**Laboratory Manual**



|  |  |
| --- | --- |
| **Name of the School:** | **School of Computing**  **Sciences and Engineering** |
| **Name of the Department:** | **Computer Science and Engineering** |
| **Name of the Programme:** | **B.Tech AIML** |
| **Course/ Course Code:** | **Programming Using Python 17YCM711** |
| **Academic Year:** | **2023-24** |

# Guidelines

## Laboratory rules

1. Attendance is required for all lab classes. Students who do not attend lab will not receive credit.

1. Ensure that you are aware of the test and its procedure before each lab class. **You will NOT be allowed to attend the class if you a renort unprepared!**

1. Personal safety is top priority. Do not use equipment that is not assigned to you.

1. All accidents must be reported to your instructor or laboratory supervisor.

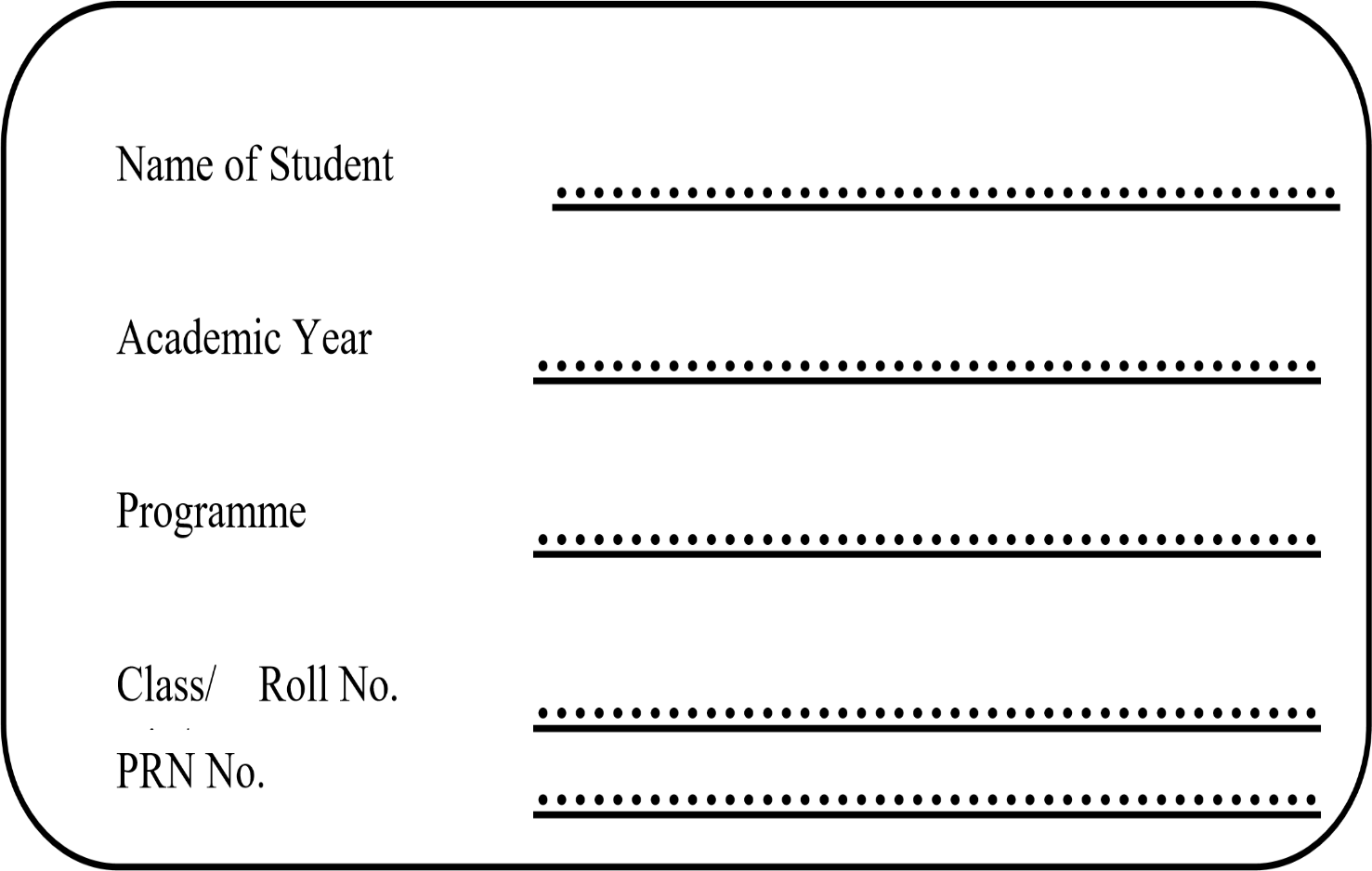
1. The surroundings near the equipment must be cleaned before leaving each lab class.

