Python and Tkinter Notes

by

Peter Hedlund

Revised 08/20/21

Table of Contents

Forward

About The Author

```
This Document was created by Peter Hedlund (p_hedlund@hotmail.com).
I have been a programmer since 1982 being self taught on Assembler for the 8051,
8098, z80, Basic, C, Modula 2, Pascal, PLM, Python. During my career I have
been guided by others. I started creating a list of notes on python about 10 years
ago because I couldn't remember the tips and tricks since I usually worked with
more than one language.
Python is my 'Crack' language.  The language I go to when frustrated and need to
clear my head.
In the spirit of being helped, I freely give this document out hoping it helps
others to save a bit of time in their coding.
Enjoy. Pete
```

About this document

```
This document was created for use with:
      Python 2.7.4,
      Python 3.7.4
      PyCharm Community Edition
      PyScripter
      Sublime Text
NOTE:  I am currently updating the samples to python 3.7.4 since Python 2.7.4 is no
longer be supported. This work is ongoing. I will also update this document with
new tips or improved old tips. If you find a mistake Please e-mail me.
I used Idle to help with some of hints in this document.
```

Recommended Reading

Python Notes for Professionals – GoalKicker.com

Python 2.7 quick Reference – New Mexico Tech www.nmt.edu/tcc

Tkinter 8.5 reference: a GUI for Python – New Mexico Tech www.nmt.deu/tcc

Python 3.2 quick reference – New Mexico Tech www.nmt.deu/tcc

Sources for Information

www.python.org

www.sourceforge.com

www.stackabuse.com

www.pythhon-course.eu

www.tutorialspoint.com/python

My Module (Library)

I have created library files that are copied to the Pythonxx/Lib Directory my_lib. The source code for them is contained in a separate document. These files have been converted to python 3.7.4.

Filename	Purpose
about.py	The about_box function shown in the Tkinter section
box_char	16 bit unicode characters for on screen boxes
clock_face.py	Draws a clock face and hands
env.py	Edit and change Enviromental variables.
file_log.py	Log data to a file
gage.py	Creates a gage and sets the pointer
help.py	A simple Help screen
knob.py	Creates a knob widget that is set via program
led_d.py	Draws a dual led (one on, one off)
led.py	On Screen round led indications shown in the Tkinter section
led_sq.py	On Screen square led indications shown in the Tkinter section
low_ascii	Contains character definitions for ascii 0 - 31
my_math.py	Contains several math routines.
p_types	C style type definitions for python structures
scrl_notebook.py	Scrollable notebook (similar to ttk.Notbook)
scrn_log.py	Sends text to a scrolling text widget (Use instead of to_log)
seven_seg.py	Creates a 7 segment graphic display
sixteen_seg.py	Creates a 16 segment graphic display
to_log.py	All the to_log functions shown in the Tkinter section (legacy code only)

Installation

Files Needed

Python (in this document I will use version 3.73) (32 bit version)	python-3.7.3.exe
Python windows 32 extensions	pywin32_227.win32-py3.7.exe
Not required but recommended to install pyscripter ide	PyScripter-3.6.3-x86-Setup.exe

Installation

```
For the purpose of this document, I am installing python on the C drive and in the
Directory User, not to be confused with Users.
Run python-3.7.3.exe
This will install python and the tkinter graphics and widgets
Upon completion
Run pywin32_227.win32-py3-7.exe
This will install the windows extensions for python.
```

Environmental Variables

To the path Add both

c:/user/python37-32

c:/user/pytyon37-32/Scripts

set PYTHONHOME = c:/user/pytyon37-32

set PYTHONPATH = c:/user/python37-32/LIB

Installing Libraries

From the command prompt go to the directory your python.exe is stored in. In this example we will install auto-py-to-exe.

pip install auto-py-to-exe

To Upgrade it

pip install auto-py-to-exe --upgrade

or on pyscripter

Tools→Tools→Install Packages with Pip

Python

Types and Conversions

TYPES	Example	Decimal
Binary	0b10000001 (binary)	129
Hex	0x81 (hexidecimal)	129
Octal	0o201 (Octal)	129

CONVERSION	USE	RESULT
Binary	A = bin(129)	A = ‘0b10000001’
Div and Mod	A, b =Divmod(100,30)	A=3, b = 10
Float	A = float(27)	A =27.0
Hexidecimal	A = hex(129)	A= ‘0x81’
Integer	A = int(‘129’)	A = 129
Integer	A = int(‘81’,16)	A = 129
Octal	A = oct(129)	A = ‘0o201’

General

Sample of try/except/else

```
def init_drive():
    COMPORT = txt1.get()
    try:
        #  port name, slave address (in decimal)
        instrument2 = minimalmodbus.Instrument(COMPORT, 1)
    except:
        to_log_err("Unable to open Com Port " + COMPORT + "\n")
        return 0
For a list of exceptions look in the Appendix.
```

Determine if running in vscode or from command line

```
import os
in_vscode = 1
try:
    lst = os.environ['TERM_PROGRAM']
except:
    in_vscode = 0
…..
if in_vscode == 0:
    input('Press Enter to continue')
```

Standard Doc String to identify Program

Use this format for a docstring to place after your includes to help describe the program.

```
"""
Title:   Decoder.py
Author:  Peter Hedlund
Created: 11/06/2017
Purpose: Decode FBUS_CW_1, FBUS_SW_1, ALARM_1 ALARM_2
"""
```

Find Directory Program is running from

c:/projects/python/test/test.py

```
import os

myloc = os.path.dirname(__file__)
print(f'Program test.py is running from directory {myloc}')
```

Program test.py is running from directory c:/projects/python/test

to make sure your ini file is in the same directory

```
import os

myloc = os.path.dirname(__file__)
print(f'Program test.py is running from directory {myloc}')

fn = os.path.dirname(__file__) + '/' + 'test.ini'

# fn now contains the full file name to the ini file

print(f'Ini File is at {fn})
```

Program test.py is running from directory c:/projects/python/test
Ini File is at c:/projects/python/test/test.ini

Arrays of methods (functions)

Assume that test_n00 through test_n12 are defined about this.

Declare the functions

```
error_func = [test_n00, test_n01, test_n02, test_n03, test_n05, test_n10, test_n11,
test_n12]

Declare the names

fault_name = ['N00 : Wait for ID', 'N01 : Undefined Cmd', 'N02 : Checksum Error',
              'N03 : Input Buffer Overrun', 'N05 : Data Field Error',
              'N10 : Invalid Data', 'N11 : Invalid Cmd', 'N12 : Cmd Not Accepted']


TO execute one of the methods based on an index  x (like from a combobox)
error_func[x]()
```

Array of strings

```
pv = [] # empty array
for x in range(0, 8)
    pv.append("MyVal{:02}".format(x))
results in pv = [MyVal00, MyVal02, MyVal03, MyVal04, MyVal05, MyVal06, MyVal07]
```

Create a list of Variable names

```
puz01 = (“abcdefg”)
puz02 = (“hijklmno”)
puz03 = (“pqrstu”)
puz04 = (“vwxyz”)
pl = [puz01, puz02, puz03, puz04]
puz = pl[0]    or if using a cobmo box  puz = pl[combobox.current()]
```

Setting Up an Interval Timer

```
Def quit():
    global stat
    root.after_close(stat)

def timed_read():
    global stat
    s = ''
    if (instrument.is_open):
        s = instrument.read(50)
        s = s.decode()
        parse2log(s)
        stat = root.after(200, timed_read)

timed_read()
```

In the example, when the main program calls timer_read it starts the function timed read after 200msec Once timed_read is called it sets itself up for additional timed reads every 200msec. Use after_close to shut down the timer (fix timer not found error.

