A yellow and black logo

Description automatically generatedA logo of a university of sciences and technology

Description automatically generated**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**

EC-350 Artificial Intelligence and Decision Support System

LAB MANUAL – 01

**Course Instructor:** Assoc Prof Dr Arsalan Shaukat

**Lab Engineer:** Kashaf Raheem

**Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Degree/ Syndicate: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# LAB # 1: OBJECT ORIENTED DESIGN PRACTICES IN PYTHON

**Lab Objective:**

* Familiarize students with the Python IDLE
* To teach students the conventional coding practices in Python
* To study object-oriented design practices in python.
* Applying object-oriented constructs to create hierarchical data structures.

**Hardware/Software required:**

Hardware: Desktop/ Notebook Computer

Software Tool: Python 3.10.0

**Lab Description:**

Python is a high-level general-purpose programming language. Because code is automatically compiled to byte code and executed, Python is suitable for use as a scripting language, Web application implementation language, etc. Because Python can be extended in C and C++, Python can provide the speed needed for even compute intensive tasks.

**Features of Python:**

* Contains in-built sophisticated data structures like strings, lists, dictionaries, etc.
* The usual control structures: if, if-else, if-elif-else, while, plus a powerful collection iterator (for).
* Multiple levels of organizational structure: functions, classes, modules, and packages. They assist in organizing code. An excellent and large example is the Python standard library.
* Compile on the fly to byte code -- Source code is compiled to byte code without a separate compile step. Source code modules can also be "pre-compiled" to byte code files.
* Python is portable language, can run on different platforms such as Windows, Linux and Unix etc.
* It is Dynamically typed language i.e. Data Type of value is decided in run-time, not in advance.
* Python uses indentation to show block structure. Indent one level to show the beginning of a block. Out-dent one level to show the end of a block. As an example, the following C- style code:

Text, letter

Description automatically generated

in Python would be:

A picture containing text

Description automatically generated

And the convention is to use four spaces (and no tabs) for each level of indentation.

**Interactive Python:**

If you execute Python from the command line with no script, Python gives you an interactive prompt. This is an excellent facility for learning Python and for trying small snippets of code. Many of the examples that follow were developed using the Python interactive prompt.

In addition, there are tools that will give you a more powerful and fancy Python interactive mode. One example is IPython, which is available at <http://ipython.scipy.org/>. You may also want to consider using IDLE. IDLE is a graphical integrated development environment for Python. It contains a Python shell which helps you code in python by Auto-indent, Debugging and Color-coding your program

**1. Installing Python:**

Download Python 3.6.2 for Windows from the following link:

<https://www.python.org/downloads/>

Install the downloaded executable file.

**2. Opening IDLE**

After installing Python, go to the start menu and search Python. Run the program labeled as Integrated Development Environment (IDLE).

**3. Print “Hello, World!” on CLI**

>>> print ("Hello, World!“)

**4. Mathematical Operations in Python:**

>>> 1 + 1

2

>>> 6-5

1

>>> 2\*5

10

>>> 5\*\*2

25

>>> 21/3

7

>>> 23/3

7

>>> 23.0/3.0

7.6666...

>>> 23%3

2

**5. Operators** **in Python**

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Name** | **Example** | **Output** |
| + | Addition | 4+5 | 9 |
| - | Subtraction | 8-5 | 3 |
| \* | Multiplication | 4\*5 | 20 |
| / | Division | 19/3 | 6 |
| % | Remainder | 19%3 | 5 |
| \*\* | Exponent | 2\*\*4 | 16 |

**6. Operator Precedence in Python:**

|  |  |
| --- | --- |
| **Operators** | **Descriptions** |
| () | Parentheses |
| \*\* | Exponent (Raise to the power) |
| \* / % // | Multiplication, Divide, Reminder, Floor Division |
| + - | Addition, Subtraction |
| >> << | Right and Left Bitwise Shift |

**7. Comments in Python:**

>>> #Commented Code

**8. Variables:**

Variables are containers for storing data values. In Python, it is not needed to declare variables by specific data type. The Data type of a variable is decided at run-time.

Text, letter

Description automatically generated

type () keyword tells the type of argument.

**Variable Names**

* Must start with underscore or with a letter.
* Cannot start with a number.
* Can contain underscores and alpha –numeric letters.
* Variables names are case-sensitive

**9. Strings:**

Strings are represented either by using double or single quotes.

