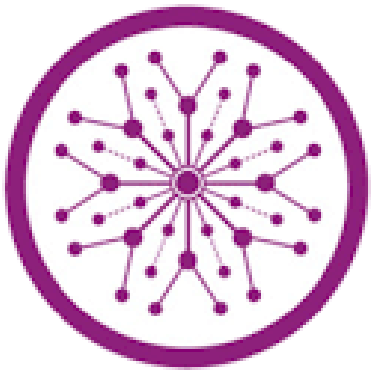
Lab Manual

Artificial Intelligence
(Python)



Faculty of Software Engineering

SUPERIOR UNIVERSITY LAHORE PAKISTAN

List of Experiments:

Lab. No	Experiment Names
1	Introduction to Python (& Installation), Syntax, Basic Functions, Control Structures, Loops & Functions
2	Lists, Tuples, Sets & Dictionary, Classes & Inheritance, Modules in Python & Examples
3	Simple Reflex Agents, Model based Reflex Agents
4	Graph in Python
5	Uninformed Search
6	Searching Algorithms: Informed/Heuristic Search
7	Game Search
8	Machine Learning (File Read, Data preprocessing)
9	Machine Learning (Naïve Bayes, D-Tree, KNN, SVM)
10	Introduction to Deep Learning
11	Data visualization using Python
12	Precision, Accuracy, Recall
13	Project
14	Project Evaluation

Lab 1-Part 1

Installing Python:

www.python.org

For Windows (32 bit): https://www.python.org/ftp/python/3.4.3/python-3.4.3.msi

For Windows (64 bit): https://www.python.org/ftp/python/3.4.3/python-3.4.3.amd64.msi

Installing Jupyter Notebook

https://www.anaconda.com/

Installing Pycharm Notebook

https://www.jetbrains.com/pycharm/download/

Opening IDLE

Go to the start menu, find Python, and run the program labeled 'IDLE'(Stands for Integrated DeveLopment Environment)

Code Example 1 - Hello, World!

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

Learning python for a C++/C# programmer

Let us try to quickly compare the syntax of python with that of C++/C#:

	C++/C#	**Python**
Comment begins with	//	#
Statement ends with	;	No semi-colon needed
Blocks of code	Defined by {}	Defined by indentation (usually four spaces)
Indentation of code and use of white space	Is irrelevant	Must be same for same block of code (for example for a set of statements to be executed after aparticular if statement)
Conditional statement	if-else if-else	if – elif – else:
Parentheses for loop execution condition	Required	Not required but loop condition followed by acolon : while a < n:print(a)

Calculations are simple with Python, and expression syntax is straightforward: the operators +, -, * and / work as expected; parentheses () can be used for grouping.

```
# Python 3: Simple arithmetic
```

```
1
>>> 1 / 2

0.5

>>> 2 ** 3             #Exponent operator

8

>>> 17 / 3          # classic division returns a float5.666666666666667

>>> 17 // 3          # floor division5

>>> 23%3               #Modulus operator2
```

Python Operators

Command	Name	Example	Output
+	Addition	4+5	9
-	Subtraction	8-5	3
*	Multiplication	4*5	20
/	Classic Division	19/3	6.3333
%	Modulus	19%3	5
**	Exponent	2**4	16
//	Floor Division	19/3	6

Comments in Python:

```
#I am a comment. I can say whatever I want!
```

Variables:

```
print ("This program is a demo of variables")

v = 1

print ("The value of v is now", v)v = v + 1
print ("v now equals itself plus one, making itworth", v)

2-print ("To make v five times bigger, you would haveto type v
= v * 5")

v = v * 5

print ("There you go, now v equals", v, "and not",v / 5)
```

Strings:

```
word1 = "Good" word2 = "Morning"
```

```
word3 = "to you too!"print (word1, word2)

sentence = word1 + " " + word2 + " " +word3print (sentence)
```

Relational operators:

Expression	Function
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
!=	not equal to
==	is equal to

Boolean Logic:

Boolean logic is used to make more complicated conditions for **if** statements that rely on more than one condition. Python's Boolean operators are **and**, **or**, and **not**. The **and** operator takes two arguments, and evaluates as **True** if, and only if, both of its arguments are True. Otherwise it evaluates to **False**.

The **or** operator also takes two arguments. It evaluates if either (or both) of its arguments are **False**.

Unlike the other operators we've seen so far, **not** only takes one argument and inverts it. Theresult of **not True** is **False**, and **not False** is **True**.

