**Cyber Calc ...**

**Source code**

from Tkinter import \*

import re #For RegEx

import math #For Sqrt

#Basic function that expects two operands as strings and returns the Addition of them as a string

def add(a,b):

if a=='':

a='0'

try:

a=int(a)

except ValueError:

a=float(a)

try:

b=int(b)

except ValueError:

b=float(b)

ans=a+b

ans=repr(ans)

try:

ans=int(ans)

except ValueError:

ans=float(ans)

return repr(ans)

#Basic function that expects two operands as strings and returns the Subtraction of them as a string

def sub(a,b):

try:

a=int(a)

except ValueError:

a=float(a)

try:

b=int(b)

except ValueError:

b=float(b)

ans=a-b

ans=repr(ans)

try:

ans=int(ans)

except ValueError:

ans=float(ans)

return repr(ans)

#Basic function that expects two operands as strings and returns the multiplication of them as a string

def mul(a,b):

try:

a=int(a)

except ValueError:

a=float(a)

try:

b=int(b)

except ValueError:

b=float(b)

ans=a\*b

ans=repr(ans)

try:

ans=int(ans)

except ValueError:

ans=float(ans)

return repr(ans)

#Basic function that expects two operands as strings and returns the division of them as a string

def divide(a,b):

a=float(a)

b=float(b)

ans=a/b

ans=repr(ans)

try:

ans=int(ans)

except ValueError:

ans=float(ans)

return repr(ans)

#Performs specific operations on elementary expression with two operands and one operator

def operation(oper1,oper2,oper):

if oper==0:

return divide(oper1,oper2)

if oper==1:

return mul(oper1,oper2)

if oper==2:

return add(oper1,oper2)

if oper==3:

return sub(oper1,oper2)

return -1

#Performs parsing for input operator and resolves expression

def operator\_parser(s,op,oper):

mid=s.find(op)

while mid!=-1:

i=mid-1;opn=-1;cls=-1

while i>-1 and s[i] not in {'+','-','\*','/'}:

i=i-1

opn=i;i=mid+1

if opn==mid-1 and op=='-':

break

while i<len(s) and s[i] not in {'+','-','\*','/'}:

i=i+1

if i==mid+1:

i=i+1

while i<len(s) and s[i] not in {'+','-','\*','/'}:

i=i+1

cls=i

oper1=s[opn+1:mid];oper2=s[mid+1:cls]

s=s[0:opn+1]+operation(oper1,oper2,oper)+s[cls:]

mid=s.find(op)

return s

#Calls for different parsing operations

def solve(s):

#Parse for division operation

s=operator\_parser(s,'/',0)

#Parse for multiplication operation

s=operator\_parser(s,'\*',1)

#Parse for Addition operation

s=operator\_parser(s,'+',2)

#Parse for Subtraction operation

s=operator\_parser(s,'-',3)

return s

#This function acts as a validation point using RegEx

def validate(s):

match=re.search(r'[^0-9+-/\\*()]',s)

if match:

return 0

match=re.search(r'\)\(',s)

if match:

return 0

match=re.search(r'\(/',s)

if match:

return 0

match=re.search(r'\(\\*',s)

if match:

return 0

match=re.search(r'\.[0-9]\*\.',s)

if match:

return 0

match=re.search(r'\+\)',s)

if match:

return 0

match=re.search(r'\-\)',s)

if match:

return 0

match=re.search(r'/\)',s)

if match:

return 0

match=re.search(r'\\*\)',s)

if match:

return 0

match=re.search(r'/\\*',s)

if match:

return 0

match=re.search(r'\\*/',s)

if match:

return 0

match=re.search(r'\+/',s)

if match:

return 0

match=re.search(r'//',s)

if match:

return 0

match=re.search(r'\\*\\*',s)

if match:

return 0

match=re.search(r'\+\\*',s)

if match:

return 0

match=re.search(r'\-/',s)

if match:

return 0

match=re.search(r'\-\\*',s)

if match:

return 0

match=re.search(r'(/\+/)|(/\+\\*)|(/\+\+)|(/\+\-)|(/\-/)|(/\-\\*)|(/\-\+)|(/\-\-)',s)

if match:

