Analogue Clock

# The structure of the program shall be:

1. importing the modules
2. defining the global variables
3. defining the classes
4. the main() function or the program

# We first import the modules and define the global variables:

1. Datetime: It is used for the fast implementation of the datetime type. Herein we use the datetime class of the datetime module.
2. Math: This module provides access to the mathematical functions defined by the C standard. Herein we use the sin() and cos() functions and the inbuilt data allocated to pi.
3. Tkinter: Tkinter is a module that provides classes which allow the display, positioning and control of widgets. The widgets that we intend to use are Tk (a top-level widget) and Canvas.
4. The global variables that we wish to define are:
   * 1. day\_list: the list of days in order
     2. month\_list: the list of months in order

# Beginning with the main() function:

1. The time\_gap is set to -5.5 (in hours), according to the Indian Standard time
2. The width and the height of the screen is set to 500 and 650 respectively.
3. The parent or the root widget that we use would be Tk.
4. We then call the default constructor of the make\_clock class passing the above values as arguments.
5. The mainloop() function to create the widget is run.

­­­­ Class make\_clock:

1. We first define the constructor in which we pass the root widget, the time gap, the width and height of the widget as parameters
2. We initialise the basic attributes of the clock like the colour of the text, the length and colour of the clock needles etc. in the constructor
3. We specify the width and the height of the widget according to the arguments passed to the constructor.
4. The canvas widget is used to create a sub-parent widget on which the elements of the clock are going to be pasted.
5. We define the viewport, the area on the canvas in which the analogue clock is going to be displayed.
6. We create an object of the find\_coords class called convert as an attribute of the class for the easy conversion of the conventional coordinate system (the origin (0,0) being at the centre) to the coordinate system followed by the widget (the origin (0,0) being at the top left corner).
7. We use the pack() function (of canvas) to allow the canvas to be filled in the remaining portion(unspecified) with the background colour.
8. The bind() function (of canvas) calls another function repeatedly on the occurrence of a particular event. In this case on the occurrence of a change in the configuration / size of the window we call the resize() function (user-defined).
9. The resize() function performs the following tasks:
   * 1. Clears all the elements on the canvas.
     2. Reset the width and the height to the present width and the height of the window.
     3. Reinitialise the convert object of the find\_coord class and the gap and viewport of the clock.
10. The constructor then calls the action() function/behaviour of the class.
11. The role of the action() function is to call the create() function infinite no. of times at intervals specified by the value of refresh\_time. This is accomplished by using the after() function of the class Tk(), the root widget.
12. The create() function:
    * 1. First deletes all the elements of the present on the canvas.
      2. Takes the value of year, month, date, hrs, min, sec and the weekday no.
      3. Initialize the month\_name
      4. The start and the step values are set (start being angle corresponding to the position vertically above the centre and the step value being the angular gap between the consecutive digits).
      5. We create the main dial in which the nos. of the clock are in white except the month no. which is replaced by the date.
      6. Dials for the month, day and date are created using the Create\_Dial() function which takes one parameter, either ‘date’, ’day’ or ‘month’ according to the dial that we wish to create.
      7. The Create\_Dial() function:
         + 1. The start and the step values are set (start being angle corresponding to the position vertically above the centre and the step value being the angular gap between the consecutive digits).
           2. The nos. are put in the respective positions and the lines are drawn from their respective centres to the current day/date/month.
           3. The dials are then named at some distance above the centre.
      8. The create\_needles() function is called which is used to create the three needles of the main clock (hr, min, sec).
         + 1. The coordinates of the end points of the needles are calculated based on the angle at which it is supposed to be present.
           2. The lines from the centre to the endpoints are drawn.
           3. A small circle is created at the centre.
      9. Lastly the create\_digital() function is called, that creates the digital clock underneath the analogue clock and also the day\_name on the side of the clock:
         + 1. The day\_name is initialised
           2. A textbox is created on the side of the clock that displays the day\_name.
           3. The digital\_time is initialised using string formatting and the time, date, month and the year.
           4. The title of the Tk() widget is set to the formatted current time.
           5. A textbox is created below the clock that displays the digital\_time.