Conditional Compile (RUN)

To conditionally use or not use code (for debugging)
DEBUG = 1 (or 0)

```
if _DEBUG_ == 1:
    do_this()
```

Allow input of strings with hex, octal or binary or decimal format.

i.e. “0x12cd”, “0o765”, “0b1011001” or “3235”

```
def get_decimal(str1):
    v = 0
    if str1.find('0x') >= 0:
        str1 = str1.lstrip('0x')
        str1 = str1.upper()
        for x in str1:
            if ((x < '0') | (x > '9')) & ((x < 'A') | (x > 'F')):
                return -1
        v = int(str1, 16)
    elif str1.find('0o') >= 0:
        str1 = str1.lstrip('0o')
        for x in str1:
            if (x < '0') | (x > '7'):
                return -1
        v = int(str1, 8)
    elif str1.find('0b') >= 0:
        str1 = str1.lstrip('0b')
        for x in str1:
            if (x < '0') | (x > '1'):
                return -1
        v = int(str1, 2)
    else:
        for x in str1:
            if (x < '0') | (x > '9'):
                return -1
        v = int(str1)
    return v
```

Literals

How to use binary, hex and octal literals in Python

Type	**A=**	**print(A)**	**A=**	**print(A)**
Binary	A=0b01100	12	A=0b10000000	128
Octal	A=01267	695	A=0100	64
HEX	A=0x12af	4783	A=0x100	256

Variables Scope (Globals)

When variables are declared they are considered global ...unless...you write to them.
For example

```
def show_my_path():
    print(my_path)

def edit_my_path():
    my_path = "test_path"

my_path = "testing/one/two"
show_my_path()
edit_my_path()
show_my_path()
```

Returns

```
testing/one/two
testing/one/two
```

Change edit_my_path to

```
def edit_my_path():
    global my_path
    my_path = "test_path"
```

then running the program again results in

```
testing/one/two
test_path
```

REMEMBER: if your going to change a variable declared outside the scope of the function make it a global.

Dictionaries

```
my_dict = {'one':1, 'two':2,'three':3,'four':4}
print(my_dict['two'])  … 2
my_dict['two'] = 4
print(my_dict['two'])  … 4
```

add to Dictionaries

```
my_dict['five'] = 5
print(my_dict['five']  … 5
```

get key if it is not there

```
print(my_dict.get('five', 0)  … 0
```

delete entry

```
del my_dict['five']
```

Clear all Entries

```
my_dict.clear()
```

delete dictionary

```
del my_dict
```

Determine Variable Types

```
To determine the type of a variable use the type() command
a = (1,2,3)
print(type(a))
<class 'tuple'>
```

Random Numbers

```
import random
random.seed()
random.random() # returns float between 0.0 and 0.999999999
random.randrange(75) # Returns 0-74
random.randrange(5, 76) returns 5 - 75
random.randrange(0, 101, 5) # Returns 0,5,10,15,20,…,95, 100 (start, stop, step)
random.randint(0,5)  # returns 1 - 5
```

Disable Warning [unused import from wildcard import]

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

Importing Constants

```
If you plan to create a group of constants to put in a file to then import to other
python files,
DO NOT USE _ as the first character of the constant name.

p_type.py
_stx = 0x02

code.py
from p_type import *
print(_stx)
Will give you a _stx is not defined NameError

instead use
stx_ = 0x02
```

Functions

Simple Function – Returning a value

```
def  sampleFunc(val_in):
    val_out = val_in * 2
    return val_out
```

Returning multiple value from a function

```
def  sampleFunc(val_in):
    val_out_a = val_in * 2
    val_out_b = val_in * 3
    val_out_c = val_in * 4
    val_out_d = val_in * 5
    return val_out_a, val_out_b, val_out_c, val_out_d
```

Default arguments in functions

```
def sampleFunc(val1, val2=23):
      print (f'val1 = {val1}  val2 = {val2}')
sampleFunc(2)       →  2, 23
sampleFunc(3,2)    →  3, 2
```

Passing variable number of arguments to a function

```
def mult(*args):
    y = 1
    for num in args:
        y *= num
    print(num)
mult(3, 7)            → 21
mult(9, 8)            → 82
mult(3, 4, 7)        → 84
mult(5, 6, 10, 8)  → 21
```

Passing a function to a function

```
import time
def a():
    sleep(.3)
def b():
    sleep(.5)
def measure(func):
    t = time()
    func()
    print(func.__name__, " took ", time() -t)

measure(a)  # a took .3
measure(b) # b took .5
```

Getting Functions / variables from a library

```
dir(library)

Warning: this will produce a long list of variable names and functions,
in short a brute force method for looking into a library.
import mylib.to_log

dir (mylib.to_log)
(at the end of the list are the user functions
[ 'to_log',
 'to_log_blue',
 'to_log_cyan',
 'to_log_green',
 'to_log_red',
 'to_log_violet',
 'to_log_white',
 'to_log_yellow'
```

Screen Output / Keyboard Input

Using Print() without linefeeds

```
print("something", end="")
```

Colors with Print

```
# Note : make sure the following code snippets is in your code to enable ansi
escape codes.

from colorama import init
init()
# Limited color selection is possible with print using Ansi Escape Codes
print('\033[92m', '[ COLORED ], '\x1b[0m') (text is green)
```

Black	'\033[90m'	Red	'\033[91m'	Green	'\033[92m'	Yellow	'\033[93m'
Blue	'\033[94m'	Magenta	'\033[95m'	Cyan	'\033[96m'	White	'\033[97m'
30-37 = foreground		40,47 = background		90-97 foreground bright		100-107 background bright	

Reading Keyboard Input

```
Name = input("Enter Your First Name")
printf("Greetings ", Name)
```

Console KBHIT / GETCH

```
In the console mode, looking for a key to be hit and getting the keycode and
the extended keycode of the key that was pressed

import msvcrt

def wait():
    while msvcrt.kbhit() == False:
        pass
    key = msvcrt.getch()
    if ord(key) == 0:
        eext = msvcrt.getch()
        print(f'Extended Key Value returned {ord(eext)}')
    else :
        print(f'Key Value returned {ord(key)}')

print('The wait has started ')
wait()
print('The wait is over')

<TAB> - key = 9
<BACKSPACE> - Key = 8
<a> - Key = 97
<HOME> - Extended = 71
<END> - Extended 79
```

Using If / elif / else with different types

Numeric

```
a = 23
b = 14
if  a>b:
   print(“a greater”)
elif b > a:
   print(“b greater”)
else
   print(“a equals b”)
```

Check for inclusion using Sets

```
Ab = “ABCDEFGHIJKLMNOPQRSTUVWXYZ”
sm = “;:-.!?’”

‘b’ in Ab     (returns False)
‘B’ in Ab     (returns True)
‘.’ in sm     (returns True)
‘>’ in sm     (returns False)
instead of
if (b >= ‘A’) and (b <= ‘Z’):
    Do_this
use
if b in Ab:
    Do_this
```

Binary

```
A = True
b = False
if b is True:
   do_this()
if b is False
   do_that()
```

Real World

Locate all active serial ports

The first function gets all the active serial ports, and returns them in a dictionary. The second function allows the re-scanning of active serial ports and puts them in the combobox widget. The example after that shows how to use the serial_ports() function in creating the combobox widget.

```
def serial_ports():
    ports = ['COM%s' % (i + 1) for i in range(256)]
    result = []
    for port in ports:
        try:
            s = serial.Serial(port)
            s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass
    return result

def rescan_com_ports():
    ports = serial_ports()
    txt1.config(values=ports)
    txt1.update_idletasks();

#  load list into a combo box
ports = serial_ports()
#  Combo Box widget
txt1 = ttk.Combobox(mainframe, state="readonly", width=10, height=20, values=ports)
txt1.grid(column=0, row=0, columnspan=1)
txt1.current(0)
```

Operating System Running

```
The output of platform.system() is as follows:
    • Linux: Linux
    • Mac: Darwin
    • Windows: Windows
```

Paste Text into the clipboard

Putting Text into the windows clipboard

```
root.clipboard_clear()
s = 'Stuff to put into clipboard'
root.clipboard_append("'" + s + "'")
root.update()
```

Restart Python Program

```
import os
import sys

os.execv(sys.executable, ['Python'] + sys.argv
```

Execute a program from within Python

```
wstr=”Notepad.exe”
os.system(wstr)
--OR--
subprocess.call(“%s %s %s” % (“Notepad.exe”, “myfile.c”, “myfile.h”))
insure each item to pass has a corresponding %s
```

Execute a Python script from within Python

```
file = 'crypto_helper.pyw';
pprog = 'pythonw '
dir_path = ‘n:\projects\python\utilities\cyrpto2’ + ‘\\’
Called script is active calling script is inactive
    subprocess.call("%s %s" % (pprog, dir_path + file))
Called script and calling scripts are active
    subprocess.Popen(pprog + dir_path + file)    #  not working
```

