INDEX

PRACTICAL -1

AIM : Develop program to understand the control structures of python.

CODE :

```
print("----Practical 1----")
lucky_numbers = [1,3,4,8,10,15,25,46,28,86,54,75,98,23,42]
number_test = int(input("Enter 2 Digit Number : "))
flag = False
for luck in lucky_numbers:
   if luck == number_test:
      flag = True
      break
if flag:
   print("You are Lucky")
else:
   print("Better Luck Next Time")
print("-----END-----")
```

OUTPUT :

```
s/[redacted]/Desktop/server.pyLocal/Programs/Python/Python37-32/python.exe c:/Users
----Practical 1----
Enter 2 Digit Number : 10
You are Lucky
-----END-----

[redacted]@DESKTOP-3R6MGMA MINGW64 ~/Desktop
s/[redacted]/Desktop/server.pyLocal/Programs/Python/Python37-32/python.exe c:/Users
----Practical 1----
Enter 2 Digit Number : 11
Better Luck Next Time
-----END-----
```

PRACTICAL -2

AIM : Develop programs to learn different types of structures (list, dictionary, tuples) in python.

CODE :

```
tup = ("Samsung", "Apple", "Blackberry", "Google")
lis = [50000, 100000, 35000, 40000]
print("----Practical 2----")
print("The Tuples are", tup)
print("The List is", lis)
print("Lets Make a dictionary with Company and Price")
dic = {}
for (company,price) in zip(tup,lis):
    dic[company] = price
print("The dictonary is", dic)
print("-----END-----")
```

OUTPUT :

```
----Practical 2----
The Tuples are ('Samsung', 'Apple', 'Blackberry', 'Google')
The List is [50000, 100000, 35000, 40000]
Lets Make a dictionary with Company and Price
The dictonary is {'Samsung': 50000, 'Apple': 100000, 'Blackberry': 35000, 'Google': 40000}
-----END-----
```

PRACTICAL -3

AIM : Develop programs to learn concept of functions scoping, recursion and list mutability.

CODE :

```
import argparse

ap = argparse.ArgumentParser()
ap.add_argument("--input",type=int, help="The input Number", default=5)
args = vars(ap.parse_args())

print("----Pra 3----")
print("----Recursion----")
def factorial(n):
   if n == 1:
      return 1
   else:
      return n*(factorial(n-1))
print("Factorial of ",args['input'] , ":",factorial(args['input']))

print("----Function Scoping----")
a = 10
if a == 10:
   print("Num1 :",a)
   b = a + 2
print("Num2 :",b)
print("B is in the Scope")
def print_fun():
   c = a + b
   print("Num3 :",c)

try:
   print("Num3 :",c)
except:
   print("C is out of Scope")
print_fun()

print("----List Mutablity----")
lis = ["Samsung", "Apple", "Blackberry", "Google"]
print("Inital List :", lis)
lis.append("Nokia")
print("After Appending :",lis)
lis.pop()
lis.pop()
print("After Popping :",lis)
print("-----END-----")
```

OUTPUT :

```
----Pra 3----
----Recursion----
Factorial of  5 : 120
----Function Scoping----
Num1 : 10
Num2 : 12
B is in the Scope
C is out of Scope
Num3 : 22
----List Mutablity----
Inital List : ['Samsung', 'Apple', 'Blackberry', 'Google']
After Appending : ['Samsung', 'Apple', 'Blackberry', 'Google', 'Nokia']
After Popping : ['Samsung', 'Apple', 'Blackberry']
-----END-----
```

PRACTICAL -4

AIM : Develop programs to understand working of exception handling and assertions.

CODE :

```
import argparse

ap = argparse.ArgumentParser()
ap.add_argument("--number", type=int, default=0, help="A number to Divide by 20")
args = vars(ap.parse_args())

print("----Practical 4----")
a = 20
num = args['number']

try:
   assert num != 0, "Hello Cannot divide by Zero"
   print(a,'/',num,'=',a//num)
except AssertionError as error:
   print(error)
print('-----END-----')
```

