2020

Lab 2: Introduction to Python

ANACONDA®

Dr. Mohammed Al-Sarem

Taibah University, Information System Department

1/1/2020

Lab 2: Introduction to Python

Lab Objectives:

Python is a powerful general-purpose programming language. It has many powerful libraries that makes the data scientist life easier. In this introductory lab we introduce Python syntax, data types, functions, and control flow tools. These Python basics are an essential part of almost every problem you will solve and almost every program you will write.

Methodology

In class task:

At the end of this lab, the student will be able to:

- Use control flows as well as loops in python
- Write a simple python function
- Solve some mathematical problem

home task:

References:

For more information, see https://www.w3schools.com/python/

1.1. Getting Started

In the previous lab, you should install the required environment that allows you to lunch Python. To do that, check that Anaconda is installed properly in your machine. Similar to what did you in the previous lab, lunch *jupyter* notebook and write the following:

```
In [1]:
from platform import python_version
print(python_version())

3.7.4
```

It is expected to see 3.7.4 which means that the used version is 3.7.4. Knowing that is necessary later in particular when new libraries are required. Python packages are very rapid developed. Thus, you have to select the right version that is compatible with python version installed in your machine.

1.2. Running Python

Python files are saved with a *.py* extension. For now, just create new python file using the option allocated at the top corner of *jupyter* notebook and choose ***Python 3*** option to create a python file. A plain Python file looks similar to the following code.

```
In [4]:
# filename.py
"""This is the file header.
The header contains basic information about the file.
"""
if __name__ == "__main__":
    pass # 'pass' is a temporary placeholder.
```

Let's explain what is in the figure above:

- The # character creates a single-line comment. Comments are ignored by the interpreter and serve as annotations for the accompanying source code. We did the same thing using *Markdown.* Do you remember that?? Refer the previous lab. Explain the different between both ways:

in markdown run as html so you can modify the text by color, size

in code comment, the text is plain and normal. also, you can use comment with codes in the same line so you can explain the code

.

- A pair of three quotes, """ """ or ''' ''', creates a multi-line string literal, which may also be used as a multi-line comment. A triple-quoted string literal at the top of the file serves as the header for the file. The header typically identifies the author and includes instructions on using the file. Executable Python code comes after the header.

❶ **Exercise 1:** Add your information to the header at the top, then add the following code.

```
if __name__ == "__main__":
    print("Hello, world!") # Indent with four spaces (NOT a tab).
```

❶ What do you see?? Which these lines mean??

this is the main method and the code print Hello wolrd!

_____

_____

_____

_____

_____.

You see in the code above that ***print()*** function is used. The propose of the function is the same as those in jave language and c++. To get more information about how to use it, try this code *help(print).* Write down here what occurred in your screen. Can you explain what is the difference??

```
Help on built-in function print in module builtins:
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

1.3. Python Basics

Arithmetic: Arithmetic operations in Python can be used in the same way as in other programing language. Python uses the regular +, -, *, and / operators for addition, subtraction, multiplication and division respectively. It uses ** for exponentiation and % for modular division.

- **Exercise 2:** Demonstrate how can python do that.

```
x = (5-6)
print(x)

x = (5*6)
print(x)

x = (5/6)
print(x)

x = (5+6)
print(x)
```

Variables: Variables are used to temporarily store data. A single equals sign = assigns one or more values (on the right) to one or more variable names (on the left). A double equals sign == is a comparison operator that returns True or False. Unlike many programming languages, Python does not require a variable's data type to be specified upon initialization[1]. Because of this, Python is called a **dynamically typed language**.

[1] Python will warry you if you declare a variable but do not use it you

Exercise 3: Check the following and see if the outputs are the same in both cases?

```
In [9]:
x= 5
y= 2**2 +1
x==y

x,y=5, 2**(2+1)
x==y
```

x is the same 5 y in the first=5 second =8

Functions: To define a function, use the def keyword followed by the function name, a parenthesized list of parameters, and a colon. Then indent the function body using exactly four spaces. Functions are defined with parameters and called with arguments, though the terms are often used interchangeably. Below, width and height are parameters for the function *area()*. The values 2 and 5 are the arguments that are passed when calling the function.