1. Ensure that readings are checked and marked by your TA for each lab period.

## Laboratory report

1. Each student has to submit the report for each experiment.
2. Your report is to be written only in the space provided in the manual
3. Please write your number and, batch and group number.
4. Your report must be neat, well organized, and make a professional impact. Label all axes and include proper units.
5. Your reports should be submitted within 7 days and before the beginning of the lab class of the next batch.
6. Your reports will be as per rubrics provided at the end of each experiment

Anyone caught plagiarizing work in the laboratory report, from a current or past student's notebook, will receive 0 on the grading scale.



# CERTIFICTAE

This is to certify that

Mr./Miss……………………………..………………………………… Roll no…………………... PRN No…………………………………

of class……… …………..has satisfactorily/unsatisfactorily

completed the Term Work of Course ……………………………….

in this School during academic year ………………….

Course Teacher Head of Department Dean Academics

**Table of Contents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Title of Experiment** | **Date** | **Remark** | **Sign** |
| 1. | Python program to read the contents of a file in reverse order. |  |  |  |
| 2. | Python program to create a class in which one method accepts a string from the user and other prints it. |  |  |  |
| 3. | Write a Python code to read csv file using pandas module and print the first and last five lines of a file. |  |  |  |
| 4. | Write a Python to double a given number and add two numbers using lambda function. |  |  |  |
| 5. | Write a Python to use filter() function for filter only even numbers from a given list.. |  |  |  |
| 6. | Create a list in python and perform the following operations. |  |  |  |
| 7. | Python program to illustrate the concept of threading . |  |  |  |
| 8. | Write a Program for Currency Converter. |  |  |  |
| 9. | Program for data visualization using Matplotlib for Scatter plot. |  |  |  |
| 10. | Program for sending current time string to the client using Socket Programming. |  |  |  |

**Experiment 1**

**Aim:** Python program to read the contents of a file in reverse order.

**Theory:** The Python program reads the contents of a file and prints them in reverse order. It accomplishes this by opening the specified file, reading its content, and then reversing the content before displaying it. The program allows users to analyze the content of a file in the opposite order from its original arrangement. This can be useful for tasks like examining logs or reviewing the contents of a file in a different perspective.

**Example 1:** Program to read the contents of a file in reverse order using Slicing.

**Code:**

# The file to be read

with open("D:/python\_rohit/file1.txt", "r") as myfile:

my\_data = myfile.read()

# Reversing the data by passing -1 for [start: end: step]

rev\_data = my\_data[::-1]

# Displaying the reversed data

print("Reversed data = ",rev\_data)

**Output :**

Reversed data = kihsaN ,ytisrevinU pidnaS ,gnireenignE dna ecneicS retupmoC fo loohcS

**Example 2:** Program to read the contents of a file in reverse order by looping.

**Code:**

# Opening the file to read

my\_data = open("D:/python\_rohit/file1.txt", "r")

# reversing the data

for myLine in my\_data:

l = len(myLine)

rev\_data = ''

while(l>=1):

rev\_data = rev\_data + myLine[l-1]

l=l-1

print("Reversed data = ",rev\_data) # Displaying the reversed data

**Output :**

Reversed data = kihsaN ,ytisrevinU pidnaS ,gnireenignE dna ecneicS retupmoC fo loohcS

**Experiment 2**

**Aim:** Python program to create a class in which one method accepts a string from the user and other prints it.

**Theory:** The provided Python program involves the creation of a class containing a method. The specific method within the class is designed to accept an argument and subsequently print the received input. The structure of the program revolves around encapsulating this functionality within a class, promoting code organization and reusability. The core purpose is to showcase the implementation of a method that handles input and produces an output through the print statement within the context of a Python class.

