**Ex.No:13**

**Date:**

**IMPLEMENTATION OF SYMBOLIC PROGRAMMING PARADIGM**

**AIM:**

To implement symbolic programming paradigm in python.

13a. Write the commands to perform the operations on substitutions and expressions

**ALGORITHM:**

1. Import sympy module
2. Evaluate the expression using sympy command
3. Print the result

**Program:**

# import sympy

from sympy import \*

x, y, z = symbols('x y z')

exp = x\*\*2 + 7 \* y + z

print("Before Substitution : {}".format(exp))

# Use sympy.subs() method

res\_exp = exp.subs([(x, 2), (y, 4), (z, 1)])

print("After Substitution : {}".format(res\_exp))

x = symbols('x')

exp = cos(x) + 7

print("Before Substitution : {}".format(exp))

# Use sympy.subs() method

res\_exp = exp.subs(x, 0)

print("After Substitution : {}".format(res\_exp))

x, y = symbols('x y')

exp = x\*\*2 + 1

print("Before Substitution : {}".format(exp))

# Use sympy.subs() method

res\_exp = exp.subs(x, y)

print("After Substitution : {}".format(res\_exp))

**Output:**

Before Substitution : x\*\*2 + 7\*y + z

After Substitution : 33

Before Substitution : cos(x) + 7

After Substitution : 8

Before Substitution : x\*\*2 + 1

After Substitution : y\*\*2 + 1

13b. To perform the following operations on matrices

**ALGORITHM:**

1. Import matrix from sympy.matrices.
2. Create the matrix
3. Print the matrix
4. Display the matrix
5. Display 0th row
6. Print first column
7. Delete the first column from the matrix
8. Insert the row into the matrix
9. Generate two matrices
10. Print addition of two matrices
11. Print the multiplication of two matrices

**Program:**

from sympy.matrices import Matrix

m=Matrix([[1,2,3],[2,3,1]])

print(m)

M=Matrix(2,3,[10,40,30,2,6,9])

print(M)

print(M.shape)

print(M.row(0))

M.col(1)

M.row(1)[1:3]

print(M)

M=Matrix(2,3,[10,40,30,2,6,9])

M.col\_del(1)

a=Matrix([[1,2,3],[2,3,1]])

print(a)

a1=Matrix([[10,30]])

a=M.row\_insert(0,M)

print(a)

a2=Matrix([40,6])

a=M.col\_insert(1,M)

print(a)

M1=Matrix([[1,2,3],[3,2,1]])

M2=Matrix([[4,5,6],[6,5,4]])

print(M1+M2)

M1=Matrix([[1,2,3],[3,2,1]])

M2=Matrix([[4,5],[6,6],[5,4]])

print(M1\*M2)

**Output:**

Matrix([[1, 2, 3], [2, 3, 1]])

Matrix([[10, 40, 30], [2, 6, 9]])

(2, 3)

Matrix([[10, 40, 30]])

Matrix([[10, 40, 30], [2, 6, 9]])

Matrix([[1, 2, 3], [2, 3, 1]])

Matrix([[10, 30], [2, 9], [10, 30], [2, 9]])

Matrix([[10, 10, 30, 30], [2, 2, 9, 9]])

Matrix([[5, 7, 9], [9, 7, 5]])

Matrix([[31, 29], [29, 31]])

13c. Write the commands to find derivative, integration, limits, quadratic equation

**ALGORITHM:**

1. Import sympy module
2. Make a symbol
3. Find the derivative of the expression
4. Print the result
5. Find the integration of the expression
6. Print the result
7. Find the limit of the expression
8. Print the result
9. Find the quadratic equation of the expression
10. Print the result

**Program:**

from sympy import \*

x = Symbol('x')

 #make the derivative of cos(x)\*e ^ x

ans1 = diff(cos(x)\*exp(x), x)

print("The derivative of the  sin(x)\*e ^ x : ", ans1)