```
In [10]:
def add(x, y):
    return x + y

add(2,5)

Out[10]:
7
```

Exercise 4: For the previous exercise, write a function that return the result of comparison.

```
def comparison(x,y):
    if (x==y):
        return True
    else:
        return False
print(comparison(5,6))
```

Python functions can also return multiple values.

```
In [16]:
def add(x, y):
    return x, y, x + y

FV, SV, S=add(2,5)

print("first variable is {} second veriable is {} and summation is {}".format(FV,SV, S))

first variable is 2 second veriable is 5 and summation is 7
```

Exercise 5: The volume of a sphere with radius r is $V = \frac{4}{3}\pi r^3$ define a function called *sphere_volume()* that accepts a single parameter r. Return the volume of the sphere of radius r, using 3.14159 as an approximation for π (for now). To test your function, call it under the if __name__ == "__main__" clause and print the returned value. Run your file to see if your answer is what you expect it to be.

```
if __name__=="__main__":
    def sphere_volume(r):
        v=(4/3)*3.14159*r**3
        return v
sphere_volume(5)
answer:523.5983333
```

It is also possible to specify default values for a function's parameters. In the following example, the function *information ()* has three parameters, and the value of *level* defaults to 8. If it is not specified in the function call, the variable *level* will contain the value 8 when the function is executed. Note, the default argument is latest variable in the order. What do you think is it possible to reorder the sequence of the variables??

```
def information(Course,Professor, level='8'):
    print(Course + 'is in ' + level +' which is delivered by' + Professor)

if __name__== "__main__":
    Course= "IS-372 Data Mining & Data Warehouse"
    Professor = "A/Prof. Mohammed Al-Sarem"
    information(Course,Professor)
```

:[19] In

```
IS-372 Data Mining & Data Warehouseis in 8 which is delivered byA/Prof. Mohammed Al-Sarem
```

What do you think is it possible to reorder the sequence of the variables??_____
Now, Specify each parameter!

Data Types and Structures: Python has four numerical data types: int, long, float, and complex. Each stores a different kind of number. The built-in function type() identifies an object's data type.

Strings: In Python, strings are created with either single or double quotes. To concatenate two or more strings, use the + operator between string variables or literals. Parts of a string can be accessed using *slicing*, indicated by square brackets [ ]. Slicing syntax is [start: stop: step]. The parameters start and stop default to the beginning and end of the string, respectively. The parameter step defaults to 1.

```
Course_Name='IS-372 Data Mining & Data Warehouse'
print(Course_Name[2])
```

:[26] In

```
-
```

Exercise 6: Using the variable Course_Name, demonstrate how to get:

- String "IS-372"! print(course_name[0:6])
- What is the result of calling Course_Name[-1] IS-372 data mining & data werehousin
- Course_Name [:5] IS-37
- Course_Name[6:] data mining & data werehousing

1.4. Lists

A Python list is created by enclosing comma-separated values with square brackets **[ ]**. Entries of a list do not have to be of the same type. Access entries in a list with the same indexing or slicing operations used with strings. Try this

my_list = ["Hello", 93.8, "world", 10]

my_list[0] 'Hello'
my_list[-2] 'world'
my_list[:2] ['Hello', 93.8]

Common list methods (functions) include *append()*, *insert()*, *remove()*, and *pop()*.

```
my_list = [1, 2] # Create a simple list of two integers.
my_list.append(4) # Append the integer 4 to the end.
my_list.insert(2, 3) # Insert 3 at location 2.
my_list.remove(3) # Remove 3 from the list.
my_list.pop() # Remove (and return) the last entry.
```

:[35] In

```
4
```

:[35]Out

What is the output after

Append () method? add element at the end
Insert () method? add element at specified location
Remove () method? remove the first element

Pop () method? remove element at specified location

1.5. Tuples

A Python tuple is an ordered collection of elements, created by enclosing comma-separated values with parentheses (and). Tuples are similar to lists, but they are much more rigid, have less built-in operations, and cannot be altered after creation. Lists are therefore preferable for managing dynamic ordered collections of objects.
When multiple objects are returned by a function, they are returned as a tuple. For example, recall that the *arithmetic* () function returns two values.