**Code:**

class Country:

def AcceptCountry(self):

self.cname=input("Enter Country Name:")

def DisplayCountry(self):

print("Country Name is:",self.cname)

class State(Country):

def AcceptState(self):

self.sname=input("Enter State Name:")

def DisplayState(self):

print("State Name is:", self.sname)

#main body

Obj=State()

Obj.AcceptCountry()

Obj.AcceptState()

Obj.DisplayCountry()

Obj.DisplayState()

**Output:**

Enter Country Name:India

Enter State Name:Maharashtra

Country Name is: India

State Name is: Maharashtra

**Experiment 3**

**Aim:** Write a Python code to read csv file using pandas module and print the first and last five lines of a file.

**Theory:** It uses the **pandas** module to read a CSV file and prints the first and last five lines of the file. It demonstrates the simplicity and efficiency of using **pandas** for reading and analyzing tabular data. The **read\_csv** function is employed to load the data, and the **head()** and **tail()** functions are utilized to display the initial and final rows of the dataset, respectively. This code is useful for quick data exploration and understanding the structure of the CSV file.

**Code:**

import pandas as pd

# Getting dataset via url

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Define the column names

col\_names = ["sepal\_length\_in\_cm",

"sepal\_width\_in\_cm",

"petal\_length\_in\_cm",

"petal\_width\_in\_cm",

"class"]

# Read data from URL

iris\_data = pd.read\_csv(url, names=col\_names)

# df = pd.read\_csv('countries.csv') (Reading from actual CSV file)

iris\_data.head()

rows=iris\_data.tail()

print(rows)

**Output:**

sepal\_length\_in\_cm sepal\_width\_in\_cm petal\_length\_in\_cm \

145 6.7 3.0 5.2

146 6.3 2.5 5.0

147 6.5 3.0 5.2

148 6.2 3.4 5.4

149 5.9 3.0 5.1

petal\_width\_in\_cm class

145 2.3 Iris-virginica

146 1.9 Iris-virginica

147 2.0 Iris-virginica

148 2.3 Iris-virginica

149 1.8 Iris-virginica

**Experiment 4**

**Aim:** Write a Python to double a given number and add two numbers using lambda function.

**Theory:** It uses a lambda function to create a concise and inline function for doubling a given number and adding two numbers. The lambda function takes a single argument, doubles it, and then adds two to the result. The script likely involves user input or predefined values for the numbers, demonstrating a quick and efficient way to perform these arithmetic operations using a lambda function in Python.

**Code:**

# Define the lambda functions

double = lambda x: x \* 2

add = lambda x, y: x + y

# Take input from the user

num = int(input("Enter a number: "))

num1 = int(input("Enter first number: "))

num2 = int(input("Enter second number: "))

# Apply the lambda functions

result1 = double(num)

result2 = add(num1, num2)

# Display the results

print("Double of", num, "is", result1)

print("Sum of", num1, "and", num2, "is", result2)

**Output:**

Enter a number: 5

Enter first number: 2

Enter second number: 2

Double of 5 is 10

Sum of 2 and 2 is 4

**Experiment 5**

**Aim:**

**Example1**: Write a Python to use filter() function for filter only even numbers from a given list..

**Theory:** Ituses the **filter()** function to selectively filter out even numbers from a given list. The **filter()** function takes a function and an iterable (in this case, a list) as its arguments. The function checks each element of the list and retains only those that satisfy a specified condition.

**Code:**

# Initialisation of list

lis1 = [1, 2, 3, 4, 5]

def is\_even(x): return x % 2 == 0

# using filter

lis2 = list(filter(is\_even, lis1))

# Printing output

print(lis2)

**Output:**

[2, 4]

**Example2**: Write a Python to use filter() function for filter out vowels from the list. # function that filters vowels.