Running a program and inserting keys

```
import win32com.client
import time

shell = win32com.client.Dispatch("WScript.Shell")
shell.Run("notepad.exe")
shell.AppActivate("notepad.exe")
time.sleep(.5)
shell.SendKeys("Now is the time\n", 0) # 1 for Pause = true 0 for no pause

shell.SendKeys("For all good men\n", 0)
shell.SendKeys("To aid The Doctor", 0)
shell.SendKeys('{HOME}{UP}{UP}', 0)
```

Sending a mouse click

```
import win32api, win32con
def click(x,y):
    win32api.SetCursorPos((x,y))
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,x,y,0,0)
    win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,x,y,0,0)
click(10,10)
```

Open Html Help file

```
import webbrowser
new = 2 # open in a new tab, if possible

// open a public URL, in this case, the webbrowser docs
url = "http://docs.python.org/library/webbrowser.html"
webbrowser.open(url,new=new)

// open an HTML file on my own (Windows) computer
url = "file://d/testdata.html"
```

Editing Environmental Variables (with registry)

```
Note: Normally changes to the environmental variables only last until the
program is terminated, then it revers back to what it was before the program was
called. This tip changes the variables stored in the registry

import _winreg

#  KEY_PATH is the key
#  REG_PATH is the entry where the variables are stored

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

def set_reg(name, value):
    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_reg(name):
    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

# Display 3 e environmental Variables
print get_reg('TESTPROJECT')
print get_reg('TESTDATAFOLDER')
print get_reg('DRV_PASSKEY')

# Change 1
print set_reg('TESTPROJECT', 'WBRA')

# display it
print get_reg('TESTPROJECT')

# When you exit the program this is the new value
```

Handling argv

Handling ARGV in calling python program.

```
import sys

print "This is the name of the script: ", sys.argv[0]
print "Number of arguments: ", len(sys.argv)
print "The arguments are: ", str(sys.argv)

Sample application

ln = len(sys.argv)
for x in range(1,ln):
    s = sys.argv[x]
    ch = s[0].upper()
    if ch == 'H':
        print('Usage : scrap.py 18 [Fxx] [H] [Sxx] [V]')
        print("  Fxx - Set filter to value xx")
        print("  H   - Show this help")
        print("  Sxx - Select record xx to view only")
        print("  V   - Verbose mode (show data)")
        exit(0)
    if ch == 'V':
        print("Verbose Mode Enabled")
        show_raw = 1
    if ch == 'F':
        filt = int(s[1:])
        print("Filter set to ", filt)
    if ch == 'S':
        select = int(s[1:])
        print('Entry Selected is ', select)


Sample Usage

sys.argv = 'scrap.py 18 F35 S20

Filter set to  35
Entry Selected is  20

[ Starting Test ]
[Processing selection 20]
[ Left to Right Flat #3 ]
[abs = x=  525 y=  447 z=    0]
[add = x= -525 y= -447 z=    0]
[x is first number]
[Negative Value]
------------------------------------------------------------
[ Test Complete ]
```

Handling Args in functions.

```
Using args in functions
def test_one(str1, str2, *args):
    print(f'VAL1 = {str1}  VAL2 = {str2}')
    print(f'Args = {args}')
    ln = len(args)
    for x in range(0,ln):
        print(f' {x} : args[{x}] = {args[x]}')
a = [1,2,3,4]
b = [2,3,4,5]
c = [3,4,5,6]
d = [4,5,6,7]
vals = a, b, c, d

# pass numbers
test_one('Testing', 'my test', 1,2,3,4,5)

#  OUTPUT
VAL1 = Testing  VAL2 = my test
Args = (1, 2, 3, 4, 5)
 0 : args[0] = 1
 1 : args[1] = 2
 2 : args[2] = 3
 3 : args[3] = 4
 4 : args[4] = 5

#pass variable lists
test_one('Testing 2', 'my test', a,b,c,d)

#  OUTPUT
VAL1 = Testing 2  VAL2 = my test
Args = ([1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6], [4, 5, 6, 7])
 0 : args[0] = [1, 2, 3, 4]
 1 : args[1] = [2, 3, 4, 5]
 2 : args[2] = [3, 4, 5, 6]
 3 : args[3] = [4, 5, 6, 7]

# pass list of variable lists (not what is desired)
test_one('3rd Try', 'list', vals)

#  OUTPUT
VAL1 = 3rd Try  VAL2 = list
Args = (([1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6], [4, 5, 6, 7]),)
 0 : args[0] = ([1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6], [4, 5, 6, 7])

# pass pointer to list of variables (desired)
test_one('4th Try', 'list', *vals)

#  OUTPUT
VAL1 = 4th Try  VAL2 = list
Args = ([1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6], [4, 5, 6, 7])
 0 : args[0] = [1, 2, 3, 4]
 1 : args[1] = [2, 3, 4, 5]
 2 : args[2] = [3, 4, 5, 6]
 3 : args[3] = [4, 5, 6, 7]
```

Sound

Add Beep to your programs

```
Import winsound
Frequency=2500 # 2.5 khz
Duration=1000 # 1 second
winsound.Beep(Frequency, Duration)

Note: The program will not continue execution until the duration is complete
See the appendix for frequency to note table.
```

Add Sound to your program

```
#import winsound
fname=”test.wav”
windsound.PlaySound(fname, windsound.SND_FILENAME)

Note: the program will not continue until the wav file is finished playing
```

Add Sound to your program without pausing

```
#import winsound
fname=”test.wav”
windsound.PlaySound(fname, windsound.SND_FILENAME | windsound.SND_ASYNC)

Note: The SND_ASYNC flag allow the program to continue while the sound file
is being played.
```

Directory / Filename

Directory Manipulation

Function	Command
does directory path exist T or F	os.path.exists(prj_path)
create directory	os.makedirs(prj_path)
returns current working directory	os.getcwd()
change current directory	os.chdir(new_dir)
creates the directory test in the current directory	os.mkdir('test')
Renames a directory or file (old to new)	os.rename(old, new)
Removes the directory 'old_dir'	os.rmdir('old_dir')

Get Parts of file name

Tpath = Sys.argv[0]; **Prog** = os.path.basename(Tpath)

Function	Command	Returns
Full Path & Prog	sys.argv[0]	N:/store/python/scrap/dir_test.py
Prog Name Only	os.path.basename(Tpath)	dir_test.py
Path to Prog	sys.path[0]	N:/store/python/scrap/
Path to Prog	os.path.dirname(Tpath)	N:/store/python/scrap/
Prog Name	os.path.splitext(Prog)[0]	dir_test
Prog Extension	os.path.splitext(Prog)[1]	.py
Full Path w/o ext	os.path.splitext(Tpath)[0]	N:/store/python/scrap/dir_test

Search directory for file

Find all *.trx files

```
import glob
for fname in glob.iglob("*.trx")
      print fname
```

Get list of files in directory

```
import glob
lst = glob.glob('*.txt')
print lst
I.e.['one.txt', 'two.txt', 'three.txt']
```

Get list of active drives

```
import win32api

dv = win32api.GetLogicalDriveStrings()
dv = dv.split('\000')[:-1]
print dv
```

Check if drive is removable

```
import win32file, win32api

def rdrive(d):
    # returns boolean, true if drive d is removable
    return win32file.GetDriveType(d)==win32file.DRIVE_REMOVABLE

dv = win32api.GetLogicalDriveStrings()
dv = dv.split('\000')[:-1]
print rdrive(dv)Convert string to int
```

File I/O

Write to a file

```
fp = open('out2.txt', 'w')
fp.write("This is text\n")
fp.write("So is this\n")
fp.close()
```

Read lines of a file

```
fp = open('out2.txt', 'r')
ctr = 0
for line in fp:
    str1 = str(line)
    if str1.find(error_msg) >= 0:
        ctr = 6
        fail = fail + 1
    if ctr > 0:
        print line
        ctr = ctr - 1
fp.close()
```

