Table of Contents

1 Objectives

After performing this lab, students shall be able to understand:

- ✔ Python data types.
- ✔ Python operators (math, comparison, boolean)
- ✔ Python condition and loops
- ✔ Python functions

2 Task Distribution

Total Time	**170 Minutes**
Python data types	15 Minutes
Python operators	10 Minutes
Python if-else conditions	10 Minutes
Python loops	15 Minutes
Python functions	20 Minutes
Exercise	90 Minutes
Online Submission	10 Minutes

3 Online Python Interpreter

We will be working on Google Collab to run Python programs.

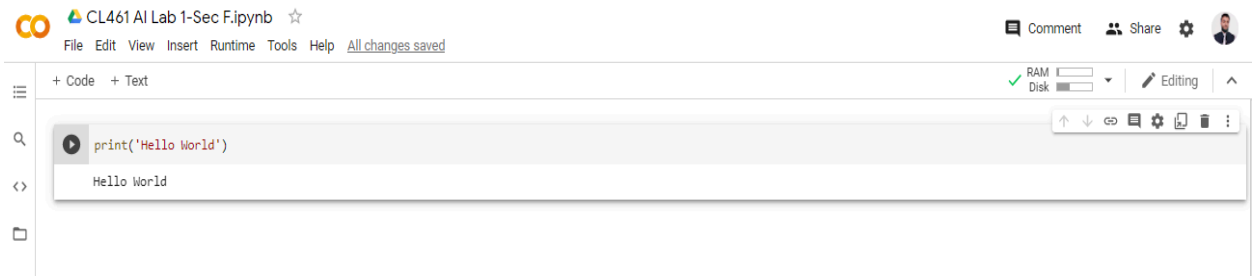
Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser without any configuration. To do that we need to create a Colab notebook.

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them.

Colab notebooks are Jupyter notebooks that are hosted by Colab. The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. Learn more about Jupyter here.

Follow the instructions given below:

- Visit the following link: https://colab.research.google.com/
- Sign in using your nu email id.
- You will be directed to 'Welcome to Colaboratory' page.
- Go to **File->New Notebook**.
- A new notebook is created by the name 'Untitled0.ipynb'.
- Rename the notebook to your roll number.
- Write your first Python program by typing the following statement in the first cell
    - `print('Hello World')`
- Execute this cell by clicking the play button on the left of the cell or by pressing Ctrl+Enter.
- You will notice that the notebook will connect to a runtime. RAM and Disk resources are allocated. (Refer to the screenshot below)



For offline usage, PyCharm IDE is recommended. Installation details for PyCharm will be shared later.

4 Data Types

The following section describes the standard types that are built into the Python interpreter. These datatypes are divided into different categories like numeric, sequences, mapping etc. Typecasting is also discussed below.

4.1 Built-in Types

The following chart summarizes the standard data types that are built into the Python interpreter.

Sr#	Categories	Data Type	Examples
1	Numeric Types	int	-2, -1, 0, 1, 2, 3, 4, 5, int(20)
2		float	-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25, float(20.5)
3		complex	1j, complex(1j)
4	Text Sequence Type	str	'a', 'Hello!', str("Hello World")
5	Boolean Type	bool	True, False, bool(5)
6	Sequence Types	list	["apple", "banana", "cherry"], list(("apple", "banana", "cherry"))
7		tuple	("apple", "banana", "cherry"), tuple(("apple", "banana", "cherry"))
8		range	range(6)
9	Mapping Type	dict	{"name" : "John", "age" : 36}, dict(name="John", age=36)

10	Set Types	set	{"apple", "banana", "cherry"}, set(("apple", "banana", "cherry"))
11		frozenset	frozenset({"apple", "banana", "cherry"})
12	Binary Sequence Types	bytes	b"Hello", bytes(5)
13		bytearray	bytearray(5)
14		memoryview	memoryview(bytes(5))

Python has no command for declaring a variable for any datatype. A variable is created the moment you first assign a value to it. Variable names are case-sensitive. Just like in other languages, Python allows you to assign values to multiple variables in one line.

