Submitted by	Fatima Shahzad BCS-22004
Submitted to	Sir Ghulam Mustafa
Course	Artificial Intelligence
Department	Information Technology
Discipline	Computer Science

Lab#1 of Artificial Intelligence

Introduction to AI and Its Application Using Python

Python is widely used in AI, including Machine Learning, NLP, and Neural Networks. It is an interpreted, interactive, object-oriented, high-level language created by Guido van Rossum (1985-1990). Like Perl, Python's source code is available under the GNU GPL.

- **Interpreted**: Runs without prior compilation, like PERL and PHP.
- **Interactive**: Allows direct interaction with the interpreter.
- **Object-Oriented**: Supports encapsulation of code within objects.
- **Beginner-Friendly**: Ideal for beginners, supporting diverse applications from text processing to games.

Programming syntax :

- Python syntax can be executed by writing directly in the Command Line:
- >>> print("Hello, World!")
  Output: Hello, World!
- Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:
- C:\Users\Your Name>python myfile.py

A Python program is read by a parser. Python was designed to be a highly readable language. The syntax of the Python programming language is the set of rules which defines how a Python program will be written.

- ## Comments in Python:

A comment begins with a **hash character(#)** which is not a part of the string literal and ends at the end of the physical line. All characters after the # character up to the end of the line are part of the comment and the Python interpreter ignores them. See

the following example. It should be noted that **Python has no multi-lines or block comments facility.**

Example1:

```
x = 1

# The initial value of x is 1.

if x > 0:

    print("These are two comments")  # Print a string.
```

Example2:

```
y = 5

# The initial value of y is 5.

if y % 2 == 1:

    print("y is an odd number")  # Check if y is odd.
```

- # Input / output in Python:

Output in Python:

```
print("Welcome to Python programming.")
#Output:Welcome to Python programming.
```

```
txt="Python is interesting language"
print(txt)
#Output:Python is interesting language
```

Taking Inputs in Python:

Inputting a String:

```
name=input("enter the name:")
```

Inputting an integer number:

```
age=int(input("enter the age:"))
```

Inputting a floating point number:

```
price=float(input("enter the price:"))
```

- **Multiple Statements on a Single Line:**

  You can write two separate statements into a single line using a **semicolon (;)** character between two line.

Example1:

```
print("Statement1")
print("Statement2")

#You can write above two statements in  a following way

print("Statement1"); print("Statement2")

#Output :
#Statement1
#Statement2
```

Example2:

```
#Writing multiple print statements on separate lines

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

```
print("Welcome to Python programming.")

#Writing the same statements on a single line using semi colon

print("Hello, World!"); print("Welcome to Python programming.")

# Output:
# Hello, World!
# Welcome to Python programming.
```

- **Indentation:**

```
# Program with Single Space Indentation

x = 1
if x > 0:
 print("This statement has a single space Indentation")
 print("This statement has a single space Indentation")


# Program with Single Tab Indentation

x = 1
if x > 0:
    print("This statement has a single tab Indentation")
    print("This statement has a single tab Indentation")
```

- **Python Reserve words:**

Python **reserved words** (keywords) are special words with predefined meanings that cannot be used as variable names. The following identifiers are used as reserved words of the language, and cannot be used as ordinary identifiers.

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	el	if	or	yield
assert	else	import	pass	
break	except	in	raise	

- Use blank lines to separate top-level function and class definitions and single blank line to separate methods definitions inside a class and larger blocks of code inside functions.
- When possible, put inline comments (should be complete sentences).
- Use spaces around expressions and statements.

- # Datatypes and Typecasting in Python:

  In Python, every value is an object with an identity, type, and value. Data types define how values can be used. Like Java and C++, Python has built-in data types, and additional types can be defined using extension modules. You can check a variable's type using `type()`. Objects whose values can change are **mutable**, while those that cannot are **immutable**.

Numbers:

Numbers are created by numeric literals. Numeric objects are immutable, which means when an object is created its value cannot be changed. Python has three distinct numeric types: integers, floating point numbers, and complex numbers. Integers represent negative and positive integers without fractional parts whereas floating point numbers represents negative and positive numbers with fractional parts. In addition, Booleans are a subtype of plain integers.

- **Integer Values:**

```
# Integer values
a = 1452
print(type(a))  # Output: <class 'int'>

b = -4587
print(type(b))  # Output: <class 'int'>

c = 0
print(type(c))  # Output: <class 'int'>
```

- **Float Values:**

```
# Float values
g = 9.31
print(type(g))  # Output: <class 'float'>

h = -11.23
print(type(h))  # Output: <class 'float'>

i = .34
print(type(i))  # Output: <class 'float'>

j = 2.2e-10
print(type(j))  # Output: <class 'float'>
```

- **Complex Numbers in Python:**

```
# Creating complex numbers
x = complex(1, 2)  # 1 + 2j
```

```
print(type(x))  # Output: <class 'complex'>
print(x)  # Output: (1+2j)

# Another way to define a complex number
z = 1 + 2j
print(type(z))  # Output: <class 'complex'>
print(z)  # Output: (1+2j)

# Adding complex numbers
z = z + 1 + 2j
print(type(z))  # Output: <class 'complex'>
print(z)  # Output: (2+4j)
```

- **Boolean:**

```
#Boolean (bool) in Python

# Boolean values in Python
x = True
print(type(x))  # Output: <class 'bool'>

y = False
print(type(y))  # Output: <class 'bool'>
```

- **Strings in Python:**

  In Python, a string type object is a sequence (left-to- right order) of characters. Strings start and end with single or double quotes Python strings are immutable. Single and double quoted strings are same and you can use a single quote within a string when it is surrounded by double quote and vice versa. Declaring a string is simple, see the following statements.