return 0

match=re.search(r'(\\*\+/)|(\\*\+\\*)|(\\*\+\+)|(\\*\+\-)|(\\*\-/)|(\\*\-\\*)|(\\*\-\+)|(\\*\-\-)',s)

if match:

return 0

match=re.search(r'(\+\+/)|(\+\+\\*)|(\+\+\+)|(\+\+\-)|(\+\-/)|(\+\-\\*)|(\+\-\+)|(\+\-\-)',s)

if match:

return 0

match=re.search(r'(\-\+/)|(\-\+\\*)|(\-\+\+)|(\-\+\-)|(\-\-/)|(\-\-\\*)|(\-\-\+)|(\-\-\-)',s)

if match:

return 0

return 1

#parsing to simplify the expression by ripping off brackets and carving out basic simple expression which doesn't have brackets

def earth(s):

#Regex Validation call

if validate(s)==0:

s="Invalid Expression"

return s

op=-1;cls=-1;i=0

while i<len(s) :

if s[i] == '(':

op=i

if s[i] == ')':

cls=i

if op==-1:

s="Invalid Expression"

break

simplify=solve(s[op+1:cls])

s=s[0:op]+simplify+s[cls+1:]

i=-1;op=-1;cls=-1

i=i+1

if op!=-1 or cls!=-1:

s="Invalid Expression"

return s

if s!="Invalid Expression":

s=solve(s)

return s

action\_text\_control=0

#Appends data to Entry for screen

def action\_append(expr\_inp,s):

global action\_text\_control

if action\_text\_control==1:

expr\_inp.delete(0,END)

action\_text\_control=0

expr\_inp.insert(END,s)

#Deletes Last character kinda backspace

def action\_trimLast(expr\_inp):

expr\_inp.delete(len(expr\_inp.get())-1)

#Clears screen

def action\_clear(expr\_inp):

expr\_inp.delete(0,END)

#Solves given expression by calling earth and displaying appropriate results

def action\_solve(expr\_inp):

global action\_text\_control

try:

s=expr\_inp.get()

s='('+s+')'

s=earth(s)

except:

s="Invalid Expression"

expr\_inp.delete(0,END)

expr\_inp.insert(0,s)

action\_text\_control=0

#Renders area screen

def switch\_areas():

peris.pack\_forget()

peri\_peri.pack\_forget()

peri\_semi .pack\_forget()

persqrt.pack\_forget()

#f1.pack\_forget()

f2.pack\_forget()

f3.pack\_forget()

areas.pack(fill=BOTH,expand=1)

rasq.select()

switch\_areas\_sq()

#Renders area screen for Triangle

def switch\_areas\_tri():

area\_sq.pack\_forget()

area\_rec.pack\_forget()

ate1.delete(0,END)

ate2.delete(0,END)

ate3.delete(0,END)

ate4.delete(0,END)

area\_tri.pack(fill=BOTH,expand=1)

#Renders area screen for Square

def switch\_areas\_sq():

area\_tri.pack\_forget()

area\_rec.pack\_forget()

aqe1.delete(0,END)

aqe2.delete(0,END)

area\_sq.pack(fill=BOTH,expand=1)

#Renders area screen for Rectangle

def switch\_areas\_rec():

area\_tri.pack\_forget()

area\_sq.pack\_forget()

are1.delete(0,END)

are2.delete(0,END)

are3.delete(0,END)

area\_rec.pack(fill=BOTH,expand=1)

#Renders Perimeter screen for Perimeter

def switch\_peri\_peri():

peri\_semi.pack\_forget()

pre1.delete(0,END)

pre2.delete(0,END)

pre3.delete(0,END)

peri\_peri.pack(fill=BOTH,expand=1)

#Renders Perimeter screen for Semi Perimeter

def switch\_peri\_semi():

peri\_peri.pack\_forget()

spe1.delete(0,END)

spe2.delete(0,END)

spe3.delete(0,END)

peri\_semi.pack(fill=BOTH,expand=1)

#Renders Perimeter screen

def switch\_peris():

areas.pack\_forget()

area\_tri.pack\_forget()

area\_sq.pack\_forget()

area\_rec.pack\_forget()

persqrt.pack\_forget()