**Python Example:**

word1 = "Good"

word2 = "Morning"

word3 = "to you too!"

print (word1, word2)

sentence = word1 + " " + word2 + " " +word3

print (sentence)

print(word1[2])

**10. Boolean Operators in Python:**

* Boolean operators return either TRUE or FALSE

|  |  |
| --- | --- |
| **Expression** | **Function** |
| **<** | **less than** |
| **<=** | **less that or equal to** |
| **>** | **greater than** |
| **>=** | **greater than or equal to** |
| **!=** | **not equal to** |
| **<>** | **not equal to (alternate)** |
| **==** | **equal to** |

**11. Conditional Statements:**

**‘if - else' - Statement**

a = 1

if a > 5:

print ("This shouldn't happen.") #condition true

else:

print ("This should happen.") #condition false

**‘elif' - Statement**

z = 4

if z > 70:

print ("Something is very wrong")

elif z < 7:

print ("This is normal")

**12. Input from user:**

**input(“ ”):**

a = input (“Enter Value for variable a: ”)

print (a)

input: Reads a string of text from user input.

***Example:***

name = input (“What's your name Lad? ")

print (name, "... what a nice name!")

Output:

What is your name Lad? Ali

Ali... what a nice name!

**13. Indexes of String:**

In Python strings are considered as arrays and characters in a string are numbered with indexes starting at 0:

***Example:***

name = "P. aishh“

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 00 | 11 | 22 | 33 | 44 | 55 | 66 | 77 |
| Character | pP | .. |  | aa | ii | ss | hh | ah |

Accessing an individual character of a string:

***variableName*** [ ***index*** ]

***Example:***

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

Output:

P. aishh starts with P

**14. 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

**15. 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"

**16. Loops in Python:**

**'while' loop**

a = 0

while a < 10:

a = a + 1

print (a)

**'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')

[**range ()**](http://docs.python.org/library/functions.html)**:**

If you do need to iterate over a sequence of numbers, the built-in function [range()](http://docs.python.org/library/functions.html) comes in handy. It generates lists 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]

**17. Functions:**

**Define a Function?**

**def** function\_name(parameter\_1, parameter\_2):

this statement is written within the function body

return;

**How to call a function?**

function\_name(parameters)

Code Example - Using a function

a = multiplybytwo(70)

The computer would see this:

a=140

**Function Scope:**

def changeme( mylist ):

"This changes a passed list into this function"

mylist = [1,2,3,4]; # This would assign new reference in mylist

print ("Values inside the function: ", mylist)

return

# Now you can call changeme function

mylist = [10,20,30];

changeme( mylist );

print ("Values outside the function: ", mylist)

**18. Lists:**

Lists are what they seem - a list of values. Each one of them is numbered, starting from zero. You can remove values from the list and add new values to the end. Example: Your many cats' names. *Compound* data types used to group together other values. The most versatile is the *list*, which can be written as a list of comma-separated values (items) between square brackets. List items need not all have the same type.

cats = ['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester']

print cats[2]

cats.append('Catherine')

#*Remove your 2nd cat, Snappy. Woe is you.*

del cats[1]

**19. Compound datatype:**

>>> a = ['spam', 'eggs', 100, 1234]

>>> a[1:-1]

['eggs', 100]

>>> a[:2] + ['bacon', 2\*2]

['spam', 'eggs', 'bacon', 4]

>>> 3\*a[:3] + ['Boo!']

['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boo!']

>>> a= ['spam', 'eggs', 100, 1234]

>>> a[2] = a[2] + 23

>>> a

['spam', 'eggs', 123, 1234]

**20. Replace some items:**

>>> a[0:2] = [1, 12]

>>> a

[1, 12, 123, 1234]

**21. Remove some items:**

>>> a[0:2] = []

>>> a

[123, 1234]

**22. Clear the list: replace all items with an empty list:**

>>> a[:] = []

>>> a

**23. Length of list:**

>>> a = ['a', 'b', 'c', 'd']

>>> len(a)

4

**24. Nest lists:**

>>> q = [2, 3]

>>> p = [1, q, 4]

>>> len(p)

3

>>> p[1]

[2, 3]

**25. Functions of lists:**

**list.append(x):** Add an item to the end of the list; equivalent to a[len(a):] = [x].

**list.extend(L):** Extend the list by appending all the items in the given list, equivalent to a[len(a):] = L.

**list.insert(i, x):** Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

**list.remove(x):** Remove the first item from the list whose value is x. It is an error if there is no such item.

**list.pop([i]):** Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list.

**list.count(x):** Return the number of times x appears in the list.

**list.sort():** Sort the items of the list, in place.

**list.reverse():** Reverse the elements of the list, in place.

**26. Tuples:**

Tuples are just like lists, but you can't change their values. Again, each value is numbered starting from zero, for easy reference. Example: the names of the months of the year.