Operator Precedence:

Operator	Description
()	Parentheses
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus
* / % //	Multiply, divide, modulo, and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive 'OR' and regular 'OR'
<= < > >=	Comparison Operators
== !=	Equality Operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Conditional Statements:

'if' - Statement

```
y = 1

if y == 1:
```

```
print ("y still equals 1, I was just checking")
```

'if - else' - Statement

```
a = 1

if a > 5:

print ("This shouldn't happen.")else:

print ("This should happen.")
```

'elif' - Statement

```
z = 4

if z > 70:

print ("Something is very wrong")elif z < 7:

print ("This is normal")
```

TASKs:

Open IDLE and run the following program. Try different integer values for separate runs of the program. Play around with the indentation of the program lines of code and run it again. See what happens. Make a note of what changes you made and how it made the program behave. Also note any errors, as well as the changes you need tomake to remove the errors.

```
x = input("Please enter an integer: ")


if x < 0:

x = 0

print('Negative changed to zero')elif x ==0:

print('Zero')elif x ==1:

print('Single')else:

print('More')
```

The **input()** function prompts for input and returns a string.`a = input ("Enter Value for variable a: ") print (a)`

Indexes of String:

Characters in a string are numbered with *indexes* starting at 0:Example:

name = "J. Smith"

Accessing an individual character of a string:

variableName [**index**]

Example:

```
print (name, " starts with", name[0])
```

Output:

J. Smith starts with J

input:

input: Reads a string of text from user input.

Example:

```
name = input("What's your name? ") print (name, "... what a nice name!")
```

Output:

What's your name? Ali

Ali... what a nice name!

String Properties:

len(*string*) - number of characters in a string (including spaces)str.lower(*string*) - lowercase version of a string

str.upper(*string*) - uppercase version of a stringExample:

```
name = "Linkin Park" length = len(name) big_name = str.upper(name)

print (big_name, "has", length, "characters")
```

Output:

LINKIN PARK has 11 characters

Strings and numbers:

ord(*text*) - converts a string into a number.

Example: ord('a') is 97, ord("b") is 98, ...

Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.

chr (*number*) - converts a number into a string.

Example: chr(99) is "c"

Loops in Python:

The 'while' loop

```
a = 0

while a < 10:

a = a + 1 print (a)
```

Range function:

Range(5) #[0,1,2,3,4]

Range(1,5) #[1,2,3,4]

Range(1,10,2) #[1,3,5,7,9]

The 'for' loop

```
for i in range(1, 5):print (i )


for i in range(1, 5):print (i)
else:
print ('The for loop is over')
```

Functions:

How to call a function?

function_name(parameters)

Code Example - Using a function

```
def greet():   #function definition
   print("Hello")
   print("Good Morning")
greet()    #function calling
```

```
def add_sub(x,y)a=x+y                                                    return
a,b
result1, result2 =add_sub(5,10)print(result1, result2)
```

```
def multiplybytwo(x):
   return x*2
a = multiplybytwo(70)
```

The computer would actually see this:

a=140

Define a Function?

def function_name(parameter_1,parameter_2):

{this is the code in the function}

return {value (e.g. text or number) to return to the main program}

range() **Function:**

If you need to iterate over a sequence of numbers, the built-in function range() comes inhandy. It generates iterator containing arithmetic progressions:

```
>>> range(10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

It is possible to let the range start at another number, or to specify a different increment(even negative; sometimes this is called the 'step'):

```
>>> list(range(5, 10))[5, 6, 7, 8, 9]
```

```
>>> list(range(0, 10, 3) )

[0, 3, 6, 9]

>>> list(range(-10, -100, -30) )

[-10, -40, -70]
```

The range() function is especially useful in loops.

Accessing values in a list

Each value in a list is indexed according to its position in the list. The item in the first position is at index 0; the item at the second position is at index 1; and so on.

It looks something like this:

```
names = ["John", "Alice", "Sarah", "George"]
print(names[2])  # Sarah
```

Adding items to a list

One neat feature of lists is that they're mutable: we can change the values inside. This means we can add values, remove them, replace them, reorder them, etc.

```
names = ["John", "Alice", "Sarah", "George"]
names.append("Simon")
print(names[-1])
```

Removing items from a list

Just like with adding items, there are several options available to use when it comes to removing items from a list.

```
names = ["John", "Sarah", "Alice", "John"]
names.remove("John")
print(names)
```

LAB 1 TASK:

Mini projects (any 1)**:**

1. **Dynamic calculator (solving : 1+2×3(4-5÷4)-(3÷5))**
2. **To-Do list (Dynamic)**
3. **Tic tac toe (Dynamic)**
4. **Hangman (Dynamic) with printing hangman**