National University of Computer and Emerging Sciences



Lab Manual 04

CL461-Artificial Intelligence Lab

|  |  |
| --- | --- |
| Course Instructor | Dr. Mubasher Baig |
| Lab Instructor (s) | Saad Ali  Mahmood Hussain |
| Section | A |
| Semester | Spring 2021 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

Table of Contents

[1](#_heading=h.30j0zll) Objectives 2

[2](#_heading=h.1fob9te) Task Distribution 3

[3](#_heading=h.3znysh7) Python Classes 3

3.1 Creating Cat Class 3

[3.2 Creating Objects](#_heading=h.uksxxhumufxn) 4

3.3Inheritance 4

[4 Python Data Visualizations 6](#_heading=h.gme2d56qwbi9)

[4.1](#_heading=h.4o4glbq7jt6g) Creating a Simple Plot 7

[4.2](#_heading=h.17dp8vu) Creating a Scatter Plot 8

5Exercise (30) 9

6Submission Instructions 13

# Objectives

After performing this lab, students shall be able to understand Python data structures which include:

* Python Class and Objects
* Python Inheritance
* Python Data Visualizations

# Task Distribution

|  |  |
| --- | --- |
| **Total Time** | **170 Minutes** |
| Python Class and Objects | 30 Minutes |
| Python Inheritance | 20 Minutes |
| Python Data Visualizations | 20 Minutes |
| Exercise | 90 Minutes |
| Online Submission | 10 Minutes |
|  |  |

# Classes

Object-oriented programming (OOP)is one of the most effective approaches to writing programs. In OOP you write classes that represent real-world things and situations, and you create objects based on these classes. When you write a class, you define the general behavior that a whole category of objects can have.

## Creating a Cat Class,

class Cat():

"""A simple attempt to model a cat."""

def \_\_init\_\_(self, name, age):

"""Initialize name and age attributes."""

self.name = name

self.age = age

def sit(self):

"""Simulate a cat sitting in response to a command."""

print(self.name.title() + " is now sitting.")

def roll\_over(self):

"""Simulate rolling over in response to a command."""

print(self.name.title() + " rolled over!")

Important points are:

* We define a class called Cat. By convention, capitalized names refer to classes

in Python. The parentheses in the class definition are empty because we’re

creating this class from scratch.

* Using """ """ We write a docstring describing what this class does.
* The *\_\_init\_\_()* is like Constructor to a class, called automatically when the object is created. This method has two leading underscores and two trailing underscores, a convention that helps prevent Python’s default method names from conflicting with your method names. Except, self, other two’s *name and age* are parameters needed to pass by when the object is created. the method call will automatically pass the self argument.
* *self* is an instance of the class within the class, class variables are accessible through self instance.

## Creating a Cat Object,

Think of a class as a set of instructions for how to make an instance. The class Cat is a set of instructions that tells Python how to make individual instances representing specific cats.

Let’s make an instance representing a specific cat:

my\_cat = Cat('willie', 6)

print("My dog's name is " + my\_cat.name.title() + ".")

print("My dog is " + str(my\_cat.age) + " years old.")

Code details are below:

* We tell Python to create a cat whose name is 'willie' and whose age is 6. When Python reads this line, it calls the \_\_init\_\_() method in Cat with the arguments 'willie' and 6
* Using *my\_cat.age*, *my\_cat.name* we are calling attributes of the class.
* To call the class functions *my\_cat.sit(), my\_cat.roll\_over()* to be used, () comes with functions

## Inheritance

You don’t always have to write from scratch when writing with class, When one class inherits from another class, it automatically takes all attributes and functions of the first class. First class is also called Parent Class, While Inheriting class is called child one. The child class inherits every attribute and method from its parent class but is also free to define new attributes and methods of its own.

Let's say we have a **Car** Class:

class Car():

"""A simple attempt to represent a car."""

def \_\_init\_\_(self, make, model, year):

self.make = make # to take manufacturer

self.model = model # to take model

self.year = year # year of manufacturing

self.odometer\_reading = 0 # meter reading set to 0

#method 1

def get\_descriptive\_name(self):

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

#method 2

def read\_odometer(self):

print("This car has " + str(self.odometer\_reading) + " miles on it.")