Read lines of a file (2nd method)

```
fp = open('outfile.dat', 'rt')
while True:
    line = fp.readline()
    if not line:
        break;
    print(line)
fp.close()

OR to compare 2 lines of a file(look for duplicates in a sorted file)

fp = open('outfile.dat', 'rt')
line = fp.readline()
while True:
    line2 = fp.readline()
    if not line:
        break;
    if len(line) == len(line2):
        x = len(line) - 6
        if line[:x] == line2[:x] :
            print(line[-5:-1], '&', line2[-5:-1])
    line = line2
fp.close()
```

File Positioning

```
Use fp.tell() to get the position of the current file.
Use fp.seek(x) to set the position for the next read/readline
```

Sorting The contents of a File

```
This sample will show how to load a file with lines terminated with a carriage
return, soft the lines,
then write them back to an output file (sorted).
Unsorted file to read is 'outfile.dat'

# Read the file into lines
    fp = open('outfile.dat', 'rt')
    lines = fp.read()
    fp.close()

# Split lines into line2 by the carriage return
    line2 = lines.split('\n')

#   Sort line2
    slines = sorted(line2)
    print("Saving sorted lines")

#  Write the sorted lines out to outsort.dat and add the carriage
#  return back in that was use to split the lines.
    fp = open('outsort.dat', 'wt')
    for x in range(len(slines)):
        fp.write(slines[x] + '\n')
    fp.close()
```

Get The Size of a File

```
Import os
      wsize = os.path.getsize("file1.baf")
```

Does File Exist

```
If os.path.exists('FileToFind.txt') == True:
      print("File does exist)
or
if os.path.isfile(file_path) == True:
      print("File does exist)
```

File Copy

```
Import shutil
shutil.copy(src, dst)      #  dst can be directory
shutil.copy2(src, dst)     #  dst can be directory, copy's metadata
shutil.copyfile(src, dst)  #  dst must be a file name
```

Write INI file

```
From configparser import ConfigParser
config_fn = 'RFCOMM.ini'
def write_config_port():
    config = ConfigParser()
    config['working_port'] = {'port' : port.get(),
                              'baudrate': baudrate.get(),
                              '# Comment': 'Optional Comment can go here'}
    with open(config_fn, 'w') as configfile:
        config.write(configfile)
```

Read INI File

```
From configparser import ConfigParser
def read_config_port():
    global ports
    config = ConfigParser()
    if os.path.exists(ocnfig_fn) == False:
        write_config_port()
    config.read(config_fn)
    port.get(config['working_port']['port'])
    baudrate.get(config['working_port']['baudrate'])

RFCOMM.ini
[working_port]
port = COM3
baudrate = 19200
# Comment = Optional Comment can go here
```

Reading A CSV Files

```
import csv
def load_data():
    cnt=[]; x=[]; y=[]; z=[]
    #  Get all text from text widget
    path = tkFileDialog.askopenfilename(parent=mainframe, filetypes=[('CSV', '*.csv')])
    if path = '':
       return (0,0,0,0)
    lc = 0
    with open(path, 'r') as csvfile:
        sr = csv.reader(csvfile, delimiter=',')
        for row in sr:
            if lc == 0:
                print(f'column Names are {" ,".join(row)}')
             else:
                 cnt.append(lc)
                x.append(int(row[0]))
                y.append(int(row[1]))
                z.append(int(row[2]))
            lc =+ 1
    return(cnt, x, y, z)
```

This Routine will prompt a user for a csv file to open, then get the entire file into list (using a comma to separate the fields). There are 3 rows to read in,The first data is split into 3 lists with the 4th containing the index number. All 4 lists are returned

Writing a JSON file (pretty printing)

```
This will create a indented json file
def write_json_pp():
    data = {}
    data['people'] = []
    data['people'].append({
        'name': 'Scott',
        'website': 'stackabuse.com',
        'from': 'Nebraska'
    })
    data['people'].append({
        'name': 'Larry',
        'website': 'google.com',
        'from': 'Michigan'
    })
    data['people'].append({
        'name': 'Tim',
        'website': 'apple.com',
        'from': 'Alabama'
    })

    with open('data.json', 'w') as outfile:
        json.dump(data, outfile, indent=4)
```

Reading a JSON file

```
import json
def read_json():

    with open('data.json') as json_file:
        data = json.load(json_file)
        for p in data['people']:
            print('Name: ' + p['name'])
            print('Website: ' + p['website'])
            print('From: ' + p['from'])
            print('')
```

Actual JSON data file (data.json)

```
{
    "people": [
        {
            "name": "Scott",
            "website": "stackabuse.com",
            "from": "Nebraska"
        },
        {
            "name": "Larry",
            "website": "google.com",
            "from": "Michigan"
        },
        {
            "name": "Tim",
            "website": "apple.com",
            "from": "Alabama"
        }
    ]
}
```

Write a bytearray file

```
ba = get_byte_array() # fill the byte array
baf = open("filename.baf", "wb")
baf.write(ba)
baf.close()
```

Read a bytearray file

```
baf = open("filename.baf", "rb")
ba = baf.read()
      or to read one record
rn = record to read
baf.seek(rn*76)
baf.read(76)
baf.close()
```

Zip Files

```
In these examples the zip file created / read is test.zip
zfile = test.zip

The files put into the zipfile are test1.txt, test2.txt and test3.txt
sfile = (test1.txt, tes2.txt, test3.txt)

These files are located in x:/test
sdir = x:/test

They will be stored in x:bkup
zdir = x:/bkup
```

Write to a Zip File

```
import zipfile

zf = zipfile.ZipFile(zdir + zfile, 'w', compression=zipfile.ZIP_DEFLATED
    for z in range(0, len(sfile)):
          zf.write(f'{sdir}{sfile[z]}')
    zf.close()
```

Read from a zip file

```
import zipfile

zf = zipfile.ZipFile(zdir + zfile, 'r')
zf.extractall(sdir)
```

Get Directory of zip file (to terminal)

```
import zipfile

zf = zipfile.ZipFile(zdir + zfile, 'r')
zf.printdir()
zf.close()
```

```
File Name                                             Modified             Size
test1.txt                                      2018-11-09 09:17:12           85
test2.txt                                      2017-07-18 09:06:16           48
test3.txt                                      2017-07-18 08:03:18           67
```

Printing to a Printer

Print to default Printer

```
# Insure pywin32 is installed.
import os, sys
import win32print
printer_name = win32print.GetDefaultPrinter ()
#
# raw_data could equally be raw PCL/PS read from
#  some print-to-file operation
#
if sys.version_info >= (3,):
  raw_data = bytes ("This is a test", "utf-8")
else:
  raw_data = "This is a test"

hPrinter = win32print.OpenPrinter (printer_name)
try:
  hJob = win32print.StartDocPrinter (hPrinter, 1, ("test of raw data", None,
"RAW"))
  try:
    win32print.StartPagePrinter (hPrinter)
    win32print.WritePrinter (hPrinter, raw_data)
    win32print.EndPagePrinter (hPrinter)
  finally:
    win32print.EndDocPrinter (hPrinter)
finally:
  win32print.ClosePrinter (hPrinter)
```

This will print ‘This is a test’ on paper from default printer.

Byte Array Conversions

String to byte array

```
S1 is a string
st = bytes(S1, ‘utf-8’)
```

Byte Array to strings

```
S1 is a byte array
st = s1.decode(‘utf-8’)
```

Hex String to byte array

```
s1 = ‘D88039F34D98’
print(bytes.fromhex(s1)) → b'\xd8\x809\xf3M\x98'
```

Byte Array to Hex string

```
s1 = b'\xd8\x809\xf3M\x98'
print(bytes.hex(s1))  →  ‘D88039F34D98’
```

Integer to byte array (2 bytes)

```
i_value = 1600   #  use h for signed 2 byte and  H is for unsigned)
ba = bytearray(struct.pack(‘h’, i_value)
print([ “0x%02x” % b for b in ba])  →  ['0x06', '0x40']
```

Byte array to integer (2 bytes)

```
# Create a signed int use ba from above  use h
# for signed 2 byte and  H is for unsigned)
[i] = struct.unpack(h,ba)
print(i)
```

Integer to byte array (1 bytes)

```
i_val = 27
ba = i_val.to_bytes(1, ‘little’)
```

Float to byte array

```
f_value = 5.1
ba = bytearray(struct.pack(‘f’, f_value)
print([ "0x%02x" % b for b in ba1 ])  →  ['0x33', '0x33', '0xa3', '0x40']
```

Byte Array to float

```
# use ba from above
[fl] = struct.unpack(‘f’, ba)
```

Character to byte array

```
Ba = bytes(c_value, 'utf-8')
```

Byte array to character

```
Ch = chr(ba)
```

NOTE:
A list of format codes for the struct.pack and struct.unpack functions is in the appendix of this document.