4.2 Typecasting

The process of explicitly converting the value of one data type (int, str, float, etc.) to another data type is called type casting. In Type Casting, loss of data may occur as we enforce the object to a specific data type.

```
# cast to float
x=float(2)
y=float(30.0)
z=float("20")
print(x)
print(y)
print(z)

# cast to str
x=str(2)
y=str(30.0)
z=str("20")
print(x)
print(y)
print(z)

# Sum two numbers using typecast
num_int = 123
num_str = "456"
print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))
num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))
num_sum = num_int + num_str
print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Notice the `type()` function used in the above example. Find out what it does. Execute the given

example in Jupyter notebook to observe the result of type casting.

5 Operators

This section contains the details of different Python operators i.e. Math operators, comparison operators and Boolean operators.

5.1 Math Operators

From **Highest** to **Lowest** precedence:

Operators	Operation	Example
**	Exponent	`2 ** 3 = 8`
%	Modulus/Remainder	`22 % 8 = 6`
//	Integer division	`22 // 8 = 2`
/	Division	`22 / 8 = 2.75`
*	Multiplication	`3 * 3 = 9`
-	Subtraction	`5 - 2 = 3`
+	Addition	`2 + 2 = 4`

5.2 Comparison Operators

Operator	Meaning
`==`	Equal to
`!=`	Not equal to
`<`	Less than
`>`	Greater Than
`<=`	Less than or Equal to
`>=`	Greater than or Equal to

5.3 Boolean Operators

There are three Boolean operators: `and`, `or`, and `not`.

The `and` Operator's *Truth* Table:

Expression	Evaluates to
`True and True`	`True`
`True and False`	`False`
`False and True`	`False`
`False and False`	`False`

The `or` Operator's *Truth* Table:

Expression	Evaluates to
`True or True`	`True`
`True or False`	`True`
`False or True`	`True`
`False or False`	`False`

The `not` Operator's *Truth* Table:

Expression	Evaluates to
`not True`	`False`
`not False`	`True`

6 If-else Conditions

Python supports conditional statements i.e. `if, elif, else.` Comparison operators and Boolean operators written in the previous section can be used in if-elif-else statements.
Python uses indentation instead of curly-brackets to define the scope in the code.

6.1 if Statement Example:

```
name = 'Alice'
if name == 'Alice':
    print('Hi, Alice.')
```

6.2 if-else Statement Example:

```
name = 'Bob'
if name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

6.3 if-elif-else Statement Example:

```
name = 'Bob'
age = 30
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
else:
    print('You are neither Alice nor a little kid.')
```

Python conditional statements only require `if` statement to execute. Both `elif` and `else` are optional and used as per requirement.

7 Loops

Python has two types of loops i.e. `while`, `for`. Use Jupyter notebook to execute all code snippets given in the examples below to observe their results.

7.1 While Loop Example with break Statement

With the `while` loop we can execute a set of statements as long as a condition is true or the loop execution reaches a `break` statement.

```
while True:
    print('Please type your name.')
    name = input()
    if name == 'your name':
        break
print('Thank you!'
```

`input()` in the above example is a built-in Python function which is discussed in the functions section below.

7.2 While Loop Example with continue Statement

When the program reaches a `continue` statement, the program execution immediately jumps back to the start of the loop.

```
while True:
    print('Who are you?')
    name = input()
    if name != 'Joe':
        continue
    print('Hello, Joe. What is the password? (It is a fish.)')
    password = input()
    if password == 'swordfish':
        break
print('Access granted.')
```

7.3 for Loop Example with range()

```
print('My name is')
for i in range(5):
    print('Jimmy Five Times ({})'.format(str(i)))
```

7.4 for Loop Example with range() arguments

The `range()` function can also be called with three arguments. The first two arguments will be the start and stop values, and the third will be the step argument. The step is the amount that the variable is increased by after each iteration.

```
for i in range(0, 10, 2):
    print(i)
```

8 Functions

This section contains the details of Python user defined or custom function along with a few examples of Python built-in functions.

8.1 Custom Functions

Programmers can define their own functions in Python. Functions can contain all types of Python statements like variables, conditions and loops etc.

8.1.1 Simple Function Example

A function in Python starts with `def` keyword followed by the function name with round brackets. Function parameters can be passed depending on the requirement.

```
def hello(name):
     print('Hello {}'.format(name))
hello('Alice') #Hello Alice
hello('Bob') #Hello Bob
```

8.1.2 Function Example with Return Statement

A return statement consists of the following:

- The return keyword.
- The value or expression that the function should return.

```
import random #Syntax to import Python libraries
```

```
def getAnswer(answerNumber):
    if answerNumber == 1:
        return 'It is certain'
    elif answerNumber == 2:
        return 'It is decidedly so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'

r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

8.2 Built-in Functions

The Python interpreter has a number of functions built into it that are always available. We have already covered a few built-in functions in the datatypes section above. Refer to this link for the complete list of Python built-in functions.

8.2.1 Built-in Function Examples

Execute the code given below in your Jupyter notebook to find the results of built-in functions.

```
# abs integer number
num = -5
print('Absolute value of -5 is:', abs(num))
# Notice print here, it is also a built-in function

# abs floating number
fnum = -1.45
print('Absolute value of 1.45 is:', abs(fnum))

# input function
x = input('Enter your name:')
print('Hello, ' + x)

# max function
number = [3, 2, 8, 5, 10, 6]
largest_number = max(number);
print("The largest number is:", largest_number)
```

```
# print usage
print('Hands-on','python','programming','lab',sep='\n')

# sum function
my_list = [1,3,5,2,4]
print "The sum of my_list is", sum(my_list)
```

Submission Instructions

Always read the submission instructions carefully.

- Rename your Jupyter notebook to your roll number and download the notebook as **.ipynb** extension.
- To download the required file, go to **File->Download .ipynb**
- Only submit the **.ipynb** file. DO NOT **zip** or **rar** your submission file
- Submit this file on Google Classroom under the relevant assignment.
- Late submissions will **NOT AT ALL** be accepted