Lab - Python

In this lab exercise, we will learn the basics of python programming language. This basics will aid us in the future lab sessions.

> First of all, you need to download this python interpreter. To write the python script in your local host, you can download various IDE's such as VSCode, Atom, PyCharm etc. However, I would suggest you to use Jupyter Notebook. Check out this video (**From 4:30 seconds**) to learn how to use Jupyter Notebook more efficiently.
>
> If you don't want to install any of these things, or your local host is not that powerful you can use google colab. It is a free online platform provided by Google to execute ipynb files. You don't have to install anything in your local machine. You just need an internet connection and you are ready to go. A short youtube survey will help you to understand all the basic usage of Google Colab.

Finally, at the end of the lab, there are few exercises to do by your own.

Python Fundamentals

- Python is an interpreted programming language.
- It can be used to develop programs based on procedural and object-oriented programming paradigms.
- Python has a very simple syntax.
- It provides many built-in programming features.

Primitive Types

There are several primitive types in python. Primitive type means a built-in type which is a part of the language. In short, python has some built-in data types and data structures which ease the overall coding experience. The most important ones are the followings:

> 1. Integer (int)
> 2. Floating Point (float)
> 3. Boolean Logic (bool)
> 4. String (str)
> 5. List ('list' or can be denoted as square brackets)
> 6. Tuple ('tuple' or can be denoted as paranthesis)
> 7. Dictionary ('dict' or can be denoted as curly braces)

Integer: numeric values without fractional part. *Examples: 1, 32, 999*

Floating Point: numeric values with fractional part. *Examples: 1.3, 3.2, 8.3454*

boolean: logical values. Either *true* or *false*.

string: an ordered sequence of alphanumeric characters. *Examples: 'Wolves', 'Python'*

list: an ordered sequence of objects in which each element is identified by an integer subscript. *Examples: [], [1, 3, 4], ['a', 3, 7.5]*

tuple: similar to the list type, but the content cannot be changed once it is created. *Examples: (), (1, 3, 4), ('a', 3, 7.5)*

dictionary: a collection of objects in which each object contains a key and a value. *Examples: {}, {1: 'orange', 2: 'banana'}*

Standard Input and Output

In [ ]:
```
"""To ask a value from the end user and store it to reuse later, first take a variable and store the value.
The "input" keyword helps to take value from the end user """

first_name = input("Please Enter Your First Name: ")
```

In [ ]:
```
"""To show the output python has a keyword "print" which prints the output"""

print(first_name)
```

Control Structure

> It is used to change the flow of control and execute a statement or a block of statements if some condition holds.

In [ ]:
```
"""Here we are asking user to enter a value and checks if its negative, non-negative or positive
by the control structure"""

value = int(input("Please Enter a Number: ")) #input keyword always takes any value as a string. Thus, we use int() function
#it is called casting. It converts a value to different data types

if value < 0:
    print('The number is negative')
elif value == 0:
    print('The number is non-negative')
else:
    print('The number is positive')
```

Loops

> There are few types of iterative processes which we call loops. but the most used ones are **for** and **while** loop depending the situation.

In [ ]:
```
"""Find the sum of all integers from 1 to 10 using while loop"""

total = 0
counter = 1

while counter <= 10: #stopping condition
    total = total + counter #sum
    counter += 1 #increament

print(total)
```

In [ ]:
```
"""Find the sum of all integers from 1 to 10 using for loop"""

total = 0

for counter in range(1, 11, 1): #initialize, stopping condition, increament.
    total = total+ counter

print(total)
```

List Modification

In [ ]:
```
"""A list can be created using a comma-separated sequence within square brackets"""

first_list = [1, 2, 3, 8]
```

In [ ]:
```
"""In a list, the individual items can be accessed by subscript/index:"""

print("First value of the list is ", first_list[0])
print("Last value of the list is ", first_list[-1])
```

In [ ]:
```
"""The elements of a list can also be accessed by traversing the list using the iterator mechanism:"""

for values in first_list:
    print(values)
```

Other List Operations

In [ ]:
```
#append
first_list.append(10)
print("After adding another element in the list, the list is ", first_list)



#modify
first_list[2] = 5
print("After modifying an element in the list, the list is ", first_list)



#insert: elements can be inserted in any positon/index
first_list.insert(2, 12)
print("After inserting an element in the list in the third position, the list is ", first_list)
```

There are also other methods for several operations such as remove, pop, delete etc. Please practice those by your own

Dictionary Modification

In [ ]:
```
"""There are lot of ways to access elements of the dictionary. However, we will parctice the most common one.
The traversal using the iterator mechanism (for statement) is made over the keys in a dictionary"""

zipCode = {'London':290, 'Leeds':578, 'Wolverhampton':911}

for key in zipCode:
    print(key, '= ', zipCode[key])
```

In [ ]:
```
"""To explicitely find the list of keys or list value we can use the following"""
key_list = zipCode.keys()
value_list = zipCode.values()

print(key_list)
print(value_list)
```

In [ ]:
```
"""To change the value of a corresponding key we can do the following"""

zipCode['London'] = 666

for key in zipCode:
    print(key, '= ', zipCode[key])
```

In [ ]:
```
"""Using the same technique we can add another entry. However, the key must be unique.
Duplicate key name will modify the value only"""

zipCode['Birmingham'] = 111

for key in zipCode:
    print(key, '= ', zipCode[key])
```

Similarly to the list operations, dictionary has also lots of other operations and students are encouraged to practice those by their own

Functions

> Writing a function is pretty easy in python using the keyword def(). After declaring the function name, we have put the arguments. We do not have to mention explicitely if the passing argument is a list, dictionary or int etc.. It can be anything.

In [ ]:
```
def oddEven(num):
    if num%2 == 0:
        print('The number is even')
    else:
        print('The number is odd')
    return

num = int(input("Please Enter a Number: "))
oddEven(num)
```

Exercises

Now that, you have known some basics of python, try to do the following exercises. These exercise questions are not limited to the functions that are given above. You have to think for the solutions and search for the functions that you need to use. In this way, you will learn a lot of operations of different data structures such as lists, dictionaries etc. This will aid you to be a better developer in python programming language as well as data analytic domain.

Exercise on Loops

- Display Fibonacci series up to 10 terms.
- Find the factorial of a given number. Ask the user to provide the number.
- Write a program to print a right angle triangle using the for loop.

Exercise on Functions

- Make a simple calculator that can add, subtract, multiply and divide two numbers using function.

Exercise on List and Dictionaries

- Remove the first duplicate element of a list

- Write a Python program to create a new dictionary by extracting the mentioned keys from the below dictionary.

sample_dict = { "name": "Anderson", "age": 35, "salary": 5000, "city": "London"}

keys = ["name", "salary"]

> Expected Output: {'name': 'Anderson', 'salary': 5000}

- Write a Python program to convert two lists into a dictionary in a way that item from list_one is the key and item from list_two is the value.

list_one = ['one', 'two', 'three']

list_two = [1, 2, 3]

> Expected Output: {'one': 1, 'two': 2, 'three': 3}

In [ ]:
```

```