# Compute (e ^ x \* sin(x)+ e ^ x \* cos(x))dx

ans2 = integrate(exp(x)\*sin(x) + exp(x)\*cos(x), x)

print("The result of  integration is : ", ans2)

# Compute definite integral of sin(x ^ 2)dx

# in b / w interval of ? and ?? .

ans3 = integrate(sin(x\*\*2), (x, -oo, oo))

print("The value of integration is : ", ans3)

# Find the limit of sin(x) / x given x tends to 0

ans4 = limit(sin(x)/x, x, 0)

print("limit is : ", ans4)

# Solve quadratic equation like, example : x ^ 2?2 = 0

ans5 = solve(x\*\*2 - 2, x)

print("roots are : ", ans5)

**Output:**

The derivative of the sin(x)\*e ^ x : -exp(x)\*sin(x) + exp(x)\*cos(x)

The result of integration is : exp(x)\*sin(x)

The value of integration is : sqrt(2)\*sqrt(pi)/2

* limit is : 1

roots are : [-sqrt(2), sqrt(2)]

RESULT:

Thus the Python program to implement symbolic program have been written and executed successfully.

**Ex.No:14**

**Date:**

**IMPLEMENTATION OF AUTOMATA PROGRAMMING PARADIGM**

**AIM:**

To implement symbolic programming paradigm in python.

14a. To write a program to convert DFA to NFA using python.

**ALGORITHM:**

1. Initialize the transitions
2. Copy the input in list
3. Parse the string of a,b in 0,1 for simplicity
4. Counter to remember the number of symbols read
5. Set the final states
6. Check for each possibility
7. Move further only if you are at non-hypothetical state
8. Read the last symbol and current state lies in the set of final states
9. Input string for next transition is input[i+1:]
10. Increment the counter
11. Print the state

**Program:**

import sys

def main():

transition = [[[0,1],[0]], [[4],[2]], [[4],[3]], [[4],[4]]]

x = input("enter the string: ")

x = list(x)

for index in range(len(x)):

if x[index]=='a':

x[index]='0'

else:

x[index]='1'

final = "3"

start=0

i=0

trans(transition, x, final, start, i)

print("rejected")

def trans(transition, x, final, state, i):

for j in range (len(x)):

for each in transition[state][int(x[j])]:

if each < 4:

state = each

if j == len(x)-1 and (str(state) in final):

print("accepted")

sys.exit()

trans(transition, x[i+1:], final, state, i)

i = i+1

main()

**Output:**

enter the string: abb

accepted

enter the string: aaaabbbb

rejected

14b.Write a program to convert NFA to DFA

**ALGORITHM:**

1. Take NFA input from User
2. Enter total no. of states
3. Enter total no. of transitions/paths eg: a,b so input 2 for a,b,c input 3
4. Enter state name eg: A, B, C, q1, q2 ..etc
5. Creating a nested dictionary
6. Enter path eg : a or b in {a,b} 0 or 1 in {0,1}
7. Enter all the end states that
8. Assign the end states to the paths in dictionary\
9. Print NFA
10. Enter final state/states of NFA
11. Holds all the new states created in dfa
12. List of all the paths
13. Conatins all the states in nfa plus the states created in
14. Compute first row of DFA transition table
15. Create a nested dictionary in dfa
16. Create a single string from all the elements of the list
17. Append it to the new\_states\_list
18. Compute the other rows of DFA transition table
19. Create a temporay list
20. Assign the state in DFA table
21. Create a single string(new state) from all the elements of the list
22. Assign the new state in the DFA table
23. Remove the first element in the new\_states\_list
24. Print the DFA created
25. Print Final states of DFA