Strings

Simple use of string.format for screen output

```
Y = 12288;   x = 0b0011000000000000
```

Sample	Output
print("0x{:04x} 0b{}".format(y, x))	0x3000 0b0011000000000000
print("|{:^8}|".format(int(y/4))	| 3072 |
print("|{:8}|".format(int(y/4))	|3072 |
Print("|{:>8}|".format(int(y/4))	| 3072|

NOTE: See appendix "Format() Type Specifiers" for complete list of format specifiers

Alternate string format

Python Version 3.6 and up (recommended)

```
Y = 12288; fn = 'test1.dat';   z = 12.3574;   b = 4; c = 25
```

COMMAND	OUTPUT
print(f'file to open {fn}')	File to open test1.dat
print(f'File {fn} has error code {y:#x}')	File test1.dat has error code 0x3000
print(f'{y:08x}')	00003000
print(f'{y:#08x}')	0x003000
print(f'{z:2.4}')	12.36
print(f'{z:2.3}')	12.3
print(f'{c:{b}}')	25

NOTE: Formatted string cannot be broken into multiple lines with print. Use print(f' stuff', end='') to span source code lines. Length specifies total number of characters in output and padding character See appendix "Format() Type Specifiers" for a complete list of format specifiers.

String Manipulation

String slicing [start:end]
Str1 = "1234567890abcdefghij"

Str1[5:]	Returns all but the first 5 chars	"1234567890abcdefghij"
Str1[:5]	Returns the first 5 chars	"1234567890abcdefghij"
Str1[-5:]	Returns the last 5 chars	"1234567890abcdefghij"
Str1[:-5]	Returns all but the last 5 chars	"1234567890abcdefghij"
Str1[5:15]	Returns from 6th through 15th chars	"1234567890abcdefghij"
Str1[5:-5]	Returns from 6th to all but the last 5	"1234567890abcdefghij"
Str1[-5:-2]	Returns last 5 char minus last 2	"1234567890abcdefghij"
Str1[2:4]	Returns 2nd pair of characters	"1234567890abcdefghij"

Sample String Parser

Get the x, y and z from a string
Looks for = then , so number of chars is not important
Gets the data between the equals sign and the comma

```
s = 'gyro x=-16, y=-475, z=-117, t=32.44'

def get_xyz(st):
    s1 = st[:-1]
    p = 0
    c = []
    while s1.find(',',p) >= 0
        p = s1.find('=', p+1)
        c = append(s1[p+1: s1.find(',',p)])

x = get_xyz(s)
for z in x
    print(z,", ")
print('\n')

-16, -475, -117, 32.44
```

Quick SubString in string

```
st = 'Now is the time for'
if 'the' in st:
   do_if_true
```

Search String (find)

To find an occurrence of one string in another
sourc_str.find('search str', optional_offset)

```
text = 'ABCDEFGABC'
```

Command	**Output**
print(text.find('DEF'))	3
print(text.find('ABC'))	0
print(text.find('ABC'),1)	7
print(text.find('CAB'))	-1

String Editing

NOTE: Strings cannot be edited directly, create a new string based on the original string. Below replace B with Z

```
text = 'ABCDEFG'
text = text[:1] + 'Z' + text[2:]
print text
AZCDEFG
```

Using String Lists

```
s1 = {
    ('from tkinter import *\n'),
    ('from tkinter import ttk\n'),
    ('import os\n'),
    ('import base64\n'),
}
s2 = (
    ('from tkinter import *\n'),
    ('from tkinter import ttk\n'),
    ('import os\n'),
    ('import base64\n'),
)
printing
for s in s1:
    print(s)

(Note s1 is sorted)
```

```
import base64\n
import os\n
from tkinter import*\n
from tkinter import ttk\n
```

```
printing
for s in s2:
    print(s)
(Note s2 is not sorted)
```

```
from tkinter import *\n
from tkinter import ttk\n
import os\n
import base64\n
```

String of characters

```
b = 5
S1 = '#' * 5                  Returns '#####'
s2 = '#' * b                  Returns '#####'
s3 = '|' + ('-' * b) + '|'    Returns '|-----|'

This method does not work using the format f'' feature
```

Hex String to Integer

st = '3c00'

int(st, 16) Shows 15360

Octal String to Integer

st = '3700'

int(st, 8) Shows 1984

Lists

Creation

```
List1 = [] # create empty list
list1 = list1 + [23] # add item to list
r=40
list1 = list1 + [r]
list1 += [200]
list1.append(50)  # add to the end
print(list1)
[20, 40, 200, 50]
list1.insert(1, 30)   # insert element at position
print(list1)
[20, 30, 40, 200, 50]
```

Copying Lists (3 ways)

```
# Create test list
lst = []  ##  Empty list
for z in range(0,100):
    lst.append(z * 10)
l2 = []   ## new empty list
l2 = lst.copy()    #  or
l2 = lst[:]        #  or
l2 = list(lst)
```

Deleting List Elements

```
Lst = [5, 10, 15, 20, 25]

Delete a list element by value
g = Lst
g.remove(15)
print(g)          [5, 10, 20, 25]

Delete an element by index
g = Lst
del g[1]
print(g)          [5, 15, 20, 25]

Delete the last list element
g = Lst
g.pop()
print(g)          [5, 10, 15, 20]

Delete the first element
g = Lst
g.pop(0)
print(g)          [10, 15, 20, 25]

Delete All list elements
g = Lst
g.clear()
print(g)          []
```

Deleting item from list

```
A = [1, 2, 3, 4, 5, 6, 7, 8, 9]
del a[0]    # Delete the first item in the list
print a  >>  [2, 3, 4, 5, 6, 7, 8, 9]
del a[-1]  # Delete the last item in the list
print a  >>  [2, 3, 4, 5, 6, 7, 8]
del a[2:4] # Delete a range from 2 to 4
print a  >>  [2, 3, 6, 7, 8]
del a        # Delete the entire list
print a  >>  Name Error 'a' is not defined

Note:
a.pop(0) will return a[0] and delete it from the list
```

Sorting Tuples or Lists

```
A = ('one', 'two', 'three', 'four')  #  TUPLE
print(A)
output : ['one', 'two', 'three', 'four']
A = sorted(A)
print(A)
output : ['four', 'one', 'three', 'two']

A = {'one', 'two', 'three', 'four'}  #  List
print(A)
output : ['one', 'two', 'three', 'four']
A = sorted(A)
print(A)
output : ['four', 'one', 'three', 'two']

NOTE: To sort with mixed up case / lower case items.
A = sorted(A, key=str.lower)   or   A = sorted(A, key=str.casefold)
```

List Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

NOTE: See the appendix for a all methods for a list.