months = ('January' , 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', ' December’)

|  |  |
| --- | --- |
| **index** | **Value** |
| 0 | Jan |
| 1 | Feb |
| 2 | Mar |
| 3 | April |
| 4 | May |
| 5 | Jun |
| 6 | Jul |
| 7 | Aug |
| 8 | Sep |
| 9 | Oct |
| 10 | Nov |
| 11 | Dec |

>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

>>> fruit = set(basket) # create a set without duplicates

>>> fruit

set(['orange', 'pear', 'apple', 'banana'])

>>> 'orange' in fruit # fast membership testing

True

>>> 'crabgrass' in fruit

False

**27. Dictionaries:**

Dictionaries are like what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition.

In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary are not numbered - they are not in any specific order, either - the key does the same thing. You can add, remove, and modify the values in dictionaries.

***Example:*** telephone book.

phonebook = {'Andrew Parson':8806336,

'Emily Everett':6784346, 'Peter Power':7658344,

'Lewis Lame':1122345}

#Add the person 'Gingerbread Man' to the phonebook:

phonebook['Gingerbread Man'] = 1234567

#Delete the person 'Andrew Parson' to the phonebook:

del phonebook['Andrew Parson']

**OOP in Python**

Python supports object-oriented programming constructs that can be utilized to create sophisticated codebase employing decent data structures. In this lab, different object-oriented programming constructs will be revised, and they will be used to create different algorithms employing different data structures.

**1. Classes**

Classes are the fundamental constructs in object-oriented paradigms, and they are used to hold methods and variables related to common entity. All the attributes within the class can be accessed through its instance or object. Following examples shows the usage of classes:

***Example 1:***

class MyClass:

i = 12345 #Class Variable

def f(self):

return 'hello world'

x = MyClass() #object of MyClass

print (x.i)

print (x.f())

***Example 2:***

class Shape:

def \_\_init\_\_(self,x,y): # Constructor

self.x = x # Instance Variable

self.y = y # Instance Variable

description = "This shape has not been described yet"

author = "Nobody has claimed 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()

**2. Inheritance**

All classes have a property that they can inherit from other classes. This is one of the fundamental concepts of object-oriented design practices. Python also supports inheritance. Following code snippet describes how you can inherit from another classes:

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): # Method Overriding

return 2 \* self.x + 3 \* self.y

**3. Modules**

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

For example:

Define a module.py file and write a following code script.

# Define a variable:

Age = 78

# Define a method

def Print():

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.")

Now make a main.py file and import the module:

import module

print (module.Age)

module.Print()

o=module.Piano()

o.PrintDetails()

**4. Trees**

Python does not have in-built support for trees. So, we would be utilizing classes to create trees. ***For example:***

class Tree():

def \_\_init\_\_(self):

self.left = None

self.right = None

self.data = None

root = Tree()

root.data = "root"

root.left = Tree()

root.left.data = "left"

root.right = Tree()

root.right.data = "right"

root.left.left = Tree()

root.left.left.data = "left 2"

root.left.right = Tree()

root.left.right.data = "left-right"

print(root.left.left.data)

**Lab Tasks:**

**Q1: Write a program that lets the user enter in some English text, then converts the text to Pig-Latin.**

To review, Pig-Latin takes the first letter of a word, puts it at the end, and appends “ay”. The only exception is if the first letter is a vowel, in which case we keep it as it is and append “hay” to the end.

E.g. “hello” -> “ellohay”, and “image” -> “imagehay”

***Hint:*** *Split the entered string through split() method and then iterate over the resultant list, e.g. “My name is John Smith”.split(“ ”) -> [“My”, “name”, “is”, “John”, “Smith”]*

**Q2: Write a method to calculate Fibonacci series up to ‘n’ points. After calculating the series, the method should return it to main.**

**Q3: Write a simple program that builds a random password generator. For password generator the user must enter total number of passwords and their lengths. Display all the passwords with random characters.**

**Q4: Create a class named ‘Complex’ that must have the following attributes:**

**Variables named ‘Real’ and ‘Imaginary’**

**Methods named Magnitude () and Orientation ()**

**Take a complex number from user in main and print its magnitude and orientation. You have a liberty to create methods signature as you like.**

**Q5: Create the following Binary Search Tree and search for the node ‘13’. You can hard code the tree as well, but it is better if you create it dynamically at run time (You must have learned in Data Structures & Algorithms). Also, tell the time performance of searching the node ‘13’ in Big-O notation.**

**Conclusion:**

In this lab, you learned the basics of Python and its important features and practiced OOP in Python.