**Code:**

def fun(variable):

letters = ['a', 'e', 'i', 'o', 'u']

if (variable in letters):

return True

else:

return False

# sequence

sequence = ['g', 'e', 'e', 'j', 'k', 's', 'p', 'r']

# using filter function

filtered = filter(fun, sequence)

print('The filtered letters are:')

for s in filtered:

print(s)

**Output:**

The filtered letters are:

e

e

**Experiment 6**

**Aim:** Create a list in python and perform the following operations.

1. Append()

2. Insert()

3. Remove()

4. Pop()

5. Len()

6. Max()

7. Min()

**Theory:**

In Python, you can perform various operations on lists, a versatile data structure. Here's a brief summary of common list operations:

**Append():**

Adds an element to the end of the list.

Example: my\_list.append(10)

**Insert**():

Inserts an element at a specific index in the list.

Example: my\_list.insert(2, 5) (Insert 5 at index 2)

**Remove():**

Removes the first occurrence of a specified value.

Example: my\_list.remove(10) (Removes the first occurrence of 10)

**Pop():**

Removes and returns the element at a specified index, or the last element if no index is provided.

Example: my\_list.pop(2) (Removes and returns the element at index 2)

**Len():**

Returns the number of elements in the list.

Example: length = len(my\_list)

**Max():**

Returns the maximum value in the list.

Example: maximum = max(my\_list)

**Min():**

Returns the minimum value in the list.

Example: minimum = min(my\_list)

These operations provide flexibility for manipulating and analyzing data in lists, making them a fundamental part of Python programming.

**Code:**

# Crating List

my\_list=[1,2,3]

my\_list.append(9)

my\_list.append(8)

print("The list after append() operation is: ",my\_list)

#O/P: The list after append() operation is: [1, 2, 3, 9, 8]

my\_list.extend([20,21])

print("The list after axtend() operator is: ",my\_list)

#O/P: The list after axtend() operator is: [1, 2, 3, 9, 8, 20, 21]

my\_list.insert(5,30)

print("The list after insert() operator is: \n",my\_list)

#O/P: The list after insert() operator is: [1, 2, 3, 9, 8, 30, 20, 21]

my\_list.remove(8)

print("The list after remove() operator is: \n",my\_list)

#O/P: The list after remove() operator is: [1, 2, 3, 9, 30, 20, 21]

my\_list.pop()

print("The list after pop() operator is:\n",my\_list)

#O/P: The list after pop() operator is: [1, 2, 3, 9, 30, 20]

print("Length of the list is: ",len(my\_list))

print("Maximum element in the list is: ",max(my\_list))

print("Minimum element in the list is: ",min(my\_list))

**Output:**

Length of the list is: 6

Maximum element in the list is: 30

Minimum element in the list is: 1

**Experiment 7**

**Aim:** Python program to illustrate the concept of threading .

**Theory:** It demonstrates the concept of threading, a technique used to execute multiple threads (lightweight sub-processes) concurrently within a single process. Threading is particularly useful for tasks that can be performed simultaneously, enhancing program efficiency and responsiveness. In this program, the threading module is likely utilized to create and manage multiple threads. The code showcases how threading can be employed to execute various tasks concurrently, allowing for better utilization of system resources and potentially improving the overall performance of the program.

**Code**:

import threading

import os

def task1():

print("Task 1 assigned to thread: {}".format(threading.current\_thread().name))

print("ID of process running task 1: {}".format(os.getpid()))

def task2():

print("Task 2 assigned to thread: {}".format(threading.current\_thread().name))

print("ID of process running task 2: {}".format(os.getpid()))

if \_\_name\_\_ == "\_\_main\_\_":

# print ID of current process

print("ID of process running main program: {}".format(os.getpid()))

# print name of main thread

print("Main thread name: {}".format(threading.current\_thread().name))

# creating threads

t1 = threading.Thread(target=task1, name='t1')

t2 = threading.Thread(target=task2, name='t2')

# starting threads

t1.start()

t2.start()

