# Python Libraries

by

Peter Hedlund

Revised 08/24/21

## Table of Contents

## Directory Structure

All of the library files are stored off the python directory (
i.e. my python is located in c:\user\python37-32)
c:\user\python37-32\Lib\mylib
That directory contains

```
    __init__.py
    about.py
    box_char.py
    clock_face.py
    data_validation.py
    env.py
    file_log.py
    gage.py
    help.py
    knob.py
    led.py
    LED_d.py
    led_sq.py
    low_ascii.py
    my_math.py
    p_types.py
    print_colors.py
    scrn_log.py
    scrl_notebook.py
    seven_seg.py
    sixteen_seg.py
    to_log.py
```

All library files contain stand alone code so that you can run the library file from python and see a demonstration of the features of the routine. This also serves as a sample of how to use the library routines.

## Documentation of Libraries

```
As of 12/12/2020 All Libraries have doc strings to identify the Library (Packages)
and Functions.
The Library doc string will contain a list of all user functions in the library.
i.e. for LED_D.py
```

### print(mylib.led_d.__doc__)

```
LED Library for dual leds (top green if on or bottom red if off).
    make_led_r    Makes 2 round led's On and off
    make_led_s    Makes 2 square led's On and Off
    set_led       Sets the leds on = green / gray  off = gray / red
    get_led       Gets the current state of the led.
```

### print(mylib.led_d.make_led_r.__doc__)

```
Creates 2 round leds one on top of the other.
       The top led is grey for off and green for on.
       The bottom led is grey for on or red for off

ALL libraries contain get_lib_version and get_lib_full_version
The first get the numeric version the second get the library file name and version.
```

# MyLib Library Routines

## __init__py

```
"""
mylib by Peter Hedlund  04/27/2021
This is a collection of modules that i have written and placed into the
mylib directory under c:/user/python????/Lib/mylib directory.

Current modules are
    About ------- Creates an about box of information
    box_char ---- 16 bit unicode characters for on screen boxes
    Clock_face -- Draws a clock face and sllow setting time on it
    env ---------    Reads / Writes enviromental variables (in Registry)
    data_validate Checksum 8, Checksum 16, crc16 and crc16_CCITT Routines
    file_log ----    Logs text string to files
    gage -------- Draws a gage and allows setting of the indicator
    help -------- Opens a Help window containing text from a file
    knob -------- Draws a knob on the screen and allows changing of position
    led_d ------- Draws a dual LED (Red / Green) for on / off type use
    led_sq ------ Draws 1 to 8 square leds and allows on/off control
    led --------- Draws 1 to 8 rount leds and allows on/off control
    low_ascii --- Contains character definitions for ascii 0 - 31
    my_math ----- A collection of math routines
    p_types ----- C style type definitions for python structures
    print_colors  Ansi escape codes to set colors and cursor position of text
    scrl_notebook Scrollable version of the Notebook widget
    scrn_log ---- Replacement for to_log easier to use.
    seven_seg --- Draws seven segment displays
    sixteen_seg - Draws 16 segment display (british fiag) alphanumeric
    to_log ------ Provides logging data to screen widget (Legacy Library)

"""
```

## About.py

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#

""" About Box
get_lib_version() Show numeric version of library.
get_lib_full_version()  Show numeric version and library filename.
about_box()  Displays data from prog_id in an about box.
"""



from tkinter import messagebox

name_about    = 'about.py'
version_about = '2.0.0'
date_about    = '02/05/21'
author_about  = 'Peter A. Hedlund'

LIB_NAME = name_about
LIB_VERSION = version_about
LIB_DATE = date_about
LIB_AUTHOR = author_about


def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION


def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs


def about_box(prog_id):
    """Shows the about box filled with prog_id information. """
    messagebox.showinfo("About " + prog_id['progname'],
                        "Filename : " + prog_id['progname'] + "\n" +
                        "Title : " + prog_id['title'] + "\n" +
                        "Version : " + prog_id['version'] + "\n" +
                    "Creation Date : " + prog_id['date'] + "\n" +
                    "Revision Date : " + prog_id['rev_date'] + "\n" +
                    "Author : " + prog_id['author'] + "\n" +
                    "Description : " + prog_id['description']
                    )


if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk

    def my_about_box(none):
        about_box(prog_id)

    prog_id = {'progname': 'about.py',
               'title': 'Test About Box',
               'version': '1.1',
               'date': "1 February 2018",
               'rev_date': '',
               'author': "Peter Hedlund",
               'description': 'Stand alone test of about box.\n'
              }

    root = Tk()
    #  prevent window resizing.
    root.resizable(0, 0)
    # Replace tk icon with your own.
    root.title(prog_id['title'])

    mainframe = ttk.Frame(root, padding="3", height=100, width=300)
    mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
    mainframe.grid_propagate(0)

    lab = Label(mainframe, text="About", relief=SUNKEN, anchor='center', width=13, bg='#ccffcc')
    lab.grid(column=1, row=0, padx=15, pady=5)
    lab.bind("<Button-1>", my_about_box)

    s = 'Filename : Version [' + get_full_lib_version() + ']\n'
    lab2 = Label(mainframe,text=s)
    lab2.grid(column=0,row=1, columnspan=2)
    for child in mainframe.winfo_children():
        child.grid_configure(padx=5, pady=5)

    root.mainloop()
```

## Box_Char.py

```
# box_char.py

"""Character defnitions for ascii box characters"""
name_box_char    = 'box_char.py'
version_box_char = '1.2.0'
date_box_char    = '04/30/21'
author_box_char  = 'Peter A. Hedlund'

# Single Line box
h_bar_     = '\u2500'   # horizontal bar 1 char wide
v_bar_     = '\u2502'   # Vertical Bar 1 char tall
ul_corner_ = '\u250c'   # upper left corner
ur_corner_ = '\u2510'   # upper right corner
ll_corner_ = '\u2514'   # Lower left corner
lr_corner_ = '\u2518'   # lower right corner
v_bar_tr_  = '\u251c'   # Vertical bar split right
v_bar_tl_  = '\u2524'   # Vertical bar split left
h_bar_td_  = '\u252c'   # Horizontal bar split up
h_bar_tu_  = '\u2534'   # horizontal bar split down
ctr_cross_ = '\u253c'   # center cross (vertical and horizontal bar)

# Double Line Box
dl_h_bar_  = '\u2550'   # horizontal bar 1 char wide
dl_v_bar_  = '\u2551'   # Vertical Bar 1 char tall
dl_ul_c_   = '\u2554'   # upper left corner
dl_ur_c_   = '\u2557'   # upper right corner
dl_ll_c_   = '\u255a'   # Lower left corner
dl_lr_c_   = '\u255d'   # lower right corner
dl_vb_tr_  = '\u2560'   # Vertical bar split right
dl_vb_tl_  = '\u2563'   # Vertical bar split left
dl_hb_tu_  = '\u2569'   # Horizontal bar split up
dl_hb_td_  = '\u2566'   # horizontal bar split down
dl_cross_  = '\u256c'   # center cross (vertical and horizontal bar)
# Double Hozizontal
dh_ul_c_   = '\u2552'   # upper left corner
dh_ur_c_   = '\u2555'   # upper right corner
dh_ll_c_   = '\u2558'   # Lower left corner
dh_lr_c_   = '\u255b'   # lower right corner
dh_vb_tr_  = '\u255e'   # Vertical bar split right
dh_vb_tl_  = '\u2561'   # Vertical bar split left
dh_hb_tu_  = '\u2567'   # Horizontal bar split up
dh_hb_td_  = '\u2564'   # horizontal bar split down
dh_cross_  = '\u256a'   # center cross (vertical and horizontal bar)
# Double Vertical
```

```
dv_ul_c_   = '\u2553'   # upper left corner
dv_ur_c_   = '\u2556'   # upper right corner
dv_ll_c_   = '\u2559'   # Lower left corner
dv_lr_c_   = '\u255c'   # lower right corner
dv_vb_tr_  = '\u255f'   # Vertical bar split right
dv_vb_tl_  = '\u2562'   # Vertical bar split left
dv_hb_tu_  = '\u2568'   # Horizontal bar split up
dv_hb_td_  = '\u2565'   # horizontal bar split down
dv_cross_  = '\u256b'   # center cross (vertical and horizontal bar)

if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk
    import mylib.scrn_log as scrn

    def send_str(txt, wstr):
        txt.insert(END,wstr)
        txt.index(END)
        txt.see(END)
        txt.update()

    root = Tk()
    #  prevent window resizing.
    root.resizable(0, 0)
    # Replace tk icon with your own.
    root.title('Box Characters DEMO')
    mainframe = ttk.Frame(root, padding="3", height=340, width=440)
    mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
    mainframe.grid_propagate(0)

    log = scrn.scrn_log(mainframe, 50, 20, 'black', 'light grey', 0, 0, 2, 10)

    log.log_scrn('Box Char Library Test\n')
    log.log_scrn(f'Small Box  {ul_corner_}{ur_corner_}\n')
    log.log_scrn(f'           {ll_corner_}{lr_corner_}\n')

    log.log_scrn(f'           {ul_corner_}{h_bar_}{ur_corner_}\n')
    log.log_scrn(f'Medium Box {v_bar_}X{v_bar_}\n')
    log.log_scrn(f'           {ll_corner_}{h_bar_}{lr_corner_}\n')

    log.log_scrn(f'           {ul_corner_}{h_bar_td_}{ur_corner_}\n')
    log.log_scrn(f'4 pane Box {v_bar_tr_}{ctr_cross_}{v_bar_tl_}\n')
    log.log_scrn(f'           {ll_corner_}{h_bar_tu_}{lr_corner_}\n')

    log.log_scrn(f'           {ul_corner_}{h_bar_}{h_bar_td_}{h_bar_}{ur_corner_}\n')
    log.log_scrn(f'4 pane Box {v_bar_tr_}{h_bar_}{ctr_cross_}{h_bar_}{v_bar_tl_}\n')
    log.log_scrn(f'          {ll_corner_}{h_bar_}{h_bar_tu_}{h_bar_}{lr_corner_}\n')

    log.log_scrn(f'          {ul_corner_}{h_bar_}{h_bar_td_}{h_bar_}{ur_corner_}\n')
    log.log_scrn(f'4 pane Box {v_bar_}X{v_bar_}X{v_bar_}\n')
    log.log_scrn(f'single     {v_bar_tr_}{h_bar_}{ctr_cross_}{h_bar_}{v_bar_tl_}\n')
    log.log_scrn(f'          {v_bar_}X{v_bar_}X{v_bar_}\n')
    log.log_scrn(f'          {ll_corner_}{h_bar_}{h_bar_tu_}{h_bar_}{lr_corner_}\n')

    log.log_scrn(f'          {dl_ul_c_}{dl_h_bar_}{dl_hb_td_}{dl_h_bar_}{dl_ur_c_}\n')
    log.log_scrn(f'4 pane Box {dl_v_bar_}X{dl_v_bar_}X{dl_v_bar_}\n')
    log.log_scrn(f'double     {dl_vb_tr_}{dl_h_bar_}{dl_cross_}{dl_h_bar_}{dl_vb_tl_}\n')
    log.log_scrn(f'          {dl_v_bar_}X{dl_v_bar_}X{dl_v_bar_}\n')
    log.log_scrn(f'          {dl_ll_c_}{dl_h_bar_}{dl_hb_tu_}{dl_h_bar_}{dl_lr_c_}\n')

    log.log_scrn(f'          {dh_ul_c_}{dl_h_bar_}{dh_hb_td_}{dl_h_bar_}{dh_ur_c_}\n')
    log.log_scrn(f'4 pane Box {v_bar_}X{v_bar_}X{v_bar_}\n')
    log.log_scrn(f'dbl horiz  {dh_vb_tr_}{dl_h_bar_}{dh_cross_}{dl_h_bar_}{dh_vb_tl_}\n')
    log.log_scrn(f'          {v_bar_}X{v_bar_}X{v_bar_}\n')
    log.log_scrn(f'          {dh_ll_c_}{dl_h_bar_}{dh_hb_tu_}{dl_h_bar_}{dh_lr_c_}\n')

    log.log_scrn(f'          {dv_ul_c_}{h_bar_}{dv_hb_td_}{h_bar_}{dv_ur_c_}\n')
    log.log_scrn(f'4 pane Box {dl_v_bar_}X{dl_v_bar_}X{dl_v_bar_}\n')
    log.log_scrn(f'dbl vert   {dv_vb_tr_}{h_bar_}{dv_cross_}{h_bar_}{dv_vb_tl_}\n')
    log.log_scrn(f'          {dl_v_bar_}X{dl_v_bar_}X{dl_v_bar_}\n')
    log.log_scrn(f'          {dv_ll_c_}{h_bar_}{dv_hb_tu_}{h_bar_}{dv_lr_c_}\n')




    for child in root.winfo_children():
        child.grid_configure(padx=5, pady=5)

    root.mainloop()
```

## Clock_Face.py

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#

# #-------------------------------------------------------------------------
# Name:        clock_face
# Purpose:
#
# Author:      PHedlund
#
# Created:     09/09/2020
# Copyright:   (c) PHedlund 2020
#-------------------------------------------------------------------------
""" See my_clock.__doc__ for details about this library.
    get_lib_version() Show numeric version of library.
    get_lib_full_version()  Show numeric version and library filename.
    my_clock() Create the clock face
    show_time(self, h, m) Shows time in Hours and Mins.
    show_full_time(self, h, m, s) Shows time in hours, Mins and secs.
"""

import math
from tkinter import *


class my_clock(object):
    """This graphic function will display a clock on the screen.
        Right now is has minuite hand in black and hour hand in red.
        The red hand moves with the second hand so at 4:55 the hour
        hand is almost pointed at 4 as a mechanical clock would be.

        get_lib_version() Returns version of code

        get_full_lib_version() Returns class name, version and date

        show_time()  Shows Hours, Mins,

        show_full_time()  Shows Hours, Mins, Secs,

        Example Useage

        alt_mtr = {'X1': 30, 'Y1': 30, 'width':150 }
        pc_clock = my_clock(cv, alt_mtr)
        tm = time.localtime(time.time())
        pc_clock.show_full_time(tm.tm_hour, tm.tm_min, tm.tm_sec) """

    name_my_clock    = 'my_clock.py'
    version_my_clock = '1.0.0'
    date_my_clock    = '11/20/20'
    author_my_clock  = 'Peter A. Hedlund'

    LIB_NAME = name_my_clock
    LIB_VERSION = version_my_clock
    LIB_DATE = date_my_clock
    LIB_AUTHOR = author_my_clock

    # LIB_NAME = 'clock_face.py'
    # LIB_VERSION = '1.0.0'
    # LIB_DATE = '11/20/20'
    deg_point = 0.0
    ctrx = 0
    ctry = 0
    _hours = None
    _min = None
    _sec = None


    ##  Class Iniialization Function
    def __init__(self, cnvs, mtr_info):
    #  orgx1, orgy1, orgx2, orgy2, color, angst, angln):
        self.cnvs = cnvs
        self.orgx1 = mtr_info['X1']
        self.orgy1 = mtr_info['Y1']
        self.width = mtr_info['width']

        self.dia = (self.width/2)
        self.h_len = self.dia * .75
        self.m_len = self.dia * .9
        self.s_len = self.dia * .95
        self.ctrx = self.orgx1 + self.dia
        self.ctry = self.orgy1 + self.dia

        self.cnvs.create_oval(self.orgx1, self.orgy1,
            self.orgx1 + self.width, self.orgy1 + self.width, width=3)
        self.draw_hour_ticks(12)
        self.draw_min_ticks(60)
        self.show_time(0, 0)
```

```
## Class External Function
def get_lib_version(self):
    """    Returns version information only. """
    return self.LIB_VERSION


def get_full_lib_version(self):
    """Returns version information and library name."""
    msg = self.get_lib_version()
    rs = 'Library Name : ' + self.LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + self.LIB_DATE
    rs = rs + '\nAuthor : ' + self.LIB_AUTHOR + '\n'
    return rs


## Class Internal Function
def get_x_y(self, angle, radius):
    if angle > 360:
        angle %= 360
    q = int(angle / 90)
    angle = angle % 90
    dia = radius

    x = dia * math.cos(math.radians(angle))
    y = dia * math.sin(math.radians(angle))
    point2 = [[self.ctrx-y, self.ctry+x],[self.ctrx-x, self.ctry-y],
              [self.ctrx+y, self.ctry-x],[self.ctrx+x, self.ctry+y]]
    return point2[q][0], point2[q][1]


## Class Internal Function
def draw_hour_ticks(self, count):
    a_count = 360  / count
    a_org = (360 ) / 2
    incr = (360 - 0) / count

    for x in range(0, count+1):
        angl = a_org + (x * a_count)
        angl %= 360
        x1,y1 = self.get_x_y(angl, self.dia)
        x2,y2 = self.get_x_y(angl, self.dia - 8)
        self.cnvs.create_line(x1, y1, x2, y2, width=2)
        x1,y1 = self.get_x_y(angl, self.dia+15)
        lbl = int(incr * x)
        if x > 0:
            self.cnvs.create_text(x1, y1, text=str(int(lbl/30)))


## Class Internal Function
def draw_min_ticks(self, count):
    a_count = 360  / count
    a_org = (360 ) / 2

    for x in range(0, count+1):
        angl = a_org + (x * a_count)
        angl %= 360
        x1,y1 = self.get_x_y(angl, self.dia)
        x2,y2 = self.get_x_y(angl, self.dia - 5)
        self.cnvs.create_line(x1, y1, x2, y2, width=2)
        x1,y1 = self.get_x_y(angl, self.dia+15)


## Class Internal Function
def draw_angle(self, anl, color, leng):
    x2, y2 = self.get_x_y(anl, self.dia * leng)
    line_id = self.cnvs.create_line(self.ctrx, self.ctry,x2, y2,
        width=2, fill=color, arrow=LAST)
    return(line_id)


## Class External Function
def show_time(self, h, m):
    """ Shows time in Hours and Mins."""
    self.cnvs.delete(self._hours)
    angl = ((h * 30) + (m / 2) + 180) % 360
    self._hours = self.draw_angle(angl, 'red', .75)
    self.cnvs.delete(self._min)
    angl = ((m * 6) + 180) % 360
    self._min = self.draw_angle(angl, 'black', .95)

## Class External Function
def show_full_time(self, h, m, s):
    """ Shows time in hours, Mins and secs."""
    self.cnvs.delete(self._hours)
    angl = ((h * 30) + (m / 2) + 180) % 360
    self._hours = self.draw_angle(angl, 'black', .60)
    self.cnvs.delete(self._min)
    angl = ((m * 6) + 180) % 360
    self._min = self.draw_angle(angl, 'black', .85)
    self.cnvs.delete(self._sec)
```

```
        angl = ((s * 6) + 180) % 360
        self._sec = self.draw_angle(angl, 'red', .95)


#
# Test Program Starts Here
#


if __name__ == "__main__":

    ##  ----------------------------------------------------------------------
    ##  Imports
    ##  ----------------------------------------------------------------------
    import os
    import sys
    import time
    import math
    from tkinter import *
    import tkinter as tk
    from tkinter import ttk
    from tkinter import font
    import winsound
    from configparser import ConfigParser


    ##  ----------------------------------------------------------------------
    ##  Classes
    ##  ----------------------------------------------------------------------


    ##  ----------------------------------------------------------------------
    ##  Controls
    ##  ----------------------------------------------------------------------


    def t_reload_time():
        tm = time.localtime(time.time())
        pc_clock.show_full_time(tm.tm_hour, tm.tm_min, tm.tm_sec)
        my_tm = f'  {tm.tm_hour:02}:{tm.tm_min:02}:{tm.tm_sec:02}'
        lbl.config(text=my_tm)
        root.after(1000, t_reload_time)


    def t_quit_prog():
```

```
    root.destroy()


##  -----------------------------------------------------------------------
##
##  GUI Program Starts Here
##
##  -----------------------------------------------------------------------

root = tk.Tk()
root.geometry("1400x650")
root.geometry('%dx%d+%d+%d' % (210, 270, 10,   10))
root.title("PyClock")

cv = tk.Canvas(root, height="210", width=205, bg='white')
cv.grid(column=0, row=1, columnspan=10)

_font = font.Font(weight='bold')

alt_mtr = {'X1': 30, 'Y1': 30, 'width':150 }
pc_clock = my_clock(cv, alt_mtr)

lbl = tk.Label(root, text='', font = _font)
lbl.grid(column=0, row=4, columnspan=3)

quit = tk.Button(root, text=' Quit ', command=t_quit_prog)
quit.grid(column=2, row=5, padx=1)
print(pc_clock.get_full_lib_version())
t_reload_time()
root.mainloop()
```