# Class find\_coords:

1. It has been created for the easy conversion of the conventional coordinate system (the origin (0,0) being at the centre) to the coordinate system followed by the widget (the origin (0,0) being at the top left corner).
2. The constructor takes the values the world (the total canvas) and the viewport (the location in we place the analogue clock) and performs the calculations of changing the coordinate system.
3. The change() function accepts the x, y coordinates and returns the new/changed x, y coordinates as a tuple.
4. The coods() function accepts the x, y coordinates of the start and end points and returns the new/changed x, y coordinates of the start and end points as a tuple of two tuples. It does so by calling the change() function twice (once for each of the points).

######## write the inbuilt functions used #######

# Inbuilt functions used:

>>> help('tkinter.Tk.mainloop')

Help on function mainloop in tkinter.Tk:

tkinter.Tk.mainloop = mainloop(self, n=0)

Call the mainloop of Tk.

>>> help("tkinter.Tk")

Help on class Tk in tkinter:

tkinter.Tk = class Tk(Misc, Wm)

Toplevel widget of Tk which represents mostly the main window of an application. It has an associated Tcl interpreter.

>>> help("datetime.timedelta")

Help on class timedelta in datetime:

datetime.timedelta = class timedelta(builtins.object)

Difference between two datetime values.

>>> help('min')

Help on built-in function min in module builtins:

min(...)

min(iterable, \*[, default=obj, key=func]) -> value

min(arg1, arg2, \*args, \*[, key=func]) -> value

With a single iterable argument, return its smallest item. The default keyword-only argument specifies an object to return if the provided iterable is empty. With two or more arguments, return the smallest argument.

>>> help('tkinter.Canvas')

Help on class Canvas in tkinter:

tkinter.Canvas = class Canvas(Widget, XView, YView)

Canvas widget to display graphical elements like lines or text.

init\_\_(self, master=None, cnf={}, \*\*kw)

Construct a canvas widget with the parent MASTER.

Valid resource names: background, bd, bg, borderwidth, closeenough,

confine, cursor, height, highlightbackground, highlightcolor,

highlightthickness, insertbackground, insertborderwidth,

insertofftime, insertontime, insertwidth, offset, relief,

scrollregion, selectbackground, selectborderwidth, selectforeground,

state, takefocus, width, xscrollcommand, xscrollincrement,

yscrollcommand, yscrollincrement.

>>> help('tkinter.Canvas.pack')

Help on function pack\_configure in tkinter.Canvas:

tkinter.Canvas.pack = pack\_configure(self, cnf={}, \*\*kw)

Pack a widget in the parent widget. Use as options:

after=widget - pack it after you have packed widget

anchor=NSEW (or subset) - position widget according to

given direction

before=widget - pack it before you will pack widget

expand=bool - expand widget if parent size grows

fill=NONE or X or Y or BOTH - fill widget if widget grows

in=master - use master to contain this widget

in\_=master - see 'in' option description

ipadx=amount - add internal padding in x direction

ipady=amount - add internal padding in y direction

padx=amount - add padding in x direction

pady=amount - add padding in y direction

side=TOP or BOTTOM or LEFT or RIGHT - where to add this widget.

>>> help('tkinter.Canvas.bind')

Help on function bind in tkinter.Canvas:

tkinter.Canvas.bind = bind(self, sequence=None, func=None, add=None)

Bind to this widget at event SEQUENCE a call to function FUNC.

SEQUENCE is a string of concatenated event

patterns. An event pattern is of the form

<MODIFIER-MODIFIER-TYPE-DETAIL> where MODIFIER is one

of Control, Mod2, M2, Shift, Mod3, M3, Lock, Mod4, M4,

Button1, B1, Mod5, M5 Button2, B2, Meta, M, Button3,

B3, Alt, Button4, B4, Double, Button5, B5 Triple,

Mod1, M1. TYPE is one of Activate, Enter, Map,

ButtonPress, Button, Expose, Motion, ButtonRelease

FocusIn, MouseWheel, Circulate, FocusOut, Property,

Colormap, Gravity Reparent, Configure, KeyPress, Key,

Unmap, Deactivate, KeyRelease Visibility, Destroy,

Leave and DETAIL is the button number for ButtonPress,

ButtonRelease and DETAIL is the Keysym for KeyPress and

KeyRelease. Examples are

<Control-Button-1> for pressing Control and mouse button 1 or

<Alt-A> for pressing A and the Alt key (KeyPress can be omitted).

