

# Create a GUI Scientific Calculator in Python using Tkinter library\_\_

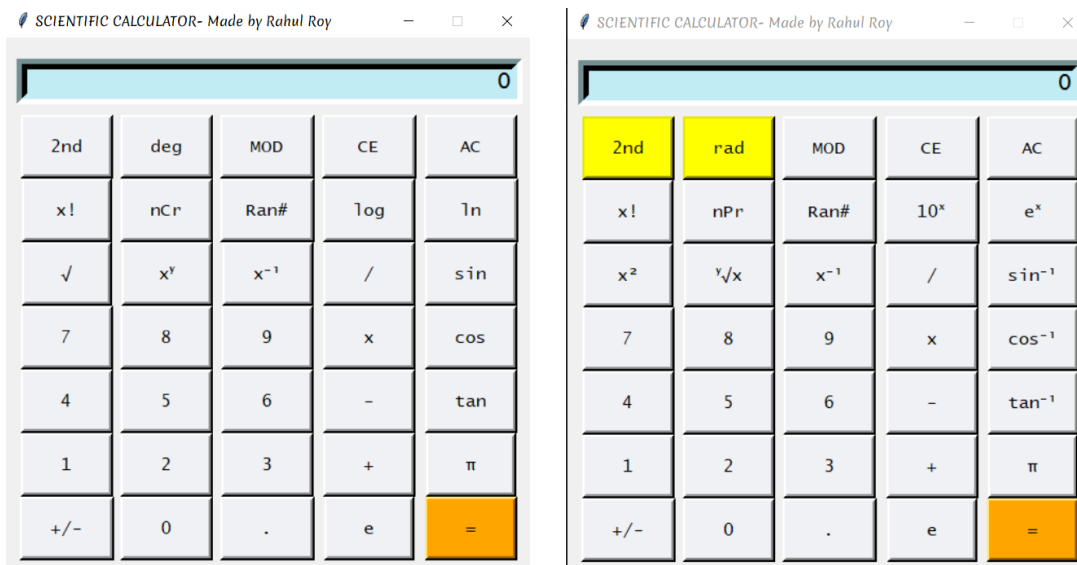
In this blog, we are going to build our own Scientific Calculator using Tkinter library of Python.

Having a scientific calculator is an absolute requirement when it comes to the daily needs of a science student. Especially in these days of online learning, having a Scientific Calculator on your screen helps a lot in online assignments.

This article is here to help you build one using Python's standard GUI toolkit - Tkinter.

## Logic behind the working of this Scientific Calculator\_\_

Our Calculator will look like this :-



1. We will create a class Calc where the instance variables shall be
  - i. 'op'(Stores the current operator clicked by user),
  - ii. 'M'(stores memory),
  - iii. 'isTrue\_2nd'(stores True if 2<sup>nd</sup> Button is clicked, it activates the second set of scientific functions like  $\sin^{-1}$ ,  $10^x$ ,  $e^x$ ,  $\sqrt[x]{x}$  ),
  - iv. 'isdegree'(True if 'deg' is on, takes radians for trigonometric operations when 'deg' is clicked to 'rad' and isdegree = False) and
  - v. 'decimalclicked'(turns True when user clicks '.')
2. We create all the functions to be invoked on clicking any operational button.
3. Then we come to the structural part of the Calculator, we will build the display box, the buttons with respective paddings ,styles(necessary fonts and colors) and commands as the function to be executed after being clicked.
4. At last, we will arrange all these buttons in a grid geometry.

## Python Code to implement the Calculator functions\_\_\_\_\_

```
class calc():
    def __init__(self):
        self.op, self.M = None, None,
        self.isTrue_2nd, self.isdegree, = False, True
        self.decimalclicked = False
```

```

# AC() clears the Memory, Sets the Display to 0
def AC(self):
    self.op, self.M, self.decimalclicked = None, None, False
    equation.set("0")

# CE() clears the Most Recent Entry
def CE(self):
    self.decimalclicked = False
    equation.set("0")

def error(self, type=""):
    self.CE()
    equation.set(type + "ERROR")

def numclick(self,num):
    curdisp = equation.get()

    if not self.op and curdisp == '0':
        equation.set(num)
    elif self.op and curdisp == self.M:
        equation.set(num)
    else:
        equation.set(curdisp + num)

def signClick(self):
    try:
        curdisp = equation.get()
        if curdisp != '0':
            equation.set(str(float(equation.get())*-1))
    except:
        self.error()

def decimalClick(self):
    curdisp = equation.get()

    if self.op and curdisp == self.M:
        equation.set("0.")
    elif not self.decimalclicked:
        equation.set(curdisp+".")
        self.decimalclicked = True

def piRanClick(self,clicked):
    self.CE()

    val = {'e':str(math.e),'pi':str(math.pi),'Ran':str(random.random())}
    equation.set(val[clicked])

def DMAS_mod_Click(self,clicked):
    self.M = equation.get()
    self.op = clicked

def rec_fact_click(self,clicked):
    cur = equation.get()