## data_validation.py

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#

""" data_validation
get_lib_version() Show numeric version of library.
get_lib_full_version()  Show numeric version and library filename.
checksum_8(pkt) Returns 8 byte checksum of all bytes in pkt
checksum_16(pkt) Returns 16 byte checksum of all bytes in pkt
crc16( st, crc) Given a binary string and starting CRC, Calc a final CRC-16
crc16_CCITT(data : bytearray, offset , length)
    Given a binary string and offset and length Calc a final CRC-16 CCITT

"""



from tkinter import messagebox

name_about    = 'data_validation.py'
version_about = '1.0.0'
date_about    = '07/16/21'
author_about  = 'Peter A. Hedlund'

LIB_NAME = name_about
LIB_VERSION = version_about
LIB_DATE = date_about
LIB_AUTHOR = author_about


def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION


def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs
```

```
INITIAL_DF1 = 0x0000

table = (
0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040 )



def checksum_8(pkt):
    """Returns 8 byte checksum of all bytes in pkt"""
    crc = 0
    for x in range(0, len(pkt)):
        crc += pkt[x]
    crc %= 256
    return crc

def checksum_16(pkt):
    """Returns 16 byte checksum of all bytes in pkt"""
    crc = 0
    for x in range(0, len(pkt)):
        crc += pkt[x]
    crc %= 65536
    return crc

def crc16( st, crc):
    """Given a binary string and starting CRC, Calc a final CRC-16 """
    for ch in st:
        crc = (crc >> 8) ^ table[(crc ^ ch) & 0xFF]
    return crc

def crc16_CCITT(data : bytearray, offset , length):
    """Given a binary string and offset and length Calc a final CRC-16 CCITT """
    if data is None or offset < 0 or offset > len(data)- 1 and offset+length > len(data):
        return 0
    crc = 0xFFFF
    for i in range(0, length):
        crc ^= data[offset + i] << 8
        for j in range(0,8):
            if (crc & 0x8000) > 0:
                crc =(crc << 1) ^ 0x1021
            else:
                crc = crc << 1
    return crc & 0xFFFF

#
# Testing Library Function
#


if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk
    import mylib.scrn_log as scrn

    def to_hex(bya):
        os = ''
        for x in bya:
            os = os + f'0x{x:02X} '
        return os
```

```
def send_str(txt, wstr):
    txt.insert(END,wstr)
    txt.index(END)
    txt.see(END)
    txt.update()

root = Tk()
#  prevent window resizing.
root.resizable(0, 0)
# Replace tk icon with your own.
root.title('Data Validation Test Program DEMO')
mainframe = ttk.Frame(root, padding="3", height=420, width=460)
mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
mainframe.grid_propagate(0)

log = scrn.scrn_log(mainframe, 52, 23, 'black', 'light grey', 0, 0, 2, 10)

ba = (0x02, 0x05, 0x5b, 0x00, 0x5d, 0x03)

log.log_scrn_color('Data Validation Library Test\n', 'green')
cs1 = checksum_8(ba)
cs2 = checksum_16(ba)
cs3 = crc16(ba,0)
cs4 = crc16_CCITT(ba,0, len(ba))

log.log_scrn_color(f'Data Packet {to_hex(ba)}\n\n', 'blue')
log.log_scrn(f'checksum-8 shoud return  0xC2    actual is 0x{cs1:02X}\n')
log.log_scrn(f'checksum-16 shoud return 0x00C2  actual is 0x{cs2:04X}\n')
log.log_scrn(f'       crc16 shoud return 0x57A6  actual is 0x{cs3:04X}\n')
log.log_scrn(f'crc16_CCITT shoud return 0x85A3  actual is 0x{cs4:04X}\n')
log.log_scrn('\n')
ba = (0x02 ,0x02, 0x5B, 0x04, 0x91, 0x62, 0x94, 0x95, 0x94, 0x0F,
  0x50, 0x47, 0x58, 0x34, 0x00, 0x40, 0x00, 0x5D, 0x03 )
cs1 = checksum_8(ba)
cs2 = checksum_16(ba)
cs3 = crc16(ba,0)
cs4 = crc16_CCITT(ba,0, len(ba))
st = f'{to_hex(ba)}\n'
log.log_scrn_color(f'Data Packet\n', 'blue')
log.log_scrn_color(f'{st[0:50]}\n{st[50:100]}\n', 'blue')
log.log_scrn(f'checksum-8 shoud return  0xE5    actual is 0x{cs1:02X}\n')
log.log_scrn(f'checksum-16 shoud return 0x04E5  actual is 0x{cs2:04X}\n')
log.log_scrn(f'       crc16 shoud return 0x4172  actual is 0x{cs3:04X}\n')
log.log_scrn(f'crc16_CCITT shoud return 0xEE66  actual is 0x{cs4:04X}\n')
```

```
for child in root.winfo_children():
    child.grid_configure(padx=5, pady=5)

root.mainloop()
```

## Env.py

```
#-------------------------------------------------------------------------------
# Name:        module1
# Purpose:
#
# Author:      USPEHED
#
# Created:     28/08/2018
# Copyright:   (c) USPEHED 2018
# Licence:     <your licence>
#-------------------------------------------------------------------------------
"""Enviormental Varable Manipulation

get_lib_version() Show numeric version of library.
get_lib_full_version()  Show numeric version and library filename.
set_env()  Set enviromental variable
get_eng()  Get value of enviormental variable.
"""

import winreg

name_env    = 'env.py'
version_env = '2.0.0'
date_env    = '02/05/21'
author_env  = 'Peter A. Hedlund'

LIB_NAME = name_env
LIB_VERSION = version_env
LIB_DATE = date_env
LIB_AUTHOR = author_env


def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION


def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs
```

```
KEY_PATH = winreg.HKEY_LOCAL_MACHINE
REG_PATH = r'SYSTEM\CurrentControlSet\Control\Session Manager\Environment'

def set_env(name, value):
    """Set enviromental variable. """
    try:
        _winreg.CreateKey(KEY_PATH, REG_PATH)
        registry_key = _winreg.OpenKey(KEY_PATH, REG_PATH, 0,
                                _winreg.KEY_WRITE)
        _winreg.SetValueEx(registry_key, name, 0, _winreg.REG_SZ, value)
        _winreg.CloseKey(registry_key)
        return True
    except WindowsError:
        return False

def get_env(name):
    """Get enviromental variable. """
    try:
        registry_key = _winreg.OpenKey(KEY_PATH, REG_PATH, 0,
                                _winreg.KEY_READ)
        value, regtype = _winreg.QueryValueEx(registry_key, name)
        _winreg.CloseKey(registry_key)
        return value
    except WindowsError:
        return None
```

## File_Log.py

```
# pylint: disable=unused-wildcard-import, method-hidden
#
# #-------------------------------------------------------------------------
# Name:        file_log
# Purpose:
#
# Author:      PHedlund
#
# Created:     02/06/2021
# Copyright:   (c) PHedlund 2021
#-------------------------------------------------------------------------
"""    Logging Function Library
    get_lib_version
    get_full_lib_version
    file_log
    log
    log_dt
    header
"""
##  ---------------------------------------------------------------------
##  Imports
##  ---------------------------------------------------------------------
import os, sys
import time
from datetime import datetime

##  ---------------------------------------------------------------------
##  Classes
##  ---------------------------------------------------------------------


class file_log():
    """
        Logging functions
        Logging data to a file

        get_lib_version() Returns version of code

        get_full_lib_version() Returns class name, version and date

        log()  Sends a string out to log file
        log_dt() Sends the date / time and string out to the log file
        header() Sends a header (depending on type and dt_type) out to
            the log file
    ##  INTERNAL FUNCTIONS  ##
    ch_line() Creates len amount of the character ch

    gen_fix_str() Generates a fixed string padded with spaces on the left
    SAMPLE USEAGE

    log1 = file_log(my_fn, fpath = my_path, new_f = 1)

    log1.header(1, 'Writing to new log file 1') # NO CR
    log1.log('Sample data to new log\n')        # CR added
    log1.log_dt('Sample data with date/time\n') # CR added """

name_file_log    = 'file_log.py'
version_file_log = '1.1.0'
date_file_log    = '04/30/21'
author_file_log  = 'Peter A. Hedlund'

LIB_NAME = name_file_log
LIB_VERSION = version_file_log
LIB_DATE = date_file_log
LIB_AUTHOR = author_file_log


def __init__(self, fname, fpath = os.path.dirname(sys.argv[0]), new_f = 1):
    self.fname = fname
    self.fpath = fpath
    self.new_f = new_f
    self.full_file = self.fpath + '/' + self.fname
    if self.new_f == 1:
        fp = open(self.full_file, 'w', encoding='utf-8')
        fp.close()


# Public Function
def get_lib_version(self):
    """    Returns version information only. """
    return self.LIB_VERSION


# Public Function
def get_full_lib_version(self):
    """Returns version information and library name."""
    msg = self.get_lib_version()
    rs = 'Library Name : ' + self.LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + self.LIB_DATE
```

```
        rs = rs + '\nAuthor : ' + self.LIB_AUTHOR + '\n'
        return rs


    def log(self, wstr):
        """ Opens file in append, writes string to file then closes the
        file.  User is responsible for carrage return character \\n """
        fp = open(self.full_file, 'a', encoding='utf-8')
        fp.write(wstr)
        fp.close()


    def log_dt(self, wstr):
        """ Open file in append, writes date / time to the file then the
        passed string wstr. closing the file when done. The user is
        responsible for added a carrage return character \\n """
        now = datetime.now()
        dt = now.strftime('%m/%d/20%y %H:%M:%S ')
        self.log(dt + wstr)


    def ch_line(self, ch, ln):
        """ This routine creates a string of character ch at a given
        length (for header function) """
        ostr = ''
        x = 0
        while x < ln:
            ostr = ostr + ch
            x = x + 1
        return ostr


    def gen_fix_str(self, wstr, ln):
        ad = self.ch_line(' ', ln - len(wstr))
        return wstr + ad


    def header(self, type, dt_type, wstr):
        """ This routine creates a header log entry depending on type,
        0 Simple -= string =-
        1 Double Line box
        2 Single line Box
        3 C Style header
        4 Python Style Header
        dt_type
        0 NONE
```

```
1 Date
2 Time
3 Date and time
This routine provies the needed carrage returns."""
now = datetime.now()
mx_len = len(wstr)
if mx_len < 10:
    mx_len = 10
s1 = self.gen_fix_str(wstr, mx_len)
if (dt_type & 1) == 1 :
    ts = now.strftime('%H:%M:%S')
    s2 = self.gen_fix_str( ts, mx_len)
if (dt_type & 2) ==  2:
    ds = now.strftime('%m/%d/20%y')
    s3 = self.gen_fix_str(ds, mx_len)

if type == 1:  #  Double Line Box
    bl = self.ch_line('=', mx_len + 4)
    self.log(bl + '\n')
    self.log('| ' + s1 + ' |\n')
    if (dt_type & 1) == 1:
        self.log('| ' + s2 + ' |\n' )
    if (dt_type & 2) == 2:
        self.log('| ' + s3 + ' |\n' )
    self.log(bl + '\n')
elif type == 2:  #  Single Line Box
    bl = self.ch_line('-', len(wstr) + 2)
    self.log('+' + bl + '+\n')
    self.log('| ' + wstr + ' |\n')
    if (dt_type & 1) == 1:
        self.log('| ' + s2 + ' |\n' )
    if (dt_type & 2) == 2:
        self.log('| ' + s3 + ' |\n' )
    self.log('+' + bl + '+\n')
elif type == 3:  #  C style Header
    bl = self.ch_line('-', len(wstr) + 2)
    self.log('// ' + bl + '\n')
    self.log('//  ' + wstr + '\n')
    if (dt_type & 1) == 1:
        self.log('// ' + s2 + '\n' )
    if (dt_type & 2) == 2:
        self.log('// ' + s3 + '\n' )
    self.log('// ' + bl + '\n')
elif type == 4:  #  Python Style Header
    bl = self.ch_line('-', len(wstr) + 2)
```

```
            self.log('# ' + bl + '\n')
            self.log('#  ' + wstr + '\n')
            if (dt_type & 1) == 1:
                self.log('# ' + s2 + '\n' )
            if (dt_type & 2) == 2:
                self.log('# ' + s3 + '\n' )
            self.log('# ' + bl + '\n')
        else:  #  Default Header Style (Sweet and simple)
            self.log('-= ' + wstr + ' =-\n')
            if (dt_type & 1) == 1:
                self.log('-= ' + s2 + ' =-\n' )
            if (dt_type & 2) == 2:
                self.log('-= ' + s3 + ' =-\n' )


if __name__ == "__main__":

    from tkinter import *
    import tkinter as tk
    import mylib.scrn_log

##  ------------------------------------------------------------------------
##  Controls
##  ------------------------------------------------------------------------

    def quit_prog():
        global root
        root.destroy()

    def btn_create():
        log1.header(0, 0, 'Log to new log file 1 Header 0 ,0')
        log1.header(1, 0, 'Log to new log file 1 Header 1, 0')
        log1.header(2, 0, 'Log to new log file 1 Header 2, 0')
        log1.header(3, 0, 'Log to new log file 1 Header 3, 0')
        log1.header(4, 0, 'Log to new log file 1 Header 4, 0')
        log1.header(1, 0, 'Header Big type 1, 0')
        log1.header(1, 1, 'Header Big type 1, 1')
        log1.header(1, 2, 'Header Big type 1, 2')
        log1.header(1, 3, 'Header Big type 1, 3')
        log1.header(2, 0, 'Header Big type 2, 0')
        log1.header(2, 1, 'Header Big type 2, 1')
        log1.header(2, 2, 'Header Big type 2, 2')
        log1.header(2, 3, 'Header Big type 2, 3')
        log1.log('Sample data to new log\n')
        log1.log_dt('Date/Time Sample\n')
        mylog.log_scrn('Created log file\n')
```

```
def btn_append():
    log2.header(0, 0, 'Log to existing log file 2 Header 0, 0')
    log2.header(0, 1, 'Log to existing log file 2 Header 0, 1')
    log2.header(0, 2, 'Log to existing log file 2 Header 0, 2')
    log2.header(0, 3, 'Log to existing log file 2 Header 0, 3')
    log2.header(2, 0, 'Log to existing log file 2 Header 2, 0')
    log2.log('Sample data to existing log\n')
    log2.log_dt('Date/Time Sample\n')
    mylog.log_scrn('appended log file\n')

def btn_clear():
    mylog.clear_scrn()

def btn_view():
    fp = open(mfn, 'r')
    while True:
        line = fp.readline()
        if not line:
            break
        mylog.log_scrn(line)
    fp.close()


my_fn = 'test.log'
my_path = 'n:/store/python/converted/scrap'
mfn = my_path + '/' + my_fn
root = tk.Tk()
root.geometry("1400x650")
root.geometry('%dx%d+%d+%d' % (520, 600, 20,   20))

root.title("Logging Demonstration")

log1 = file_log(my_fn, fpath = my_path, new_f = 1)
log2 = file_log(my_fn, fpath = my_path, new_f = 0)
brow = 0
btn1 = tk.Button(root, text='Create Log File', command=btn_create)
btn1.grid(column=0, row=brow)
brow += 1
btn2 = tk.Button(root, text='Append Log File', command=btn_append)
btn2.grid(column=0, row=brow)
brow += 1
btn3 = tk.Button(root, text='View Log file', command=btn_view)
btn3.grid(column=0, row=brow)
```

```
brow += 1
btn4 = tk.Button(root, text='Clear Window', command=btn_clear)
btn4.grid(column=0, row=brow)
brow += 1

btn5 = tk.Button(root, text='Quit', command=quit_prog)
btn5.grid(column=0, row=brow)
# Text box Widget
mylog = mylib.scrn_log.scrn_log(root, 45, 34, 'lightgreen', 'black', 2, 0, 6, 20)


for child in root.winfo_children():
    child.grid_configure(padx=5, pady=5)

root.mainloop()
```

## Gage.py

```
# pylint: disable=unused-wildcard-import, method-hidden
#
# #-------------------------------------------------------------------------
# Name:       gage
# Purpose:
#
# Author:     PHedlund
#
# Created:    01/06/2021
# Copyright:  (c) PHedlund 2021
#-------------------------------------------------------------------------
""" Gage Function Library
    get_lib_version
    get_full_lib_version
    gage
    gage.draw_value
"""
##  ---------------------------------------------------------------------
##  Imports
##  ---------------------------------------------------------------------
import math
from tkinter import *
import tkinter as tk


##  ---------------------------------------------------------------------
##  Classes
##  ---------------------------------------------------------------------


class gage(object):
    """ Graphic Guage, this will sit on a existing canvas cnvs
        orgx and orgy are the starting points
        widx, and widy are the width
        color is the color of the guage
        angst is the starting angle (0 is at the bottom center)
        angln is the number of degrees the guage is drawn to

        get_lib_version() Returns version of code

        get_full_lib_version() Returns class name, version and date

        draw_value() will calculate the angle based on set_range()
        then call draw_angle() to draw the pointer.