OUTPUT :

```
$ C:/Users/[redacted]/AppData/Local/Programs/Python/Python37-32/python.exe c:/Users/[redacted]/Desktop/prac.py --number 25
----Practical 4----
20 / 25 = 0
-----END-----

[redacted]@DESKTOP-3R6MGMA MINGW64 ~/Desktop
$ C:/Users/[redacted]/AppData/Local/Programs/Python/Python37-32/python.exe c:/Users/[redacted]/Desktop/prac.py --number 0
----Practical 4----
sorry! Cannot divide by Zero
-----END-----
```

PRACTICAL -5

AIM : Develop programs for data structure algorithms using python – searching, sorting and hash tables

CODE :

```
import random
print("----Practical 5----")
lucky_numbers = [1,3,4,8,10,15,25,46,28,86,54,75,98,23,42]
new = sorted(lucky_numbers)
print('Unsorted Number :', lucky_numbers)
print('Sorted Number :',new)
num = int(input('Enter a Number to Search in Sorted List : '))
flag = True
index = 0
for temp in new:
    if temp == num:
        print('Number Exist at Index ', index)
        flag = False
    index = index + 1
if flag:
    print('Number does not Exist in the List')
hash_table = {}
while True:
    new_data = input("Enter a data to generate Hash(Type break to exit and print hash table)
: ")
    if 'break' in new_data.lower():
        break
    new_key = len(new_data)
    new_key = ( random.randrange(1000) + new_key ) + new_key * new_key
    hash_table[str(new_key)] = new_data
print(hash_table)
print("-----END-----")
```

OUTPUT :

```
----Practical 5----
Unsorted Number : [1, 3, 4, 8, 10, 15, 25, 46, 28, 86, 54, 75, 98, 23, 42]
Sorted Number : [1, 3, 4, 8, 10, 15, 23, 25, 28, 42, 46, 54, 75, 86, 98]
Enter a Number to Search in Sorted List : 86
Number Exist at Index  13
Enter a data to generate Hash(Type break to exit and print hash table) : chandler
Enter a data to generate Hash(Type break to exit and print hash table) : bing
Enter a data to generate Hash(Type break to exit and print hash table) : 170130116055
Enter a data to generate Hash(Type break to exit and print hash table) : IT 4th
Enter a data to generate Hash(Type break to exit and print hash table) : GECG
Enter a data to generate Hash(Type break to exit and print hash table) : break
{'931': 'chandler', '125': 'bing', '791': '170130116055', '676': 'IT 4th', '628': 'GECG'}
-----END-----
```

PRACTICAL -6

AIM : Develop programs to learn regular expressions using python.

CODE :

```
import re
print("----Practical 6----")
NAME_REGEX = re.compile(r"[a-zA-Z]+")
EMAIL_REGEX = re.compile(r"[^@]+@[^@]+\.[^@]+")
PHONE_REGEX = re.compile(r"[6-9]{1}[0-9]{9}")
while True:
    name = input('Enter Name : ')
    if not NAME_REGEX.match(name):
        print("Please Enter Valid Name")
    else:
        break
while True:
    email = input('Enter Email : ')
    if not EMAIL_REGEX.match(email):
        print("Please enter Valid Email address %s" % (name))
    else:
        break
while True:
    phone = input("Enter Mobile Number : ")
    if not PHONE_REGEX.match(phone):
        print("Please Enter valid Phone Number %s" % (name))
    else:
        break
print("\nHere are Your Details you Entered\n")
print("Name : %s\nEmail : %s\nPhone Number : %s" % (name, email, phone))
print("-----END-----")
```

OUTPUT :

```
----Practical 6----
Enter Name : Rachel Ross Geller
Enter Email : rachel 0101
Please enter Valid Email address Rachel Ross Geller
Enter Email : wewereonabreak@gmail.com
Enter Mobile Number : fuyu
Please Enter valid Phone Number Rachel Ross Geller
Enter Mobile Number : 9988776655

Here are Your Details you Entered

Name : Rachel Ross Geller
Email : wewereonabreak@gmail.com
Phone Number : 9988776655
-----END-----
```

PRACTICAL -7

AIM : Develop chat room application using multithreading.