#method 3

def update\_odometer(self, mileage):

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

#method 4

def increment\_odometer(self, miles):

self.odometer\_reading += miles

#method 5

def fill\_gas\_tank(self):

print("Gas tank needs to be filled")

Points:

* We start with Car. When you create a child class, the parent class must be part of the current file and must appear before the child class in the file.
* It has the four attributes, make, model, year and odometer\_reading. (Note: odometer\_reading is initialized through default value)
* The class contains 5 functions

Now, we opted to create a child class, *ElectricCar*

class ElectricCar(Car):

"""Represent a car, specific to electric vehicles."""

#constructor

def \_\_init\_\_(self, make, model, year):

"""Initialize attributes of the parent class."""

super().\_\_init\_\_(make, model, year)

# Attribute of the child class

self.battery\_size = 70

#function 1

def describe\_battery(self):

"""Print a statement describing the battery size."""

print("This car has a " + str(self.battery\_size) + "-kWh battery.")

# overriding a function in child class

def fill\_gas\_tank(self):

"""Electric cars don't have gas tanks."""

print("This car doesn't need a gas tank!")

Important points here are:

* class ElectricCar(Car)defines the name of the current class which is ElectricCar and the Parent Class which is Car, for inheritance one has to pass the parent class name in the Constructor of Child class.
* super().\_\_init\_\_(make, model, year) One inheritance rule is, Child has to call the parent constructor first before executing its own. We are calling through parent class object (super, in child we call it so), by passing values that are required to call its constructor.
* Child class can have its own attributes.
* Acknowledge, fill\_gas\_tank() is a function that is defined in both classes, *parent and child,* This terminology is called Method Overriding. Which is when both parent and children can have a function with the same name and signature.Python deals the scenario by examining object type, if the object is of Parent Class, then the Parent version will be called and vice versa.

# Matplotlib: Data Visualizations

Data visualization involves exploring data through visual representations. It’s closely

associated with data mining, which uses code to explore the patterns and connections in a

data set. A data set can be just a small list of numbers that fits in one line of code or many gigabytes of data.

Python has inbuilt data visualization capabilities.i.e Matplotlib, which is a mathematical plotting library. Matplotlib comes with Jupyter, Spyder Installation of Anaconda, If you wish to install it on **Linux**.

sudo apt-get install python3-matplotlib

Using **Pip**:

pip install --user matplotlib

**4.1** **Plotting a Simple Line Plot**

Let's plot a simple line plot using matplotlib, First, import the matplotlib module pyplot, Matplotlib is an open source library, For python graphing, we use the pyplot module of it.

import matplotlib.pyplot as plt

To draw a line plot between the list of squares and their natural counterparts, the following snippet is to be used.

import matplotlib.pyplot as plt

#data

x = [1,2,3,4,5]

squares = [1, 4, 9, 16, 25]

#draw lineplot(by default)

plt.plot(x,squares)

#label to x coordinate

plt.xlabel('x')

#label to y coordinate

plt.ylabel('Square(x)')

#title of graph

plt.title('x vs Square(x)')

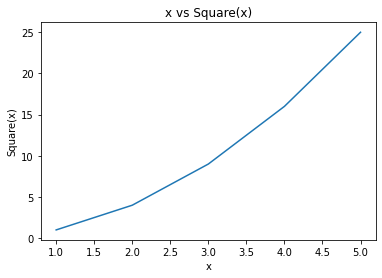
#show the commulative result

plt.show()

Important Points to graph are:

* first line import the pyplot library as plt alias
* x and square are the two lists, created containing data to plot
* *xlabel* is function to show label along x-coordinate of the graph
* *ylabel* is function to show label along y-coordinate of the graph
* *title* shows the title of the graph
* show function call is an option, it is used to draw cumulative graphs.

The resultant graph is:



**4.2** **Plotting a Scatter Plot**

ScatterPlot is used to draw a series of points, we can pass multiple lists in there, first one depicting values of x-axis and second one for y-axis, for scatter plotting we use *scatter* function instead of *plot* function.