Time/Date

Get Today's Date (day MONTH year)

```
def todays_date():
    tm = time.localtime()
    str1 = time.strftime("%d %B 20%y", tm)
    my_date.set(str1)
```

Elapsed Time (Seconds)

```
Import time
start = time.time()
print"Started  on " + time.asctime(time.localtime(start)) + '\n'
endtm = time.time()
print"Finsihed on " + time.asctime(time.localtime(endtm)) + '\n'
elapsed = endtm - start
str1 = time.strftime("%X", time.gmtime(elapsed))
print "Elapsed  Time " + str1 + '\n'

Note 1 : Time is in the string format of YYMMDDHHMMSSss
```

Elapsed Time (More then 1. resolution)

```
tm1 = time.process_time() # (use perf_counter if timing with sleep function)
# performs a function
tm2 = time.process_time() # (use perf_counter if timing with sleep function)
print(tm2-tm1)    # i.e. 0.09861660000001393
```

Seconds to String

```
from datetime import datetime
import time

def sec_to_str(tm):
    wstr = datetime.fromtimestamp(tm).strftime("%y%m%d%H%M%S")
    return(wstr)
```

String to Seconds

```
def str_to_sec(st1):
    tms = datetime(2000 + int(st1[0:2]), int(st1[2:4]), int(st1[4:6]),
                   int(st1[6:8]), int(st1[8:10]), int(st1[10:12]) )
    tm = time.mktime(tms.timetuple())
    return tm
```

Time To String

```
def time_to_str():
    now = datetime.now()
    s1 = now.strftime("%y%m%d%H%M%S")
    s2 = now.strftime("%f")
    s1 = s1 + s2[0:2]
    return s1
```

Network Sockets

Client.py Sample

```
import socket

def main():
    host = '127.0.0.1'
    port = 5000
    mySocket = socket.socket()
    mySocket.connect((host,port))
    message = input(" -> ")
    while message != 'q':
        mySocket.send(message.encode())
        data = mySocket.recv(1024).decode()
        print ('Received from server: ' + data)
        message = input(" -> ")
    mySocket.close()

if __name__ == '__main__':
    main()
```

Server.py Sample

```
import socket

def main():
    host = "127.0.0.1"
    port = 5000
    mySocket = socket.socket()
    mySocket.bind((host,port))
    mySocket.listen(1)
    conn, addr = mySocket.accept()
    print ("Connection from: " + str(addr))
    while True:
        data = conn.recv(1024).decode()
        if not data:
            break
        print("from connected  user: " + str(data))
        data = str(data).upper()
        print ("sending: " + str(data))
        conn.send(data.encode())
    conn.close()

if __name__ == '__main__':
   main()
```

Brief Introduction to Classes

```
This demo software show creation of a class and a sample of it useage
The first section is the class definition, the second variable assignment and
the third is the sample usage of the class.

from tkinter import messagebox

class about:
    """
        class comment
    """
    Version = '3.0'


    def __init__(self, progid):
        self.progid = progid


    def get_lib_version(self):
        return self.Version


    def get_full_lib_version(self):
        msg = self.get_lib_version()
        rs = 'About.py Version : ' + msg
        return rs


    def show_about(self):
        messagebox.showinfo("About " + self.progid['progname'],
                      "Filename : " + self.progid['progname'] + "\n" +
                      "Title : " + self.progid['title'] + "\n" +
                      "Version : " + self.progid['version'] + "\n" +
                      "Creation Date : " + self.progid['date'] + "\n" +
                      "Revision Date : " + self.progid['rev_date'] + "\n" +
                      "Author : " + self.progid['author'] + "\n" +
                      "Description : " + self.progid['description'])


prog_id = {'progname': 'Crypto_Helper.py',
           'title': 'Add in solving cryptoquotes',
           'version': '1.0',
           'date': "30 July 2019",
           'rev_date': '',
           'author': "Peter Hedlund",
           'description': 'To set compare tables to run against \n'
           ' a dictionary file to find possbile solutions.'}

# Testing the class
t1 = about(prog_id)
print('Version ', t1.Version)
print(t1.get_lib_version())
print(t1.get_full_lib_version())
t1.show_about()
```

Tkinter

Variable Types in Tkinter

Some widgets (like text entry widgets, radio buttons and so on) can be connected directly to application variables by using special options: variable, textvariable, onvalue, offvalue, and value. This connection works both ways: if the variable changes for any reason, the widget it's connected to will be updated to reflect the new value. These Tkinter control variables are used like regular Python variables to keep certain values. It's not possible to hand over a regular Python variable to a widget through a variable or textvariable option. The only kinds of variables for which this works are variables that are subclassed from a class called Variable, defined in the Tkinter module. They are declared like this:

TYPE	**DESCRIPTION**
BooleanVar()	Holds a boolean, returns 0 for False and 1 for True
DoubleVar()	Holds a float; default value 0.0
IntVar()	Holds an integer; default value 0
StringVar()	Holds a string; default value ""

To read the current value of such a variable, call the method get(). The value of such a variable can be changed with the set() method.

GUI Programming

Sample GUI Program

```
from tkinter import *
from tkinter import ttk

def quit_prog():
    root.destroy()

wstr = ""
root = Tk()
#  prevent window resizing.
root.resizable(0, 0)
root.title('')
mainframe = ttk.Frame(root, padding='3', height=120, width=180)
mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
mainframe.grid_propagate(0)

works = StringVar()

lbl = ttk.Label(mainframe, text=prompt)
lbl.grid(column=0, row=0)
ans = ttk.Entry(mainframe, textvariable=works)
ans.grid(column=0, row=1)
quitProg = ttk.Button(mainframe, text='Done', command=quit_prog)
quitProg.grid(column=0, row=2)

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

Prevent Window Resizing

```
root = Tk()
#  prevent window resizing.
root.resizable(0, 0)
root.title("Modbus Setup GUI")
mainframe = ttk.Frame(root, padding="3", height=480, width=800)
mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
mainframe.grid_propagate(0)
```

Add Icon to GUI

```
root = Tk()
#  prevent window resizing.
root.resizable(0,0)
# Replace tk icon with your own.
root.iconbitmap(r'serial.ico')
root.title("Modbus Analog GUI")
```

Get GUI Position

```
Import win32gui
handle = win32gui.GetForegroundWindow()
rect = win32gui.GetWindowRect(handle)
```

```
x = rect[0] # x position of window
y = rect[1] # y position of window
OR
rv = root.geometry()
 RV is a string that contains a string
  '300x300+10+20'
    W   H   X  Y
```

Get Mouse Position

```
def get_xy(event):
    print(f'X={event.x_root} Y={event.y_root}')
# in gui code
root.bind('<Button-3>', get_xy)
# Right click to send the x y position to the terminal.
```

Start GUI at a specific position

```
Start gui at screen position 210 x, 200 y
Screen width is 800 and height is 610

high = 610
wide = 800
root = Tk()
#  prevent window resizing.
root.resizable(0, 0)
root.geometry('%dx%d+%d+%d' % (wide,high,210,200))
root.title("Test File Location Manipulator")
mainframe = ttk.Frame(root, padding="3", height=high, width=wide)
mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
mainframe.grid_propagate(0)

OR
root.geometry('+{}+{}'.format(210,200))
OR
root.geometry('+210+200')
```

Center GUI On Screen

```
Wd = 350
Ht = 450
# Create mainframe
posx = int(root.winfo_screenwidth() / 2 - Wd / 2)
poxy = int(root.winfo_screenheight() / 2 - Ht / 2)
root.geometry("+{}+{}".format(posx, posy)
```

Force GUI On Top of Other Windows

Force tkinter window to be on top of all other windows.

```
Root = Tk()
root.wm_attributes("-topmost", 1)
```

Create an Embedded Icon (Python 2.7)

1. Start with the icon you want to embed in your tkinter program. (small_test.ico)

2. Run this program and capture the output, save as icon.py

```
import binascii
import itertools

iconfile = 'decode.ico'
STR_LENGTH = 60
VARIABLE = 'iconhexdata ='
INDENT = ' ' * (len(VARIABLE)+1)

def grouper(n, iterable):
    "s -> (s0,s1,...sn-1), (sn,sn+1,...s2n-1), (s2n,s2n+1,...s3n-1), ..."
    return itertools.izip_longest(*[iter(iterable)]*n, fillvalue='')

with open(iconfile, 'rb') as imgfile:
    imgdata = imgfile.read()
    hexstr = binascii.b2a_hex(imgdata)

hexlines = (''.join(group) for group in grouper(STR_LENGTH, hexstr))

print VARIABLE,
print (' \\\n' + INDENT).join((repr(line) for line in hexlines))
```

3. In your tkinter program add the following imports

```
import os, binascii, tempfile, icon
```

4. Create a temp file

```
# create temp file
with tempfile.NamedTemporaryFile(delete=False) as iconfile:
    iconfile.write(binascii.a2b_hex(icon.iconhexdata))
```

5. Install the icon

```
root = Tk()
#  prevent window resizing.
root.resizable(0, 0)
root.iconbitmap(iconfile.name)
```

6. Clean up the temp file at the end.