SERVER CODE :

```
import socket
import threading
import time

def accept_client():
    while True:
        conn, addr = server_socket.accept()
        CONNECTION_LIST.append(conn)
        client_thread = threading.Thread(target=broadcast_user, args=(conn, ))
        client_thread.start()

def broadcast_user(cli_socket):
    while True:
        try:
            data = cli_socket.recv(1024).decode()
            if data:
                broadcast_message_client(cli_socket, data)
        except Exception as x:
            print(x.message)
            break

def broadcast_message_client(cli_socket, message):
    for client in CONNECTION_LIST:
        if client != cli_socket:
            client.send(message.encode())

if __name__ == '__main__':
    CONNECTION_LIST = []
    host = socket.gethostname()
    port = 5001
    server_socket = socket.socket()
    server_socket.bind((host, port))
    server_socket.listen(5)
    print('Chat Room Started on port %d' % (port))
    server_thread = threading.Thread(target=accept_client, args=())
    server_thread.start()
```

CLIENT CODE :

```
import socket
import threading
```

```
def send_message(uname):
    while True:
        message = input('\n')
        data = uname + ' : ' + message
        client_socket.send(data.encode())

def recieve_message():
    while True:
        data = client_socket.recv(1024).decode()
        print('\n\t\t' + str(data))

if __name__ == '__main__':
    host = socket.gethostname()
    port = 5001
    client_socket = socket.socket()
    uname = input('Enter Your Name in Chat : ')

    client_socket.connect((host,port))
    print('Connected to remote host')

    send_thread = threading.Thread(target=send_message, args=(uname, ))
    send_thread.start()

    recieve_thread = threading.Thread(target=recieve_message, args=())
    recieve_thread.start()
```

OUTPUT :

```
$  env C:\\Users\\[redacted]\\AppData\\Local\\Programs\\Python\\Python37-32\\python.exe c:\\Users\\[redacted]\\.vscode\\extensions\\ms-python.py
thon-2020.6.91350\\pythonFiles\\lib\\python\\debugpy\\launcher 3047 -- c:\\Users\\[redacted]\\Desktop\\server.py
Chat Room Started on port 5001
```

```
Enter Your Name in Chat : Mike
Connected to remote host

hello


                watson : hi
how are you?


                watson : i am fine, where are u from?
ahmedabad and u?


                watson : surat
ohh nice. okay bye


                watson : bbye!
```

```
Enter Your Name in Chat : watson
Connected to remote host

hi


                Mike : how are you?
i am fine, where are u from?


                Mike : ahmedabad and u?
surat


                Mike : ohh nice. okay bye
bbye!
```