*.*

#import the library

import matplotlib.pyplot as plt

#data

x\_values = [1, 2, 3, 4, 5]

y\_values = [1, 4, 9, 16, 25]

#scatter instead of plot, s is pointsize

plt.scatter(x\_values, y\_values,s=80)

# label to x coordinate

plt.xlabel('x')

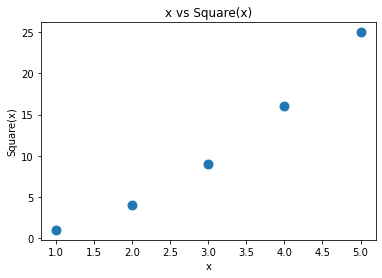
#label to y coordinate

plt.ylabel('Square(x)')

#title of graph

plt.title('x vs Square(x)')

The Resultant graph is:



One can also assign custom point colors and color of the edges

plt.scatter(x\_values, y\_values, c='red', edgecolor='none', s=40)

#s is the point size

**4.3** **Saving a Plot**

If you want your program to automatically save the plot to a file, you can replace the call to plt.show() with a call to plt.savefig().

plt.savefig('squares\_plot.png', bbox\_inches='tight')

The first argument is a filename for the plot image, which will be saved in the same directory as the current file. The second argument trims extra whitespace from the plot. If you want the extra whitespace around the plot, you can omit this argument.

# Exercise (20 marks)

## Generate a Random Walk (10)

In this exercise, We are using Python to create data for a random walk. After creating the data, One has to use matplotlib to plot it. Random walk is a path having no clear direction, at each point or after a set of points one has to decide either to move left or right.

Create a class named *RandomWalk.py*

*RandomWalk*

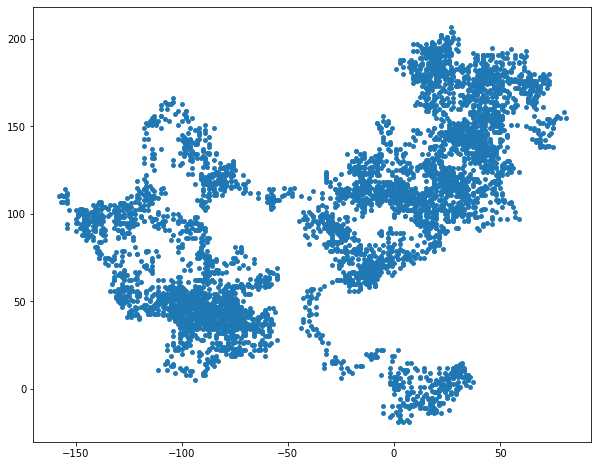
no\_of\_step: Int

x\_values: List

y\_values: List

fill\_walk()

After generating data for the desired number of steps, one also has to plot it. Using *Scatterplot.* The Output should be similar to this:



## 

Person Student

LMS

## Implementing Multiple Inheritance (10+2 marks)

Suppose, You are required to make LMS for university. Three classes enlisting details are given below.

***Person:***

Name, Phone\_Number, Age (class variables)

showName (), showPhoneNumber(), describe()(show name and phone number with age)

are the functions of class.

***Student:***

student\_id(class variable)

describe(shows student\_id)

Person

name : String

phone : String

age : Int

showName()

showPhone()

describe()

Student

student\_id: Int

describe()

***LMS:***

inherits both Person and Student class.

main function is given below:

"""Object 1"""

id1 = LMS('John', 30,'123456' ,'102')

print('Name of id1 is ')

id1.showName()

print('Description for id1 returned by portal are ')

id1.describe()

"""Object 2"""

id2 = LMS('Ahmed', 22,'78601' ,'007')

print('Name of id2 is ')

id2.showName()

print('Description for id2 returned by portal are ')

id2.describe()

Bonus:

Your output should match mine:

Name of id1 is

John

Description for id1 returned by portal are

My Id is 102

Name of id2 is

Ahmed

Description for id2 returned by portal are

My Id is 007

# Submission Instructions

Always read the submission instructions carefully.

* Rename your Jupyter notebook to your roll number and download the notebook as **.ipynb** extension.
* To download the required file, go to **File->Download .ipynb**
* Only submit the **.ipynb** file. DO NOT **zip** or **rar** your submission file
* Submit this file on Google Classroom under the relevant assignment.
* Late submissions will not be accepted