    ##  INTERNAL FUNCTIONS  ##

    get_x_y() given the angle and diameter returns the x, y point
    from center to draw a line.

    draw_ticks() will place ticks from outside arc to inside arc
    equidistant based on count

    draw_angle(angle,color) draws line from center to outside arc
        angle 0 is pointed straight down, using color provided

    clear_line()  erases the last line drawn with draw_angle()

    set_range() will calculate the angles per count given max and min
    values.

    set_label() will place a text label at the bottom center of the gage

    To Simplify the creation and management of the gauges, i have decided
    to use a dictionary to set all inital values to the gauge

    SAMPLE USEAGE
    # assume cv is the canvas object and you are displaying
      an altitude gauge

    alt_mtr = {'X1': 50, 'Y1': 50, 'X2':150, 'Y2':150,
       'color': 'black', 'angl_st': 40 , 'angl_ln' : 280,
       'rng_lo':0, 'rng_hi': 1000,
       'title': 'Altitude', 'num_ticks': 10}

    g = guage(cv, alt_mtr)
    g.draw_value(100, 'red')  """

name_gage    = 'gage.py'
version_gage = '1.0.0'
date_gage    = '07/15/20'
author_gage  = 'Peter A. Hedlund'

LIB_NAME = name_gage
LIB_VERSION = version_gage
LIB_DATE = date_gage
LIB_AUTHOR = author_gage

# LIB_NAME = 'gage.py'
# LIB_VERSION = '1.0.0'
# LIB_DATE = '7/15/20'
line_id = None
deg_point = 0.0
min_r = 0
max_r = 0
ctrx = 0
ctry = 0
text_id = None


def __init__(self, cnvs, mtr_info):
#  orgx1, orgy1, orgx2, orgy2, color, angst, angln):
    self.cnvs = cnvs
    self.orgx1 = mtr_info['X1']
    self.orgy1 = mtr_info['Y1']
    self.orgx2 = mtr_info['X2']
    self.orgy2 = mtr_info['Y2']
    self.color = mtr_info['color']
    self.angst = mtr_info['angl_st']
    self.angln = mtr_info['angl_ln']
    self.rng_st = mtr_info['rng_lo']
    self.rng_len = mtr_info['rng_hi']
    self.title = mtr_info['title']
    self.ticks = mtr_info['num_ticks']

    self.dia = (self.orgx2 - self.orgx1)/2
    self.ctrx = ((self.orgx2-self.orgx1)/2)+self.orgx1
    self.ctry = ((self.orgy2-self.orgy1)/2)+self.orgy1
    self.cnvs.create_arc(self.orgx1, self.orgy1,
        self.orgx2, self.orgy2,
        start=270 + self.angst, extent=self.angln,
        width=3, style =ARC)
    self.cnvs.create_arc(self.orgx1+10, self.orgy1+10,
        self.orgx2-10, self.orgy2-10,
        start=270 + self.angst, extent=self.angln,
        width=3, style =ARC)
    self.cnvs.create_rectangle(self.orgx1 - 30, self.orgy1 - 25,
                               self.orgx2 + 30, self.orgy2 + 20, width=3)
    self.set_range(self.rng_st,self.rng_len)
    self.set_label(self.title)
    self.draw_ticks(self.ticks)


# Public Function
```

```
def get_lib_version(self):
    """    Returns version information only. """
    return self.LIB_VERSION


def get_full_lib_version(self):
    """Returns version information and library name."""
    msg = self.get_lib_version()
    rs = 'Library Name : ' + self.LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + self.LIB_DATE
    rs = rs + '\nAuthor : ' + self.LIB_AUTHOR + '\n'
    return rs


def get_x_y(self, angle, radius):
    if angle > 360:
        angle %= 360
    q = int(angle / 90)
    angle = angle % 90

    dia = self.ctrx - self.orgx1
    # dia2 = self.ctry - self.orgy1
    dia = radius

    x = dia * math.cos(math.radians(angle))
    y = dia * math.sin(math.radians(angle))
    point2 = [[self.ctrx-y, self.ctry+x],[self.ctrx-x, self.ctry-y],
              [self.ctrx+y, self.ctry-x],[self.ctrx+x, self.ctry+y]]
    return point2[q][0], point2[q][1]


def draw_ticks(self, count):
    a_count = self.angln / count
    a_org = (360 - self.angln) / 2
    incr = (self.max_r - self.min_r) / count

    for x in range(0, count+1):
        angl = a_org + (x * a_count)
        x1,y1 = self.get_x_y(angl, self.dia)
        x2,y2 = self.get_x_y(angl, self.dia - 10)
        self.cnvs.create_line(x1, y1, x2, y2, width=2)
        x1,y1 = self.get_x_y(angl, self.dia+15)
        lbl = int(self.min_r + (incr * x))
        self.cnvs.create_text(x1, y1, text=str(lbl))
```

```
   def draw_angle(self, anl, color):
      x2, y2 = self.get_x_y(anl, self.dia)
      self.line_id = self.cnvs.create_line(self.ctrx, self.ctry,x2, y2,
         width=2, fill=color, arrow=LAST)


   def clear_line(self):
      self.cnvs.delete(self.line_id)


   def set_range(self, min_rng, max_rng):
      self.min_r = min_rng
      self.max_r = max_rng
      self.deg_point = (max_rng - min_rng) / self.angln


   def set_label(self, text2):
      self.cnvs.create_text(self.ctrx,self.orgy2,text=text2)

   # Public Function
   def draw_value(self, value, color):
      """draw_value() will calculate the angle based on set_range()
      then call draw_angle() to draw the pointer. with selected color."""

      if self.deg_point == 0.0:
         return
      v = (value - self.min_r) / self.deg_point
      self.clear_line()
      st = (360 - self.angln) / 2
      self.draw_angle(v + st, color)
      self.cnvs.delete(self.text_id)
      self.text_id = self.cnvs.create_text(self.ctrx, self.orgy2-15,
         text=str(value))



if __name__ == "__main__":
   from tkinter import *
   from tkinter import ttk

##  -----------------------------------------------------------------------
##  Controls
##  -----------------------------------------------------------------------

   def quit_prog():
    global root
    root.destroy()


def upd_fuel(_f_):
    fuel.draw_value(int(_f_), 'red')


def upd_speed(_s_):
    speed.draw_value(int(_s_), 'red')


root = tk.Tk()
root.geometry("1400x650")
root.geometry('%dx%d+%d+%d' % (380, 250, 20,   20))

root.title("Gage Demonstration")

cv = tk.Canvas(root, height="180", width=380, bg='white')

cv.grid(column=0, row=1, columnspan=10)

speed_mtr = {'X1': 50, 'Y1': 50, 'X2':150, 'Y2':150,
         'color': 'black', 'angl_st': 40 , 'angl_ln' : 280,
         'rng_lo':-100, 'rng_hi': 100,
         'title': 'Speed', 'num_ticks': 10}
speed = gage(cv, speed_mtr)
speed.draw_value(0, 'red')

fuel_mtr = {'X1': 230, 'Y1': 50, 'X2':330, 'Y2':150,
         'color': 'black', 'angl_st': 40 , 'angl_ln' : 280,
         'rng_lo':0, 'rng_hi': 2000,
         'title': 'Fuel', 'num_ticks': 10}
fuel = gage(cv, fuel_mtr)
fuel.draw_value(0, 'red')

pdx = 5
sl1 = tk.Scale(root, label='Speed',from_=-100, to=100,orient=tk.HORIZONTAL,
         command=upd_speed)
sl1.grid(column=0, row=2, padx=pdx)
sl1 = tk.Scale(root, label='Fuel',from_=0, to=2000,orient=tk.HORIZONTAL,
         command=upd_fuel)
sl1.grid(column=1, row=2, padx=pdx)
sdirh = tk.Button(root, text='  Quit  ', command=quit_prog, bg='saddlebrown',
    fg='yellow')
sdirh.grid(column=2, row=2, padx=pdx)
```

```
root.mainloop()
```

## Help.py

```
#-------------------------------------------------------------------------------
# Name:        Help.py
# Purpose:
#
# Author:      USPEHED
#
# Created:     25/07/2018
# Copyright:   (c) USPEHED 2018
# Licence:     <your licence>
#-------------------------------------------------------------------------------

#
# pylint: disable=unused-wildcard-import, method-hidden
#

"""Reads a text file and displays it in a toplevel window.
    get_lib_version() Show numeric version of library.
    get_lib_full_version()  Show numeric version and library filename.
    example:
    Dialog(mainframe, title='TEST HELP', filename = 'help.demo')
"""

from tkinter import *
from tkinter import ttk

import os

name_help    = 'help.py'
version_help = '2.0.0'
date_help    = '02/05/21'
author_help  = 'Peter A. Hedlund'

LIB_NAME = name_help
LIB_VERSION = version_help
LIB_DATE = date_help
LIB_AUTHOR = author_help


def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION


def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs


class Dialog(Toplevel):
    """This is the called routine  see example below for useage. """
    def __init__(self, parent, title = None, filename = None):

        Toplevel.__init__(self, parent)
        self.transient(parent)

        if title:
            self.title(title)
        if filename == None:
            filename = 'read.me'

        self.parent = parent

        self.result = None

        body = Frame(self)
        self.initial_focus = self.body(body, filename)
        body.pack(padx=5, pady=5)

        self.buttonbox()

        self.grab_set()

        if not self.initial_focus:
            self.initial_focus = self

        self.protocol("WM_DELETE_WINDOW", self.ok)

        self.geometry("+%d+%d" % (parent.winfo_rootx()+50,
                                  parent.winfo_rooty()+50))

        self.initial_focus.focus_set()

        self.wait_window(self)

    #
```

```
# construction hooks

def body(self, master, fn):
    # create dialog body.  return widget that should have
    # initial focus.  this method should be overridden
    txt1 = Text(master, width=60, height=34, fg='lightgreen', bg='black', padx=15, pady=15)
    txt1.grid(column=2, row=0, columnspan=6, rowspan=20, sticky="nsew")
    txt1.insert(1.0, "Help File\n")

    with open(fn, 'rb') as f:
        txt1.insert(END, f.read())

    #  Scrollbar linked to text box above
    scrollb = ttk.Scrollbar(master, command=txt1.yview)
    scrollb.grid(row=0, column=7, rowspan=20, sticky='nse')
    txt1['yscrollcommand'] = scrollb.set


def buttonbox(self):
    # add standard button box. override if you don't want the
    # standard buttons

    box = Frame(self)

    w = Button(box, text="OK", width=10, command=self.ok, default=ACTIVE)
    w.pack(side=LEFT, padx=5, pady=5)

    self.bind("<Return>", self.ok)

    box.pack()

#
# standard button semantics

def ok(self, event=None):

    if not self.validate():
        self.initial_focus.focus_set() # put focus back
        return

    self.withdraw()
    self.update_idletasks()

    self.apply()
    self.parent.focus_set()
    self.destroy()
    #
    # command hooks

    def validate(self):

        return 1 # override

    def apply(self):

        pass # override



if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk

    def call_help():
        Dialog(mainframe, title='TEST HELP', filename = 'help.demo')

    prog_id = {'progname': 'help.py',
               'title': 'help box',
               'version': '1.0',
               'date': "31 July 2018",
               'rev_date': '',
               'author': "Peter Hedlund",
               'description': 'Stand alone test of help box.\n'
              }

    root = Tk()
    #  prevent window resizing.
    root.resizable(0, 0)
    # Replace tk icon with your own.
    root.title(prog_id['title'])

    mainframe = ttk.Frame(root, padding="3", height=100, width=300)
    mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
    mainframe.grid_propagate(0)
    mainframe.tk.call('tk', 'scaling', 1.2)

    btn = ttk.Button(root, text='Help Me', command=call_help)
    btn.grid(column=1, row=0)
    s = 'Filename : Version [' + get_full_lib_version() + ']\n'
```

```
lab2 = Label(mainframe,text=s)
lab2.grid(column=0,row=3, columnspan=2)

for child in mainframe.winfo_children():
    child.grid_configure(padx=5, pady=5)