#f1.pack\_forget()

f2.pack\_forget()

f3.pack\_forget()

peris.pack(fill=BOTH,expand=1)

pesq.select()

switch\_peri\_peri()

#Renders Percentage Sqrt screen

def switch\_persqrt():

areas.pack\_forget()

area\_tri.pack\_forget()

area\_sq.pack\_forget()

area\_rec.pack\_forget()

peris.pack\_forget()

peri\_peri.pack\_forget()

peri\_semi .pack\_forget()

#f1.pack\_forget()

f2.pack\_forget()

f3.pack\_forget()

pqe1.delete(0,END)

pqe2.delete(0,END)

persqrt.pack(fill=BOTH,expand=1)

#Find Perimeter

def find\_peri(pre1,pre2,pre3):

n=pre1.get()

s=pre2.get()

n='('+n+')'

s='('+s+')'

try:

s=earth('('+n+'\*'+s+')')

except:

s="Invalid Input"

pre3.delete(0,END)

pre3.insert(0,s)

#Find SemiPerimter

def find\_semi(pre1,pre2,pre3):

n=pre1.get()

s=pre2.get()

n='('+n+')'

s='('+s+')'

try:

s=earth('(('+n+'\*'+s+')/2)')

except:

s="Invalid Input"

pre3.delete(0,END)

pre3.insert(0,s)

#Find Area of Triangle using Heroine's Formulae

def find\_area\_triangle(a,b,c):

a=float(a)

b=float(b)

c=float(c)

s=(a+b+c)/2

sa=s-a

sb=s-b

sc=s-c

return math.sqrt(s\*sa\*sb\*sc)

#Handler for handling calculate triangle area event and set answer field

def find\_triarea(pre1,pre2,pre3,pre4):

a=pre1.get()

b=pre2.get()

c=pre3.get()

try:

a=int(a)

except:

s='Invalid Side A'

pre4.delete(0,END)

pre4.insert(0,s)

return

try:

b=int(b)

except:

s='Invalid Side B'

pre4.delete(0,END)

pre4.insert(0,s)

return

try:

c=int(c)

except:

s='Invalid Side C'

pre4.delete(0,END)

pre4.insert(0,s)

return

s=str(find\_area\_triangle(a,b,c))

pre4.delete(0,END)

pre4.insert(0,s)

#Handler for handling calculate square area event and set answer field

def find\_sqarea(pre1,pre2):

n=pre1.get()

try:

n=int(n)

except:

s='Invalid Side'

pre2.delete(0,END)

pre2.insert(0,s)

return

s=str(n\*n)

pre2.delete(0,END)

pre2.insert(0,s)

#Handler for handling calculate Rectangle area event and set answer field

def find\_recarea(pre1,pre2,pre3):

n=pre1.get()

s=pre2.get()

try:

n=int(n)

except:

s='Invalid Length'

pre3.delete(0,END)

pre3.insert(0,s)

return

try:

s=int(s)

except:

s="Invalid Breadth"

pre3.delete(0,END)

pre3.insert(0,s)

return

s=str(n\*s)

pre3.delete(0,END)

pre3.insert(0,s)

#Handler for handling calculate Percentage Sqrt event and set answer field

def find\_persqrt(pre1,pre2):

n=pre1.get()

try:

n=float(n)

except:

s='Invalid Percentage'

pre2.delete(0,END)

pre2.insert(0,s)

return

s=str(math.sqrt(n/100))

pre2.delete(0,END)

pre2.insert(0,s)

#Created basic container for window

root = Tk()

root.wm\_title("CyberCalc")

root.resizable(0,0)

#Created basic Menu container

menu=Menu(root)

root.config(menu=menu)

#Menu Created

submenu=Menu(menu,tearoff=0)

submenu1=Menu(menu,tearoff=0)

menu.add\_cascade(label="Operations",menu=submenu)

submenu.add\_command(label="Basic",command=lambda:construct\_main\_calc())

submenu.add\_command(label="Area",command=lambda: switch\_areas())

submenu.add\_command(label="Peri & Semi-peri",command=lambda: switch\_peris())

submenu.add\_command(label="Percent sqrt",command=lambda:switch\_persqrt())

menu.add\_cascade(label="Exit", menu=submenu1)

submenu1.add\_command(label="Exit", command=quit)