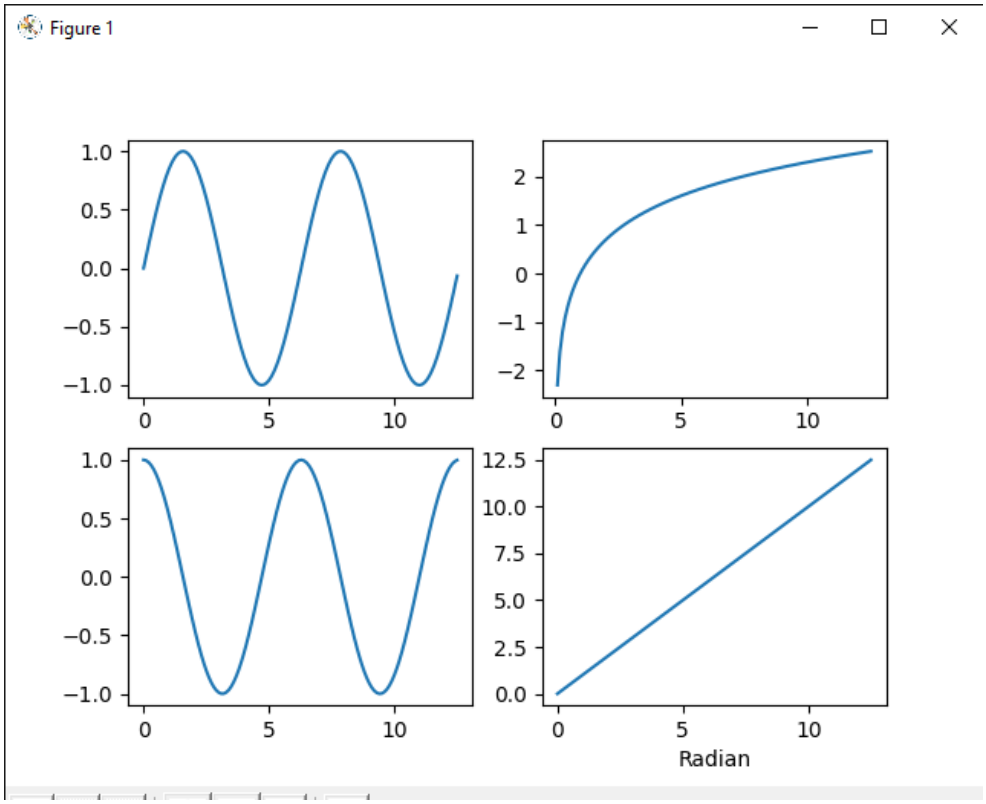
PRACTICAL -8

AIM : Learn to plot different types of graphs using PyPlot.

CODE :

```
import numpy as np
import matplotlib.pyplot as plt
fig, ax = plt.subplots(2, 2)
x = np.arange(0, 4*np.pi, 0.1)
ax[0, 0].plot(x, np.sin(x))
ax[1, 0].plot(x, np.cos(x))
ax[0, 1].plot(x, np.log(x))
ax[1, 1].plot(x, x)
plt.xlabel('Radian')
plt.show()
```

OUTPUT :



PRACTICAL -9

AIM : Implement classical ciphers using python.

CODE :

```
import argparse
ap = argparse.ArgumentParser()
ap.add_argument('--encrypt', default=None, help='Text to Encrypt')
ap.add_argument('--decrypt', default=None, help='Text to Decrypt')
ap.add_argument('--key', type=int, required=True, help='Key to Encrypt/Decrypt')
args = vars(ap.parse_args())
print("----Pra 9----")
print("Caser Cipher")
if args['encrypt'] is not None:
    pt = args['encrypt'].lower()
    ct = ""
    key = args['key']
    for pt1 in pt:
        asc = ord(pt1)
        if asc < 123 and asc > 96:
            ct = ct + chr(((asc - 96 + key) % 26) + 64)
        elif ord(pt1) == 32:
            ct = ct + pt1
    print("Ciper Text : " + ct)
if args['decrypt'] is not None:
    ct = args['decrypt'].upper()
    pt = ""
    key = args['key']
    for ct1 in ct:
        asc = ord(ct1)
        if asc < 91 and asc > 64:
            pt = pt + chr(((asc - 64 - key) % 26) + 96)
        elif ord(ct1) == 32:
            pt = pt + ct1
    print("Plain Text : "+ pt)
if args['encrypt'] is None and args['decrypt'] is None:
    print("Please give Plain Text or Ciper Text")
print("-----END-----")
```

OUTPUT :

```
$ python c:/Users/[redacted]/Desktop/server.py --encrypt Meet --key 4
----Pra 9----
Caser Cipher
Ciper Text : QIIX
-----END-----

[redacted]@DESKTOP-3R6MGMA MINGW64 ~/Desktop
$ python c:/Users/[redacted]/Desktop/server.py --decrypt Jay --key 2
----Pra 9----
Caser Cipher
Plain Text : hyw
-----END-----

[redacted]@DESKTOP-3R6MGMA MINGW64 ~/Desktop
$ python c:/Users/[redacted]/Desktop/server.py --decrypt Mitul --key 4
----Pra 9----
Caser Cipher
Plain Text : iepqh
-----END-----
```

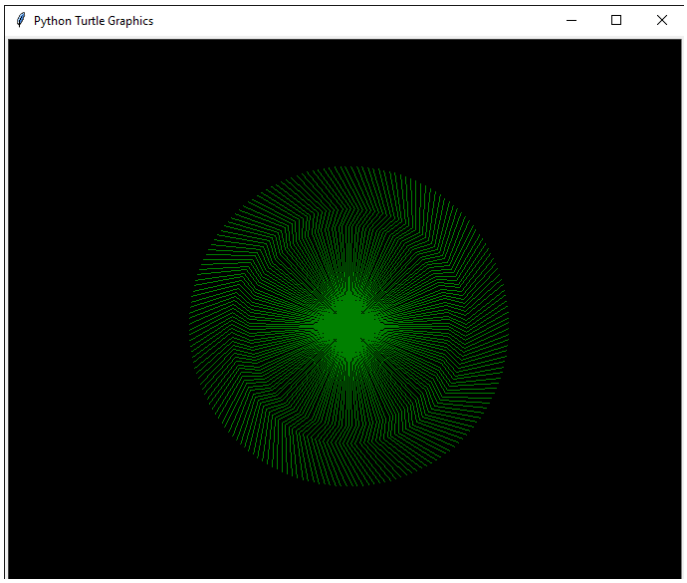
PRACTICAL -10

AIM : Draw graphics using Turtle.