root.mainloop()
```

## Knob

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#
""" Knob.py
This is a knob indicator widget. It can be read and written to. The knob
is user scalable as well as user colorable.
knob(cnvs, mtr_info)
get_lib_version()
get_full_lib_version()
show_angle(angl)
draw_value(val)
get_knob() returns current position
"""

# #-------------------------------------------------------------------------
# Name:        knob
# Purpose:     Draw a knob on the canvas that will hopefully be animated
#              by the mouse.
#
# Author:      PHedlund
#
# Created:     09/09/2020
# Copyright:   (c) PHedlund 2020
#-------------------------------------------------------------------------
""" See knob.__doc__ for details about this library. """

import math
from tkinter import *


class knob(object):
    """This graphic function will display a knob on the screen.
        This function uses a dictionary to define knob parameters
            'X1'      Canvas X pos
            'Y1'      Canvas Y pos
            'width'   Width of knob (height and witdh the same)
            'mn_angl' Minimum Angle of adjustment
            'ticks'   Number of ticks to be shown
            'MinVal': Minimum value for knob
            'MaxVal': Maximum value for knob
            'b_color' Knobs base color
            't_color' Knobs top center color
            'p_color' knobs pointer color
        get_lib_version() Returns version of code
        get_full_lib_version() Returns class name, version and date
        show_angle() draws new line at angle and color.
        draw_value() Draws the value converted to angle.
        get_knob()   returns the value of the knob in scale."""

    name_knob    = 'knob.py'
    version_knob = '1.0.0'
    date_knob    = '01/07/21'
    author_knob  = 'Peter A. Hedlund'

    LIB_NAME = name_knob
    LIB_VERSION = version_knob
    LIB_DATE = date_knob
    LIB_AUTHOR = author_knob

    # LIB_NAME = 'knob.py'
    # LIB_VERSION = '1.0.0'
    # LIB_DATE = '01/07/21'

    value = 0
    txt_id = 0


    ##  Class Iniialization Function
    def __init__(self, cnvs, mtr_info):

        self.cnvs = cnvs
        self.orgx1 = mtr_info['X1']
        self.orgy1 = mtr_info['Y1']
        self.width = mtr_info['width']
        self.value = 0
        self.pointer = 0
        self.txt_id = 0
        self.mn_angl = mtr_info['mn_angl']
        self.ticks = mtr_info['ticks']
        self.mnval = mtr_info['MinVal']
        self.mxval = mtr_info['MaxVal']
        self.base_c = mtr_info['b_color']
        self.top_c = mtr_info['t_color']
        self.ptr_c = mtr_info['p_color']
        self.dia = (self.width/2)
        # self.h_len = self.dia * .75
        # self.m_len = self.dia * .9
        # self.s_len = self.dia * .95
    self.ctrx = self.orgx1 + self.dia + 10
    self.ctry = self.orgy1 + self.dia + 10
    # outside ring
    self.cnvs.create_oval(self.orgx1+10, self.orgy1+10,
        self.orgx1 + self.width+10, self.orgy1 + self.width+10, width=3,
        fill=self.base_c)
    # inside ring
    cng = self.width / 3
    self.cnvs.create_oval(self.orgx1 + cng+10, self.orgy1 + cng+10,
        self.orgx1 - cng+10 + self.width, self.orgy1 - cng+10 + self.width,
        width=3, fill=self.top_c)
    self.draw_ticks(self.mn_angl,self.ticks)
    # self.value = self.show_angle(0, 'red')


##  Class External Function
def get_lib_version(self):
    """    Returns version information only. """
    return self.LIB_VERSION


def get_full_lib_version(self):
    """Returns version information and library name."""
    msg = self.get_lib_version()
    rs = 'Library Name : ' + self.LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + self.LIB_DATE
    rs = rs + '\nAuthor : ' + self.LIB_AUTHOR + '\n'
    return rs


##  Class Internal Function
def get_x_y(self, angle, radius):
    """ Returns the x and y points based on angle and radius."""
    if angle > 360:
        angle %= 360
    q = int(angle / 90)
    angle = angle % 90
    dia = radius

    x = dia * math.cos(math.radians(angle))
    y = dia * math.sin(math.radians(angle))
    point2 = [[self.ctrx-y, self.ctry+x],[self.ctrx-x, self.ctry-y],
              [self.ctrx+y, self.ctry-x],[self.ctrx+x, self.ctry+y]]
    return point2[q][0], point2[q][1]
```

```
## Class Internal Function
def draw_ticks(self, start, count):
    """Draws the count number of ticks starting at angle start."""
    a_count = (360 - (start * 2))  / (count)
    a_org = start
    wval = (self.mxval - self.mnval) / count
    for x in range(0, count+1):
        angl = a_org + (x * a_count)
        angl %= 360
        x1,y1 = self.get_x_y(angl, self.dia + 2)
        x2,y2 = self.get_x_y(angl, self.dia + 8)
        self.cnvs.create_line(x1, y1, x2, y2, width=2)
        x2,y2 = self.get_x_y(angl, self.dia + 15)
        self.cnvs.create_text(x2, y2, text=str(int(wval * x)))


## Class Internal Function
def draw_angle(self, anl, leng):
    """ Draws a line at defined angle and length in  defined color."""
    x1, y1 = self.get_x_y(anl, self.dia * (leng*.43))
    x2, y2 = self.get_x_y(anl, self.dia * leng)
    line_id = self.cnvs.create_line(x1, y1,x2, y2,
        width=2, fill=self.ptr_c, arrow=LAST)
    return(line_id)


## Class Exnternal Function
def show_angle(self, angl):
    """ Delets existing line and creates a new line at angle."""
    self.cnvs.delete(self.value)
    self.value = self.draw_angle(angl, .95)


## Class Internal Function
def show_number(self, val):
    """Displays a number at the bottom center of the knob"""
    self.cnvs.delete(self.txt_id)
    x1 = self.ctrx
    y1 = self.ctry + 12 + (self.width / 2)
    self.txt_id = self.cnvs.create_text(x1, y1, text=str(val))


## Class Exnternal Function
def draw_value(self, value):
    """ Draws the pointer with new value on the knob."""
```

```
        angle = re_scale(self.mnval, self.mxval, 360-(self.mn_angl * 2),
        value ) + self.mn_angl
        # angle = int(val)
        self.pointer = value
        self.show_angle(angle)
        self.show_number(value)


    def get_knob(self):
        """ Returns the current value of the knob."""
        return self.pointer


#
# Test Program Starts Here
#


if __name__ == "__main__":

    ##  ----------------------------------------------------------------------
    ##  Imports
    ##  ----------------------------------------------------------------------
    import os
    import sys
    import time
    import math
    from tkinter import *
    import tkinter as tk
    from tkinter import ttk
    from tkinter import font

    from mylib.my_math import *

    ##  ----------------------------------------------------------------------
    ##  Classes
    ##  ----------------------------------------------------------------------


    ##  ----------------------------------------------------------------------
    ##  Controls
    ##  ----------------------------------------------------------------------
```

```
def adj_knob(val):
    volume.draw_value(int(val))
    # print(volume.get_knob())

def t_quit_prog():
    root.destroy()


##  ----------------------------------------------------------------------
##
##  GUI Program Starts Here
##
##  ----------------------------------------------------------------------

root = tk.Tk()
root.geometry("1400x650")
root.geometry('%dx%d+%d+%d' % (170, 240, 10,   10))
root.title("Knob")

cv = tk.Canvas(root, height="160", width=160, bg='white')
cv.grid(column=0, row=1, columnspan=10)
my_angle = IntVar()
direct = IntVar()
direct.set(0)
my_angle.set(0)
_font = font.Font(weight='bold')

# alt_mtr = {'X1': 30, 'Y1': 30, 'width':150 }
alt_mtr = {'X1': 20, 'Y1': 20, 'width':100 , 'mn_angl':30, 'ticks':10,
           'MinVal':0, 'MaxVal':1000, 'b_color':'azure',
           't_color': 'tomato', 'p_color':'black' }
volume = knob(cv, alt_mtr)

lbl = tk.Label(root, text='', font = _font)
lbl.grid(column=0, row=4, columnspan=3)
scl = tk.Scale(root, label='Value', from_=0, to=1000, command=adj_knob,
               orient=tk.HORIZONTAL)
scl.grid(column=0, row=2, padx=1)
quit = tk.Button(root, text=' Quit ', command=t_quit_prog)
quit.grid(column=2, row=2, padx=1)
adj_knob(0)
root.mainloop()
```

## Led.py

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#

""" LED library for round leds, 1, 2, 4 and 8 wide
    Single LED ONLY
    get_lib_version() Show numeric version of library.
    get_lib_full_version()  Show numeric version and library filename.
    make_led_1r   Makes a single led with a text label on the side
    set_led_1r    Sets the led to a color
    get_led_1r    Gets the current color of the led
    BANK LEDS ONLY
    make_led_2r  creates a bank of leds
    make_led_4r
    make_led_8r
    led2num_2r    Sets the leds in the bank the binary values
    led2num_4r          colors are light grey off and red on
    led2num_8r
    set_leds_r    sets the led to a color at an index
    get_leds_r    Gets the color of the led at an indes
"""

from tkinter import *
import tkinter as tk
name_led    = 'led.py'
version_led = '2.0.0'
date_led    = '02/05/21'
author_led  = 'Peter A. Hedlund'

LIB_NAME = name_led
LIB_VERSION = version_led
LIB_DATE = date_led
LIB_AUTHOR = author_led

# LIB_NAME = 'led.py'
# LIB_VERSION = "2.0.0"
# LIB_DATE = '02/05/21'


def get_lib_version():
    """Returns version information only."""
    return LIB_VERSION
```

```
def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs


def make_led_1r(mf, label):
    """Creates a single round led widget with a label next to it. """
    lsize = 18

    scratch = Canvas()
    id = scratch.create_text((0, 0), text=label,anchor=tk.W)
    size = scratch.bbox(id)
    # size is a tuple: (x1, y1, x2, y2)
    # since x1 and y1 will be 0, x2 and y2 give the string width and height

    tw = size[2]-size[0]
    w = Canvas(mf,
               width=lsize + tw + 2,
               height=lsize)
    y = int(lsize) - 2
    led = w.create_oval(2, 2, y-2, y-2, fill='lightgrey')
    w.create_text(18, 8, text=label,anchor=tk.W)
    return w, led


def set_led_1r(w, led, color):
    """Sets the single led to the defined color. """
    w.itemconfig(led, fill=color)


def get_led_1r(c, led):
    """Gets the color of the led widget. """
    return c.itemcget(led, 'fill')


def make_led_r(mf, sz, leds, num):
    """Creates Multiple Round LEDs """
    c = Canvas(mf, height=(sz * 2) + 2, width=sz * num + 2)
    c.create_rectangle(2, 2, sz*num, sz*2)
    for x in range(0, num):
        coord = sz * x + 3, 4, sz * (x + 1) - 2, sz - 2
        leds.insert(x, c.create_oval(coord, fill='lightgrey'))
        c.create_text(x * sz + 10, sz * 2 - 8, text=str(num - 1 - x))
#    c.tag_raise(g)
    leds.reverse()
    return c, leds


def leds2num_r(c, leds, val, num):
    """Set array of leds in one function. """
    for x in range(0, num):
        cl = 'lightgrey'
        if val & (1 << x):
            cl = 'red'
        set_leds_r(c, leds, x, cl)


def set_leds_r(c, leds, ndx, color):
    """Set individual leds in a multiple led widget """
    c.itemconfig(leds[ndx], fill=color)


def get_leds_r(c, leds, ndx):
    """Gets the status of one led in an array of leds. """
    return c.itemcget(leds[ndx], 'fill')


def make_led_2r(mf, sz, leds):
    """Makes 2 led's side by side """
    c, leds = make_led_r(mf, sz, leds, 2)
    return c, leds


def led2num_2r(c, leds, val):
    """Set individual leds in a multiple led widget """
    leds2num_r(c, leds, val, 2)


def make_led_4r(mf, sz, leds):
    """Makes 4 led's side by side """
    c, leds = make_led_r(mf, sz, leds, 4)
    return c, leds


def led2num_4r(c, leds, val):
    """Set individual leds in a multiple led widget """
    leds2num_r(c, leds, val, 4)


def make_led_8r(mf, sz, leds):
    """Makes 8 led's side by side """
    c, leds = make_led_r(mf, sz, leds, 8)
    return c, leds


def led2num_8r(c, leds, val):
    """Set individual leds in a multiple led widget """
    leds2num_r(c, leds, val, 8)


if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk


    def led_1_stat(level):
        if level:
            led1stat.configure(text='LED ON ')
        else:
            led1stat.configure(text='LED OFF')

    Red_c = 'red'
    Dark_c = 'lightgrey'


    def led_on():
        global lval
        if lval == 0:
            lval = 1
        set_led_1r(w, ind, Red_c)
        led_1_stat(get_led_1r(w, ind) == Red_c)
#        print lval
        led2num_2r(w2, ind2, lval % 4)
        led2num_4r(w4, ind4, lval % 16)
        led2num_8r(w8, ind8, lval % 256)
#        x = get_leds_r(w2, ind2, 0)
#        if x == Red_c:
#            print 'Bank2 bit 0 is on '
#        else:
#            print 'Bank2 bit 0 is off'
        lbl_count.config(text='Count=' + str(lval))
        lval = lval * 2
    lval = lval % 256


def led_off():
    global lval
    set_led_1r(w, ind, Dark_c)
    led_1_stat(get_led_1r(w, ind) == Dark_c)
    led2num_2r(w2, ind2, 0)
    led2num_4r(w4, ind4, 0)
    led2num_8r(w8, ind8, 0)
    lval = 0
    lbl_count.config(text='Count=' + str(lval))


root = Tk()
#  prevent window resizing.
root.resizable(0, 0)
# Replace tk icon with your own.
root.title('LED Test Program')

mainframe = ttk.Frame(root, padding="3", height=300, width=300)
mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
mainframe.grid_propagate(0)
lval = 1
w, ind = make_led_1r(mainframe, 'BUSY')
w.grid(column=0, row=1)
led1stat = ttk.Label(mainframe, text='LED OFF')
led1stat.grid(column=1, row=1)
leds2 = []
w2, ind2 = make_led_2r(mainframe, 18, leds2)
w2.grid(column=0, row=2)
lbl_count = ttk.Label(mainframe, text='Count=0')
lbl_count.grid(column=1, row=2)
leds4 = []
w4, ind4 = make_led_4r(mainframe, 18, leds4)
w4.grid(column=0, row=3)
leds8 = []
w8, ind8 = make_led_8r(mainframe, 18, leds8)
w8.grid(column=0, row=4)

opn_prt = ttk.Button(mainframe, text="LED ON", command=led_on, width=15)
opn_prt.grid(column=0, row=0, padx=15, pady=5)
cse_prt = ttk.Button(mainframe, text="LED OFF", command=led_off, width=15)
cse_prt.grid(column=1, row=0, padx=15, pady=5)
s = 'Filename : Version [' + get_full_lib_version() + ']\n'
```

```
lab2 = Label(mainframe,text=s)
lab2.grid(column=0,row=5, columnspan=2)

for child in mainframe.winfo_children():
    child.grid_configure(padx=5, pady=5)

root.mainloop()
```

## LED_d.py

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#
"""LED Library for dual leds (top green if on or bottom red if off).

    get_lib_version() Show numeric version of library.
    get_lib_full_version()  Show numeric version and library filename.
    make_led_dr   Makes 2 round led's On and off
    make_led_ds   Makes 2 square led's On and Off
    set_led       Sets the leds on = green / gray  off = gray / red
    get_led       Gets the current state of the led.
    """


from tkinter import *
name_led_d    = 'led_d.py'
version_led_d = '1.0.0'
date_led_d    = '02/05/21'
author_led_d  = 'Peter A. Hedlund'

LIB_NAME = name_led_d
LIB_VERSION = version_led_d
LIB_DATE = date_led_d
LIB_AUTHOR = author_led_d

# LIB_NAME = 'led_d.py'
# LIB_VERSION = "1.0.0"
# LIB_DATE = '02/05/21'

def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION


def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs

def make_led_dr(mf):
    """Creates 2 round leds one on top of the other.
      The top led is grey for off and green for on.
      The bottom led is grey for on or red for off"""
    lsize = 18

    scratch = Canvas()
    id = scratch.create_text((0, 0), text='on')
    size = scratch.bbox(id)
    # size is a tuple: (x1, y1, x2, y2)
    # since x1 and y1 will be 0, x2 and y2 give the string width and height

    tw = size[2]-size[0]
    w = Canvas(mf,
              width=lsize + tw + 2,
              height=lsize * 2)
    leda = w.create_oval(2, 5, 10 ,15, fill='lightgrey')
    ledb = w.create_oval(2, 20,10, 30, fill='red')
    w.create_text(22, 9, text='ON')
    w.create_text(22, 24, text='OFF')
    return w, leda, ledb


def make_led_ds(mf):
    """Creates 2 square leds one on top of the other.
      The top led is grey for off and green for on.
      The bottom led is grey for on or red for off"""
    lsize = 18

    scratch = Canvas()
    id = scratch.create_text((0, 0), text='on')
    size = scratch.bbox(id)
    # size is a tuple: (x1, y1, x2, y2)
    # since x1 and y1 will be 0, x2 and y2 give the string width and height

    tw = size[2]-size[0]
    w = Canvas(mf,
              width=lsize + tw + 2,
              height=lsize * 2)
    leda = w.create_rectangle(2, 5, 10 ,15, fill='lightgrey')
    ledb = w.create_rectangle(2, 20,10, 30, fill='red')
    w.create_text(22, 9, text='ON')
    w.create_text(22, 24, text='OFF')
    return w, leda, ledb


def set_led(w, led1, led2, state):
    """Turns the led on or off (state) If on top, led is green bottom is grey
       if off, top led is grey and bottom is red."""
    if state == 1:
        col1 = 'green'
        col2 = 'lightgrey'
    else:
        col1 = 'lightgrey'
        col2 = 'red'
    w.itemconfig(led1, fill=col1)
    w.itemconfig(led2, fill=col2)


def get_led(c, led):
    """Returns the status of the led."""
    return c.itemcget(led, 'fill')


#
# Testing Library Function
#


if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk

    print("you are here")

    def test_led_1_stat(level):
        if level:
            led1stat.configure(text='LED ON ')
        else:
            led1stat.configure(text='LED OFF')

    Red_c = 'red'
    Dark_c = 'lightgrey'


    def test_led_on():
        set_led(w, ind1, ind2, 1)
        test_led_1_stat(get_led(w, ind1) == 'green')


    def test_led_off():
        set_led(w, ind1, ind2, 0)
    test_led_1_stat(get_led(w, ind1) == 'green')


def test_power():
    if pwr.get() == 0:
        set_led(w2, ind3, ind4, 1)
        pwr.set(1)
    else:
        set_led(w2, ind3, ind4, 0)
        pwr.set(0)
        set_led(w3, ind3, ind4, 0)
        shields.set(0)
        set_led(w4, ind3, ind4, 0)
        impulse.set(0)
        set_led(w5, ind3, ind4, 0)
        warp.set(0)


def test_shieldp():
    if pwr.get() == 0:
        return
    if shields.get() == 0:
        set_led(w3, ind3, ind4, 1)
        shields.set(1)
    else:
        set_led(w3, ind3, ind4, 0)
        shields.set(0)


def test_impulsep():
    if pwr.get() == 0:
        return
    if impulse.get() == 0:
        set_led(w4, ind3, ind4, 1)
        impulse.set(1)
    else:
        set_led(w4, ind3, ind4, 0)
        impulse.set(0)


def test_warpp():
    if pwr.get() == 0:
        return
    if warp.get() == 0:
        set_led(w5, ind3, ind4, 1)
        warp.set(1)
    else:
        set_led(w5, ind3, ind4, 0)
        warp.set(0)


root = Tk()
#  prevent window resizing.
root.resizable(0, 0)
# Replace tk icon with your own.
root.title('Dual LED Test Program DEMO')
pwr = IntVar()
shields = IntVar()
impulse = IntVar()
warp = IntVar()
pwr.set(0)
impulse.set(0)
shields.set(0)
warp.set(0)
mainframe = ttk.Frame(root, padding="3", height=400, width=300)
mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
mainframe.grid_propagate(0)

w, ind1, ind2 = make_led_dr(mainframe)
w.grid(column=0, row=1)
led1stat = ttk.Label(mainframe, text='LED OFF')
led1stat.grid(column=1, row=1)

opn_prt = ttk.Button(mainframe, text="LED ON", command=test_led_on, width=15)
opn_prt.grid(column=0, row=0, padx=15, pady=5)
cse_prt = ttk.Button(mainframe, text="LED OFF", command=test_led_off, width=15)
cse_prt.grid(column=1, row=0, padx=15, pady=5)
s = 'Filename : Version [' + get_full_lib_version() + ']\n'
lab2 = Label(mainframe,text=s)
lab2.grid(column=0,row=5, columnspan=2)
btns = ttk.Button(mainframe, text='Shields', command=test_shieldp)
btns.grid(column=0, row=6, sticky='e')
w3, ind3,ind4 = make_led_ds(mainframe)
w3.grid(column=1, row=6, sticky='w')
btni = ttk.Button(mainframe, text='Impulse', command=test_impulsep)
btni.grid(column=0, row=7, sticky='e')
w4, ind3,ind4 = make_led_ds(mainframe)
w4.grid(column=1, row=7, sticky='w')
btnw = ttk.Button(mainframe, text='Warp', command=test_warpp)
btnw.grid(column=0, row=8, sticky='e')
w5, ind3,ind4 = make_led_ds(mainframe)
```

```
w5.grid(column=1, row=8, sticky='w')
btn = ttk.Button(mainframe, text='Power', command=test_power)
btn.grid(column=0, row=9, sticky='e')
w2, ind3,ind4 = make_led_dr(mainframe)
w2.grid(column=1, row=9, sticky='w')
for child in mainframe.winfo_children():
    child.grid_configure(padx=5, pady=5)

root.mainloop()
```

