Artificial Intelligence

CSL-325

Lab Manual

Spring 2022

Department of Computer Science, Bahria University, Islamabad, Campus

List of Experiments:

Week. No	Topics Covered	Modules/IDE
1A, 1B, 1C, 1D	Introduction to Python (& Installation), Syntax, Basic Functions, Control Structures, Loops & Functions, Classes & Inheritance, Modules in Python & Examples	Anaconda Navigator, IDE (Spyder, Jupyter Notebook, Pycharm) ***They can use any one of them
2A, 2B	Lists, Tuples, Sets & Dictionary, Numpy, Pandas	Spyder, Numpy, Pandas
3	Simple Reflex Agents, Model based Reflex Agents	Spyder
4A,4B	Graph in Python, Matplotlib	Spyder, Matplotlib
5A, 5B	Uninformed Search (DFS, DLS, IDDFS), Informed Search (A star, Heuristic Search)	Spyder
6	Adversarial Search	Spyder
7	Tkinter (GUI)	Spyder, Tkinter
8	Unsupervised Learning (K-means, Agglomerative)	Spyder, Numpy, Sklearn/scikit-learn
9	Dimensionality Reduction, Principal Component Analysis (PCA)	Spyder, Numpy, PCA, Matplotlib
10	Supervised Learning (Artificial Neural Network)	Spyder, Keras, Tensorflow, Numpy
11	Naïve Bayes, K-nearest Neighbor	Spyder, GaussianNB, Label Encoder, Sklearn
12	Introduction to Prolog, Knowledgebase	Swish, Swipl, SwiPrologEditor Spyder, Pytholog
13	Expert System, Forward Chaining, Backward Chaining, Recursion	Swish, Swipl, SwiPrologEditor Spyder, Pytholog
14	Constraint Satisfaction Problem (CSP)	Spyder, Numpy, Pandas

Lab 1-A

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

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++ programmer

Let us try to quickly compare the syntax of python with that of 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 a particular if statement)
Conditional statement	if-else if-else	if – elif – else:
Parentheses for loop execution condition	Required	Not required but loop condition followed by a colon : while a < n: print(a)
Escape characters (e.g. –\n \t \‘ |	\	\

Lab 1-B

Math in Python

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 / 2
0.5
>>> 2 ** 3     #Exponent operator
8
>>> 17 / 3  # classic division returns a float
5.666666666666667
>>> 17 // 3  # floor division
5
>>> 23%3       #Modulus operator
2
```

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 it worth", v)
print ("To make v five times bigger, you would have to 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 + " " +word3
print (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 evaluate 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. The result 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")
```

Lab Journal 1-B

1. 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 to make 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')
```

Lab 1-C

Input from user:

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–

Index	0	1	2	3	4	5	6	7
Character	J	.		S	m	i	t	h

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 string

Example:

```
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 )
```

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 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 in handy. 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‘):

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

```
[0, 3, 6, 9]

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

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

The range() function is especially useful in loops.

Lambda Function:

Often, we quickly define mathematical functions with a one-line function called a *lambda* function. Lambda functions are great because they enable us to write functions without having to name them, ie, they're *anonymous*. No return statement is needed.

```
square = lambda x: x*x
print(square(3))
hypotenuse = lambda x, y: x*x + y*y

## Same as

# def hypotenuse(x, y):
#     return(x*x + y*y)

hypotenuse(3,4)
```

Default Arguments:

Functions may also have *default* argument values. Functions with default values are used extensively in many libraries. The default values are assigned when the function is defined.

```
def get_multiple(x, y=1):
    return x, y, x*y

print("With x and y:", get_multiple(10, 2))
print("With x only:", get_multiple(10))
```

Note that you can use the name of the argument in functions, but you must either use all the names, or get the position of the argument correct in the function call:

```
get_multiple(x=3), get_multiple(x=3, y=4)
get_multiple(y=4, x=3), get_multiple(3, y=4)
get_multiple(y=4, 3)
```

Lab Journal 1-C:

2. Write a simple calculator program. Follow the steps below:
    a. Declare and define a function named Menu which displays a list of choices for user such as addition, subtraction, multiplication, & classic division. It takes the choice from user as an input and return.
    b. Define and declare a separate function for each choice.
    c. In the main body of the program call respective function depending on user‘s choice.
    d. Program should not terminate till user chooses option to—Quit‖.

