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_ca
lc())
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,ip
ady=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=("Helveti
ca", 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=("Helv
etica", 10))
area_radio_control=IntVar()
rasq = Radiobutton(areas, text="Square's Area",
variable=area_radio_control,highlightthickness=0,value=1,bg="#4343
43",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=la
mbda: 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=la
mbda: 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),ipad
x=10,ipady=5,sticky=E+W)
rarec.grid(row=2,column=7,columnspan=3,padx=(10,5),pady=(5,5),ipad
x=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=("Helv
etica", 12))
atl1t1 = Label(area_tri,
text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helv
etica", 12))
atl1t2 = Label(area_tri,
text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helv
etica", 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=("He
lvetica", 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),ipad
x=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),ipa
dx=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),ipa
dx=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=("Helv
etica", 12))
aql1t1 = Label(area_sq,
text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helv
etica", 12))
aql2 = Button(area_sq, text="Area of
Square",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helv
etica", 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),ipad
x=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),ipa
dx=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=("Helv
etica", 12))
arl1t1 = Label(area_rec,
text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helv
etica", 12))
arl1t2 = Label(area_rec,
text="",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helv
etica", 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=("H
elvetica", 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),ipad
x=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),ipa
dx=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),ipa
dx=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=("Helv
etica", 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),ipad
x=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=("Helve
tica", 12))
prl2 = Label(peri_peri, text="Length of
side",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvet
ica", 12))
prl3 = Button(peri_peri,
text="Perimeter",bg="#434343",fg="#FFFFFF",highlightthickness=0,fo
nt=("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=("Helve
tica", 12))
spl2 = Label(peri_semi, text="Length of
side",highlightthickness=0,bg="#434343",fg="#FFFFFF",font=("Helvet
```

```
ica", 12))
spl3 = Button(peri_semi, text="Semi-
Perimeter",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("H
elvetica", 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=("Helv
etica", 10))
pql1 = Label(persqrt,
text="Percentage",highlightthickness=0,bg="#434343",fg="#FFFFFF",f
ont=("Helvetica", 12))
pql2 = Button(persqrt, text="Percentage
Sqrt",bg="#434343",fg="#FFFFFF",highlightthickness=0,font=("Helvet
ica", 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