**Program:**

import pandas as pd

def main():

nfa = {}

n = int(input("No. of states : "))

t = int(input("No. of transitions : "))

for i in range(n):

state = input("state name : ")

nfa[state] = {}

for j in range(t):

path = input("path : ")

print("Enter end state from state {} travelling through path {} : ".format(state,path))

reaching\_state = [x for x in input().split()]

nfa[state][path] = reaching\_state

print("\nNFA :- \n")

print(nfa)

print("\nPrinting NFA table :- ")

nfa\_table = pd.DataFrame(nfa)

print(nfa\_table.transpose())

print("Enter final state of NFA : ")

nfa\_final\_state = [x for x in input().split()]

new\_states\_list = []

dfa = {}

keys\_list = list(list(nfa.keys())[0])

path\_list = list(nfa[keys\_list[0]].keys())

dfa[keys\_list[0]] = {}

for y in range(t):

var = "".join(nfa[keys\_list[0]][path\_list[y]])

dfa[keys\_list[0]][path\_list[y]] = var

if var not in keys\_list:

new\_states\_list.append(var)

keys\_list.append(var)

while len(new\_states\_list) != 0:

dfa[new\_states\_list[0]] = {}

for \_ in range(len(new\_states\_list[0])):

for i in range(len(path\_list)):

temp = []

for j in range(len(new\_states\_list[0])):

temp += nfa[new\_states\_list[0][j]][path\_list[i]]

s = ""

s = s.join(temp)

if s not in keys\_list:

new\_states\_list.append(s)

keys\_list.append(s)

dfa[new\_states\_list[0]][path\_list[i]] = s

new\_states\_list.remove(new\_states\_list[0])

print("\nDFA :- \n")

print(dfa)

print("\nPrinting DFA table :- ")

dfa\_table = pd.DataFrame(dfa)

print(dfa\_table.transpose())

dfa\_states\_list = list(dfa.keys())

dfa\_final\_states = []

for x in dfa\_states\_list:

for i in x:

if i in nfa\_final\_state:

dfa\_final\_states.append(x)

break

print("\nFinal states of the DFA are : ",dfa\_final\_states)

main()

**Output:**

No. of states : 4

No. of transitions : 2

state name : A

path : a

Enter end state from state A travelling through path a :

A B

path : b

Enter end state from state A travelling through path b :

A

state name : B

path : a

Enter end state from state B travelling through path a :

C

path : b

Enter end state from state B travelling through path b :

C

state name : C

path : a

Enter end state from state C travelling through path a :

D

path : b

Enter end state from state C travelling through path b :

D

state name : D

path : a

Enter end state from state D travelling through path a :

A

path : b

Enter end state from state D travelling through path b :

A

NFA :-

{'A': {'a': ['A', 'B'], 'b': ['A']}, 'B': {'a': ['C'], 'b': ['C']}, 'C': {'a': ['D'], 'b': ['D']}, 'D': {'a': ['A'], 'b': ['A']}}

Printing NFA table :-

a b

A [A, B] [A]

B [C] [C]

C [D] [D]

D [A] [A]

Enter final state of NFA :

D

**RESULT:**

Thus the Python program to implement the conversion of  DFA to NFA and NFA to DFA have been written and executed successfully.

**Ex.No:15**

**Date:**

**IMPLEMENTATION OF GUI PROGRAMMING PARADIGM**

**AIM:**

To implement GUI  programming paradigm in python.

15a. Design a calculator to perform all mathematical operations using python

**Algorithm:**

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

**Program:**

from tkinter import \*

expression = ""

def press(num):

global expression

expression = expression + str(num)

equation.set(expression)

def equalpress():

try:

global expression

total = str(eval(expression))

equation.set(total)

expression = ""

except:

equation.set(" error ")

expression = ""

def clear():

global expression

expression = ""

equation.set("")