# wait until all threads finish

t1.join()

t2.join()

**Output:**

ID of process running main program: 14644

Main thread name: MainThread

Task 1 assigned to thread: t1

ID of process running task 1: 14644

Task 2 assigned to thread: t2

ID of process running task 2: 14644

**Experiment 8**

**Aim**: Write a Program for Currency Converter.

**Theory:** The program for a Currency Converter is designed to convert one currency to another based on the user's input. It typically involves fetching the latest exchange rates, receiving user input for the amount to convert, and performing the conversion calculation. The user interface may include prompts for selecting the source and target currencies. The program ensures accurate and up-to-date exchange rates to provide users with precise conversion results. It serves as a practical tool for individuals or businesses dealing with multiple currencies, facilitating seamless financial transactions across different monetary units.

**Code**:

# Import the modules needed

import requests

class Currency\_convertor:

# empty dict to store the conversion rates

rates = {}

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

data = requests.get(url).json()

# Extracting only the rates from the json data

self.rates = data["rates"]

# function to do a simple cross multiplication between

# the amount and the conversion rates

def convert(self, from\_currency, to\_currency, amount):

initial\_amount = amount

if from\_currency != 'EUR' :

amount = amount / self.rates[from\_currency]

# limiting the precision to 2 decimal places

amount = round(amount \* self.rates[to\_currency], 2)

print('{} {} = {} {}'.format(initial\_amount, from\_currency, amount,

to\_currency))

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

# YOUR\_ACCESS\_KEY = 'GET YOUR ACCESS KEY FROM fixer.io'

url = str.\_\_add\_\_('http://data.fixer.io/api/latest?access\_key=',

YOUR\_ACCESS\_KEY)

c = Currency\_convertor(url)

from\_country = input("From Country: ")

to\_country = input("TO Country: ")

amount = int(input("Amount: "))

**Input :**

From Country: USD

TO Country: INR

Amount: 1

**Output :**

1 USD = 70.69 INR

**Experiment 9**

**Aim**:

**Example1:** Program for data visualization using Matplotlib for Scatter plot.

**Code:**

import pandas as pd

import matplotlib.pyplot as plt

# reading the database

data = pd.read\_csv("tips.csv")

# Scatter plot with day against tip

plt.scatter(data['day'], data['tip'], c=data['size'],

s=data['total\_bill'])

# Adding Title to the Plot

plt.title("Scatter Plot")

# Setting the X and Y labels

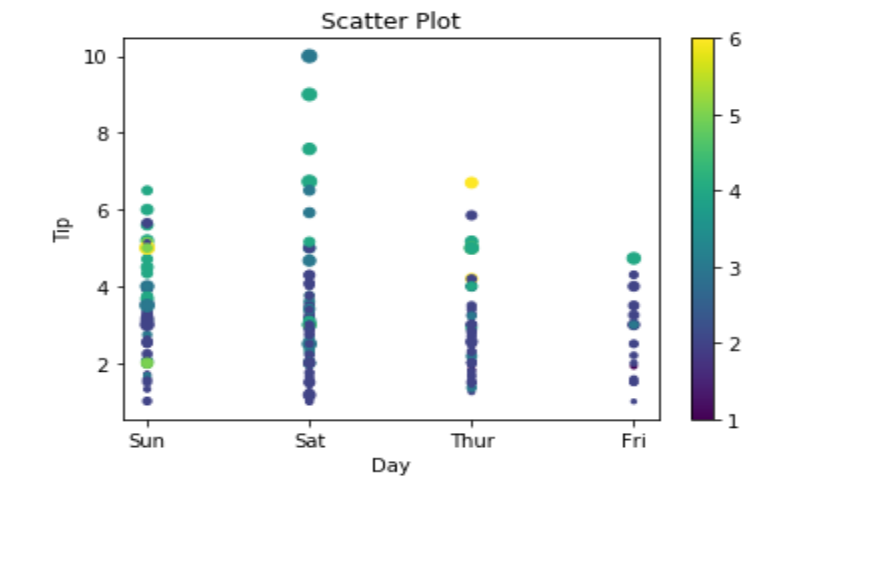
plt.xlabel('Day')

plt.ylabel('Tip')

plt.colorbar()

plt.show()