## Led_sq.py

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#

"""LED library for Square leds, 1, 2, 4 and 8 wide
    Single LED ONLY
    get_lib_version() Show numeric version of library.
    get_lib_full_version()  Show numeric version and library filename.
    make_led_1s   Makes a single led with a text label on the side
    set_led_1s    Sets the led to a color
    get_led_1s    Gets the current color of the led
    BANK LEDS ONLY
    make_led_2s  creates a bank of leds
    make_led_4s
    make_led_8s
    led2num_2s    Sets the leds in the bank the binary values
    led2num_4s          colors are light grey off and red on
    led2num_8s
    set_leds_s    sets the led to a color at an index
    get_leds_s    Gets the color of the led at an indes
    """

from tkinter import *
name_led_sq    = 'led_sq.py'
version_led_sq = '2.0.0'
date_led_sq    = '02/05/21'
author_led_sq  = 'Peter A. Hedlund'

LIB_NAME = name_led_sq
LIB_VERSION = version_led_sq
LIB_DATE = date_led_sq
LIB_AUTHOR = author_led_sq

# LIB_NAME = 'led_sq.py'
# LIB_VERSION = "2.0.0"
# LIB_DATE = '02/05/21'

def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION


def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs


def make_led_1s(mf, label):
    """Creates a single square led widget with a label next to it. """
    lsize = 18

    scratch = Canvas()
    id = scratch.create_text((0, 0), text=label)
    size = scratch.bbox(id)
    # size is a tuple: (x1, y1, x2, y2)
    # since x1 and y1 will be 0, x2 and y2 give the string width and height

    tw = size[2]-size[0]
    w = Canvas(mf,
            width=lsize + tw + 2,
            height=lsize)
    y = int(lsize) - 2
    led = w.create_rectangle(2, 2, y-2, y-2, fill='lightgrey')
    w.create_text(lsize*2, lsize/2, text=label)
    return w, led


def set_led_1s(w, led, color):
    """Sets the single led to the defined color. """
    w.itemconfig(led, fill=color)


def get_led_1s(c, led):
    """Gets the color of the led widget. """
    return c.itemcget(led, 'fill')


def make_led_s(mf, sz, leds, num):
    """Creates Multiple Square LEDs """
    c = Canvas(mf, height=(sz * 2) + 2, width=sz * num + 2)
    c.create_rectangle(2, 2, sz*num, sz*2)
    for x in range(0, num):
        coord = sz * x + 3, 4, sz * (x + 1) - 2, sz - 2
        leds.insert(x, c.create_rectangle(coord, fill='lightgrey'))
        c.create_text(x * sz + 10, sz * 2 - 8, text=str(num - 1 - x))
```

```
#    c.tag_raise(g)
    leds.reverse()
    return c, leds


def leds2num_s(c, leds, val, num):
    """Set array of leds in one function. """
    for x in range(0, num):
        cl = 'lightgrey'
        if val & (1 << x):
            cl = 'red'
        set_leds_s(c, leds, x, cl)


def set_leds_s(c, leds, ndx, color):
    """Set individual leds in a multiple led widget """
    c.itemconfig(leds[ndx], fill=color)


def get_leds_s(c, leds, ndx):
    """ Gets the status of one led in an array of leds. """
    return c.itemcget(leds[ndx], 'fill')


def make_led_2s(mf, sz, leds):
    """Makes 2 led's side by side """
    c, leds = make_led_s(mf, sz, leds, 2)
    return c, leds


def led2num_2s(c, leds, val):
    """Set individual leds in a multiple led widget """
    leds2num_s(c, leds, val, 2)


def make_led_4s(mf, sz, leds):
    """Makes 4 led's side by side """
    c, leds = make_led_s(mf, sz, leds, 4)
    return c, leds


def led2num_4s(c, leds, val):
    """Set individual leds in a multiple led widget """
    leds2num_s(c, leds, val, 4)
```

```
def make_led_8s(mf, sz, leds):
    """Makes 8 led's side by side """
    c, leds = make_led_s(mf, sz, leds, 8)
    return c, leds


def led2num_8s(c, leds, val):
    """Set individual leds in a multiple led widget """
    leds2num_s(c, leds, val, 8)


if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk


    def led_1_stat(level):
        if level:
            led1stat.configure(text='LED ON ')
        else:
            led1stat.configure(text='LED OFF')

    Red_c = 'red'
    Dark_c = 'lightgrey'


    def led_on():
        global lval
        if lval == 0:
            lval = 1
        set_led_1s(w, ind, Red_c)
        led_1_stat(get_led_1s(w, ind) == Red_c)
#       print lval
        led2num_2s(w2, ind2, lval % 4)
        led2num_4s(w4, ind4, lval % 16)
        led2num_8s(w8, ind8, lval % 256)
#       x = get_leds_s(w2, ind2, 0)
#       if x == Red_c:
#           print 'Bank2 bit 0 is on '
#       else:
#           print 'Bank2 bit 0 is off'
        lbl_count.config(text='Count=' + str(lval))
        lval = lval * 2
        lval = lval % 256
```

```
def led_off():
    global lval
    set_led_1s(w, ind, Dark_c)
    led_1_stat(get_led_1s(w, ind) == Dark_c)
    led2num_2s(w2, ind2, 0)
    led2num_4s(w4, ind4, 0)
    led2num_8s(w8, ind8, 0)
    lval = 0
    lbl_count.config(text='Count=' + str(lval))


root = Tk()
#  prevent window resizing.
root.resizable(0, 0)
# Replace tk icon with your own.
root.title('LED Test Program')

mainframe = ttk.Frame(root, padding="3", height=300, width=300)
mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
mainframe.grid_propagate(0)
lval = 1
w, ind = make_led_1s(mainframe, 'BUSY')
w.grid(column=0, row=1)
led1stat = ttk.Label(mainframe, text='LED OFF')
led1stat.grid(column=1, row=1)
leds2 = []
w2, ind2 = make_led_2s(mainframe, 18, leds2)
w2.grid(column=0, row=2)
lbl_count = ttk.Label(mainframe, text='Count=0')
lbl_count.grid(column=1, row=2)
leds4 = []
w4, ind4 = make_led_4s(mainframe, 18, leds4)
w4.grid(column=0, row=3)
leds8 = []
w8, ind8 = make_led_8s(mainframe, 18, leds8)
w8.grid(column=0, row=4)

opn_prt = ttk.Button(mainframe, text="LED ON", command=led_on, width=15)
opn_prt.grid(column=0, row=0, padx=15, pady=5)
cse_prt = ttk.Button(mainframe, text="LED OFF", command=led_off, width=15)
cse_prt.grid(column=1, row=0, padx=15, pady=5)
s = 'Filename : Version [' + get_full_lib_version() + ']\n'
lab2 = Label(mainframe,text=s)
lab2.grid(column=0,row=5, columnspan=2)
```

```
for child in mainframe.winfo_children():
    child.grid_configure(padx=5, pady=5)

root.mainloop()
```

## Low_ASCII.py

```
# low_ascii.py
"""Character defnitions for ascii characters 0 - 31 """
name_low_ascii    = 'low_ascii.py'
version_low_ascii = '1.0.0'
date_low_ascii    = '04/22/21'
author_low_ascii  = 'Peter A. Hedlund'
# ====================================================
# ascii codes 0 = 31  are non printable control codes
# ====================================================

nul_  = 0x00        # null
soh_  = 0x01        # Start of heading
stx_  = 0x02        # Start of text
etx_  = 0x03        # End of text
eot_  = 0x04        # End of transmit
ack_  = 0x06        # acknowledge
bel_  = 0x07        # audible bell
bksp_ = 0x08        # backspace
ht_   = 0x09        # horizontal tab
lf_   = 0x0a        # line feed
vt_   = 0x0b        # vertical tab
ff_   = 0x0c        # form feed
cr_   = 0x0d        # carriage return
si_   = 0x0e        # Shift in
so_   = 0x0f        # shift out
dle_  = 0x10        # Data link escape
dc1_  = 0x11        # Device Control 1
dc2_  = 0x12        # Device Control 2
dc3_  = 0x13        # Device Control 3
dc4_  = 0x14        # Device Control 4
nak_  = 0x15        # Nag acknowledge
is_   = 0x16        # Synchronus idle
etb_  = 0x17        # End Trans Block
can_  = 0x18        # Cancel
eom_  = 0x19        # End of medium
sub_  = 0x1a        # Substution
esc_  = 0x1b        # Escape
fs_   = 0x1c        # File Seperator
gs_   = 0x1d        # group seperator
rs_   = 0x1e        # Record Seperator

pb_ = 0x5b          # [
pe_ = 0x5d          # ]
```

## My_Math.py

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#
"""My_Math.py My math routines
    get_lib_version() Show numeric version of library.
    get_lib_full_version()  Show numeric version and library filename.
    inside()       returns true / false depening if value is within limits.
    outsied()      returns true / false depending if value is ouside limits
    re-scale()     returns scaled value based on limits and new range
    c_to_f()       returns c from f
    f_to_c()       returns f from c
    sec_2_hms()    returns hh:mm:ss string from seconds passed
    current_time() returns hh:mm:ss string of current time
    filter_float() returned filtered value based on coeficcent.
"""
import time

name_my_math    = 'my_math.py'
version_my_math = '1.2.0'
date_my_math    = '05/19/21'
author_my_math  = 'Peter A. Hedlund'

LIB_NAME = name_my_math
LIB_VERSION = version_my_math
LIB_DATE = date_my_math
LIB_AUTHOR = author_my_math

# LIB_NAME = 'my_math.py'
# LIB_VERSION = "1.0.0"
# LIB_DATE = '02/05/21'

def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION



def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs

def inside(lo, hi, ck):
    """Checks that ck is within lo to hi range returns 1 if true 0 if false"""
    if lo < ck and hi > ck:
        return 1
    return 0


def outside(lo, hi, ck):
    """Checks that ck is ouside of lo to hi range returns 1 if true 0 if
    false"""
    if lo > ck and hi < ck:
        return 1
    return 0


def re_scale(lo, hi, rng, val):
    """Scale value val to range rng bases on lo and hi limits """
    x = (hi - lo) / rng
    ov = val / x
    return ov


def f_to_c(f_val):
    """ Returns  celsius from fahrenheit"""
    return (f_val-32) * 5.0 / 9.0


def c_to_f(c_val):
    """ Returns  fahrenheit from celsius"""
    return (c_val * 9 / 5) + 32.0


def sec_2_hms(seconds):
    """Convert seconds to HH:MM:SS string"""
    seconds = int(seconds)
    m, s = divmod(seconds, 60)
    h, m = divmod(m, 60)
    return f'{h:02}:{m:02}:{s:02}'


def current_time():
    """returns current time in a string HH:MM:SS."""
    tm = time.localtime()
    str1 = time.strftime('%H:%M:%S',tm)
    return str1
```

```
def filter_float( old_value, new_value, filter_coe):
    """ Filter Function  (old value, new value, filter coeffecient)"""
    if filter_coe >= 1.0:
        output = old_value
    else:
        if filter_coe <= 0.0:
            output = new_value
        else:
            output = ( old_value * filter_coe ) + ( new_value * ( 1  - filter_coe ) )
    return output

#
# Testing Library Function
#


if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk
    import mylib.scrn_log as scrn

    def send_str(txt, wstr):
        txt.insert(END,wstr)
        txt.index(END)
        txt.see(END)
        txt.update()

    root = Tk()
    #  prevent window resizing.
    root.resizable(0, 0)
    # Replace tk icon with your own.
    root.title('Dual LED Test Program DEMO')
    mainframe = ttk.Frame(root, padding="3", height=420, width=440)
    mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
    mainframe.grid_propagate(0)
    raw=[10,15,5,20,20,10]
    log = scrn.scrn_log(mainframe, 50, 23, 'black', 'light grey', 0, 0, 2, 10)
    log.log_scrn_color('MyMath Library Test\n', 'green')
    log.log_scrn_color('[TESTING Inside]\n', 'blue')
    log.log_scrn(f'15 inside range 10-20 {inside(10,20,15)} (expect 1)\n')
    log.log_scrn(f'25 inside range 10-20 {inside(10,20,25)} (expect 0)\n')
    log.log_scrn_color('[TESTING Outside]\n', 'blue')
    log.log_scrn(f'25 outside range 10-20 {outside(10,20,25)} (expect 1)\n')
    log.log_scrn(f'15 outside range 10-20 {outside(10,20,15)} (expect 0)\n')
    log.log_scrn_color('[TESTING re_scale]\n', 'blue')
log.log_scrn(f'45 re_scaled from range 0-90 {re_scale(0,90,50,45)}'
        ' (expect 25)\n')
log.log_scrn(f'22.5 re_scaled from range 0-90 {re_scale(0,90,50,22.5)}'
        ' (expect 12.5)\n')
log.log_scrn(f'67.5 re_scaled from range 0-90 {re_scale(0,90,50,67.5)}'
        ' (expect 37.5)\n')
log.log_scrn_color('[Testing sec_to_hms]\n', 'blue')
log.log_scrn(' 4166 seconds is 01:09:26\n')
st = sec_2_hms(4166)
log.log_scrn(f'Return of sec_2_hms(4166) is {st}\n')
log.log_scrn(f'70 F is {f_to_c(70)} C (21.11111)\n')
log.log_scrn(f'22 C is {c_to_f(22)} C (71.6)\n')
log.log_scrn_color('[Testing current_time]\n', 'blue')
log.log_scrn(f'Current time is {current_time()}\n')
log.log_scrn_color('[Testing filter_float]\n', 'blue')

log.log_scrn(f'Raw Data = {raw}\n')
filt = []
filt.append(0)

for x in range(1,len(raw)):
    filt.append(filter_float(filt[x-1], raw[x], .80))
log.log_scrn(f'Filtering with a coeffiencent of .80\n')
log.log_scrn(f'Filtered data = \n        ')
for x in range(0,len(filt)):
    b = int(filt[x]*100) / 100
    log.log_scrn(f'{b}, ')

log.log_scrn(f'\n')
log.log_scrn(f'expected 0.0, 2.99, 3.39, 6.71, 9.37, 9.5\n')

for child in root.winfo_children():
    child.grid_configure(padx=5, pady=5)

root.mainloop()
```

## P_Types.py

```
# p_types.py
"""C style type definitions for python structures"""

name_p_types    = 'p_types.py'
version_p_types = '1.0.0'
date_p_types    = '04/22/21'
author_p_types  = 'Peter A. Hedlund'

# =============================================
# C type casting for use in python structures
# =============================================

char8_   = 'b'      # Bytes 1
uchar8_  = 'B'      # Bytes 1
bool_    = '?'      # Bytes 1
int16_   = 'h'      # Bytes 2
uint16_  = 'H'      # Bytes 2
int32_   = 'i'      # Bytes 4
uint32_  = 'I'      # Bytes 4
int64_   = 'q'      # Bytes 8
uint64_  = 'Q'      # Bytes 8
float32_ = 'f'      # Bytes 4
float64_ = 'd'      # Bytes 8
str_     = 's'      # Bytes ? put number before str_
bpacked_ = '='      # Bytes 0
```