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

gui = Tk()

gui.configure(background="light green")

gui.title("Simple Calculator")

gui.geometry("270x150")

equation = StringVar()

expression\_field = Entry(gui, textvariable=equation)

expression\_field.grid(columnspan=4, ipadx=70)

button1 = Button(gui, text=' 1 ', fg='black', bg='red',

command=lambda: press(1), height=1, width=7)

button1.grid(row=2, column=0)

button2 = Button(gui, text=' 2 ', fg='black', bg='red',

command=lambda: press(2), height=1, width=7)

button2.grid(row=2, column=1)

button3 = Button(gui, text=' 3 ', fg='black', bg='red',

command=lambda: press(3), height=1, width=7)

button3.grid(row=2, column=2)

button4 = Button(gui, text=' 4 ', fg='black', bg='red',

command=lambda: press(4), height=1, width=7)

button4.grid(row=3, column=0)

button5 = Button(gui, text=' 5 ', fg='black', bg='red',

command=lambda: press(5), height=1, width=7)

button5.grid(row=3, column=1)

button6 = Button(gui, text=' 6 ', fg='black', bg='red',

command=lambda: press(6), height=1, width=7)

button6.grid(row=3, column=2)

button7 = Button(gui, text=' 7 ', fg='black', bg='red',

command=lambda: press(7), height=1, width=7)

button7.grid(row=4, column=0)

button8 = Button(gui, text=' 8 ', fg='black', bg='red',

command=lambda: press(8), height=1, width=7)

button8.grid(row=4, column=1)

button9 = Button(gui, text=' 9 ', fg='black', bg='red',

command=lambda: press(9), height=1, width=7)

button9.grid(row=4, column=2)

button0 = Button(gui, text=' 0 ', fg='black', bg='red',

command=lambda: press(0), height=1, width=7)

button0.grid(row=5, column=0)

plus = Button(gui, text=' + ', fg='black', bg='red',

command=lambda: press("+"), height=1, width=7)

plus.grid(row=2, column=3)

minus = Button(gui, text=' - ', fg='black', bg='red',

command=lambda: press("-"), height=1, width=7)

minus.grid(row=3, column=3)

multiply = Button(gui, text=' \* ', fg='black', bg='red',

command=lambda: press("\*"), height=1, width=7)

multiply.grid(row=4, column=3)

divide = Button(gui, text=' / ', fg='black', bg='red',

command=lambda: press("/"), height=1, width=7)

divide.grid(row=5, column=3)

equal = Button(gui, text=' = ', fg='black', bg='red',

command=equalpress, height=1, width=7)

equal.grid(row=5, column=2)

clear = Button(gui, text='Clear', fg='black', bg='red',

command=clear, height=1, width=7)

clear.grid(row=5, column='1')