**Output:**



**Example 2**: Program for data visualization using Matplotlib for Bar Chart

**Code:**

import pandas as pd

import matplotlib.pyplot as plt

# reading the database

data = pd.read\_csv("tips.csv")

# Bar chart with day against tip

plt.bar(data['day'], data['tip'])

plt.title("Bar Chart")

# Setting the X and Y labels

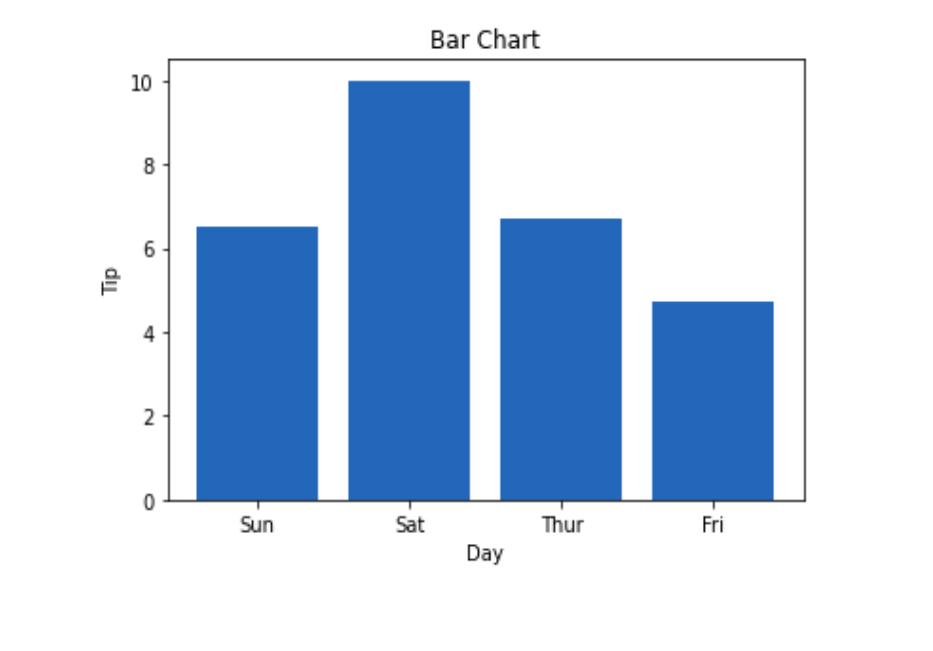
plt.xlabel('Day')

plt.ylabel('Tip')

# Adding the legends

plt.show()

**Output**:



**Example 3**: Program for data visualization using Matplotlib for Histogram

**Code:**

import pandas as pd

import matplotlib.pyplot as plt

# reading the database

data = pd.read\_csv("tips.csv")

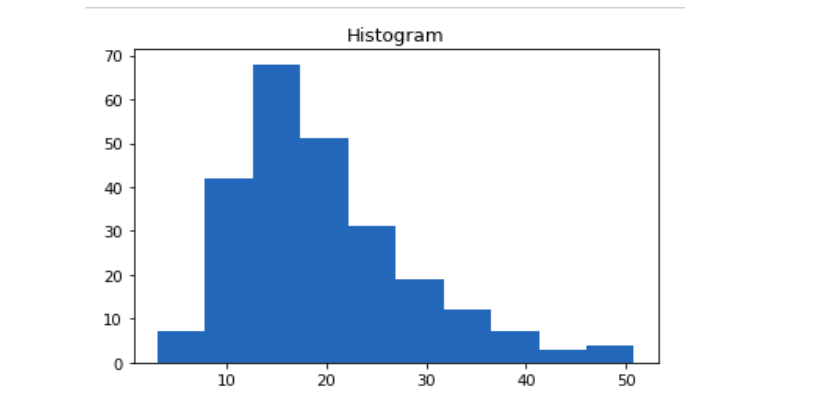
# histogram of total\_bills

plt.hist(data['total\_bill'])

plt.title("Histogram")