#Defining all frames required

##f1=Frame(root)

##f1.pack(fill=BOTH,expand=1)

f2=Frame(root,background="#FFFFFF")

f2.pack(fill=BOTH,expand=1,ipadx=0,ipady=0)

f3=Frame(root,background="#434343")

f3.pack(fill=BOTH,expand=1)

areas=Frame(root,background="#434343")

area\_tri=Frame(root,background="#434343")

area\_sq=Frame(root,background="#434343")

area\_rec=Frame(root,background="#434343")

peris=Frame(root,background="#434343")

peri\_peri=Frame(root,background="#434343")

peri\_semi=Frame(root,background="#434343")

persqrt=Frame(root,background="#434343")

#expr screen defined and gridded

expr\_inp = Entry(f2,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=26)

expr\_inp.grid(row=1,columnspan=6,padx=(0,0),pady=(0,0),ipadx=10,ipady=5,sticky=E+W)

#Added key bindings for enter event

def temp\_func(event):

if event.keycode==104:

action\_solve(expr\_inp)

root.bind('<Return>', lambda event: action\_solve(expr\_inp))

root.bind('<Key>', temp\_func)

#Basic buttons Created and gridded

b\_7 = Button(f3, text=" 7 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'7'))

b\_7.grid(row=1, column=1,columnspan=1,padx=(10,5),pady=(10,5),sticky=E+W)

b\_8 = Button(f3, text=" 8 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'8'))

b\_8.grid(row=1, column=2,columnspan=1,padx=(5,5),pady=(10,5),sticky=E+W)

b\_9 = Button(f3, text=" 9 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'9'))

b\_9.grid(row=1, column=3,columnspan=1,padx=(5,5),pady=(10,5),sticky=E+W)

b\_div = Button(f3, text=" / ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'/'))

b\_div.grid(row=1, column=4,columnspan=1,padx=(5,5),pady=(10,5),sticky=E+W)

b\_back = Button(f3, text=" AC ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_clear(expr\_inp))

b\_back.grid(row=1, column=5,columnspan=1,padx=(5,5),pady=(10,5),sticky=E+W)

b\_4 = Button(f3, text=" 4 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'4'))

b\_4.grid(row=2, column=1,columnspan=1,padx=(10,5),pady=(5,5),sticky=E+W)

b\_5 = Button(f3, text=" 5 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'5'))

b\_5.grid(row=2, column=2,columnspan=1,padx=(5,5),pady=(5,5),sticky=E+W)

b\_6 = Button(f3, text=" 6 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'6'))

b\_6.grid(row=2, column=3,columnspan=1,padx=(5,5),pady=(5,5),sticky=E+W)

b\_mul = Button(f3, text=" \* ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'\*'))

b\_mul.grid(row=2, column=4,columnspan=1,padx=(5,5),pady=(5,5),sticky=E+W)

b\_oc = Button(f3, text=" ( ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'('))

b\_oc.grid(row=2, column=5,columnspan=1,padx=(5,5),pady=(5,5),sticky=E+W)

b\_1 = Button(f3, text=" 1 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'1'))

b\_1.grid(row=3, column=1,columnspan=1,padx=(10,5),pady=(5,5),sticky=E+W)

b\_2 = Button(f3, text=" 2 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'2'))

b\_2.grid(row=3, column=2,columnspan=1,padx=(5,5),pady=(5,5),sticky=E+W)

b\_3 = Button(f3, text=" 3 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'3'))

b\_3.grid(row=3, column=3,columnspan=1,padx=(5,5),pady=(5,5),sticky=E+W)

b\_sub = Button(f3, text=" - ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'-'))

b\_sub.grid(row=3, column=4,columnspan=1,padx=(5,5),pady=(5,5),sticky=E+W)

b\_cb = Button(f3, text=" ) ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,')'))

b\_cb.grid(row=3, column=5,columnspan=1,padx=(5,5),pady=(5,5),sticky=E+W)

b\_0 = Button(f3, text=" 0 ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'0'))

b\_0.grid(row=4, column=1,columnspan=1,padx=(10,5),pady=(5,10),sticky=E+W)

b\_deci = Button(f3, text=" . ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'.'))