CODE :

```
import turtle

win = turtle.Screen()
win.bgcolor('black')

Watson = turtle.Turtle()
Watson.color('Red')
Watson.pensize(1)
Watson.speed(1000)
for i in range(180):
    Watson.forward(100)
    Watson.right(30)
    Watson.forward(20)
    Watson.left(60)
    Watson.forward(50)
    Watson.right(30)

    Watson.penup()
    Watson.setposition(0, 0)
    Watson.pendown()

    Watson.right(2)
win.exitonclick()
```

OUTPUT :



PRACTICAL -11

AIM : Develop programs to learn GUI programming using Tkinter.

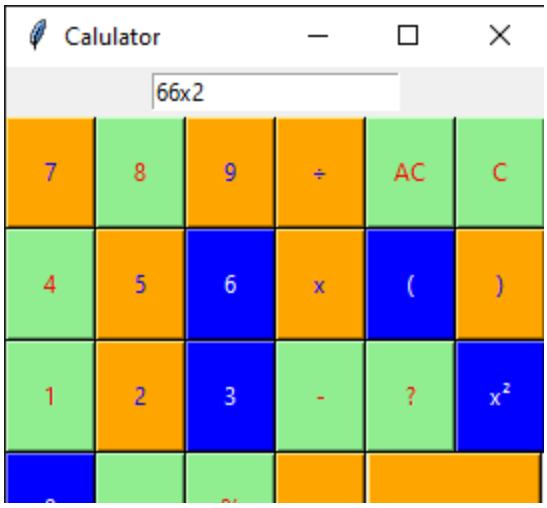
CODE :