# Adding the legends

plt.show()

**Output**: 

**Example 4**: Program for data visualization using Matplotlib for Line plot

**Code:**

import seaborn as sns

import matplotlib.pyplot as plt

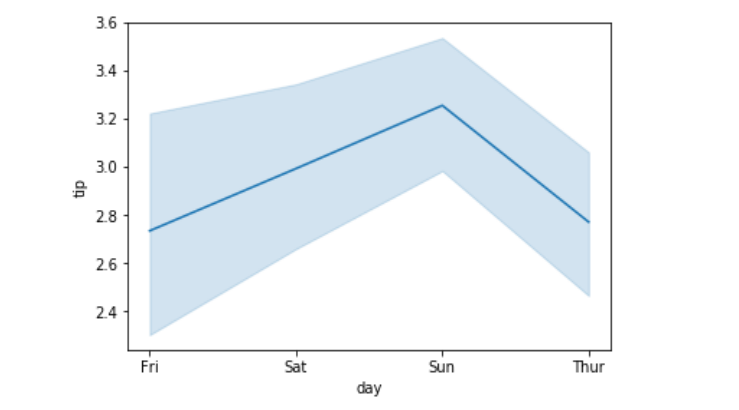
import pandas as pd

# reading the database

data = pd.read\_csv("tips.csv")

sns.lineplot(x='day', y='tip', data=data)

plt.show()

**Output**: 

**Example 5**: Program for data visualization using Matplotlib for Box plot

**Code:**

# importing packages

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

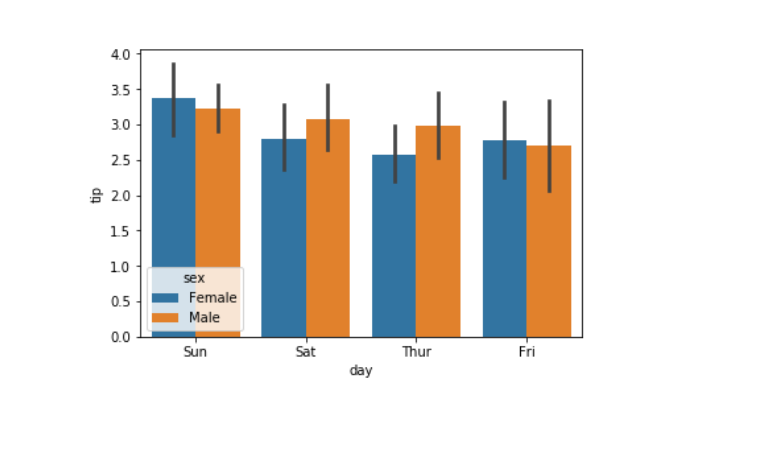
# reading the database

data = pd.read\_csv("tips.csv")

sns.barplot(x='day',y='tip', data=data, hue='sex')

plt.show()

**Output**:



**Experiment 10**

**Aim:** Program for sending current time string to the client using Socket Programming.

**Code:**

# server.py

import socket

import time

# create a socket object

serversocket = socket.socket(

socket.AF\_INET, socket.SOCK\_STREAM)

# get local machine name

host = socket.gethostname()

port = 9999

# bind to the port

serversocket.bind((host, port))

# queue up to 5 requests

serversocket.listen(5)

while True:

# establish a connection

clientsocket,addr = serversocket.accept()

print("Got a connection from %s" % str(addr))

currentTime = time.ctime(time.time()) + "\r\n"

clientsocket.send(currentTime.encode('ascii'))

clientsocket.close()

# client.py

import socket

# create a socket object

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# get local machine name

host = socket.gethostname()

port = 9999

# connection to hostname on the port.

s.connect((host, port))

# Receive no more than 1024 bytes

tm = s.recv(1024)

s.close()

print("The time got from the server is %s" % tm.decode('ascii'))

**Output:**

$ python server.py &

Got a connection from ('127.0.0.1', 54597)

$ python client.py

The time got from the server is Wed Jan 29 19:14:15 201