + y;
print(z);
```

Sequence types: to store a collection of data (More examples in Section 3)

tuple: A tuple contains an **ordered** collection of items. Tuples are **immutable**, which means you cannot change a tuple after it is defined. Tuple uses ( ) to define its items. Notice that the items in a tuple do not need to have the same data type.

Indexing: Since the items in a tuple are ordered, each item has a unique **index**. Like many other programming languages, indexing in Python starts from zero. So tuple[0] refers to the first item, tuple[8] refers to the 9-th item, etc.

```
tuple_item = (0, "string", 3.14, False)
type(tuple_item)
print(tuple_item)
tuple_student = ("17991234", "CHAN TAI MAN", "Male", "2001-01-01")
print(tuple_student[0])
print(tuple_student[1])
print(tuple_student[2])
print(tuple_student[3])
```

Trial-and-Error Exercise 2.2:

Try the following statement in the IPython console, what will happen?

```
print(tuple_student[4])
```

list: A list also contains an ordered collection of items. Lists are **mutable**. It is one of the most commonly used data type in Python. List uses [ ] to define its items. Notice that the items in a list do not need to have the same data type. We can access a list item through indexing.

```
list_item = [0, "string", 3.14, False]
type(list_item)
print(list_item)
list_student = ["17991234", "CHAN TAI MAN", "Male", "2001-01-01"]
print(list_student)
print(list_student[0])
print(list_student[1])
print(list_student[2])
print(list_student[3])
list_student[3] = "2001-12-31"
print(list_student[3])
print(list_student)
```

Trial-and-Error Exercise 2.3:

Try the following statement, what will happen? Can you see the difference between **tuple** and **list**?

```
tuple_student[3] = "2001-12-31"
```

range: A range represents an **immutable** sequence of integers. It is generated by the built-in function **range()**. The index will start from **0**. More examples about range will be given later.

Function range() can be used in different ways.

Case 1: range(stop), where stop is a positive integer. **range(stop)** will generate a range of integers 0, 1, 2, …, stop - 1. E.g., range(5)

```
num = range(5) # 0, 1, 2, 3, 4
print(num[0]) # 0
print(num[1]) # 1
print(num[2]) # 2
print(num[3]) # 3
print(num[4]) # 4
```

Case 2: range(start, stop), where start and stop are two integers (start is supposed to be smaller than stop; otherwise, the range will be empty). **range(start, stop)** will generate a range of integers start, start + 1, …, stop - 1. In fact, range(stop) is the same as range(0, stop). Notice that start and stop can be negative integers.

```
num = range(2, 5) # 2, 3, 4
print(num[0]) # 2
print(num[1]) # 3
print(num[2]) # 4
num = range(-5, -2) # -5, -4, -3
print(num[0]) # -5
print(num[1]) # -4
print(num[2]) # -3
```

Case 3: range(start, stop, step), where start, stop, step are three integers. It will generate a range of integers start, start + step, start + 2 x step, …, until it reaches (or almost reaches) the value of stop.

```
num = range(3, 10, 2) # 3, 5, 7, 9
print(num[0]) # 3
print(num[1]) # 5
print(num[2]) # 7
print(num[3]) # 9
```

str: Strings are **immutable** sequences of Unicode code points. They are used to represent textual data. In Python, a string can be enclosed in single quotes ('…'), double quotes ("…"), or triple quotes ('''…''', """…"""). If a string contains the double quote character, we can choose single quotes to enclose the string; if a string contains the single quote character, we can then choose double quotes.

Example 2.2: Try the user-defined **multiply()** function, which makes conversions between string and numbers

```
#%%
def multiply():
    numstr1 = input("Enter a number: ")
    numstr2 = input("Enter another number: ")
    num1 = float(numstr1)
    num2 = float(numstr2)
    result = num1 * num2
    print("The product of " + numstr1 + " and " + numstr2 + " is " +
str(result))
    print("The product of {} and {} is {}".format(num1, num2, result))
    print("The product of %f and %f is %f"%(num1, num2, result))
```

In Example 2.2, the function **multiply()** first reads two strings from the keyboard and assign them to two variables **numstr1** and **numstr2**. We then convert these two strings into two float numbers using the built-in **float()** function. We multiply the two numbers and store the result in another variable **result**. At last, we use three different approaches to printing out the result:

- In the first print(), we concatenate a set of strings through +. We also use **str()** function to convert a float into a string.
- In the second print(), we use **str.format()** function to replace the three **placeholders** by num1, num2, and the result, respectively. A pair of curly braces {} is a placeholder. This is a new style in Python.
- In the third print(), we use **%f** as a placeholder for float numbers. This is an old style in Python.

Summary:

In this lab session, we have learnt the following built-in functions:

- print(): to display message on the screen
- input(): to get data from keyboard
- type(): to show the data type of a data
- range(): to generate a range of integers
- float(): to convert an object (such as a string or integer) into float type
- str(): to convert an object into a string type

Reference of all Python built-in functions: https://docs.python.org/3.6/library/functions.html

We have also learnt the following data types:

- Numeric types: int, float, complex, bool
- Sequence types: tuple, list, range, str

What to submit:

Please save all your source codes as a single Python file and submit through BU Moodle.