b\_deci.grid(row=4, column=2,columnspan=1,padx=(5,5),pady=(5,10),sticky=E+W)

b\_per = Button(f3, text=" C ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_trimLast(expr\_inp))

b\_per.grid(row=4, column=3,columnspan=1,padx=(5,5),pady=(5,10),sticky=E+W)

b\_add = Button(f3, text=" + ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 12),command= lambda: action\_append(expr\_inp,'+'))

b\_add.grid(row=4, column=4,columnspan=1,padx=(5,5),pady=(5,10),sticky=E+W)

b\_eq = Button(f3, text=" = ",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica",12),command= lambda: action\_solve(expr\_inp))

b\_eq.grid(row=4, column=5,padx=(5,5),pady=(5,10),sticky=E+W)

#Main calc screen rendered

def construct\_main\_calc():

areas.pack\_forget()

area\_tri.pack\_forget()

area\_sq.pack\_forget()

area\_rec.pack\_forget()

peris.pack\_forget()

peri\_peri.pack\_forget()

peri\_semi .pack\_forget()

persqrt.pack\_forget()

#f1.pack(fill=BOTH,expand=1)

f2.pack(fill=BOTH,expand=1,ipadx=0,ipady=0)

f3.pack(fill=BOTH,expand=1)

#Widgets for other screens Created and gridded

#Area screen Widgets

#arbk = Button(areas, text="Back",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 10),command=lambda:construct\_main\_calc())

arbk = Label(areas, text="",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 10))

area\_radio\_control=IntVar()

rasq = Radiobutton(areas, text="Square's Area", variable=area\_radio\_control,highlightthickness=0,value=1,bg="#434343",fg="#FFFFFF",selectcolor="#434343",command=lambda: switch\_areas\_sq(),font=("Helvetica", 12))

ratri = Radiobutton(areas, text="Triangle's Area", variable=area\_radio\_control,highlightthickness=0, value=2,bg="#434343",fg="#FFFFFF",selectcolor="#434343",command=lambda: switch\_areas\_tri(),font=("Helvetica", 12))

rarec = Radiobutton(areas, text="Rectangle's Area", variable=area\_radio\_control,highlightthickness=0, value=3,bg="#434343",fg="#FFFFFF",selectcolor="#434343",command=lambda: switch\_areas\_rec(),font=("Helvetica", 12))

arbk.grid(row=1,column=8,columnspan=2,padx=(10,5),pady=(5,5),ipadx=2,ipady=2,sticky=E+W)

rasq.grid(row=2,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

ratri.grid(row=2,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

rarec.grid(row=2,column=7,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

#Area Triangle Widgets

atl1 = Label(area\_tri, text="Side A",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

atl1t = Label(area\_tri, text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

atl1t1 = Label(area\_tri, text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

atl1t2 = Label(area\_tri, text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

atl2 = Label(area\_tri, text="Side B",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

atl3 = Label(area\_tri, text="Side C",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

atl4 = Button(area\_tri, text="Area of Triangle",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 11),command=lambda:find\_triarea(ate1,ate2,ate3,ate4))

ate1 = Entry(area\_tri,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

ate2 = Entry(area\_tri,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

ate3 = Entry(area\_tri,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

ate4 = Entry(area\_tri,highlightthickness=0,font=("Helvetica", 10),borderwidth=0,justify=RIGHT,width=10)

atl1t.grid(row=1,column=1,columnspan=2,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

atl1.grid(row=1,column=3,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

atl2.grid(row=1,column=6,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

atl3.grid(row=1,column=9,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

atl1t1.grid(row=2,column=1,columnspan=2,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

ate1.grid(row=2,column=3,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

ate2.grid(row=2,column=6,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

ate3.grid(row=2,column=9,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

atl1t2.grid(row=3,column=1,columnspan=2,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

atl4.grid(row=3,column=3,columnspan=3,padx=(10,5),pady=(5,5),ipadx=0,ipady=0,sticky=E+W)

ate4.grid(row=3,column=6,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

#Area Square Widgets

aql1 = Label(area\_sq, text="Side",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

aql1t = Label(area\_sq, text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

aql1t1 = Label(area\_sq, text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

aql2 = Button(area\_sq, text="Area of Square",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 11),command=lambda:find\_sqarea(aqe1,aqe2))