```

```

    try:
        if clicked == 'r':
            cur = str(float(cur)**-1)
        if clicked == '!':
            cur = str(math.factorial(float(cur)))
        equation.set(cur)

    except:
        self.error(type="MATH ")

def trig_log_root_click(self, clicked):
    cur = equation.get()

    try:
        if clicked == 'g':
            cur = str((math.log10(float(cur)), math.pow(10, float(cur))) [
self.isTrue_2nd])
        elif clicked == 'n':
            cur = str((math.log(float(cur)), math.pow(math.e, float(cur)))
[self.isTrue_2nd])
        elif clicked == 'r':
            cur = str((math.sqrt(float(cur)), math.pow(float(cur), 2)) [se
lf.isTrue_2nd])

        if clicked in "gnr":
            equation.set(cur)
            return

        if self.isdegree and not self.isTrue_2nd:
            cur = float(cur)*math.pi/180
        else:
            cur = float(cur)

        if clicked == 's':
            cur = str(round((math.sin(cur), math.asin(cur)) [self.isTrue_2
nd], 10))
        elif clicked == 'c':
            cur = str(round((math.cos(cur), math.acos(cur)) [self.isTrue_2
nd], 10))
        elif clicked == 't':
            cur = str(round((math.tan(cur), math.atan(cur)) [self.isTrue_2
nd], 10))

        equation.set(cur)
    except:
        self.error(type="MATH ")

def pow_pc_clicked(self, clicked):
    self.M = equation.get()

    inv_func = {'pow': 'yroot', 'c': 'p'}
    self.op = {True: inv_func[clicked], False: clicked} [self.isTrue_2nd]

def equalpress(self):

```

```

# = pressed Just after op clicked
if self.op and equation.get() == self.M:
    self.error()
    return

cur = equation.get()
try:

    if self.op == '+':
        cur = str(float(self.M) + float(cur))
    elif self.op == '-':
        cur = str(float(self.M) - float(cur))
    elif self.op == '*':
        cur = str(float(cur) * float(self.M))
    elif self.op == '/':
        cur = str(float(self.M)/float(cur))
    elif self.op == 'mod':
        cur = str(float(self.M) % float(cur))
    elif self.op == 'pow':
        cur = str(math.pow(float(self.M),float(cur)))
    elif self.op == 'yroot':
        cur = str(math.pow(float(self.M),1/float(cur)))
    elif self.op == 'c':
        cur = str(math.comb(int(float(self.M)),int(float(cur))))
    elif self.op == 'p':
        cur = str(math.perm(int(float(self.M)),int(float(cur))))

    self.AC()

    if '.' in cur:
        self.decimalclicked = True
        equation.set(cur)
except:
    self.error(type="MATH ")

def deg_2nd_click(self, clicked):
    if clicked == 'deg':
        self.isdegree = not self.isdegree
        btn_deg["text"] = {True: "rad", False: "deg"} [btn_deg["text"] ==
"deg"]
        btn_deg["bg"] = {True: "yellow", False: "#eff1f4"} [btn_deg["bg"]
=="#eff1f4"]

        elif clicked == '2nd':
            self.isTrue_2nd = not self.isTrue_2nd
            btn_permucombo["text"] = {True: "nPr", False: "nCr"} [btn_permucom
bo["text"] == "nCr"]
            btn_log10["text"] = {True: "10"+get_super('x'), False: "log"} [btn
_log10["text"] == "log"]
            btn_loge["text"] = {True: "e"+get_super('x'), False: "ln"} [btn_lo
ge["text"] == "ln"]
            btn_root["text"] = {True: "x"+get_super('2'), False: "√"} [btn_roo
t["text"] == "√"]