An event pattern can also be a virtual event of the form

<<AString>> where AString can be arbitrary. This

event can be generated by event\_generate.

If events are concatenated they must appear shortly

after each other.

FUNC will be called if the event sequence occurs with an

instance of Event as argument. If the return value of FUNC is

"break" no further bound function is invoked.

An additional boolean parameter ADD specifies whether FUNC will

be called additionally to the other bound function or whether

it will replace the previous function.

Bind will return an identifier to allow deletion of the bound function with

unbind without memory leak.

If FUNC or SEQUENCE is omitted the bound function or list

of bound events are returned.

>>> help('tkinter.Canvas.delete')

Help on function delete in tkinter.Canvas:

tkinter.Canvas.delete = delete(self, \*args)

Delete items identified by all tag or ids contained in ARGS.

>>> help('tkinter.Canvas.winfo\_width')

Help on function winfo\_width in tkinter.Canvas:

tkinter.Canvas.winfo\_width = winfo\_width(self)

Return the width of this widget.

>>> help('tkinter.Canvas.winfo\_height')

Help on function winfo\_height in tkinter.Canvas:

tkinter.Canvas.winfo\_height = winfo\_height(self)

Return height of this widget.

>>> help('tkinter.Tk.after')

Help on function after in tkinter.Tk:

tkinter.Tk.after = after(self, ms, func=None, \*args)

Call function once after given time.

MS specifies the time in milliseconds. FUNC gives the

function which shall be called. Additional parameters

are given as parameters to the function call. Return

identifier to cancel scheduling with after\_cancel.

>>> help('datetime.datetime.timetuple')

Help on method\_descriptor in datetime.datetime:

datetime.datetime.timetuple = timetuple(...)

Return time tuple, compatible with time.localtime().

>>> help('math.sin')

Help on built-in function sin in math:

math.sin = sin(...)

sin(x)

Return the sine of x (measured in radians).

>>> help('math.cos')

Help on built-in function cos in math:

math.cos = cos(...)

cos(x)

Return the cosine of x (measured in radians).

>>> help('tkinter.Canvas.create\_text')

Help on function create\_text in tkinter.Canvas:

tkinter.Canvas.create\_text = create\_text(self, \*args, \*\*kw)

Create text with coordinates x1,y1.

>>> help('eval')

Help on built-in function eval in module builtins:

eval(source, globals=None, locals=None, /)

Evaluate the given source in the context of globals and locals.

The source may be a string representing a Python expression

or a code object as returned by compile().

The globals must be a dictionary and locals can be any mapping,

defaulting to the current globals and locals.

If only globals is given, locals defaults to it.

>>> help('tkinter.Canvas.create\_line')

Help on function create\_line in tkinter.Canvas:

tkinter.Canvas.create\_line = create\_line(self, \*args, \*\*kw)

Create line with coordinates x1,y1,...,xn,yn.

>>> help('tkinter.Canvas.create\_oval')

Help on function create\_oval in tkinter.Canvas:

tkinter.Canvas.create\_oval = create\_oval(self, \*args, \*\*kw)

Create oval with coordinates x1,y1,x2,y2.

>>> help('tkinter.Tk.title')

Help on function wm\_title in tkinter.Tk:

tkinter.Tk.title = wm\_title(self, string=None)

Set the title of this widget.

>>> help('str.join')

Help on method\_descriptor in str:

str.join = join(...)

S.join(iterable) -> str

Return a string which is the concatenation of the strings in the

iterable. The separator between elements is S.