aqe1 = Entry(area\_sq,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

aqe2 = Entry(area\_sq,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

aql1t.grid(row=1,column=1,columnspan=2,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

aql1.grid(row=1,column=3,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

aqe1.grid(row=1,column=6,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

aql1t1.grid(row=2,column=1,columnspan=2,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

aql2.grid(row=2,column=3,columnspan=3,padx=(10,5),pady=(5,5),ipadx=0,ipady=0,sticky=E+W)

aqe2.grid(row=2,column=6,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

#Area Rectangle Widgets

arl1 = Label(area\_rec, text="Length",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

arl1t = Label(area\_rec, text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

arl1t1 = Label(area\_rec, text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

arl1t2 = Label(area\_rec, text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

arl2 = Label(area\_rec, text="Breadth",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

arl3 = Button(area\_rec, text="Area of Rectangle",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 11),command=lambda:find\_recarea(are1,are2,are3))

are1 = Entry(area\_rec,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

are2 = Entry(area\_rec,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

are3 = Entry(area\_rec,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

arl1t.grid(row=1,column=1,columnspan=2,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

arl1.grid(row=1,column=3,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

are1.grid(row=1,column=6,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

arl1t1.grid(row=1,column=1,columnspan=2,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

arl2.grid(row=2,column=3,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

are2.grid(row=2,column=6,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

arl1t2.grid(row=1,column=1,columnspan=2,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

arl3.grid(row=3,column=3,columnspan=3,padx=(10,5),pady=(5,5),ipadx=0,ipady=0,sticky=E+W)

are3.grid(row=3,column=6,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

#Perimeter Widgets

#pebk = Button(peris, text="Back",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 10),command=lambda:construct\_main\_calc())

pebk = Label(peris, text="",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 10))

peea\_radio\_control=IntVar()

pesq = Radiobutton(peris, text="Perimeter", variable=peea\_radio\_control, value=1,highlightthickness=0,bg="#434343",fg="#FFFFFF",selectcolor="#434343",command=lambda: switch\_peri\_peri(),font=("Helvetica", 12))

petri = Radiobutton(peris, text="SemiPerimter", variable=peea\_radio\_control, value=2,highlightthickness=0,bg="#434343",fg="#FFFFFF",selectcolor="#434343",command=lambda: switch\_peri\_semi(),font=("Helvetica", 12))

pebk.grid(row=1,column=5,columnspan=2,padx=(10,5),pady=(5,5),ipadx=2,ipady=2,sticky=E+W)

pesq.grid(row=2,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

petri.grid(row=2,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

#Perimeter Perimeter screen Widgets

prl1 = Label(peri\_peri, text="No. of sides",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

prl2 = Label(peri\_peri, text="Length of side",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

prl3 = Button(peri\_peri, text="Perimeter",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 11),command=lambda:find\_peri(pre1,pre2,pre3))

pre1 = Entry(peri\_peri,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

pre2 = Entry(peri\_peri,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

pre3 = Entry(peri\_peri,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

prl1.grid(row=1,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

pre1.grid(row=1,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

prl2.grid(row=2,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

pre2.grid(row=2,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

prl3.grid(row=3,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=1,ipady=1,sticky=E+W)

pre3.grid(row=3,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

#Perimeter SemiPerimter Widgets

spl1 = Label(peri\_semi, text="No. of sides",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

spl2 = Label(peri\_semi, text="Length of side",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

spl3 = Button(peri\_semi, text="Semi-Perimeter",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 11),command=lambda:find\_semi(spe1,spe2,spe3))

spe1 = Entry(peri\_semi,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

spe2 = Entry(peri\_semi,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

spe3 = Entry(peri\_semi,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

spl1.grid(row=1,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

spe1.grid(row=1,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

spl2.grid(row=2,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

spe2.grid(row=2,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

spl3.grid(row=3,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=0,ipady=0,sticky=E+W)

spe3.grid(row=3,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

#Percentage Sqrt Widgets

#pqbk = Button(persqrt, text="Back",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 10),command=lambda:construct\_main\_calc())

pqbk = Label(persqrt, text="",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 10))

pql1 = Label(persqrt, text="Percentage",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvetica", 12))