```
root.mainloop()
# delete temp file
os.remove(iconfile.name)
```

Create an Embedded Icon (Python 3.7)

Step 1 Convert icon to base64 string. In this example the icon is timer_clock.ico and the output will be icon.py

```
import base64

with open("timer_clock.ico", "rb") as imageFile:
    str = base64.b64encode(imageFile.read())
fp = open('icon.py', "wb")
fp.write(b'wstr = """')
fp.write(str)
fp.write(b'"""')
fp.close
```

Step 2 Use the icon.py to embed a temporary file into your program. Note, wstr is the string name created in the above step.

```
from tkinter import *
from tkinter import ttk
import base64
import os
from icon import *
icondata = base64.b64decode(wstr)
## The temp file is icon.ico

tempFile = "icon.ico"
iconfile = open(tempFile, "wb")
## Extract the icon
iconfile.write(icondata)
iconfile.close()

root = Tk()
root.wm_iconbitmap(tempFile)
os.remove(tempFile)
root.mainloop()
```

Same Program—Different Computers---Different screens

To keep your programs looking to scale on different systems.

mainframe.tk.call('tk', 'scaling', 1.2)

Your program should look the same on different computers.

Widgets

Getting x, y position of a character in a text widget

```
Txt is the text widget
pos = txt.index(CURRENT) # get position "1.2"
p = pos.find('.') # get position of the decimal point in the string
x = int(pos[:p])
y = int(pos[y+1:])
```

Put a character at an x,y position in a text widget

```
txt.insert("%d.%d" % (x,y),'\n')
```

Things to pass to text widget index

```
 txt.insert(1.3, "test")  # absolute position line 1, col 3
 txt.insert(CURRENT,"Test")
 txt.insert(END, "test")  # used most often
 txt.insert(INSERT, "test")
 txt.insert("%d.%d" % (x,y),"test)# put at x,y position
```

Replace string in text widget

```
txt.delete('1.0','2.0')
txt.insert('1.0','New String')
Note: position entry is in string format col.row
```

Putting string at position x, y of text widget

```
def puts_xy(x, y, stng):
    txt.configure(state='normal')
    pos1 = '{}.{}'.format(x,y)
    pos2 ='{}.{}'.format(x+1,y)
    txt.delete(pos1,pos2)
    txt.insert(pos1,stng)
    txt.configure(state='disabled')
    txt.update()
```

Clear all widgets in a frame

```
pf is our frame widget to clear all other widgets inside pf
for widgets in pf.winfo_children():
    Widget.destroy()
```

Get the color of a widget

```
w = Label(mainframe, text="TESTING")
w.grid(column=0, row=0)

w.cget('background') #gets the background color
# Normal backgound color is SystemButtonFace
#  see APPENDIX G: System Colors Definitions
#  for a complete list of colors.
```

Radio Buttons

```
# Sample Use
if reportType.get() == 1:
       do_this()

# Integer variables
reportType = IntVar()

# Radio Buttons
r1 = Radiobutton(mainframe, text='Show Log', variable=reportType, value=1)
r1.grid(column=5, row=2, sticky='w')
r2 = Radiobutton(mainframe, text='Show Errors', variable=reportType, value=2)
r2.grid(column=5, row=4, sticky='w')
reportType.set(2)
r1.update()
```

Array of StringVar() or IntVar()

In the main gui section of the program is where you define your array.

```
ivar=[]
svar=[]
for x in range(0,10)
    ivar.append(0)
    ivar[x] = IntVar()
    svar.append('')
    svar[x] = StringVar()

Then use in your program
ivar[x].set(3)
y = ivar[x].get()
svar[x].set('test')
y = svar[x].get()
```

Array of Radio Buttons (2 per array element)

Setting up an array of 10 sets of 2 radio buttons, all set to 0 (num)

```
radio = { 0:0, 1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0}
for x in range(0,10):
    radio[x] = IntVar()
    rb = Radiobutton(inp, text="num", value=0, variable=radio[x])
    rb.grid(column=3, row=x+1, padx=5, pady=5)
    rb.select()
    lb = Radiobutton(inp, text="char", value=1, variable=radio[x])
    lb.grid(column=5, row=x+1, padx=5, pady=5)
To get the value of the 2nd Radio button (item 1)
   x = check[1]
   print(x.get())   # will print out 0
To progromatically change the value to 1 (check buttons will change on screen
   x = check[1]
   x.set(1)
```

Array of Entries

```
Setting up an array of 10 entry boxes (similar to radio buttons)
lndx = { 0:"", 1:"", 2:"", 3:"",4:"",5:"",6:"",7:"",8:"",9:""}
for x in range(0,10):
    lndx[x] = StringVar()
    lmp = ttk.Entry(inp, width=5, textvariable=lndx[x])
    lmp.grid(column=2, row=x+1, padx=5, pady=5)
to get the value of an entrybox textvariable (in this case the 3rd line
     r = lndx[2]
    val = r.get()
or if its a number you want
    r = lndx[2]
   val = int(r.get())Array of Labels
Create an array of labels (75 in this case).
lab = {}
for x in range(0, 15):
    lab[x] = ttk.Label(mainframe, text=str(x+1), width=5)
    lab[x].grid(column=2, row=x+1)
    lab[x+15] = ttk.Label(mainframe, text=str(x+16), width=5)
    lab[x+15].grid(column=3, row=x+1)
    lab[x+30] = ttk.Label(mainframe, text=str(x+31), width=5)
    lab[x+30].grid(column=4, row=x+1)
    lab[x+45] = ttk.Label(mainframe, text=str(x+46), width=5)
    lab[x+45].grid(column=5, row=x+1)
    lab[x+60] = ttk.Label(mainframe, text=str(x+61), width=5)
    lab[x+60].grid(column=6, row=x+1)
To change the color of label 74 to red
    lab[74].config(foreground='Red')
```

Array of Labels/CheckButtons/Entries

```
 See array of stringVar and IntVar section on how to make the arrays
al_state, hr, mn, sc.  This loop creates 8 rows of 7 columns showing
number, set (yes/no) hours, minuets, seconds

       for x in range(0,8):
            tk.Label(top, text='  ').grid(column=0, row=x+1)
            lb = tk.Label(top, text=str(x+1))
            lb.grid(column=1, row=x+1, padx=5)
            r = ttk.Checkbutton(top, variable=al_state[x])
            r.grid(column=2, row=x+1)
            en = tk.Entry(top, textvariable=hr[x], width=3)
            en.grid(column=3, row=x+1)
            lb = tk.Label(top, text=' : ')
            lb.grid(column=4, row=x+1)
            en = tk.Entry(top, textvariable=mn[x], width=3)
            en.grid(column=5, row=x+1)
            lb = tk.Label(top, text=' : ')
            lb.grid(column=6, row=x+1)
            en = tk.Entry(top, textvariable=sc[x], width=3)
            en.grid(column=7, row=x+1)
            tk.Label(top, text='   ').grid(column=8, row=x+1)
```

Array of Buttons

```
import tkinter as tk

def  button_press1():
    print('1 pressed')

def  button_press2():
    print('2 pressed')

def  button_press3():
    print('3 pressed')

def  button_press4():
    print('4 pressed')

def  button_press5():
    print('5 pressed')

app=tk.Tk()
butt = []
my_data = (
( 'Button 1', button_press1),
( 'Button 2', button_press2),
( 'Button 3', button_press3),
( 'Button 4', button_press4),
( 'Button 5', button_press5),
)
for x in range(0, len(my_data)):
    button = tk.Button(app, text=my_data[x][0], command=my_data[x][1])
    button.grid(column=0, row=x+1)
app.mainloop()
```

Passing values to Array of buttons

This example creates 5 buttons with defined return values for each one even though there is only one button handler created

```
import tkinter as tk
from functools import partial

def but_p(val):
    print(f'Pressed {val}')

app = tk.Tk()
butt = []
for x in range(0,5):
    butt.append(0)
    butt[x] = tk.Button(app, text=f'CMD {x}', command=partial(but_p, x))
    butt[x].grid(column=0, row=x+1)

app.mainloop()
```

Set the Label Font

```
lbl = Label(mainframe, text='Hi There", font=('Helvetica',24))
 bl.grid(column=0, row=0)
```

Set Text Label to BOLD

```
from tkinter import font
_font = font.Font(weight='bold')
lbl = Label(mainframe, text='test', font=_font)
 lbl.grid(column=0, row=0)
```

Set the Default font for the GUI text