3. Write a unit conversion calculator program in in python. Follow the steps given below:
    a. Declare and define a function named Menu which displays a list of choices for user such as meter to km, km to m, centimetre to meter, & centime to millimetre. It takes the choice from user as an input and return.
    b. Define and declare a separate function for each choice.
    c. In the main body of the program call respective function depending on user‘s choice.
    d. Program should not terminate till user chooses option to―Quit‖.

Bonus Lab Task (Extra Credit):

To get extra credit you MUST complete and show me your program during the same Lab.

Implement the following functions for the calculator you created in the above task. (Task 2)

e. Factorial
f. x_power_y (x raised to the power y)
g. log
h. ln (Natural log)

Lab 1-D

Classes & Inheritance:

The word 'class' can be used when describing the code where the class is defined.
A variable inside a class is known as an *Attribute*
A function inside a class is known as a *method*

- A class is like a
  - Prototype
  - Blue-print
  - An object creator
- A class defines potential objects
  - What their structure will be
  - What they will be able to do
- Objects are instances of a class
  - An object is a container of data: attributes
  - An object has associated functions: methods

Syntax:
# Defining a class
class class_name:
[statement 1]
[statement 2]
[statement 3] [etc]

Inheritance Syntax:

```
class child_class(parent_class):
      def_init__(self,x):
            # it will modify the _init_ function from parent class
      # additional methods can be defined here
```

Example1:

```
class MyClass:
   i = 12345
   def f(self):
               return 'hello world'
x = MyClass()
print (x.i)
print (x.f() )
```

Example2:

```
class Complex:
    def_init_(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart
x = Complex(3.0, -4.5)
print (x.r,"      ",x.i )
```

Example3:

```
class Shape:
    def__init__(self,x,y):      #The__init__function always runs first
        self.x = x
        self.y = y
    description = "This shape has not been described yet"
    author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.x * self.y
    def perimeter(self):
        return 2 * self.x + 2 * self.y
    def describe(self,text):
        self.description = text
    def authorName(self,text):
        self.author =  text
    def scaleSize(self,scale):
        self.x = self.x * scale
        self.y = self.y * scale

a=Shape(3,4)
print (a.area())
```

Inheritance Example:

```
class Square(Shape):
   def_init__(self,x):
         self.x = x
         self.y = x

class DoubleSquare(Square):
   def_init__(self,y):
         self.x = 2 * y
         self.y = y
   def perimeter(self):
         return 2 * self.x + 2 * self.y
```

Module:

A module is a python file that (generally) has only definitions of variables, functions, and classes.

Example: Module name mymodule.py

```
# Define some variables:
ageofqueen = 78

# define some functions
def printhello():
    print ("hello")
# define a class
class Piano:
    def_init__(self):
        self.type = input("What type of piano?: ")
        self.height = input("What height (in feet)?: ")
        self.price = input("How much did it cost?: ")
        self.age = input("How old is it (in years)?: ")

    def printdetails(self):
```

```
        print ("This piano is a/an " + self.height + " foot")
        print (self.type, "piano, " + self.age, "years old
and costing " +self.price + " dollars.")
```

Importing module in main program:

```
### mainprogam.py ##
# IMPORTS ANOTHER MODULE

import mymodule
print
(mymodule.ageo
fqueen )
cfcpiano =
mymodule.Piano
()
cfcpiano.print
details()
```

Another way of importing the module is:

```
from mymodule import
Piano, ageofqueenprint
(ageofqueen)
cfcpiano = Piano()
cfcpiano.printdetail
s()
```

Lab Journal 1-D:

1. Create a class name basic_calc with following attributes and methods;Two integers (values are passed with instance creation)
   Different methods such as addition, subtraction, division, multiplication
   Create another class inherited from basic_calc named s_calc which should have thefollowing additional methods;
   Factorial, x_power_y,log, ln etc

2. Modify the classes created in the above task under as follows:
   Create a module name basic.py having the class name basic_calc with all the attributesand methods defined before.
   Now import the basic.py module in your program and do the inheritance step definedbefore i.e.
   Create another class inherited from basic_calc named s_calc which should have thefollowing additional methods;
   Factorial, x_power_y, log, ln etc