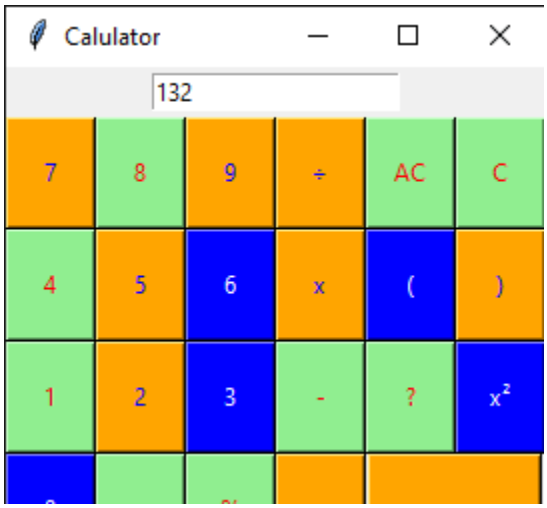
```
from tkinter import *
import math

class calc:

	def getandreplace(self):

		"""replace x with * and ÷ with /"""
		self.expression = self.e.get()
		self.newtext=self.expression.replace('/','/')
		self.newtext=self.newtext.replace('x','*')


	def equals(self):
		"""when the equal button is pressed"""
		self.getandreplace()
		try:
			# evaluate the expression using the eval function
			self.value= eval(self.newtext)
		except SyntaxError or NameError:
			self.e.delete(0,END)
			self.e.insert(0,'Invalid Input!')
		else:
			self.e.delete(0,END)
			self.e.insert(0,self.value)

	def squareroot(self):
		"""squareroot method"""
		self.getandreplace()
		try:
			# evaluate the expression using the eval function
			self.value= eval(self.newtext)
		except SyntaxError or NameError:
			self.e.delete(0,END)
			self.e.insert(0,'Invalid Input!')
		else:
			self.sqrtval=math.sqrt(self.value)
			self.e.delete(0,END)
			self.e.insert(0,self.sqrtval)

	def square(self):
```

```
            """square method"""
            self.getandreplace()
            try:
                #evaluate the expression using the eval function
                self.value= eval(self.newtext)
            except SyntaxError or NameError:
                self.e.delete(0,END)
                self.e.insert(0,'Invalid Input!')
            else:
                self.sqval=math.pow(self.value,2)
                self.e.delete(0,END)
                self.e.insert(0,self.sqval)

        def clearall(self):
            """when clear button is pressed,clears the text input area"""
            self.e.delete(0,END)

        def clear1(self):
            self.txt=self.e.get()[:-1]
            self.e.delete(0,END)
            self.e.insert(0,self.txt)

        def action(self,argi):
            """pressed button's value is inserted into the end of the text area"""
            self.e.insert(END,argi)

        def __init__(self,master):
            """Constructor method"""
            master.title('Calulator')
            master.geometry()
            self.e = Entry(master)
            self.e.grid(row=0,column=0,columnspan=6,pady=3)
            self.e.focus_set() #Sets focus on the input text area

            # Generating Buttons
            Button(master,text="=",width=11,height=3,fg="blue",bg="orange",
command=lambda:self.equals()).grid(row=4, column=4,columnspan=2)
            Button(master,text='AC',width=5,height=3, fg="red", bg="light green",
command=lambda:self.clearall()).grid(row=1, column=4)
            Button(master,text='C',width=5,height=3, fg="red",bg="light green",
command=lambda:self.clear1()).grid(row=1, column=5)
            Button(master,text="+",width=5,height=3, fg="blue",bg="orange",
command=lambda:self.action('+')).grid(row=4, column=3)
            Button(master,text="x",width=5,height=3, fg="blue",bg="orange",
command=lambda:self.action('x')).grid(row=2, column=3)
            Button(master,text="-",width=5,height=3, fg="red",bg="light green",
command=lambda:self.action('-')).grid(row=3, column=3)
```

```
        Button(master,text="÷",width=5,height=3, fg="blue",bg="orange",
command=lambda:self.action('/')).grid(row=1, column=3)
        Button(master,text="%",width=5,height=3, fg="red",bg="light green",
command=lambda:self.action('%')).grid(row=4, column=2)
        Button(master,text="7",width=5,height=3, fg="blue",bg="orange",
command=lambda:self.action('7')).grid(row=1, column=0)
        Button(master,text="8",width=5,height=3, fg="red",bg="light green",
command=lambda:self.action(8)).grid(row=1, column=1)
        Button(master,text="9",width=5,height=3, fg="blue",bg="orange",
command=lambda:self.action(9)).grid(row=1, column=2)
        Button(master,text="4",width=5,height=3, fg="red",bg="light green",
command=lambda:self.action(4)).grid(row=2, column=0)
        Button(master,text="5",width=5,height=3, fg="blue",bg="orange",
command=lambda:self.action(5)).grid(row=2, column=1)
        Button(master,text="6",width=5,height=3, fg="white",bg="blue",
command=lambda:self.action(6)).grid(row=2, column=2)
        Button(master,text="1",width=5,height=3, fg="red",bg="light green",
command=lambda:self.action(1)).grid(row=3, column=0)
        Button(master,text="2",width=5,height=3, fg="blue",bg="orange",
command=lambda:self.action(2)).grid(row=3, column=1)
        Button(master,text="3",width=5,height=3, fg="white",bg="blue",
command=lambda:self.action(3)).grid(row=3, column=2)
        Button(master,text="0",width=5,height=3, fg="white",bg="blue",
command=lambda:self.action(0)).grid(row=4, column=0)
        Button(master,text=".",width=5,height=3, fg="red",bg="light green",
command=lambda:self.action('.')).grid(row=4, column=1)
        Button(master,text="(",width=5,height=3, fg="white",bg="blue",
command=lambda:self.action('(')).grid(row=2, column=4)
        Button(master,text=")",width=5,height=3, fg="blue",bg="orange",
command=lambda:self.action(')')).grid(row=2, column=5)
        Button(master,text="?",width=5,height=3, fg="red",bg="light green",
command=lambda:self.squareroot()).grid(row=3, column=4)
        Button(master,text="x²",width=5,height=3, fg="white",bg="blue",
command=lambda:self.square()).grid(row=3, column=5)


root = Tk()
obj=calc(root) # object instantiated
root.mainloop()
```

OUTPUT :


Calulator
66x2
7
8
9
÷
AC
C
4
5
6
x
(
)
1
2
3
-
?
x²


Calulator
132
7
8
9
÷
AC
C
4
5
6
x
(
)
1
2
3
-
?
x²