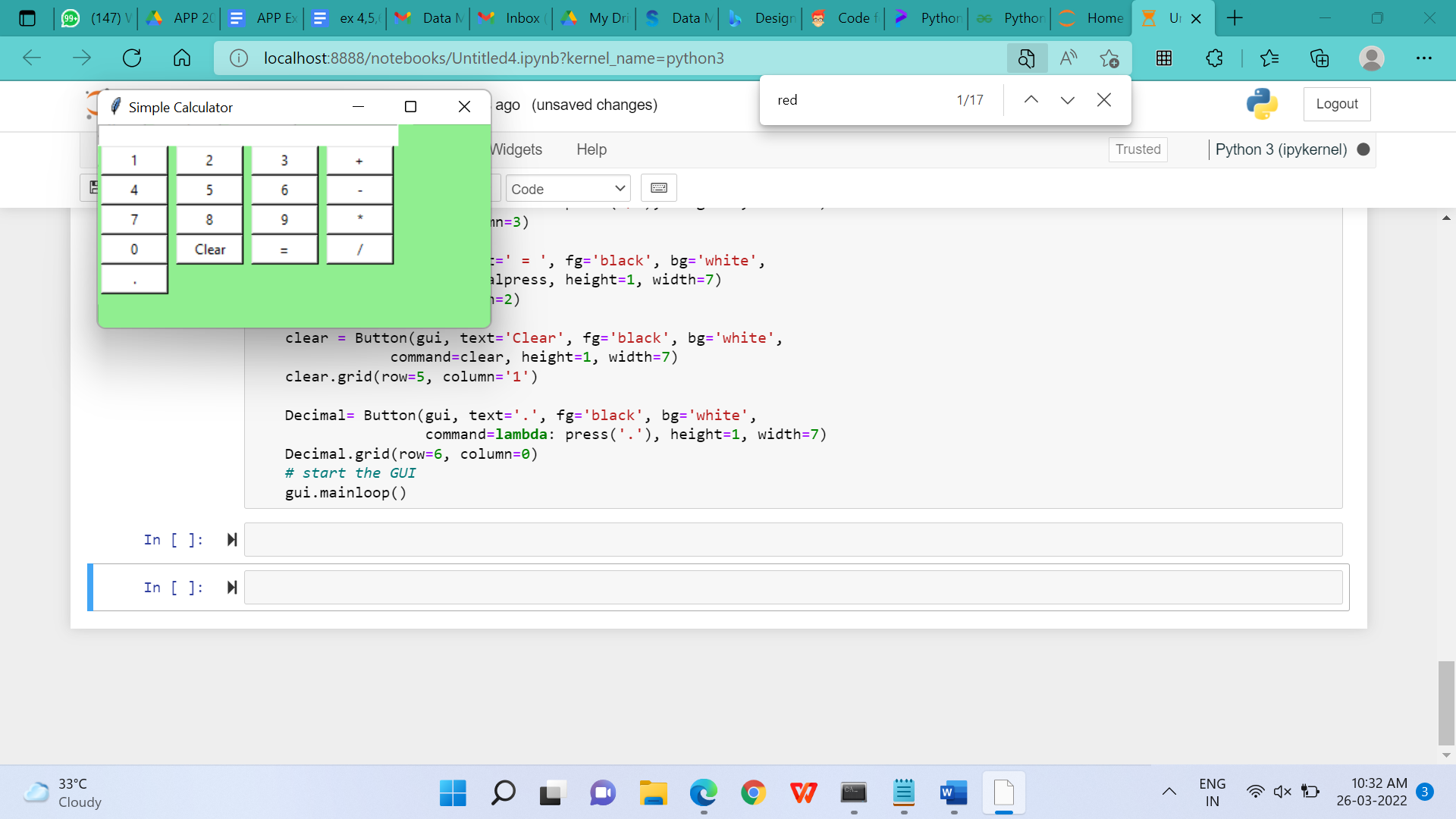
Decimal= Button(gui, text='.', fg='black', bg='red',

command=lambda: press('.'), height=1, width=7)

Decimal.grid(row=6, column=0)

gui.mainloop()

**Output:**



15.b Write a program to implement employee salary calculation using python. Input the employee name and the basic salary calculate and display the net salary using following condition.

(i)if the basic salary is more than 50000 then include 10% tax.

(ii)iif the basic salary is more than 30000 then include 5% tax.

**Algorithm:**

**Program:**

from tkinter import \*

def Ok():

result = float(e2.get())

if(result > 50000) :

tax = result \* 10/100

elif(result > 30000) :

tax = result \* 5/100

else :

tax = 0

taxText.set(tax)

nsal = result - tax

nsalText.set(nsal)

root = Tk()

root.title("Employee Salary Calculation System")

root.geometry("300x400")

global e1

global e2

global taxText

global nsalText

taxText = StringVar()

nsalText = StringVar()

Label(root, text="Employee Name").place(x=10, y=10)

Label(root, text="Salary").place(x=10, y=40)

Label(root, text="Tax").place(x=10, y=80)

Label(root, text="Total:").place(x=10, y=110)

e1 = Entry(root)

e1.place(x=100, y=10)

e2 = Entry(root)

e2.place(x=100, y=40)

tax = Label(root, text="", textvariable=taxText).place(x=100,y=80)

nsal = Label(root, text="", textvariable=nsalText).place(x=100, y=110)

