Department of Information Engineering Technology

The Superior University, Lahore

Artificial Intelligence Lab

Experiment No.1

Python Overview Basic of Python Syntax

Prepared for

_____

By:

Name: _____

ID: _____

Section: _____

Semester: _____

Total Marks: _____

Obtained Marks: _____

Signature: _____

Date: _____

Experiment No. 1

Python Overview Basic of Python Syntax

(Rubrics)

Name: _____ **Roll No. : _____**

A. PSYCHOMOTOR

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Follow Procedures	1	0	0.25	0.5	0.75	1	
2	Software and Simulations	2	0	0.5	1	1.5	2	
3	Accuracy in Output Results	3	0	0.75	1.5	2.25	3	
Sub Total		**6**	**Sub Total marks Obtained in Psychomotor(P)**					

B. AFFECTIVE

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Respond to Questions	1	0	0.25	0.5	0.75	1	
2	Lab Report	1	0	0.25	0.5	0.75	1	
3	Assigned Task	2	0	0.5	1	1.5	2	
Sub Total		**4**	**Sub Total marks Obtained in Affective (A)**					

Instructor Name:_____ **Total Marks (P+A): _____**

Instructor Signature:_____ **Obtained Marks (P+A): _____**

Introduction to Python

Python is a versatile and popular programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python has gained immense popularity over the years and is widely used for various applications, including web development, data analysis, artificial intelligence, scientific computing, and more. In this introduction to Python, we'll cover some fundamental aspects of the language:

- **Readability:** Python emphasizes clean and readable code, making it accessible to beginners and experienced developers alike. It uses indentation (whitespace) to define code blocks, which promotes good coding practices and readability.
- **Interpreted Language:** Python is an interpreted language, which means you don't need to compile your code before running it. You write the code, and an interpreter executes it line by line. This makes development and debugging quicker and more straightforward.
- **High-Level Language:** Python is a high-level programming language, which means it provides abstractions and tools to help developers work with complex tasks without having to manage low-level details like memory management.
- **Dynamic Typing**: Python uses dynamic typing, meaning you don't need to declare variable types explicitly. The interpreter determines the data type at runtime.
- **Extensive Standard Library:** Python's standard library includes a wide range of modules and packages that simplify common programming tasks. You can perform operations such as file handling, network communication, and more with ease.
- **Cross-Platform:** Python is available on various platforms, including Windows, macOS, and Linux. This cross-platform compatibility makes it an excellent choice for developing applications that need to run on different operating systems.
- **Community and Ecosystem:** Python has a large and active community of developers. This community has contributed to a rich ecosystem of libraries and frameworks, such as Django for web development, NumPy for scientific computing, TensorFlow for machine learning, and many others.
- **Versatility:** Python is a versatile language suitable for a wide range of applications, from web development (with frameworks like Flask and Django) to data science (with libraries like pandas and Matplotlib) to machine learning and artificial intelligence (with libraries like scikit-learn and TensorFlow).
- **Open Source:** Python is an open-source language, which means it's freely available, and you can modify and distribute it as needed. This openness has contributed to its widespread adoption and continuous improvement.
- **Community and Resources:** Python has an extensive online community, including forums, tutorials, and documentation. You can find plenty of resources to help you learn and troubleshoot any issues you encounter.

We'll be using **Google Colab** in our lab. Google Colab, short for Google Colaboratory, is a cloud-based platform provided by Google that allows you to write, run, and share Python code using Jupyter notebooks. It is an excellent tool for data science, machine learning, and general Python development, especially for those who may not have access to high-end hardware or want to collaborate on projects with others

Basic of Python Syntax

Python is known for its clean and readable code, and its syntax reflects this simplicity. Here are some fundamental aspects of Python's syntax:

1. **Comments:** Comments in Python begin with a hash symbol (#) and are used to add explanations or notes within your code. Comments are ignored by the Python interpreter.

```
#This is a comment
```

2. **Variables:** In Python, you can create variables without explicitly declaring their data types. The type is determined dynamically at runtime.

```
name = "John"  # A string variable
age = 30       # An integer variable
height = 6.2   # A float variable
```

3. **Data Types:** Python supports various data types, including integers, floating-point numbers, strings, lists, tuples, dictionaries, and more.

```
my_integer = 42
my_float = 3.14
my_string = "Hello, World!"
my_list = [1, 2, 3, 4, 5]
my_dict = {"name": "Alice", "age": 25}
```

4. **Print Function:** You can use the print() function to display output in Python

```
print("Hello, World!")
```

5. **Strings:** Python supports string manipulation with a wide range of methods and operations

```
my_string = "Python"
print(my_string[0])        # Accessing characters by index
print(len(my_string))       # String length
print(my_string.upper())    # Convert to uppercase
print(my_string.lower())    # Convert to lowercase
```

6. **Indentation:** Python uses indentation to define code blocks, such as loops, conditionals, and functions. Indentation is crucial for code structure and readability. Typically, four spaces (or a tab) are used for each level of indentation.

```
if True:
    print("This is indented correctly")
else:
    print("This is indented correctly")
```

7. **Whitespace:** Be careful with spaces and indentation. Python is sensitive to whitespace, and improper indentation can lead to errors
8. **Append:** The append() method allows you to add an element to the end of an existing list.