```

```

        btn_power["text"] = {True: get_super('y')+"√x", False: "x"+get_super('y')} [btn_power["text"] == "x"+get_super('y')]
        btn_sin["text"] = {True: "sin"+get_super('-1'), False: "sin"} [btn_sin["text"] == "sin"]
        btn_cos["text"] = {True: "cos"+get_super('-1'), False: "cos"} [btn_cos["text"] == "cos"]
        btn_tan["text"] = {True: "tan"+get_super('-1'), False: "tan"} [btn_tan["text"] == "tan"]

        btn_log10["padx"] = {True: 23, False: 20} [btn_log10["padx"] == 20]

        btn_root["padx"] = {True: 24, False: 29} [btn_root["padx"] == 29]
        btn_power["padx"] = {True: 22, False: 27} [btn_power["padx"] == 27]

        btn_sin["padx"] = {True: 11, False: 20} [btn_sin["padx"] == 20]
        btn_cos["padx"] = {True: 11, False: 20} [btn_cos["padx"] == 20]
        btn_tan["padx"] = {True: 11, False: 20} [btn_tan["padx"] == 20]

        btn_2nd["bg"] = {True: "yellow", False: "#eff1f4"} [btn_2nd["bg"] == "#eff1f4"]

c = calc()

```

## Python Code to Set up the GUI interface

```

from tkinter import *
import tkinter.font
import math, random

calc_root = Tk()
calc_root.title("SCIENTIFIC CALCULATOR GUI")
calc_root.resizable(width=False, height=False)
root = Frame(master=calc_root)
root.pack(padx=10, pady=10)
myfont = tkinter.font.Font(family = "Lucida Console", size = 12)

equation = StringVar(value='0')
input = Entry(root, textvariable=equation, bg='#c1ecf4', bd=10, width=35, font=("Lucida Console", 16), justify=RIGHT)
input.grid(row=0, column=0, columnspan=5, pady=10)

# Get the Superscripts

def get_super(x):
    normal = "xy12-()"
    superscript = "xy12-0"
    res = x.maketrans(''.join(normal), ''.join(superscript))
    return x.translate(res)

```

```

btn_1 = Button(root, text="1", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('1'))
btn_2 = Button(root, text="2", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('2'))
btn_3 = Button(root, text="3", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('3'))
btn_4 = Button(root, text="4", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('4'))
btn_5 = Button(root, text="5", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('5'))
btn_6 = Button(root, text="6", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('6'))
btn_7 = Button(root, text="7", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('7'))
btn_8 = Button(root, text="8", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('8'))
btn_9 = Button(root, text="9", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('9'))
btn_0 = Button(root, text="0", padx=30, pady=15, bd=4, bg="#eff1f4", font=myfont, co
mmand=lambda: c.numclick('0'))
btn_point = Button(root, text=".", padx=31, pady=15, bd=4, bg="#eff1f4", font=myfont
, command=lambda: c.decimalClick())

```

#### #Standard Operators

```

btn_add = Button(root, text="+", padx=30, pady=15, bd=4, font=myfont, bg="#eff1f4",
command=lambda: c.DMAS_mod_Click('+'))
btn_sub = Button(root, text="-", padx=30, pady=15, bd=4, font=myfont, bg="#eff1f4", command=lambda: c.DMAS_mod_Cli
ck('-'))
btn_mult = Button(root, text="x", padx=30, pady=15, bd=4, font=myfont, bg="#eff1f4",
command=lambda: c.DMAS_mod_Click('*'))
btn_div = Button(root, text="/", padx=30, pady=15, bd=4, font=myfont, bg="#eff1f4",
command=lambda: c.DMAS_mod_Click('/'))
btn_sign = Button(root, text="+/-", padx=21, pady=15, bd=4, font=myfont, bg="#eff1f4", command=lambda: c.signClick())

btn_equals = Button(root, text="=", padx=30, pady=15, font=myfont, bd=4, bg='orange'
, command=lambda: c.equalpress())