## print_colors.py

```
"""Print color and cursor commands
# Note : make sure the following code snippets is in your code to
enable ansi escape codes.

from colorama import init
init()

Colors are for foreground, background, bright foreground, bright background

Cursor Functions

def pr_cursor_up(lines):
def pr_cursor_down(lines):
def pr_cursor_forward(ch):
def pr_cursor_back(ch):
def pr_cursor_next_ln(line):
def pr_cursor_prev_ln(line):
def pr_cursor_horiz(ch):
def pr_cursor_pos(x, y):
def pr_erase_in_dsp(n):
def pr_erase_in_line(n):
def pr_scroll_up(n):
def pr_scroll_down(n):
def pr_cursor_save():
def pr_corsor_restore():
def pr_cursor_show():
def pr_cursor_hide():
"""
# Print Color setting commands

# Normal Foreground
pr_f_black   = '\033[30m'
pr_f_red     = '\033[31m'
pr_f_green   = '\033[32m'
pr_f_yellow  = '\033[33m'
pr_f_blue    = '\033[34m'
pr_f_magenta = '\033[35m'
pr_f_cyan    = '\033[36m'
pr_f_white   = '\033[37m'

# Normal Background
pr_b_black   = '\033[40m'
pr_b_red     = '\033[41m'
pr_b_green   = '\033[42m'
pr_b_yellow  = '\033[43m'
pr_b_blue    = '\033[44m'
pr_b_magenta = '\033[45m'
pr_b_cyan    = '\033[46m'
pr_b_white   = '\033[47m'

# Bright Foreground
pr_fb_black   = '\033[90m'
pr_fb_red     = '\033[91m'
```

```
pr_fb_green   = '\033[92m'
pr_fb_yellow  = '\033[93m'
pr_fb_blue    = '\033[94m'
pr_fb_magenta = '\033[95m'
pr_fb_cyan    = '\033[96m'
pr_fb_white   = '\033[97m'

# Bright Background
pr_bb_black   = '\033[100m'
pr_bb_red     = '\033[101m'
pr_bb_green   = '\033[102m'
pr_bb_yellow  = '\033[103m'
pr_bb_blue    = '\033[104m'
pr_bb_magenta = '\033[105m'
pr_bb_cyan    = '\033[106m'
pr_bb_white   = '\033[107m'

# Reset color to default
pr_normal     = '\x1b[0m'
pr_bold       = '\x1b[1m'
pr_itallic    = '\x1b[2m'
pr_underline  = '\x1b[3m'

def pr_cursor_up(lines):
    """ Move cursor up <lines> number of lines."""
    return(f'\x1b[{lines}A')


def pr_cursor_down(lines):
    """ Move cursor down <lines> number of lines."""
    return(f'\x1b[{lines}B')


def pr_cursor_forward(ch):
    """ Move cursor forward <ch> number of characters."""
    return(f'\x1b[{ch}C')


def pr_cursor_back(ch):
    """ Move cursor backward <ch> number of characters."""
    return(f'\x1b[{ch}D')


def pr_cursor_next_ln(line):
    """ Move cursor to the beginning of next line (lines) down."""
    return(f'\x1b[{line}E')


def pr_cursor_prev_ln(line):
    """ Move cursor to the beginning of next line (lines) up."""
    return(f'\x1b[{line}F')


def pr_cursor_horiz(ch):
    """Place cursor at position ch on current line."""
    return(f'\x1b[{ch}G')
```

```
def pr_cursor_pos(x, y):
    """Place cursor at horizontal x, vertical y."""
    return(f'\x1b[{x},{y}H')


def pr_erase_in_dsp(n):
    """Erase in display n is 0-3."""
    return(f'\x1b[{n}J')


def pr_erase_in_line(n):
    """Erease in line n is 0-3."""
    return(f'\x1b[{n}K')


def pr_scroll_up(n):
    """ Scroll screen up n lines."""
    return(f'\x1b[{n}S')


def pr_scroll_down(n):
    """ Scroll screen down n lines"""
    return(f'\x1b[{n}T')


def pr_cursor_save():
    """Save current cursor position."""
    return(f'\x1b[s')


def pr_corsor_restore():
    """Restore cursor position to last saved."""
    return(f'\x1b[t')


def pr_cursor_show():
    """Show Cursor."""
    return('\x1b[?25h')


def pr_cursor_hide():
    """Hide cursor."""
    return('\x1b[?25l')
```

## Scrollable Notebook

```
# -*- coding: utf-8 -*-

# Copyright (c) Muhammet Emin TURGUT 2020
# For license see LICENSE
# https://github.com/muhammeteminturgut/ttkScrollableNotebook
#
# Modifications by Peter Hedlund


from tkinter import *
from tkinter import ttk
name_SCRL_Notebook    = 'SCRL_Notebook.py'
version_SCRL_Notebook = '1.0.0'
date_SCRL_Notebook    = '04/26/21'
author_SCRL_Notebook  = 'Muhammet Emin TURGUT'
LIB_NAME = name_SCRL_Notebook
LIB_VERSION = version_SCRL_Notebook
LIB_DATE = date_SCRL_Notebook
LIB_AUTHOR = author_SCRL_Notebook


class ScrollableNotebook(ttk.Frame):
    """Scrollable Notebook widget (Similar to ttk.Notebook)"""

    def __init__(self,parent,wheelscroll=False,tabmenu=False,*args,**kwargs):
        ttk.Frame.__init__(self, parent, *args)
        self.xLocation = 0
        self.notebookContent = ttk.Notebook(self,**kwargs)
        self.notebookContent.pack(fill="both", expand=True)
        self.notebookTab = ttk.Notebook(self,**kwargs)
        self.notebookTab.bind("<<NotebookTabChanged>>",self._tabChanger)
        if wheelscroll==True: self.notebookTab.bind("<MouseWheel>", self._wheelscroll)
        slideFrame = ttk.Frame(self)
        slideFrame.place(relx=1.0, x=0, y=1, anchor=NE)
        self.menuSpace=30
        if tabmenu==True:
            self.menuSpace=50
            bottomTab = ttk.Label(slideFrame, text=" \u2630 ")
            bottomTab.bind("<1>",self._bottomMenu)
            bottomTab.pack(side=RIGHT)
        leftArrow = ttk.Label(slideFrame, text=" \u276E")
        leftArrow.bind("<1>",self._leftSlide)
        leftArrow.pack(side=LEFT)
        rightArrow = ttk.Label(slideFrame, text=" \u276F")
        rightArrow.bind("<1>",self._rightSlide)
        rightArrow.pack(side=RIGHT)
        self.notebookContent.bind("<Configure>", self._resetSlide)

    def get_lib_version(self):
        """Returns version information only."""
        return LIB_VERSION


    def get_full_lib_version(self):
        """Returns version information and library name."""
        msg = self.get_lib_version()
        rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
        rs = rs + '\nDate : ' + LIB_DATE
        rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
        return rs

    def _wheelscroll(self, event):
        if event.delta > 0:
            self._leftSlide(event)
        else:
            self._rightSlide(event)

    def _bottomMenu(self,event):
        tabListMenu = Menu(self, tearoff = 0)
        for tab in self.notebookTab.tabs():
            tabListMenu.add_command(label=self.notebookTab.tab(tab, option="text"),command=
lambda temp=tab: self.select(temp))
        try:
            tabListMenu.tk_popup(event.x_root, event.y_root)
        finally:
            tabListMenu.grab_release()

    def _tabChanger(self,event):
        try: self.notebookContent.select(self.notebookTab.index("current"))
        except: pass

    def _rightSlide(self,event):
        if self.notebookTab.winfo_width()>self.notebookContent.winfo_width()-self.menuSpace:
            if (self.notebookContent.winfo_width()-(self.notebookTab.winfo_width()
+self.notebookTab.winfo_x()))<=self.menuSpace+5:
                self.xLocation-=20
                self.notebookTab.place(x=self.xLocation,y=0)
    def _leftSlide(self,event):
        if not self.notebookTab.winfo_x()== 0:
            self.xLocation+=20
            self.notebookTab.place(x=self.xLocation,y=0)

    def _resetSlide(self,event=None):
        self.notebookTab.place(x=0,y=0)
        self.xLocation = 0

    def add(self,frame,**kwargs):
        if len(self.notebookTab.winfo_children())!=0:
            self.notebookContent.add(frame, text="",state="hidden")
        else:
            self.notebookContent.add(frame, text="")
        self.notebookTab.add(ttk.Frame(self.notebookTab),**kwargs)

    def forget(self,tab_id):
        self.notebookContent.forget(self.__ContentTabID(tab_id))
        self.notebookTab.forget(tab_id)

    def hide(self,tab_id):
        self.notebookContent.hide(self.__ContentTabID(tab_id))
        self.notebookTab.hide(tab_id)

    def identify(self,x, y):
        return self.notebookTab.identify(x,y)

    def index(self,tab_id):
        return self.notebookTab.index(tab_id)

    def __ContentTabID(self,tab_id):
        return self.notebookContent.tabs()[self.notebookTab.tabs().index(tab_id)]

    def insert(self,pos,frame, **kwargs):
        self.notebookContent.insert(pos,frame, **kwargs)
        self.notebookTab.insert(pos,frame,**kwargs)

    def select(self,tab_id=None):
        if tab_id == None:
            return self.notebookTab.select()
##          self.notebookContent.select(self.__ContentTabID(tab_id))
        self.notebookTab.select(tab_id)

    def tab(self,tab_id, option=None, **kwargs):
        kwargs_Content = kwargs.copy()
        kwargs_Content["text"] = "" # important
        self.notebookContent.tab(self.__ContentTabID(tab_id), option=None, **kwargs_Content)
        return self.notebookTab.tab(tab_id, option=None, **kwargs)
    def tabs(self):
##        return self.notebookContent.tabs()
        return self.notebookTab.tabs()

    def enable_traversal(self):
        self.notebookContent.enable_traversal()
        self.notebookTab.enable_traversal()

if __name__ == "__main__":

    root=Tk()
    root.title("Example")
    notebook=ScrollableNotebook(root,wheelscroll=True,tabmenu=True)
    frame1=Frame(notebook)
    frame2=Frame(notebook)
    frame3=Frame(notebook)
    frame4=Frame(notebook)
    notebook.add(frame1,text="I am Tab One")
    notebook.add(frame2,text="I am Tab Two")
    notebook.add(frame3,text="I am Tab Three")
    notebook.add(frame4,text="I Forgot How to Count")
    notebook.pack(fill="both",expand=True)
    text=Text(frame1)
    text.pack()
    Label(frame2,text="I am Frame 2").pack()
    Label(frame3,text="I am Frame 3").pack()
    Label(frame4,text="You know i'm Frame 4").pack()
    text.insert(INSERT,"Hello World!")
    root.mainloop()
```

## Scrn_Log

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#
""" Logging functions Library
   scrn_log()
   get_lib_version() Show numeric version of library.
   get_lib_full_version()  Show numeric version and library filename.
   log_scrn()  sends string to text window in last color
   log_scrn_color()  sends string to text window in color
   log_scrn_raw() non colorized version without updateing every time.
      Acceptable colors are 'red','yellow','green','blue''cyan'
      'white', 'violet', sky blue, hot pink, lightgrey, and brown4
   get_pos() returns x.y of cursor position of window.
   clear_scrn()  clears the text window
   save_scrn()  saves the text window to a user selected file
   clipboard_scrn() returns a copy of the screen in the clipboard
"""

from tkinter import *
from tkinter import filedialog
from tkinter import ttk

name_scrn_log    = 'scrn_log.py'
version_scrn_log = '1.1.0'
date_scrn_log    = '04/29/21'
author_scrn_log  = 'Peter A. Hedlund'

LIB_NAME = name_scrn_log
LIB_VERSION = version_scrn_log
LIB_DATE = date_scrn_log
LIB_AUTHOR = author_scrn_log

class scrn_log(object):
   """frame=mainframe, wd=width, ht=height, fgc=foreground color,
   bgc=background color, col=column, rw=row, cs=columnspan, rs=rowspan"""

   update = 1   #  Set to 0 to speed up screen writes in loops

   def __init__(self, frame, wd, ht, fgc, bgc, col, rw, cs, rs ):
   #  Text box Widget
      txt = Text(frame, width=wd, height=ht, fg=fgc, bg=bgc, padx=2, pady=2)
      txt.grid(column=col, row=rw, columnspan=cs, rowspan=rs, sticky="nsew",
      padx=5, pady=5)
      txt.configure(state='disabled')
```

```
    #  Scrollbar linked to text box above
    scrollb = ttk.Scrollbar(frame, command=txt.yview)
    scrollb.grid(column=col+cs+1, row=rw, rowspan=rs, sticky='nse')
    txt['yscrollcommand'] = scrollb.set
    self.txt = txt
    self.colors = {'red': 'junk1','yellow': 'junk2','green': 'junk3',
    'blue': 'junk4','cyan': 'junk5','white': 'junk6','violet':'junk7',
    'sky blue': 'junk8', 'hot pink': 'junk9', 'lightgrey':'junk10',
    'brown4':'junk11'}

def get_lib_version(self):
    """Returns version information only."""
    return LIB_VERSION


def get_full_lib_version(self):
    """Returns version information and library name."""
    msg = self.get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + "\nDate : " + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs


def log_scrn(self, tstr):
    """Sends string to text window. """
    self.txt.configure(state='normal')
    self.txt.index(END)
    self.txt.see(END)
    self.txt.insert(END, tstr)
    if self.update == 1:
        self.txt.update()
    self.txt.configure(state='disabled')


def log_scrn_color(self, tstr, color):
    """ Sends the string to the text window in specified color
    see colors dictionary in the init section for color selection
    Current Colors : 'red','yellow','green','blue','cyan','white',
    'violet', 'sky blue', 'hot pink', 'lightgrey', and 'brown4'"""
    if (color in self.colors) == False:
        color = 'green'
    self.txt.configure(state='normal')
    self.txt.mark_set(INSERT, END)
    st = self.txt.index('insert')
```

```
        self.txt.insert(END, tstr)
        ed = self.txt.index('insert')
        self.txt.tag_add(self.colors[color], st, ed)
        self.txt.tag_config(self.colors[color], foreground=color)
        self.txt.see(END)
        if self.update == 1:
            self.txt.update_idletasks()
        self.txt.configure(state='disabled')


    def log_scrn_raw(self, tstr):
        """Sends string to text window. low overhead for multiple calls. """
        self.txt.configure(state='normal')
        self.txt.insert(END, tstr)
        self.txt.configure(state='disabled')


    def get_pos(self):
        """ returns current position of cursor x.y """
        self.txt.configure(state='normal')
        rval = self.txt.index(INSERT)
        self.txt.configure(state='disabled')
        return  rval


    def clear_scrn(self):
        """Clear the text widget. """
        #  Enable writing to text widget
        self.txt.configure(state='normal')
        #  Delete from first position to end
        self.txt.delete(1.0, END)
        #  Disable writing to text widget
        self.txt.configure(state='disabled')


    def save_scrn(self, main):
        """Get all text from existing text widget and save it to a
        user defined text file. NO COLORED TEXT"""
        lines = self.txt.get("1.0", END).splitlines()
        path = filedialog.asksaveasfilename(parent=main)
        if path == '':
            return
        fo = open(path, 'w', encoding='utf-8')
        for line in lines:
            fo.write(line + "\n")
        fo.close()
    def clipboard_scrn(self, main):
        """Get all text from existing text widget and save it to
        the clipboard"""
        lines = self.txt.get("1.0", END).splitlines()
        main.clipboard_clear()
        for line in lines:
            main.clipboard_append(line + '\n')
        main.update()


if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk
    import os
    import sys

    def quit_prog():
        root.destroy()


    def my_clear_log():
        screen.clear_scrn()


    def log_clip():
        screen.clipboard_scrn(mainframe)


    def my_save_log():
        screen.save_scrn(mainframe)


    def get_position():
        pos = screen.get_pos()
        screen.log_scrn(f'Position is {pos}\n')


    def part_text():
        screen.log_scrn('no CR on this line ')


    def test_log():
        screen.log_scrn('Name ' + os.path.basename(sys.argv[0]) + '\n' + screen.get_full_lib_version() +
'\n')
        screen.log_scrn('Sample Text Normal\n')
        screen.log_scrn_color('Sample Text Blue\n', 'blue')
        screen.log_scrn_color('Sample Text Cyan\n', 'cyan')
        screen.log_scrn_color('Sample Text Red\n','red')
        screen.log_scrn_color('Sample Text Violet\n','violet')
        screen.log_scrn_color('Sample Text Yellow\n', 'yellow')
        screen.log_scrn_color('Sample Text Green\n', 'green')
        screen.log_scrn_color('Sample Text White\n', 'white')
        screen.log_scrn_color('Sample Text Sky Blue\n', 'sky blue')
        screen.log_scrn_color('Sample Text Hot Pink\n', 'hot pink')
        screen.log_scrn_color('Sample Text Light Grey\n', 'lightgrey')
        screen.log_scrn_color('Sample Text Brown\n', 'brown4')
        screen.log_scrn('to_log function without color\n')

    root = Tk()
    #  prevent window resizing.
    root.resizable(0, 0)
    # Replace tk icon with your own.
    root.title('Testing Log file')

    mainframe = ttk.Frame(root, padding="3", height=650, width=700)
    mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
    mainframe.grid_propagate(0)

    s1 = ttk.Style()
    s1.configure('red.TButton', background='Red', relief='sunken')
    s2 = ttk.Style()
    s2.configure('blue.TButton', background='Blue')
    s3 = ttk.Style()
    s3.configure('green.TButton', background='Green')
    s4 = ttk.Style()
    s4.configure('cyan.TButton', background='Cyan', relief='raised')

    btn = (('Clear Log', my_clear_log, 'cyan.TButton'),
           ('Save Log', my_save_log, 'cyan.TButton'),
           ('TEST', test_log, 'red.TButton'),
           ('Text no LF', part_text, ''),
           ('Get Pos', get_position, ''),
           ('Log 2 Clipboard', log_clip, 'blue.TButton' ),
           ('Quit', quit_prog, 'red.TButton'))

    for x in range(0, len(btn)):
        bbtn = ttk.Button(mainframe, text=btn[x][0], command=btn[x][1],
                style=btn[x][2], width=15)
        bbtn.grid(column=0, row=x+1, padx=15, pady=5)
```

```
screen = scrn_log(mainframe, 50, 34, 'lightgreen', 'black', 2, 1, 6, 20)
s = 'Filename : Version [' + screen.get_full_lib_version() + ']\n'
lab2 = Label(mainframe,text=s)
lab2.grid(column=2,row=0, columnspan=2)

for child in mainframe.winfo_children():
    child.grid_configure(padx=5, pady=5)

root.mainloop()
```

## Seven_Seg.py

```
"""Seven Segment display This library consists of 7 segment line characters
as well as 7 segment led characters. Digits 0 -9 and a-f are supported along
with characters 16-22 which are individual segments. (16 is a, 17 is b ... )

    get_lib_version() Show numeric version of library.
    get_lib_full_version()  Show numeric version and library filename.
    class Digit() creates line type 7 segment display character
        show()  displays numeric value in character
    class Counter() creates line type 7 segment display with dig characters
        and base of 10 or 16 (decimal or hex)
        num() displays the number on the counter display 1000 on a 3 digit
        display is 000
    Class Counter_Dp() creates a counter of dig characters with a decimal
        point at char dp and dp+1
        num() displays the number on the counter display with the whole number
            on the left side of the decimal point and the fractional number
            on the right side of the decimal point.
            display is 000
    class Clock() Creates a 7 segment line clock mode=0 = HH:MM mode=1
        = HH:MM:SS.
        num() Pass H, M and S to routinte to dispay time
    class Digit_Led() creates led type 7 segment display character
        show()  displays numeric value in character
    class Counter_Led() creates Led type 7 segment display with dig
        characters and base of 10 or 16 (decimal or hex)
        num() displays the number on the counter display 1000 on a 3 digit
            display is 000
    Class Counter_Led_Dp() creates a counter of dig characters with a decimal
        point at char dp and dp+1
        num() displays the number on the counter display with the whole number
            on the left side of the decimal point and the fractional number
            on the right side of the decimal point.
            display is 000
    class Clock_Led() Creates a 7 segment LED clock mode=0 = HH:MM mode=1
        = HH:MM:SS.
        num() Pass H, M and S to routinte to dispay time
        """
'''Seven segment display of hex digits.'''
import tkinter as tk