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)  # Output: [1, 2, 3, 4]
```

9. **type() Function:** The type() function is used to determine the data type of an object in Python. It returns a reference to the type object, which represents the data type of the given object. You can use type() to check if an object is of a specific type or to determine the type dynamically.

```
x = 42
y = "Hello, World!"
z = [1, 2, 3]
print(type(x))  # Output: <class 'int'>
print(type(y))  # Output: <class 'str'>
print(type(z))  # Output: <class 'list'>
```

10. **len() Function:** The len() function is used to find the length or the number of elements in various Python objects, such as strings, lists, tuples, and dictionaries. It returns an integer representing the length of the object.

```
my_string = "Python"
my_list = [1, 2, 3, 4, 5]
print(len(my_string))  # Output: 6
print(len(my_list))    # Output: 5
```

11. **Arithmetic Operators:** Python supports standard arithmetic operators like +, -, *, /, and % for addition, subtraction, multiplication, division, and modulo, respectively.

```
a = 10
b = 5
result = a + b
print(result) #Output: 15
```

12. **Control Structures:** Python supports common control structures like if, elif, else for conditional statements and for and while for loops.

    - **if Statement:** The if statement is used for conditional branching. It allows you to execute a block of code if a specified condition is true. If the condition is true, the code inside the if block is executed; otherwise, it is skipped.

```
#if Statment
x = 10
if x > 5:
    print("x is greater than 5")
```

- **elif Statement:** The elif statement is used in conjunction with if to handle multiple conditions. If the initial if condition is false, Python checks the conditions specified in elif statements one by one. The first elif block with a true condition (if any) will execute, and the remaining elif and else blocks are skipped

```
x = 10
if x > 20:
    print("x is greater than 20")
elif x > 10:
    print("x is greater than 10 but not 20")
else:
    print("x is 10 or less")
```

- **else Statement:** The else statement is used to define a block of code that executes when none of the preceding if or elif conditions are true. It provides a fallback option when no other conditions match

```
x = 5
if x > 10:
    print("x is greater than 10")
else:
    print("x is 10 or less")
```

- **while Loop:** The while statement is used to create a loop that repeatedly executes a block of code as long as a specified condition is true. Be cautious with while loops, as they can lead to infinite loops if the condition is never met

```
count = 0
while count < 5:
    print("Count:", count)
    count += 1
```

- **for Loop:** The for loop in Python is used to iterate over a sequence (such as a list, tuple, string, or range) or any iterable object. It allows you to execute a block of code for each item in the sequence.

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

- **Nested loop:** A nested loop in Python is a loop inside another loop. This allows you to iterate over a collection of elements using two or more iterators, creating a multi-dimensional loop structure. Nested loops are often used when you need to perform repetitive operations on elements that have multiple levels of organization, such as rows and columns in a grid or items in nested lists.

```
m=2
n=4
for i in range(0,n):
  print('welcome')
  for j in range(0,m):
    print('welcome to AI Lab')
print('course')
```

- **Break Statement:** The break statement is used within loops (e.g., for or while) to exit the loop prematurely when a certain condition is met.

```
for letter in 'welcometo':
  if letter=='l':
    break
  print('current letter:', letter)
```

- **Continue Statement:** The continue statement in Python is used to skip the rest of the current iteration of a loop (either a for loop or a while loop) and move on to the next iteration. It allows you to control the flow of your loop by skipping specific iterations based on a condition.

```
for letter in 'welcometo':
  if letter=='w'or letter=='t':
    continue
  print('current letter:', letter)
```

13. **List:** A list is an ordered collection of items, where each item can be of any data type. Lists are defined using square brackets [] and can contain items separated by commas. Lists are mutable, meaning you can change their content (add, remove, or modify items) after they are created. Lists are commonly used when you need a dynamic collection of items.

```
#Creation of list
List=[]
List1=[10,20,30,40,'AI','Lab']
#Accessing elements from the List
List1[5] #Output: 'Lab'
#Removing Elements
List.remove(50)
#Adding Elements to a List
List.append(50)
```

14. **Dictionary (Dict):** A dictionary is an unordered collection of key-value pairs, where each key is unique. Dictionaries are defined using curly braces {} or the dict() constructor and consist of key-value pairs separated by colons : Dictionaries are mutable, allowing you to add, update, or remove key-value pairs. Dictionaries are useful for associating values with unique identifiers (keys).