```
def arithmetic(a, b):
    """Return the difference and the product of the two inputs."""
    return a - b, a * b
x, y = arithmetic(5,2) # Get each value individually,
print(x, y)
both = arithmetic(5,2) # or get them both as a tuple.
print(both)
```

:[36] In

```
10 3
(10 ,3)
```

Exercise 7: Write a function called list_ops(). Define a list with the entries "bear", "ant", "cat", and "dog", in that order. Then perform the following operations on the list:

1. Append "eagle".
2. Replace the entry at index 2 with "fox".
3. Remove (or pop) the entry at index 1.
4. Sort the list in reverse alphabetical order.
5. Replace "eagle" with "hawk".

(Hint: the list's index() method may be helpful.)

6. Add the string "hunter" to the last entry in the list.

Return the resulting list.

```
def list_ops():
    my_list=["bear","ant","cat","dog"]
    my_list.append("eagle")
    my_list.pop(2)
    my_list.insert(2,"fox")
    my_list.pop(1)
    my_list.reverse()
    my_list.pop(0)
    my_list.insert(0,"hawk")
    my_list.append("hunter")
    return my_list
print(list_ops())
```

1.6. Sets

A Python set is an unordered collection of distinct objects. Objects can be added to or removed from a set after its creation. Initialize a set with curly braces { }, separating the values by commas, or use *set ()* to create an empty set. Like mathematical sets, Python sets have operations like union, intersection, difference, and symmetric difference.

```
In [40]:
# Initialize some sets. Note that repeats are not added.
DataMining_Professors = {"Muhannad, Al-Mohaeemid", "Faisal, Saeed", "Mohammed, Al-Sarem"}
print(DataMining_Professors)

DataMining_Professors.add("Wadii, Boulila") # Add an object to the set.
DataMining_Professors.discard("Muhannad, Al-Mohaeemid") # Delete an object from the set.
print(DataMining_Professors)

Database_Professors = {"Muhannad, Al-Mohaeemid", "Faisal, Saeed", "Essa, Hizzam", "Wadii, Boulila"}

DataMining_Professors.intersection(Database_Professors)
DataMining_Professors.difference(Database_Professors)
```

```
{'Faisal, Saeed', 'Muhannad, Al-Mohaeemid', 'Mohammed, Al-Sarem'}
{'Faisal, Saeed', 'Mohammed, Al-Sarem', 'Wadii, Boulila'}

Out[40]:
{'Mohammed, Al-Sarem'}
```

1.7. Dictionaries

Like a set, a Python dict (dictionary) is an unordered data type. A dictionary stores key-value pairs, called items. The values of a dictionary are indexed by its keys. Dictionaries are initialized with curly braces, colons, and commas. Use *dict ()* or {} to create an empty dictionary.

```
In [46]:
DataScience_Track = {"IS-372": "Data Mining & Data Warehouse",
                     "IS-472": "Decision Support System",
                     "IS-476": "Information Search, Retrieval & Visualization",
                     "IS-453": "Special Topics in Data Management"}
print(DataScience_Track["IS-453"])

# Display the keys and values.
DataScience_Track.keys()

DataScience_Track.values()
```

```
Special Topics in Data Management

Out[46]:
dict_values(['Data Mining & Data Warehouse', 'Decision Support System', 'Information Search, Retrieval & Visualization', 'Special Topics in Data Management'])
```

2.1. Control Flow Tools

An if statement executes the indented code if (and only if) the given condition holds. The elif statement is short for "else if" and can be used multiple times following an if statement, or not at all. The else keyword may be used at most once at the end of a series of if/elif statements.

```
if (condition):
    Statement 1
Elif (condition):
    Statement 2
Else:
    Statement 3
```

2.2. The While Loop

A while loop executes an indented block of code while the given condition holds.

```
i = 0
while i < 10:
    print(i, end=' ') # Print a space instead of a newline.
    i += 1 # Shortcut syntax for i = i+1.
```

```
0 1 2 3 4 5 6 7 8 9
```

2.3. The For Loop

A *for* loop iterates over the items in any iterable. Iterables include (but are not limited to) strings, lists, sets, and dictionaries.

```
colors = ["red", "green", "blue", "yellow"]
for entry in colors:
    print(entry + "!")
```

Exercise 8: What output is produced by the following code?

```
s = "stab"
for i in range(len(s)):
    print (s[0 : i : 1])
```

```
s
st
sta
```