# For Reference:

from datetime import timedelta,datetime

from math import sin, cos, pi

from tkinter import \*

day\_list = ['MONDAY','TUESDAY','WEDNESDAY','THURSDAY','FRIDAY','SATURDAY','SUNDAY']

month\_list = ['January','February','March','April','May','June','July','August','September','October','November','December']

class find\_coords:

def \_\_init\_\_(self,world,viewport):

x\_min,y\_min,x\_max,y\_max = world

X\_min,Y\_min,X\_max,Y\_max = viewport

self.delta = min((X\_max-X\_min)/(x\_max-x\_min),(Y\_max-Y\_min)/(y\_max-y\_min))

x\_c,y\_c = 0.5\*(x\_min+x\_max),0.5\*(y\_min+y\_max)

X\_c,Y\_c = 0.5\*(X\_min+X\_max),0.5\*(Y\_min+Y\_max)

self.c\_1 = X\_c-self.delta\*x\_c

self.c\_2 = Y\_c-self.delta\*y\_c

def change(self,x,y):

X\_new,Y\_new = (self.delta\*x+self.c\_1),(self.delta\*(-y)+self.c\_2)

return X\_new,Y\_new

def coords(self,x1,y1,x2,y2):

return self.change(x1,y1),self.change(x2,y2)

class make\_clock:

def \_\_init\_\_(self,root,time\_gap=0,w=1000,h=650):

self.def\_world = [-1,-1,1,1.5]

self.color\_bg = 'black'

self.color\_clock\_details= 'white'

self.color\_date\_at\_month= 'red'

self.color\_main\_needles = 'light blue'

self.color\_inner\_circles= 'blue'

self.color\_inner\_needles= 'red'

self.font\_main\_details = 'Harrington 20'

self.font\_inner\_titles = 'Papyrus 10'

self.center\_circle\_rad = 0.05/2

self.len\_hrs = 0.50

self.len\_min = 0.80

self.len\_sec = 0.90

self.inner\_cl\_no\_font = {'Month':'calibri 9','Day':'calibri 9','Date':'calibri 7 bold'}

self.inner\_cl\_radius = {'Month':0.2,'Day':0.2,'Date':0.35}

self.inner\_cl\_loc = {'Month':(0,0.4),'Day':(-0.5,0),'Date':(0,-0.5)}

self.inner\_cl\_count = {'Month':12,'Day':7,'Date':31}

self.inner\_cl\_width\_needles = {'Month':4,'Day':4,'Date':4}

self.inner\_cl\_detail = {'Month':'self.month','Day':'self.w\_day','Date':'self.date'}

width,height = w,h

self.refresh\_time = 500

self.gap = min(width,height)/16

self.width\_hrs\_hand = (self.gap-10)/3

self.width\_min\_hand = (self.gap-10)/6

self.time\_gap = timedelta(hours = time\_gap)

viewport = (self.gap,self.gap,width-self.gap,height-25)

self.convert = find\_coords(self.def\_world,viewport)

self.root = root

self.canvas = Canvas(root,width=width,height=height,background=self.color\_bg)

self.canvas.pack(fill=BOTH,expand=True)

self.canvas.bind("<Configure>",self.resize)

self.action()

def resize(self,event):

self.canvas.delete(ALL)

width,height = self.canvas.winfo\_width(),self.canvas.winfo\_height()

self.gap = min(width,height)/16

viewport = (self.gap,self.gap,width-self.gap,height-self.gap)

self.convert = find\_coords(self.def\_world,viewport)

def action(self):

self.create()

self.root.after(self.refresh\_time,self.action)

def create(self):

self.canvas.delete(ALL)

self.year,self.month,self.date,self.hrs,self.min,self.sec,self.w\_day,\*extra = datetime.timetuple(datetime.utcnow()-self.time\_gap)

self.month\_name = month\_list[self.month-1]

start,step = pi/2,pi/6

for i in range(12):

angle = start-(i\*step)

x\_coord,y\_coord = cos(angle),sin(angle)

if i==self.month or (i+12)==self.month:

self.canvas.create\_text(self.convert.change(x\_coord,y\_coord),fill=self.color\_date\_at\_month,text=' '.join([str(self.date),self.month\_name[:3],str(self.year)]),font=self.font\_main\_details)