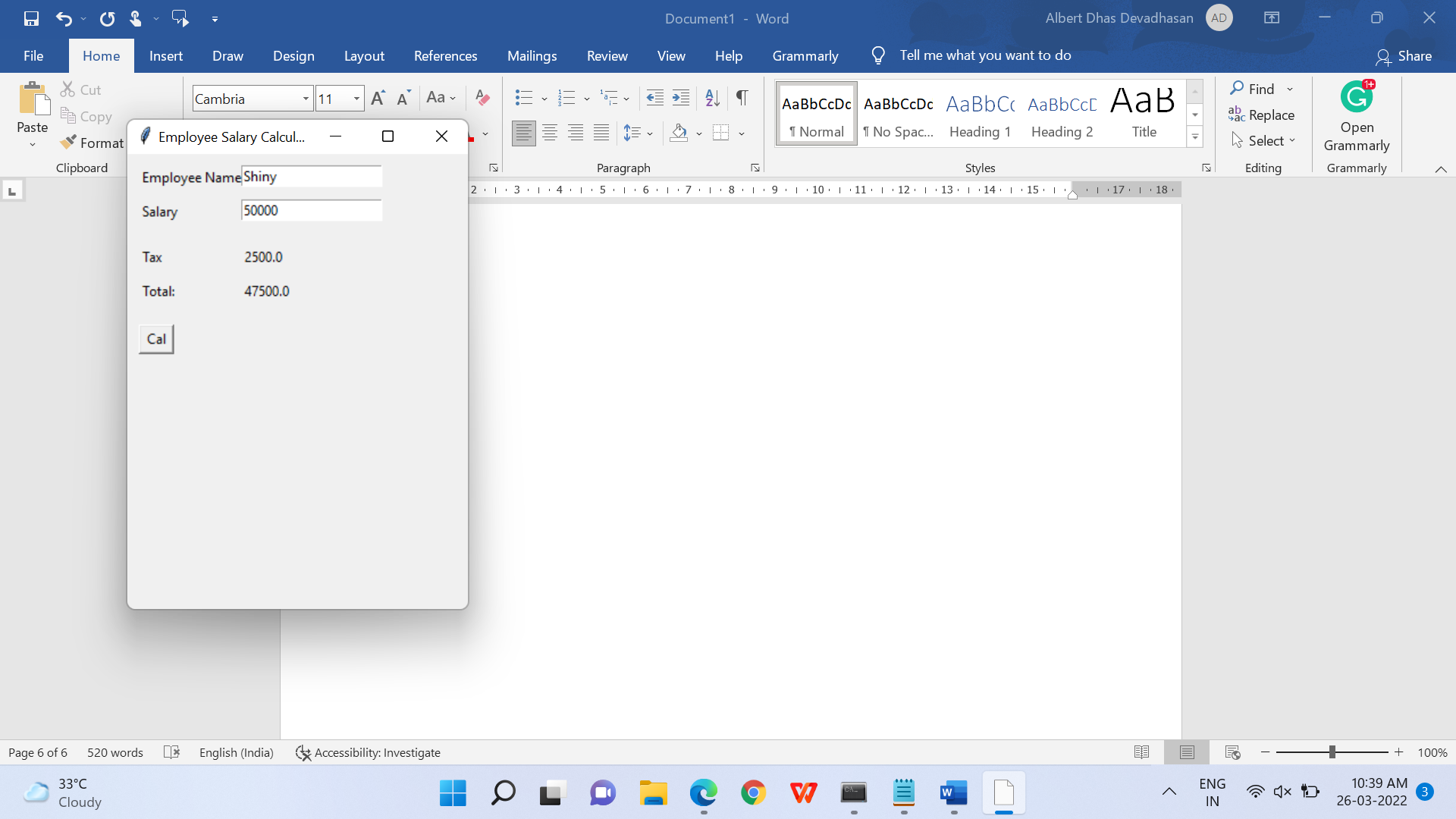
Button(root, text="Cal", command=Ok ,height = 1, width =3).place(x=10, y=150)

empname = Entry(root)

empsal = Entry(root)

root.mainloop()

**Output:**



**RESULT:**

Thus, the Python GUI program for calculator and employee salary calculation have been written and executed successfully.