```

#### # Scientific operators

```

btn_reciprocal = Button(root, text='x'+get_super('-
1'), padx=21, pady=15, bd=4, bg="#eff1f4", font=myfont, command=lambda: c.rec_fact_c
lick('r'))
btn_power = Button(root, text='x'+get_super('y'), padx=27, pady=15, bd=4, bg="#eff1f
4", font=myfont, command=lambda: c.pow_pc_clicked('pow'))
btn_root = Button(root, text="√", padx=29, pady=15, bd=4, font=myfont, bg="#eff1f4",
command=lambda: c.trig_log_root_click('r'))
btn_fact = Button(root, text="x!", padx=24, pady=15, bd=4, font=myfont, bg="#eff1f4"
, command=lambda: c.rec_fact_click('!'))

btn_permucombo = Button(root, text="nCr", padx=20, pady=15, bd=4, bg="#eff1f4", font
=myfont, command=lambda: c.pow_pc_clicked('c'))
btn_random = Button(root, text="Ran#", padx=16, pady=15, bd=4, bg="#eff1f4", font=my
font, command=lambda: c.epiRanClick('Ran'))

```

```

btn_mod = Button(root, text="MOD", padx=21, pady=15, bd=4, bg="#eff1f4", font=myfont,
    command=lambda: c.DMAS_mod_Click('mod'))

btn_e = Button(root, text="e", padx=29, pady=15, bd=4, bg="#eff1f4", font=myfont, co
    mmand=lambda: c.epiRanClick('e'))
btn_pi = Button(root, text="π", padx=29, pady=15, bd=4, bg="#eff1f4", font=myfont, c
    ommand=lambda: c.epiRanClick('pi'))

btn_sin = Button(root, text="sin", padx=20, pady=15, bd=4, bg="#eff1f4", font=myfont
    , command=lambda: c.trig_log_root_click('s'))
btn_cos = Button(root, text="cos", padx=20, pady=15, bd=4, bg="#eff1f4", font=myfont
    , command=lambda: c.trig_log_root_click('c'))
btn_tan = Button(root, text="tan", padx=20, pady=15, bd=4, bg="#eff1f4", font=myfont
    , command=lambda: c.trig_log_root_click('t'))

btn_log10 = Button(root, text="log", padx=20, pady=15, bd=4, bg="#eff1f4", font=myfo
    nt, command=lambda: c.trig_log_root_click('g'))
btn_log_e = Button(root, text="ln", padx=26, pady=15, bd=4, bg="#eff1f4", font=myfont
    , command=lambda: c.trig_log_root_click('n'))

btn_2nd = Button(root, text="2nd", padx=20, pady=15, bd=4, bg="#eff1f4", font=myfont
    , command=lambda: c.deg_2nd_click('2nd'))
btn_deg = Button(root, text="deg", padx=20, pady=15, bd=4, bg="#eff1f4", font=myfont
    , command=lambda: c.deg_2nd_click('deg'))

#Reset Btns
btn_ce = Button(root, text="CE", padx=26, pady=15, bd=4, bg="#eff1f4", font=myfont,
    command=lambda: c.CE())
btn_ac = Button(root, text="AC", padx=25, pady=15, bd=4, bg="#eff1f4", font=myfont,
    command=lambda: c.AC())

# Arrange in Grid
btn_2nd.grid(row=1, column=0)
btn_deg.grid(row=1, column=1)
btn_mod.grid(row=1, column=2)
btn_ce.grid(row=1, column=3)
btn_ac.grid(row=1, column=4)

btn_fact.grid(row=2, column=0)
btn_permucombo.grid(row=2, column=1)
btn_random.grid(row=2, column=2)
btn_log10.grid(row=2, column=3)
btn_log_e.grid(row=2, column=4)

btn_root.grid(row=3, column=0)
btn_power.grid(row=3, column=1)
btn_reciprocal.grid(row=3, column=2)
btn_div.grid(row=3, column=3)
btn_sin.grid(row=3, column=4)

btn_7.grid(row=4, column=0)
btn_8.grid(row=4, column=1)
btn_9.grid(row=4, column=2)
btn_mult.grid(row=4, column=3)
btn_cos.grid(row=4, column=4)

```

```
btn_4.grid(row=5, column=0)
btn_5.grid(row=5, column=1)
btn_6.grid(row=5, column=2)
btn_sub.grid(row=5, column=3)
btn_tan.grid(row=5, column=4)

btn_1.grid(row=6, column=0)
btn_2.grid(row=6, column=1)
btn_3.grid(row=6, column=2)
btn_add.grid(row=6, column=3)
btn_pi.grid(row=6, column=4)

btn_sign.grid(row=7, column=0)
btn_0.grid(row=7, column=1)
btn_point.grid(row=7, column=2)
btn_e.grid(row=7, column=3)
btn_equals.grid(row=7, column=4)

root.mainloop()
```