elif i==0:

self.canvas.create\_text(self.convert.change(x\_coord,y\_coord),fill=self.color\_clock\_details,text=str(12),font=self.font\_main\_details)

else:

self.canvas.create\_text(self.convert.change(x\_coord,y\_coord),fill=self.color\_clock\_details,text=str(i),font=self.font\_main\_details)

self.Create\_Dial('Month')

self.Create\_Dial('Day')

self.Create\_Dial('Date')

self.create\_needles()

self.create\_digital()

def Create\_Dial(self,name):

start,step = pi/2,2\*pi/self.inner\_cl\_count[name]

for i in range(self.inner\_cl\_count[name]):

angle = start-(i\*step)

x\_coord,y\_coord = self.inner\_cl\_radius[name]\*cos(angle),self.inner\_cl\_radius[name]\*sin(angle)

self.canvas.create\_text(self.convert.change(x\_coord+self.inner\_cl\_loc[name][0],y\_coord+self.inner\_cl\_loc[name][1]),fill=self.color\_inner\_circles,text=str(i+1),font=self.inner\_cl\_no\_font[name])

angle = start-((eval(self.inner\_cl\_detail[name])-1)\*step)

x\_month,y\_month = (self.inner\_cl\_radius[name]-0.05)\*cos(angle),(self.inner\_cl\_radius[name]-0.05)\*sin(angle)

self.canvas.create\_line(self.convert.coords(self.inner\_cl\_loc[name][0],self.inner\_cl\_loc[name][1],x\_month+self.inner\_cl\_loc[name][0],y\_month+self.inner\_cl\_loc[name][1]),fill=self.color\_inner\_needles,width=self.inner\_cl\_width\_needles[name])

self.canvas.create\_text(self.convert.change(self.inner\_cl\_loc[name][0],self.inner\_cl\_loc[name][1]+0.05),fill=self.color\_clock\_details,text=name,font=self.font\_inner\_titles)

def create\_needles(self):

angle = (pi/2)-(pi/6)\*(self.hrs+(self.min/60.0))

x\_hrs,y\_hrs = cos(angle)\*self.len\_hrs,sin(angle)\*self.len\_hrs

angle = (pi/2)-(pi/30)\*(self.min+(self.sec/60.0))

x\_min,y\_min = cos(angle)\*self.len\_min,sin(angle)\*self.len\_min

angle = (pi/2)-(pi/30)\*self.sec

x\_sec,y\_sec = cos(angle)\*self.len\_sec,sin(angle)\*self.len\_sec

draw\_line = self.canvas.create\_line

draw\_line(self.convert.coords(0,0,x\_hrs,y\_hrs),fill=self.color\_main\_needles,width=self.width\_hrs\_hand)

draw\_line(self.convert.coords(0,0,x\_min,y\_min),fill=self.color\_main\_needles,width=self.width\_min\_hand)

draw\_line(self.convert.coords(0,0,x\_sec,y\_sec),fill=self.color\_main\_needles,arrow='last')

self.canvas.create\_oval(self.convert.coords(-self.center\_circle\_rad,-self.center\_circle\_rad,self.center\_circle\_rad,self.center\_circle\_rad),fill=self.color\_inner\_circles)

def create\_digital(self):

day\_name=day\_list[self.w\_day]

self.canvas.create\_text(self.convert.change(0.5,0),fill=self.color\_clock\_details,text=day\_name,font=self.font\_inner\_titles)

time\_display = '%02i : %02i : %02i'%(self.hrs,self.min,self.sec)

date\_display = ' '.join([str(self.date),self.month\_name,str(self.year)])

digital\_time = time\_display+"\n"+date\_display

self.root.title(time\_display)

self.canvas.create\_text(self.convert.change(0,-1.4),fill=self.color\_clock\_details,text=" "+digital\_time,font=self.font\_main\_details)

time\_gap = -5.5

w,h = 500,650

root = Tk()

make\_clock(root,time\_gap,w,h)

root.mainloop()