# DONE Add Clock face to LED Display
# DONE Add Update_time to LED_Clock
# DONE Add Clock face to line display
# DONE Add update_time to Line display clock
# DONE Add Counter with Decimal Point to LED Display
# DONE Add Show to decimal LED display
# DONE Add Counter with Decimal Point to Line Display
# DONE Add Show to decimal line display

segments = (
    (0, 3, 4, 0, 16, 0, 20, 3, 16, 7, 4, 7), # a
    (20, 4, 23, 8, 23, 20, 20, 24, 17, 20, 17, 8 ), # b
    (20,24, 23, 28, 23, 40,20, 44, 17, 40, 17, 28), # c
    (20,45, 16, 48, 4, 48, 0, 45, 4, 42, 16,42), # d
    (0, 44, -3, 40, -3, 28, 0, 24, 3, 28, 3, 40), # e
    (0, 4, 3, 8, 3, 20, 0, 24, -3, 20, -3, 8), #f
    (20, 24, 16, 21, 4, 21, 0, 24, 4, 27, 15, 27), #g
)

# Order 7 segments clockwise from top left, with crossbar last.
# Coordinates of each segment are (x0, y0, x1, y1)
# given as offsets from top left measured in segment lengths.

offsets = (
    (0, 0, 1, 0),  # top
    (1, 0, 1, 1),  # upper right
    (1, 1, 1, 2),  # lower right
    (0, 2, 1, 2),  # bottom
    (0, 1, 0, 2),  # lower left
    (0, 0, 0, 1),  # upper left
    (0, 1, 1, 1),  # middle
)
# Segments used for each digit; 0, 1 = off, on.
digits = (
    (1, 1, 1, 1, 1, 1, 0),  # 0
    (0, 1, 1, 0, 0, 0, 0),  # 1
    (1, 1, 0, 1, 1, 0, 1),  # 2
    (1, 1, 1, 1, 0, 0, 1),  # 3
    (0, 1, 1, 0, 0, 1, 1),  # 4
    (1, 0, 1, 1, 0, 1, 1),  # 5
    (1, 0, 1, 1, 1, 1, 1),  # 6
    (1, 1, 1, 0, 0, 0, 0),  # 7
    (1, 1, 1, 1, 1, 1, 1),  # 8
    (1, 1, 1, 1, 0, 1, 1),  # 9
    (1, 1, 1, 0, 1, 1, 1),  # 10=A
    (0, 0, 1, 1, 1, 1, 1),  # 11=b
    (1, 0, 0, 1, 1, 1, 0),  # 12=C
    (0, 1, 1, 1, 1, 0, 1),  # 13=d
    (1, 0, 0, 1, 1, 1, 1),  # 14=E
    (1, 0, 0, 0, 1, 1, 1),  # 15=F
    (1, 0, 0, 0, 0, 0, 0),  # Top horiz Bar
    (0, 1, 0, 0, 0, 0, 0),  # Top Right Vert
    (0, 0, 1, 0, 0, 0, 0),  # Bot right Vert
    (0, 0, 0, 1, 0, 0, 0),  # Bot horiz
    (0, 0, 0, 0, 1, 0, 0),  # Bot Left Vert
    (0, 0, 0, 0, 0, 1, 0),  # Top Left Vert
    (0, 0, 0, 0, 0, 0, 1),  # Ctr horiz bar
)
name_seven_seg    = 'seven_seg.py'
version_seven_seg = '1.0.0'
date_seven_seg    = '02/05/21'
author_seven_seg  = 'Peter A. Hedlund'

LIB_NAME = name_seven_seg
LIB_VERSION = version_seven_seg
LIB_DATE = date_seven_seg
LIB_AUTHOR = author_seven_seg

# LIB_NAME = 'seven_seg.py'
# LIB_VERSION = "1.0.0"
# LIB_DATE = '02/05/21'


def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION


def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs

#
# 7 Segment LED DISPLAY SECTION
#
class Digit_Led:
    """ Creates a 7 segment led type display (one character)"""
    def __init__(self, canvas, *, x=10, y=10, length=20, width=3,
                 myfill='red'):
        self.canvas = canvas
        self.segs7 = []
        for x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6 in segments:
            self.segs7.append(canvas.create_polygon(
                x+x1, y+y1, x+x2, y+y2, x+x3, y+y3, x+x4, y+y4, x+x5, y+y5,
                x+x6, y+y6,x+x1, y+y1, fill = myfill, state='hidden'))


    def show(self, num):
        """ Show the number num on the single character led 7 segment display
        """
        for iid, on in zip(self.segs7, digits[num]):
            self.canvas.itemconfigure(iid, state = 'normal' if on else 'hidden')


class Counter_Led:
    """Creates a set of 7 segment led style (dig = num characters)"""
    def __init__(self, canvas, *, x=10, y=10, dig=4, base=10, myfill='red'):
        self.canvas = canvas
        self.dig = dig
        self.base = base
        self.poss=[]
        for w in range(0, self.dig):
            self.poss.append(Digit_Led(canvas, x = w*30 + x, y=y,
                        myfill=myfill))


    def num(self, n):
        """ Shows the number on counter created if number > digits
        number wraps around 3 digits 1000 is 000, 1001 is 001"""
        div = (1000000, 100000, 10000,1000,100,10,1)
        div_h = (16777216, 1048576, 65536, 4096, 256, 16, 1)
        st = len(div) - self.dig
        for q in range(0, self.dig):
            if self.base == 16:
                self.poss[q].show(int(n/div_h[st+q]) % 16)
            else:
                self.poss[q].show(int(n/div[st+q]) % 10)

class Counter_Led_Dp:
    """Creates a set of 7 segment led style (dig = num characters, dp
    location of decimal point)"""
    def __init__(self, canvas, *, x=10, y=10, dig=4, base=10, dp=1,
                myfill='red'):
        self.canvas = canvas
        self.dig = dig
        self.base = base
        self.dp = dp
        self.poss=[]
        for w in range(0, self.dig):
            self.poss.append(Digit_Led(canvas, x = w*34 + x, y=y, myfill=myfill))
        if dp > 0:
            x1 = x-10 + ((dig - dp) * 34)
            x2 = x1 + 5
            canvas.create_oval(x1, y + 31, x2, y + 37,  fill=myfill)


    def num(self, n):
        """ Shows the number on counter created if number > digits
        number wraps around 3 digits 1000 is 000, 1001 is 001"""
        mul = (10,100,1000,10000)
        i = int(n)
        r = n - i
        ad = int(r * mul[self.dp-1])
        v = '00000000' + str(i) + str(ad)
        ln = len(v)
        for x in range(0, self.dig):
            self.poss[x].show(int(v[ln - self.dig + x]))



class Clock_Led:
    """ Creates a 7 segment line clock in either HH:MM or HH:MM:SS.
    mode 0 = HH:MM mode 1 = HH:MM:SS. """
    def __init__(self,canvas, *, x=10, y=10, mode=0, myfill='red'):
        self.canvas = canvas
        self.mode = mode
        self.poss=[]
        dig = 4
        if self.mode==1:
            dig = 6
        for w in range(0, dig):
            self.poss.append(Digit_Led(canvas, x = w*34 + x, y=y,
                myfill=myfill))

        x1 = x-10 + ((4 - 2) * 34)
        x2 = x1 + 5
        canvas.create_oval(x1, y + 11, x2, y + 17,  fill=myfill)
        canvas.create_oval(x1, y + 31, x2, y + 37,  fill=myfill)
        if mode == 1:
            x1 = x-10 + ((dig - 2) * 34)
            x2 = x1 + 5
            canvas.create_oval(x1, y + 11, x2, y + 17,  fill=myfill)
        canvas.create_oval(x1, y + 31, x2, y + 37,  fill=myfill)


    def num(self, h,m,s=0):
        """ Shows the time (Hh:MM:SS) """

        self.poss[0].show(int(h/10) % 10)
        self.poss[1].show((h) % 10)
        self.poss[2].show(int(m/10) % 10)
        self.poss[3].show(m % 10)

        if self.mode == 1:
            self.poss[4].show(int(s/10) % 10)
            self.poss[5].show(s % 10)

#
# 7 Segment Line DISPLAY SECTION
#

class Digit:
    """ Creates a 7 segment line type display (one character)"""
    def __init__(self, canvas, *, x=10, y=10, length=23, width=3,
                myfill='green'):
        self.canvas = canvas
        l = length
        self.segs = []
        for x0, y0, x1, y1 in offsets:
            self.segs.append(canvas.create_line(
                x + x0*l, y + y0*l, x + x1*l, y + y1*l,
                width=width, state = 'hidden', fill=myfill))


    def show(self, num):
        """ Show the number num on the single character line 7 segment display
        """
        for iid, on in zip(self.segs, digits[num]):
            self.canvas.itemconfigure(iid, state = 'normal' if on else 'hidden')


class Counter:
    """Creates a set of 7 segment line style (dig = num characters)"""
    def __init__(self, canvas, *, x=10, y=10, dig=4, base=10, myfill='green'):
        self.canvas = canvas
        self.dig = dig
        self.base = base
        self.poss=[]
        for w in range(0, self.dig):
            self.poss.append(Digit(canvas, x = w*30 + x, y=y, myfill=myfill))


    def num(self, n):
        """ Shows the number on counter created if number > digits
        number wraps around 3 digits 1000 is 000, 1001 is 001"""
        div = (1000000, 100000, 10000,1000,100,10,1)
        div_h = (16777216, 1048576, 65536, 4096, 256, 16, 1)
        st = len(div) - self.dig
        for q in range(0, self.dig):
            if self.base == 16:
                self.poss[q].show(int(n/div_h[st+q]) % 16)
            else:
                self.poss[q].show(int(n/div[st+q]) % 10)


class Counter_Dp:
    """Creates a set of 7 segment line style (dig = num characters, dp
    location of decimal point)"""
    def __init__(self, canvas, *, x=10, y=10, dig=4, base=10, dp=1,
                myfill='green'):
        self.canvas = canvas
        self.dig = dig
        self.base = base
        self.dp = dp
        self.poss=[]
        for w in range(0, self.dig):
            self.poss.append(Digit(canvas, x = w*34 + x, y=y, myfill=myfill))
        if dp > 0:
            x1 = x-8 + ((dig - dp) * 34)
            x2 = x1 + 5
            canvas.create_oval(x1, y + 31, x2, y + 37,  fill=myfill)


    def num(self, n):
        """ Shows the number on counter created if number > digits
        number wraps around 3 digits 1000 is 000, 1001 is 001"""
        mul = (10,100,1000,10000)
        i = int(n)
        r = n - i
        ad = int(r * mul[self.dp-1])
        v = '00000000' + str(i) + str(ad)
        ln = len(v)
        for x in range(0, self.dig):
            self.poss[x].show(int(v[ln - self.dig + x]))


class Clock:
    """ Creates a 7 segment line clock in either HH:MM or HH:MM:SS.
    mode 0 = HH:MM mode 1 = HH:MM:SS. """
    def __init__(self,canvas, *, x=10, y=10, mode=0, myfill='green'):
        self.canvas = canvas
        self.mode = mode
        self.poss=[]
        dig = 4
        if self.mode==1:
            dig = 6
        for w in range(0, dig):
            self.poss.append(Digit(canvas, x = w*34 + x, y=y, myfill=myfill))
        x1 = x-8 + ((4 - 2) * 34)
        x2 = x1 + 5
        canvas.create_oval(x1, y + 11, x2, y + 17,  fill=myfill)
        canvas.create_oval(x1, y + 31, x2, y + 37,  fill=myfill)
        if mode == 1:
            x1 = x-8 + ((dig - 2) * 34)
            x2 = x1 + 5
            canvas.create_oval(x1, y + 11, x2, y + 17,  fill=myfill)
            canvas.create_oval(x1, y + 31, x2, y + 37,  fill=myfill)


    def num(self, h,m,s=0):
        """ Shows the time (Hh:MM:SS) """

        self.poss[0].show(int(h/10) % 10)
        self.poss[1].show((h) % 10)
        self.poss[2].show(int(m/10) % 10)
        self.poss[3].show(m % 10)

        if self.mode == 1:
            self.poss[4].show(int(s/10) % 10)
            self.poss[5].show(s % 10)



if __name__ == "__main__":

    def update():
        global n
        # update 4 individual line characters the hard way
    dig1.show(int(n/1000) % 10)
    dig2.show(int(n/100) % 10)
    dig3.show(int(n/10) % 10)
    dig4.show(n % 10)
    # update 4 individual led characters the hard way
    dig1L.show(int(n/1000) % 10)
    dig2L.show(int(n/100) % 10)
    dig3L.show(int(n/10) % 10)
    dig4L.show(n % 10)

    dig5.show((n % 6) + 16)
    dig5L.show((n % 6 ) + 16)
    # update 5 character line  display the easy way
    ctr4.num(n)
    # update 4 character led  display the easy way
    ctr5.num(n)

    tm.num(0,int(n/60) %60, n%60)
    n = (n+1) % 100000
    root.after(100, update)


root = tk.Tk()
screen = tk.Canvas(root, bg='black', width=430)
screen.grid()

# 4 character line 7 segment display the hard way
dig1 = Digit(screen)
dig2 = Digit(screen, x=40)
dig3 = Digit(screen, x=70)
dig4 = Digit(screen, x=100)
dig5 = Digit(screen, x = 290)
# 4 character led 7 segment display the hard way
dig1L = Digit_Led(screen, x=150)
dig2L = Digit_Led(screen, x=180)
dig3L = Digit_Led(screen, x=210)
dig4L = Digit_Led(screen, x=240)
dig5L = Digit_Led(screen, x = 330)
n = 0
# 5 character line 7 segment display (decimal) the easy way
ctr4 = Counter(screen, y=70, dig=5 )
# 4 character led 7 segment display (hex) the easy way
ctr5 = Counter_Led(screen, x=180, y=70, base=16, myfill='navy')
tm = Clock(screen, y=140, mode=1 )
tm.num(1,12,37)
tm2 = Clock_Led(screen, x = 220, y=140, mode=1)
```

```
tm2.num(5,22,37)
ctr6 = Counter_Dp(screen, y = 210,dig=6, dp=2)
ctr6.num(43.1450)
ctr7 = Counter_Led_Dp(screen, x=225, y=210, dig=5, dp=2)
ctr7.num(43.21435)

root.after(100, update)
root.mainloop()
```

## Sixteen_Seg.py

```
"""16 segment alphanumeric line display
   get_lib_version() Show numeric version of library.
   get_lib_full_version()  Show numeric version and library filename.
   class seg16Digit() creates 1 character of a 16 segment display
      show()  displays numeric value in character
   class strDisp() creates multiple 16 segment display characters
      show() displays the sting
   NOTE: Characters 0x20 to 0x7e have been encoded.
   Characters 0x80 - 0x8E are for making animations
   the show method of the strDisp class will automatically subtract
   0x20 from the characters in the string.
   """
import tkinter as tk

# Order 16 segments  or british flag display.
#
#   --a1-- --a2--
#  |\   |   /|
#  f  h   i   j  b
#  |  \  |  /  |
#  |   \|/   |
#   --g1-- --g2--
#  |    /|\    |
#  |   / | \   |
#  e   m  l  k   c
#  | /  |  \ |
#   --d1-- --d2--
#
# clockwise from top left, with crossbar last.
# Coordinates of each segment are (x0, y0, x1, y1)
# given as offsets from top left measured in segment lengths.

offsets = (
   (0, 0, 1, 0),  # a1 top
   (1, 0, 2, 0),  # a2 top
   (2, 0, 2, 1),  # b upper right
   (2, 1, 2, 2),  # c lower right
   (1, 2, 2, 2),  # d2 bottom
   (0, 2, 1, 2),  # d1 bottom
   (0, 1, 0, 2),  # e lower left
   (0, 0, 0, 1),  # f upper left
   (0, 1, 1, 1),  # g1 middle
   (1, 1, 2, 1),  # g2 middle
   (0, 0, 1, 1),  # h left diag top
```

```
    (1, 0, 1, 1),  # i ctr vert top
    (2, 0, 1, 1),  # j right diag top
    (1, 1, 2, 2),  # k right diag btm
    (1, 1, 1, 2),  # l ctr vert btm
    (1, 1, 0, 2),  # m left diag btm

)

# ascii character 0x20 through 0x7e
# offset to numbers is 0x20, offset to letters is 0x20
# Segments used for each digit; 0, 1 = off, on.
digits = (
   # a1 a1 b  c  d2 d1 e  f  g1 g2 h  i  j  k  l  m
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x20 space
    (1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0),  # 0x21 !
    (0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0),  # 0x22 "
    (0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0),  # 0x23 #
    (1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0),  # 0x24 $
    (0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1),  # 0x25 %
    (1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0),  # 0x26 &
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0),  # 0x27 '
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0),  # 0x28 (
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1),  # 0x29 )
    (0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1),  # 0x2A *
    (0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0),  # 0x2B +
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1),  # 0x2C ,
    (0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x2D -
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0),  # 0x2E .
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1),  # 0x2F /
    (1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1),  # 0x30 0
    (0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x31 1
    (1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x32 2
    (1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x33 3
    (0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x34 4
    (1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x35 5
    (1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x36 6
    (1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x37 7
    (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x38 8
    (1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x39 9
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),  # 0x3A :
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1),  # 0x3B ;
    (0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1),  # 0x3C <
    (0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x3D =
    (0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0),  # 0x3E >
    (1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0),  # 0x3F ?
```