```
#Creation of Dict
Dict={'Lab Instructor':'Waqar', 1:[10,20,30,40]}
#one mixed key and one is integer
Dict2={1:'welcome', 2:'to', 3:'AI'}
print(Dict)
print(Dict2)
#Creating empty Dict
my_dict = dict({1:'Welcome',2:'to',3:'lab'})
#Adding Elements
Dict[0]='welcome'
Dict[1]='to'
Dict[2]='AI'
Dict[3]='Lab'
```

15. **Tuple:** A tuple is an ordered collection of items, similar to a list, but unlike lists, tuples are immutable, meaning their contents cannot be changed after creation. Tuples are defined using parentheses () and can contain items separated by commas. Tuples are often used when you have a collection of values that should not change during program execution, such as coordinates or configuration settings.

```
#Creation of Tuples
var=("Welcome","to","AI")
#Concatenation of Tuples
tuple1=(10,20,30)
tuple2=('AI',"Lab")
print(tuple1+tuple2) #Output: (10, 20, 30, 'AI', 'Lab')
#Nesting of Tuples
tuple1=(10,20,30)
tuple2=('AI',"Lab")
tuple3=(tuple1,tuple2)
print(tuple3) #Output: ((10, 20, 30), ('AI', 'Lab'))
```

16. **Functions:** You can define functions using the def keyword. In Python, a function is a reusable block of code that performs a specific task or set of tasks. Functions are used to organize code into smaller, manageable pieces, promote code reuse, and make code more readable and maintainable.

```
#Example 01
def multiply(a,b):
  x=a*b
  return x
y = multiply(5,8)
y #Output: 40

#Example 02
def listednew():
    list =['welcome','to','AI','Lab']
    for i in list:
      print(i)
    return(0)
listednew()

#Example 03
def factorial(x):
    if x==1:
        return x
    else:
      return x*factorial(x-1)
```

17. **Class:** Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects. A very basic class would look something like this:

```
class MyClass:
  variable = "Python"
  def function(self):
   print("Artificial Intelligence is the future")
```

- **Accessing Object Variables:** To access the variable inside of the newly created object "myobjectx" you would do the following:

```
class MyClass:
  variable="Section B"
  def function(self):
   print("Artificial Intelligence is the future.")
myobjectx = MyClass()
print (myobjectx.variable) # Output: Section B
```

- **Accessing Object Functions:** To access a function inside of an object you use notation similar to accessing a variable:

```
class MyClass:
  variable = "Section B"
  def function(self):
   print("Artificial Intelligence is the future.")
myobjectx=MyClass()
myobjectx.function() # Output: Artificial Intelligence is the future.
```

- **Initializing Values and Calling Function** :
  - **Case Study:** We have a class defined for vehicles. Create two new vehicles called car1 and car2. Set car1 to be a red convertible worth $60,000.00 with a name of Fer, and car2 to be a blue van named Jump worth $10,000.00.

```
class Vehicle:
  name=""
  kind="car"
  color=""
  value=100.00
  def description(self):
    desc_str = "%s is a %s %s worth $%.2f." % \
              (self.name, self.color, self.kind, self.value)
    return desc_str

carl = Vehicle()
carl.name="Fer"
carl.color="red"
carl.kind="convertible"
carl.value = 60000.00

car2 = Vehicle()
car2.name = "Jump"
car2.color="blue"
car2.kind="van"
car2.value = 10000.00

print(carl.description()) #Output: Fer is a red convertible worth
$60000.00.
print(car2.description()) #Output: Jump is a blue van worth $10000.00.
```

- **The *__init__*() Function:** All classes have a function called *__init__*(), which is always executed when the class is being initiated. Use the *__init__*() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

```
class Vehicle:
  def __init__(self, name, color, kind, value):
   self.name=name
   self.kind=kind
   self.color=color
   self.value=value
  def description(self):
   desc_str = "%s is a %s %s worth $%.2f." % (self.name, self.color,
self.kind, self.value)
   return desc_str

#your code goes here
carl=Vehicle("fer", "red", "convertible", 68000)
```

```
print(carl.description()) #Output: fer is a red convertible worth
$68000.00.

car2=Vehicle("Jump", "blue", "van", 10080)
print(car2.description()) #Output: Jump is a blue van worth $10080.00.
```

Lab Task

1. Write a calculator program in Python using ADD, SUB, MUL, DIV functions. Take input values of Two numbers, an Operator and call the corresponding functions to perform these tasks.
2. Write a program to take values from users to create a list, ask him how many numbers of values he wants to add and print them.
3. A person wants to go from Faisalabad to Karachi, if the distance between these cities is 1,115.3 km Ask person if he wants to travel By air or by road.
Aeroplan speed is 800 km/h
Car speed is 100 km/h
1. Write program in python to creating a class Travel , and two functions Fly and Road which can calulate travel time by each means , ask him about his preferences to travel and tell him how much time he will take to reach.

Home Task

1. Write a Program for a Library System
Ask user about his name, create a dictionary to store his name and roll number, If his roll number exist say him Hi with his name and give him choices to select from different books category after selecting books category and then books at the end give him all the information which book he selects and random time to return books.
Use classes for writing program.
2. Create a Covid-19 Detection System. Ask user about his symptoms and tell him either he is Covid-19 positive or not.
Use Dictionary to create data set