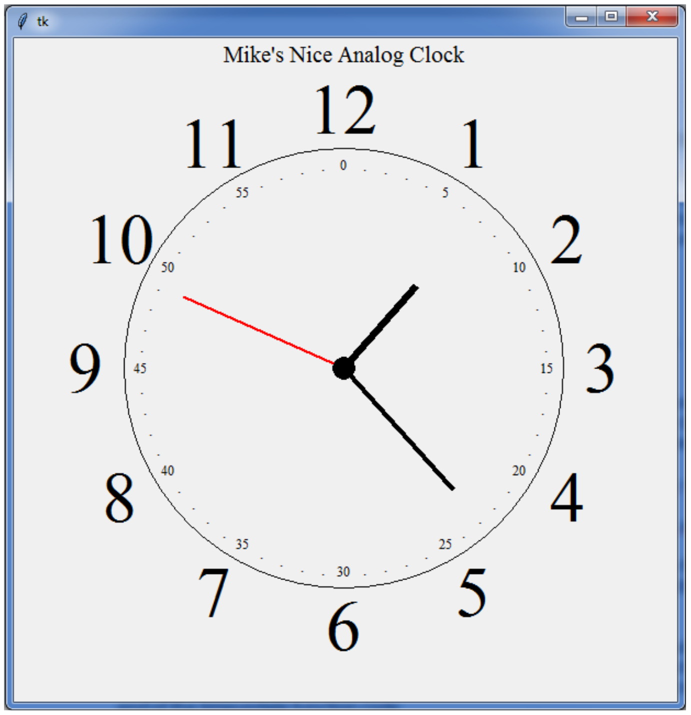
CSC 4800 Python Applications Programming

HW/Lab #8: Analog Clock Tkinter Application Final Project

Due: Wednesday, March 15, 2017, 11:59pm deadline, Submit solution in Canvas

Design and implement a Python Tkinter GUI application that draws a simple 12-hour "analog" clock with (hour, minute, second) hands in the main window. The clock hands must reflect the current time on the PC, and must be updated every second. Text "Numerals" for the hours and some minute markers are required. You are welcome to "fancy up" your clock display, but be sure to show at least these minimum details.



- Your application must use the Tkinter GUI library and graphical Widgets. If you prefer, you may use the Ttk module instead, but it is not required.

- The main root window should be covered with a Canvas widget, approximately 600x600 in size. All graphical output will be drawn on this canvas. The instructor baseline sample uses ovals, lines, and text items, but you are welcome to consider other options.

- You can obtain the current time (24-hour time) by importing the datetime module, e.g.,

```
from datetime import datetime
 t = datetime.now().time()   #  t.hour, t.min, t.sec
```

- The best way to implement a "periodic timer" that you can use to update the time being displayed is to simply force the main root window to repeatedly callback a function after an interval delay. For example, for the main window 'root':

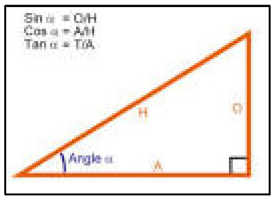
```
root.after(500, timeupdate)
```

  Once the root window has finished drawing itself on the screen, it will wait approximately 500 ms (1/2 second), and then call function timeupdate() which you must provide in your code. Place this 'root.after(…)' line in your code right before the mainloop(), and a second time at the end of the timeupdate function code.

Note that in the timeupdate() function, it is much more accurate to call the datetime functions to obtain the time, rather than trying to "count the seconds" yourself. Using a 1/2 second refresh interval is sufficient to keep a "second hand" updated accurately. Also, using the root.after(…) strategy is more efficient that using a "sleep(…)" strategy.

- The clock hands must originate in the center of the canvas, and point outward towards the correct hour, minute and second times. The length of the clock hands depends on the width and height of the canvas, and it is easiest to work with a square canvas. The hour hand should be shorter than the minute and second hands (for example, the length of the hour hand is about 50% of the clock circle radius, the minute hand is about 75%, etc.). The hands should be drawn using colors or other attributes that distinguish them from one another.

- To determine a hand direction and coordinates, imagine a (invisible) circle centered in the window. If it was, say 13:23:49, then you can use radian-based sin/cos trig functions to calculate the coordinates of where the tip of the hour hand would touch the circle. Remember that a circle has 2 **PI** radians, so a 3-hour (or 15-minute) quadrant has **PI** /2. If you assume that the clock face represents 60 intervals, then it is straightforward to map minutes and seconds on the clock face. For a 12-hour clock, one hour includes 5 of these small intervals, and the minute value is an additional percentage (that is, 30 minutes should position the hour hand half-way between the hour labels). The equations would look something like this (different quadrants will have slightly different equations):



```
from math import  pi, sin, cos

fHr = ( (iHr % 12) * 5) + ((iMin) / 60.0) * 5);  // exact position of hour hand: (13 % 12) == 1pm
int tipx, tipy;

if( fHr <= 15.0 )          // top right quadrant
{
  double angle = ( (15 – fHr) / 15) * (pi / 2.0);
                                                    // 1:23 hour hand is upper right quadrant
  tipx = cos(angle) * lenHourHand);
  tipy = sin(angle) *  lenHourHand);
}
```

note

Then the Hour line coordinates would be:

(centerX, centerY) → (centerX + tipx, centerY – tipy)

that is, to the right of center in the X direction, and above the center in the Y direction.

- Some examples of creating items on a canvas widget:

```
canvas.create_text(x, y, text='Hi There', font=('Times', '16') )
canvas.create_oval(x1, y1, x2, y2, width=1)  #topleft, bottomright coords
canvas.create_line(x1, y1, x2, y2, width=1, fill="red")
```

Lab Turn in:

Submit a ZIP of your source code solution via Canvas by the deadline.
This is the final exam deadline, no extensions are permitted.