pql2 = Button(persqrt, text="Percentage Sqrt",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvetica", 11),command=lambda:find\_persqrt(pqe1,pqe2))

pqe1 = Entry(persqrt,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

pqe2 = Entry(persqrt,highlightthickness=0,font=("Helvetica", 12),borderwidth=0,justify=RIGHT,width=10)

pqbk.grid(row=1,column=5,columnspan=2,padx=(10,5),pady=(5,5),ipadx=2,ipady=2,sticky=E+W)

pql1.grid(row=2,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

pqe1.grid(row=2,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

pql2.grid(row=3,column=1,columnspan=3,padx=(10,5),pady=(5,5),ipadx=0,ipady=0,sticky=E+W)

pqe2.grid(row=3,column=4,columnspan=3,padx=(10,5),pady=(5,5),ipadx=10,ipady=5,sticky=E+W)

root.mainloop()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***CyberCalc**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\***Notes**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Just as any other Calculator out there this one also performs the same Addition, Subtraction, Multiplication, Divsion and some other operations as area calculation, perimeter calculation and percentage square root.

\*The code involves a macro view which is a calculator broken down to smaller view as a backend part which deals with problem solving and a front part which deals with the GUI and how things look.

\***The backend part**\*

Involves basically accepting an expression and ripping off things according to BODMAS rule to break down the expression into elementary solvable expressions. The result of these are later combined to form the overall answer to expression.

Functions are defined for separete parts to be solved for a possible expression working in order of earth(),validate(),solve(),operator\_parser(),and rest are elementarty functions for simple addition,subtraction,division

and subtraction.

Validate function performs expression validation using RegEx and definning rules which must be followed for an

expression to be atleast valid for evalution.

Parsing is to simplify the expression by ripping off the brackets and carving out basic simple expression which

doesn't have brackets.

earth(s)

#This function acts as a validation point using RegEx.

validate(s)

#Calls for different parsing operations.

solve(s)

#Performs parsing for input operator and resolves expression.

operator\_parser(s,op,oper)

#Basic function that accepts two operands as strings and returns the Addition of them as a string.

add(a,b)

#Basic function that accepts two operands as strings and returns the Subtraction of them as a string.

sub(a,b)

#Basic function that accepts two operands as strings and returns the multiplication of them as a string.

mul(a,b)

#Basic function that accepts two operands as strings and returns the division of them as a string.

divide(a,b)

The problem here for backend part is a given mathematical expression and we need to evaluate it.

Possible solutions are:

1.) Create a custom set of functions which rips off the operators according to BODMAS.

2.) Use python's inbuilt Eval function

**Solutions used here are:**

First solution i.e. Create a custom set of functions which rips off operators according to BODMAS, has been used.

**Why Not 2nd and easier solution:**

1.) Python's eval function can execute any string as python code. This is a security loop hole as, an and expression which

might be malicious code may get executed.

2.) Python's eval function doesn't give always the right answer owing to implicit typecasting nature.

For example,

Python's eval function would result in 0 if given an expression 2/3.

Expected behaviour is 2/3 = 0.666666667

Actual Behaviour is 2/3 = 0

Hence this solution was dropped for the first solution.

\* **The Front end part** \*

The front end part consists of GUI constructed using Tkinter library of python.

The front end GUI consists of:

1.) A main Root element which holds the main container window.

2.) Different frames are required for different tasks.

-Package manager used for placing widgets in tkinter, pack manager for main root container, grid manager for within frames.

-A menu bar has been used for providing switching facility between different screens available.

Screens available namely are:

1.) Basic

2.) Area

3.) Perimeter and semi-perimeter

4.) Percentage Square root

Labels, Buttons, Entry fields have been used to provide an interactive GUI environment for the user.

User has the liberty to provide data input both from keyboard or an onscreen numeric pad displayed along with the basic screen.

**Project Team:**

|  |  |  |
| --- | --- | --- |
| Srno. | Name | Roll no. |
| 1 | Adhyan | CO14305 |
| 2 | Aditya Bansal | CO14306 |
| 3 | Akanksha | CO14307 |
| 4 | Akanksha Kumari | CO14308 |
| 5 | Akriti | LCO14364 |
| 6 | Tania | LCO14376 |