```
 (1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0),  # 0x40 @
# a1 a1 b  c  d2 d1 e  f  g1 g2 h  i  j  k  l  m
 (1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x41 A
 (1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0),  # 0x42 B
 (1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x43 C
 (1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),  # 0x44 D
 (1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x45 E
 (1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x46 F
 (1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0),  # 0x47 G
 (0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x48 H
 (1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0),  # 0x49 I
 (0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x4A J
 (0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0),  # 0x4B K
 (0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x4C L
 (0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0),  # 0x4D M
 (0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0),  # 0x4E N
 (1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x4F O
 (1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x50 P
 (1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0),  # 0x51 Q
 (1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0),  # 0x52 R
 (1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0),  # 0x53 S
 (1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),  # 0x54 T
 (0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x55 U
 (0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1),  # 0x56 V
 (0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1),  # 0x57 W
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1),  # 0x58 X
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0),  # 0x59 Y
 (1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1),  # 0x5A Z
# a1 a1 b  c  d2 d1 e  f  g1 g2 h  i  j  k  l  m
 (0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),  # 0x5B [
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0),  # 0x5C \
 (1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),  # 0x5D ]
 (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0),  # 0x5E ^
 (0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x5F _
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0),  # 0x60 `

# a1 a1 b  c  d2 d1 e  f  g1 g2 h  i  j  k  l  m
 (1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x61 a
 (0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x62 b
 (0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x63 c
 (0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x64 d
 (1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x65 e
 (0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0),  # 0x66 f
 (1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x67 g
 (0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x68 h
 (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0),  # 0x69 i
```

```
    (0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x6A j
    (0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0),  # 0x6B k
    (0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),  # 0x6C l
    (0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0),  # 0x6D n
    (0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x6E m
    (0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x6F o
    (1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0),  # 0x70 p
    (1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0),  # 0x71 q
    (0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x72 r
    (0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0),  # 0x73 s
    (0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0),  # 0x74 t
    (0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x75 u
    (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0),  # 0x76 v
    (0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1),  # 0x77 w
    (0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1),  # 0x78 x
    (0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x79 y
    (0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1),  # 0x7A z
    (0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0),  # 0x7B {
    (0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x7C |
    (1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0),  # 0x7D }
    (0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0),  # 0x7E ~
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x7f

   #  Diagnostics / animation frames 0x80 and 0x87
    (1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x80 top horiz Bar
    (0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x81 Top Right Vert
    (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x82 Bot right Vert
    (0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x83 Bot horiz
    (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x84 Bot Left Vert
    (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0),  # 0x85 Top Left Vert
    (0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0),  # 0x86 Ctr horiz bar
    (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0),  # 0x87 ctr left horiz
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0),  # 0x88 top left slant
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0),  # 0x89 top ctr vert
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0),  # 0x8a top right slant
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0),  # 0x8b ctr right vert
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0),  # 0x8c bot right slant
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0),  # 0x8d bot ctr vert
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1),  # 0x8e bot left slant
)
name_sixteen_seg    = 'sixteen_seg.py'
version_sixteen_seg = '1.0.0'
date_sixteen_seg    = '02/05/21'
author_sixteen_seg  = 'Peter A. Hedlund'
```

```
LIB_NAME = name_sixteen_seg
LIB_VERSION = version_sixteen_seg
LIB_DATE = date_sixteen_seg
LIB_AUTHOR = author_sixteen_seg

# LIB_NAME = 'sixteen_seg.py'
# LIB_VERSION = "1.0.0"
# LIB_DATE = '02/05/21'

def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION


def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs

#
# 16 Segment Line DISPLAY SECTION
#

class seg16Digit:
    """Creates a 16 segment line type display (one alphanumeric character)"""
    def __init__(self, canvas, *, x=10, y=10, length=23, width=3,
                myfill='green'):
        self.canvas = canvas
        self.myfill = myfill
        l = length
        w = length/2
        self.segs = []
        for x0, y0, x1, y1 in offsets:
            self.segs.append(canvas.create_line(
                x + x0*w, y + y0*l, x + x1*w, y + y1*l,
                width=width, state = 'hidden', fill=myfill))


    def show(self, num, color='green'):
        """Show the character number num  on the single character line
        16 segment display in the designated color"""
        for iid, on in zip(self.segs, digits[num]):
            self.canvas.itemconfigure(iid, state = 'normal' if on else 'hidden',
                              fill = color)


class StrDisp:
    """Creates a set of 16 segment line style (dig = num characters)"""
    def __init__(self, canvas, *, x=10, y=10, dig=4, myfill='green'):
        self.canvas = canvas
        self.dig = dig
        self.poss=[]
        for w in range(0, self.dig):
            self.poss.append(seg16Digit(canvas, x = w*32 + x, y=y, myfill=myfill))


    def show(self, st, color='green'):
        """Shows the string converted to uppercase in the display in the designated
          color."""
        ep = self.dig
        if len(st) < ep:
            ep = len(st)
        for q in range(0, ep):
            v = ord(st[q])
            self.poss[q].show(v - 0x20, color)


if __name__ == "__main__":

    import time

    def update():
        global a, b, n, toc
        a = (a+1) % 15
        toc = (toc+1) % 10
        if toc == 0:
            n = (n+1) % 30
            b = (b+1) % 0x41
        dig1.show(0x60 + a, 'red')
        dig2.show( b, 'red')
        digx.show(0x41 + n, 'cyan')
        tm = time.localtime()
        str1 = time.strftime('%H %M %S', tm)
        dsp6.show(str1, 'royalblue')
        root.after(100, update)


    def quit_prog():
```

```
    root.destroy()


root = tk.Tk()
screen = tk.Canvas(root, bg='black', width=430, height=200)
screen.grid(column=0, row=0, columnspan=5)
bt = tk.Button(root, text='   Quit   ', command=quit_prog)
bt.grid(column=0, row=1)
screen2 = tk.Canvas(root, bg='gray10', width=430, height=70)
screen2.grid(column=0, row=2, columnspan=5)
n = 0;  a = 0;  b = 0; toc = 0
# 4 character line 7 segment display the hard way
dig1 = seg16Digit(screen, x=395)
dig1.show(0x41, 'red')
dig2 = seg16Digit(screen, x=395, y=120)
dig2.show(0x48, 'red')
digx = seg16Digit(screen, x=395, y=64)
digx.show(0x61, 'red')
dsp3 = StrDisp(screen, dig=12)
dsp3.show('ABCDEFGHIJK')
dsp4 = StrDisp(screen, y=64, dig=12)
dsp4.show('LMNOPQRSTUV','yellow')
dsp5 = StrDisp(screen, y=120, dig=12)
dsp5.show('WXYZ()[]  ')
dsp6 = StrDisp(screen2, x=10, y=19, dig=13)
dsp6.show('NCC-1701 TIME', 'royalblue')
root.after(1000, update)
root.mainloop()
```

## To_Log.py

```
#
# pylint: disable=unused-wildcard-import, method-hidden
#
""" Logging functions Library

    get_lib_version() Show numeric version of library.
    get_lib_full_version()  Show numeric version and library filename.
    to_log()  sends string to text window in last color
    to_log_red()  sends string to text window in red
    to_log_yellow()  sends string to text window in  yellow
    to_log_green()  sends string to text window in  green
    to_log_blue()  sends string to text window in  blue
    to_log_cyan()  sends string to text window in  cyan
    to_log_white()  sends string to text window in  white
    to_log_violet()  sends string to text window in  violet
    clear_log()  clears the text window
    save_log()  saves the text window to a user selected file
"""

from tkinter import *
from tkinter import filedialog
name_to_log    = 'to_log.py'
version_to_log = '2.2.0'
date_to_log    = '04/29/21'
author_to_log  = 'Peter A. Hedlund'

LIB_NAME = name_to_log
LIB_VERSION = version_to_log
LIB_DATE = date_to_log
LIB_AUTHOR = author_to_log

# LIB_NAME = "to_log.py"
# LIB_VERSION = "2.1.0"
# LIB_DATE = '02/09/21'

def get_lib_version():
    """    Returns version information only. """
    return LIB_VERSION


def get_full_lib_version():
    """Returns version information and library name."""
    msg = get_lib_version()
    rs = 'Library Name : ' + LIB_NAME + '\nVersion : ' + msg
    rs = rs + '\nDate : ' + LIB_DATE
    rs = rs + '\nAuthor : ' + LIB_AUTHOR + '\n'
    return rs



def to_log(txt, tstr):
    """Sends string to text window. """

    txt.configure(state='normal')
    txt.insert(END, tstr)
    txt.index(END)
    txt.see(END)
    txt.update()
    txt.configure(state='disabled')


def to_log_red(txt, tstr):
    """Sends string to text window in red """
    # Enable changes to text widget
    txt.configure(state='normal')
    # position cursor at the end of the text
    txt.mark_set(INSERT, END)
    #  get starting text position
    st = txt.index('insert')
    #  Add text to text widget at the end
    txt.insert(END, tstr)
    #  get ending text position
    ed = txt.index('insert')
    # print "Start = {}  End = {}  String = {}".format(st,ed,str)   #  debug
    #  tag area bound by start and stop
    txt.tag_add("junk1", st, ed)
    #  Change color of tagged area
    txt.tag_config("junk1", foreground='red')
    #  Go to end of text window (auto scroll)
    txt.see(END)
    #  Update idle tasks
    txt.update_idletasks()
    # disable changes to text widget
    txt.configure(state='disabled')


def to_log_yellow(txt, tstr):
    """Sends string to text window in yellow """
    # Enable changes to text widget
    txt.configure(state='normal')
    # position cursor at the end of the text
    txt.mark_set(INSERT, END)
    #  get starting text position
    st = txt.index('insert')
    #  Add text to text widget at the end
    txt.insert(END, tstr)
    #  get ending text position
    ed = txt.index('insert')
    # print "Start = {}  End = {}  String = {}".format(st,ed,str)   #  debug
    #  tag area bound by start and stop
    txt.tag_add("junk2", st, ed)
    #  Change color of tagged area
    txt.tag_config("junk2", foreground='yellow')
    #  Go to end of text window (auto scroll)
    txt.see(END)
    #  Update idle tasks
    txt.update_idletasks()
    # disable changes to text widget
    txt.configure(state='disabled')


def to_log_green(txt, tstr):
    """Sends string to text window in green """
    # Enable changes to text widget
    txt.configure(state='normal')
    # position cursor at the end of the text
    txt.mark_set(INSERT, END)
    #  get starting text position
    st = txt.index('insert')
    #  Add text to text widget at the end
    txt.insert(END, tstr)
    #  get ending text position
    ed = txt.index('insert')
    # print "Start = {}  End = {}  String = {}".format(st,ed,str)   #  debug
    #  tag area bound by start and stop
    txt.tag_add("junk3", st, ed)
    #  Change color of tagged area
    txt.tag_config("junk3", foreground='green')
    #  Go to end of text window (auto scroll)
    txt.see(END)
    #  Update idle tasks
    txt.update_idletasks()
    # disable changes to text widget
    txt.configure(state='disabled')
```

```
def to_log_blue(txt, tstr):
    """Sends string to text window in blue """
    # Enable changes to text widget
    txt.configure(state='normal')
    # position cursor at the end of the text
    txt.mark_set(INSERT, END)
    #  get starting text position
    st = txt.index('insert')
    #  Add text to text widget at the end
    txt.insert(END, tstr)
    #  get ending text position
    ed = txt.index('insert')
    # print "Start = {}  End = {}  String = {}".format(st,ed,str)   #  debug
    #  tag area bound by start and stop
    txt.tag_add("junk4", st, ed)
    #  Change color of tagged area
    txt.tag_config("junk4", foreground='lightblue')
    #  Go to end of text window (auto scroll)
    txt.see(END)
    #  Update idle tasks
    txt.update_idletasks()
    # disable changes to text widget
    txt.configure(state='disabled')


def to_log_cyan(txt, tstr):
    """Sends string to text window in cyan """
    # Enable changes to text widget
    txt.configure(state='normal')
    # position cursor at the end of the text
    txt.mark_set(INSERT, END)
    #  get starting text position
    st = txt.index('insert')
    #  Add text to text widget at the end
    txt.insert(END, tstr)
    #  get ending text position
    ed = txt.index('insert')
    # print "Start = {}  End = {}  String = {}".format(st,ed,str)   #  debug
    #  tag area bound by start and stop
    txt.tag_add("junk5", st, ed)
    #  Change color of tagged area
    txt.tag_config("junk5", foreground='cyan')
    #  Go to end of text window (auto scroll)
    txt.see(END)
    #  Update idle tasks
    txt.update_idletasks()
    # disable changes to text widget
    txt.configure(state='disabled')


def to_log_white(txt, tstr):
    """Sends string to text window in white """
    # Enable changes to text widget
    txt.configure(state='normal')
    # position cursor at the end of the text
    txt.mark_set(INSERT, END)
    #  get starting text position
    st = txt.index('insert')
    #  Add text to text widget at the end
    txt.insert(END, tstr)
    #  get ending text position
    ed = txt.index('insert')
    # print "Start = {}  End = {}  String = {}".format(st,ed,str)   #  debug
    #  tag area bound by start and stop
    txt.tag_add("junk6", st, ed)
    #  Change color of tagged area
    txt.tag_config("junk6", foreground='lightgrey')
    #  Go to end of text window (auto scroll)
    txt.see(END)
    #  Update idle tasks
    txt.update_idletasks()
    # disable changes to text widget
    txt.configure(state='disabled')


def to_log_violet(txt, tstr):
    """Sends string to text window in violet """
    # Enable changes to text widget
    txt.configure(state='normal')
    # position cursor at the end of the text
    txt.mark_set(INSERT, END)
    #  get starting text position
    st = txt.index('insert')
    #  Add text to text widget at the end
    txt.insert(END, tstr)
    #  get ending text position
    ed = txt.index('insert')
    # print "Start = {}  End = {}  String = {}".format(st,ed,str)   #  debug
    #  tag area bound by start and stop
    txt.tag_add("junk7", st, ed)
    #  Change color of tagged area
    txt.tag_config("junk7", foreground='violet')
    #  Go to end of text window (auto scroll)
    txt.see(END)
    #  Update idle tasks
    txt.update_idletasks()
    # disable changes to text widget
    txt.configure(state='disabled')


def clear_log(txt):
    """Clear the text widget. """
    #  Enable writing to text widget
    txt.configure(state='normal')
    #  Delete from first position to end
    txt.delete(1.0, END)
    #  Disable writing to text widget
    txt.configure(state='disabled')


def save_log(txt, main):
    """Get all text from existing text widget and save it to a
    user defined text file. """
    lines = txt.get("1.0", END).splitlines()
    path = filedialog.asksaveasfilename(parent=main)
    if path == '':
        return
    fo = open(path, 'w', encoding='utf-8')
    for line in lines:
        fo.write(line + "\n")
    fo.close()


if __name__ == "__main__":
    from tkinter import *
    from tkinter import ttk


    def quit_prog():
        root.destroy()


    def my_clear_log():
        clear_log(txt)
```

```
  def my_save_log():
     save_log(txt, mainframe)


  def test_log():
     to_log(txt, 'Name:\n' + get_full_lib_version() + '\n')
     to_log(txt, 'Sample Text Normal\n')
     to_log_blue(txt, 'Sample Text Blue\n')
     to_log_cyan(txt, 'Sample Text Cyan\n')
     to_log_red(txt, 'Sample Text Red\n')
     to_log_violet(txt, 'Sample Text Violet\n')
     to_log_yellow(txt, 'Sample Text Yellow\n')
     to_log_green(txt, 'Sample Text Green\n')
     to_log_white(txt, 'Sample Text White\n')


  root = Tk()
  #  prevent window resizing.
  root.resizable(0, 0)
  # Replace tk icon with your own.
  root.title('Testing Log file')

  mainframe = ttk.Frame(root, padding="3", height=680, width=650)
  mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
  mainframe.grid_propagate(0)

  clearlog = ttk.Button(mainframe, text="Clear Log", style='cyan.TButton', command=my_clear_log,
width=15)
  clearlog.grid(column=0, row=1, padx=15, pady=5)
  savelog = ttk.Button(mainframe, text="Save Log", style='cyan.TButton', command=my_save_log,
width=15)
  savelog.grid(column=1, row=1, padx=15, pady=5)
  testProg = ttk.Button(mainframe, text="TEST", style='red.TButton', command=test_log, width=15)
  testProg.grid(column=0, row=2, padx=15, pady=5)
  quitProg = ttk.Button(mainframe, text="Quit", style='red.TButton', command=quit_prog,
width=15)
  quitProg.grid(column=1, row=2, padx=15, pady=5)
  s = 'Filename : ' + get_full_lib_version() + ']\n'
  lab2 = Label(mainframe,text=s)
  lab2.grid(column=2,row=0, columnspan=6)

  #  Text box Widget
  txt = Text(mainframe, width=45, height=34, fg='lightgreen', bg='black', padx=15, pady=15)
  txt.grid(column=2, row=1, columnspan=6, rowspan=20, sticky="nsew")
txt.insert(1.0, "Start log file\n")
txt.configure(state='disabled')

#  Scrollbar linked to text box above
scrollb = ttk.Scrollbar(mainframe, command=txt.yview)
scrollb.grid(row=1, column=7, rowspan=20, sticky='nse')
txt['yscrollcommand'] = scrollb.set

for child in mainframe.winfo_children():
    child.grid_configure(padx=5, pady=5)

root.mainloop()
```