```
# Defining strings with single and double quotes
str1 = "String"  # Strings start and end with double quotes
print(str1)   #Output:String

# Using single quotes inside double quotes
str2 = "Day's"  # Single quote within double quotes
print(str2)  # Output: Day's

# Using double quotes inside single quotes
```

```
str3 = 'Day"s'  # Double quote within single quotes
print(str3)  # Output: Day"s
```

Special characters in strings:

The backslash **(\)** character is used to introduce a special character. See the following:

Escape Sequence Characters

- \n **Newline**
- \t **Horizontal Tab**
- \\ **Backslash**
- \' **Single Quote**
- \" **Double Quote**

```
print("The is a backslash (\\) mark.")

#Output:The is a backslash (\)

print("This is tab \t key")

#Output:This is tab    key

print("These are \'single quotes\'")

#Output:These are 'single quotes'

print("These are \"double quotes\"")

#Output:These are "double quotes"

print("This is a new line\nNew line")

#Output:
#This is a new line
#New line
```

String indices and accessing string elements:

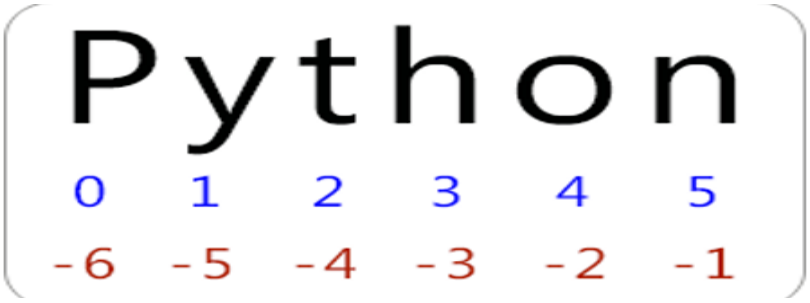
• Strings are arrays of characters and elements of an array can be accessed using indexing. Indices start with 0 from left side and -1 when starting from right side.

string1 ="PYTHON TUTORIAL"

```
print(string1[0])  # Print first character
#output=P

print(string1[-15])  # Print first character
#output=P

print(string1[14])  # Print last character
#output=L

print(string1[-1])  # Print last character
#output=L

print(string1[4])  # Print 4th character
#output=O

print(string1[-11])  # Print 4th character
#output=O

print(string1[16])  # Out of index range
#index error string out of range
```

- **String Slicing**

To cut a substring from a string is called string slicing. Here two indices are used separated by a colon (:). A slice 3:7 means indices characters of 3rd, 4th, 5th and 6th positions. The second integer index i.e. 7 is not included. You can use negative indices for slicing. See the following statements.



Lists :

A list is a container which holds comma-separated values (items or elements) between square brackets where Items or elements need not all have the same type.

Examples:

```
my_list1 = [5, 12, 13, 14]  # the list contains all integer values
print(my_list1)
#Output:[5, 12, 13, 14]
my_list2 = ['red', 'blue', 'black', 'white']  # the list contains all string
values
print(my_list2)
#Output:['red', 'blue', 'black', 'white']
my_list3 = ['red', 12, 112.12]  # the list contains a string, an integer and a
float values
print(my_list3)
#Output:['red', 12, 112.12]

my_list4 = [10, 'Python', False, 3.14]  # the list contains mixed data types
print(my_list4)
#Output:[10, 'Python', False, 3.14]
```

A list without any element is called an empty list.

Example:

```
my_list=[]
print(my_list)
#Output:[]
```

List indices:

List indices work the same way as string indices, list indices start at 0. If an index has a positive value it counts from the beginning and similarly it counts backward if the index has a negative value. As positive integers are used to index from the left end and negative integers are used to index from the right end, so every item of a list gives two alternatives indices.

Example:

```
color_list=["RED", "Blue", "Green", "Black"]  # The List has four elements;
indices start at 0 and end at 3

color_list[0]  # Return the First Element
#Output:'RED'

print(color_list[0],color_list[3])  # Print First and Last Elements
#Output:RED Black

color_list[-1]  # Return Last Element
#Output:'Black'

print(color_list[4])  # Creates Error as the index is out of range
#Output:#Error  indexes are out of range
```

List Slice :

Lists can be sliced like strings and other sequences. The syntax of list slices is easy; sliced_list =

List_Name[startIndex:endIndex]

Example:

```
color_list=["RED", "Blue", "Green", "Black"]  # The List has four elements;
indices start at 0 and end at 3
```

```
print(color_list[0:2])  # Cut first two items
#Output:['RED', 'Blue']

print(color_list[1:2])  # Cut second item
#Output:['Blue']

print(color_list[1:-2])  # Cut second item
#Output:['Blue']

print(color_list[0:3])  # Cut first three items
#Output:['RED', 'Blue', 'Green']

print(color_list[:])  # Creates copy of the original list
#Output:['RED', 'Blue', 'Green', 'Black']
```

Conditional Statements :

Python supports the usual logical conditions from mathematics:

Equals: a == b

Not Equals: a != b

Less than: a < b

Less than or equal to: a <= b

Greater than: a > b

Greater than or equal to: a >= b

e.g if b > a:

print("b is greater than a")