```
root.option_add("*font", "Arial 16 bold")
```

Set the Label Text Position

```
To set the position of the text in the label use anchorj
lblx = ttk.Label(mainframe, text="Calling ", width=10, anchor='center'))
selections for anchor are :
'nw'   'n'     'ne'
'w'  'center'  'e'
'sw'   's'     'se'
```

Toggle Buttons

```
def led00_toggle():
    if btn30.config('relief')[-1] == 'sunken':
        btn30.config(relief="raised", text='LED01 ON')
    else:
        btn30.config(relief="sunken",text='LED01 OFF')

btn30 = Button(tab3, text="LED00 ON", command=led00_toggle, width=15,
  relief="raised", state=DISABLED)
btn30.grid(column=0, row=4, padx=15, pady=5, sticky='w')
```

—OR- USING IMAGES for the Buttons

```
def toggle_h():  # Horizontal Button
    if tb.get() == 0:
        t_btn.config(image=_on_)
    else:
        t_btn.config(image=_off_)
    tb.set(tb.get() ^ 1)
def toggle_v():  # Vertical Button
    if tb2.get() == 0:
        t_btn2.config(image=v_on_)
    else:
        t_btn2.config(image=v_off_)
    tb2.set(tb2.get() ^ 1)

root = tk.Tk()
_on_ = tk.PhotoImage(file="n:\\Store\\Python\\Converted\\7seg\\on.gif")
_off_ = tk.PhotoImage(file="n:\\Store\\Python\\Converted\\7seg\\off.gif")
v_on_ = tk.PhotoImage(file="n:\\Store\\Python\\Converted\\7seg\\on_v.gif")
v_off_ = tk.PhotoImage(file="n:\\Store\\Python\\Converted\\7seg\\off_v.gif")
tb = tk.IntVar();  tb.set(0)
tb2 = tk.IntVar();  tb2.set(0)
t_btn = tk.Button(image=_off_, command=toggle_h)
t_btn.pack(pady=5)
t_btn2 = tk.Button(image=v_off_, command=toggle_v)
t_btn2.pack(pady=5)
```

Notebook (Tabs)

```
#  Create Tab Control
tabControl = ttk.Notebook(mainframe)

#  Create tabs frame
tab1 = ttk.Frame(tabControl)
#  Add the first tab
tabControl.add(tab1, text='Project Info')

#  Add a second tab
tab2 = ttk.Frame(tabControl)
#  Make second tab visible
tabControl.add(tab2, text='About Info')

#  Pack to make visible (for entire window)
tabControl.pack(expand=1, fill="both", padx=5, pady=5)
# or ….
# tabControl.grid(column=0, row=0, padx=5, pady=5)

#  Put stuff into tabs

#  Page 1  Project Info
lb = Label(tab1, text="Program File Name")
lb.grid(column=0, row=1, sticky="w")
en0 = Entry(tab1, textvariable=my_filename)
en0.grid(column=1, row=1, sticky="w")

#  Page 2  About Info
lb = Label(tab2, text="Program Name")
lb.grid(column=0, row=0, sticky="w")
en21 = Entry(tab2, text="", textvariable=my_progname)
en21.grid(column=1, row=0, sticky="w")
```

Handler for changing Notebook tabs

```
Use code above for this example
At the end of the tab definitions add this line
tabControl.bind('<<NoteBookTabChanged>>',tab_changed)

create the tab_changed handler
def tab_changed(None):
    cur_tab = tabControl.indes(tabControl.select())
    #  cur_tab now has the value of the tab selected (for 8 tabs that would be 0-7)
    if cur_tab == 2:
      do something
```

Enable / Disable Notebook Tab

```
Using above example
TabControl.tab(tab1, state=NORMAL)
tabControl.tab(tab2, state=DISABLED)
```

Open File Dialog box

```
from tkinter.filedialog import askopenfilename

path = askopenfilename(filetypes=((‘CSV Files’,’*.csv’),(“All Files”,”*.*”)))
if path == ‘’:
    return
file_name = os.path.basename(path)
```

Message Box

```
From tkinter import messagebox
messagebox.showinfo(“ERROR”,”No file selected”)
```

Progress Bar

```
If you don’t know the length of the data use Indeterminate. It creates a cylon bar
to show activity
prt = ttk.Progressbar(mainframe, mode=’indeterminate’, length=260)
prg.grid(column=1, row=2,padx=15, pady=5, columnspan=2)
prg.start([optional interval ms default is 50ms) #  Starts the progress bar motion
prt.stop()  # stops the progress bar
If you know the position of the progress bar then
prt = ttk.Progressbar(mainframe, mode=’determinate’, length=260, variable=level)
prg.grid(column=1, row=2,padx=15, pady=5, columnspan=2)
use level.set(pos) to set the level of the progress bar.
```

Progress Bar With Color

```
To use colors with progress bars you must use a style.
s1 = ttk.Style()
s1.theme(‘clam’)     # see appendix for style themes
s1.configure(‘green.Vertical.Tprogressbar’, foreground=’green’, background=’green’)
fuel_pg = ttk.Progressbar(mainframe, orent=VERTICAL, variable=fuel, length=150,
maximum=2000,
                        style=’green.Vertical.Tprogressbar’)
fuel_pg.grid(column=0, row=0, rowspan=4)
```

Clearing an Entry Widget

```
count_en = ttk.entry(mainframe, text=’now is the time’
count_en.grid(column=1, row=2)
To clear the Entry box
count_en.delete(0, ‘end’)
```

Entry Widget with Variable

```
If you plan to use an entry widget with a variable, do NOT use the text=’’ entry in
the definition.
This will prevent the variable from affecting the entry.
```

Using Colorized Buttons

```
Use the style widget to change the button color
s1 = ttk.Style()
s1.configure('red.TButton', background='Red')

quitProg = ttk.Button(mainframe, text="Quit", style='red.TButton',
command=quit_prog)
quitProg.grid(column=0, row=15)
```

Run Button after its declaration

```
Add () after function name in the creation window as shown below.

dbug_card = ttk.Button(windframe, text='Show Card',command=debug_show_card())
dbug_card.grid(column=7, row=2)
```

Button Color Styles

```
s1 = ttk.Style()
s1.configure('black.TButton', background='Black')
s2 = ttk.Style()
s2.configure('blue.TButton', background='Blue')
s3 = ttk.Style()
s3.configure('cyan.TButton', background='Cyan')
s4 = ttk.Style()
s4.configure('green.TButton', background='Green')
s5 = ttk.Style()
s5.configure('magenta.TButton', background='Magenta')
s6 = ttk.Style()
s6.configure('red.TButton', background='Red')
s7 = ttk.Style()
s7.configure('white.TButton', background='White')
s8 = ttk.Style()
s8.configure('yellow.TButton', background='Yellow')
```

Adding a Browse Button

```
from tkinter.filedialog import askopenfilename

def browse_btn():
    global full_path
    path = askopenfilename(filetypes=(('CSV Files', '*.csv'),
                                                ('All Files', '*.*')))
    if path == '':
        return
    full_path = path

brws = ttk.Button(mf, text="Browse", command=browse_btn)
brws.grid(column=2, row=0, padx=15, pady=5)
```

Creating a Vertical Button (grid method)

```
Assuming you have 3 horizontal ttk.buttons at Col=5 row=0 - row2
Create a ttk.button with a width of 4 and in the text for the name insert \n
between each letter. Use the grid method to place the button at col 6 row 0 and use
rowspan=3.
```

Setting ComboBox selected value

```
ports = ['COM2',''COM3',''COM4',''COM5',''COM6',''COM7',''COM11',''COM12']
prt_sel = ttk.ComboBox(mainframe, state='readonly', value=ports

#Set value displayed in the combo box to COM11
x = ports.index('COM11')
prt_sel.current(x)

#Get the name of the selected port
wstr = ports[prt_sel.current()]
# wstr='COM11' # assuming COM11 was selected in the combo box.
```

Using Keyboard with button click

i.e. pressing shift while clicking button 1. button 1 calls the function process_b1. This uses the python library keyboard (pip install keyboard)

```
import keyboard

def process_b1():\
    if keyboard.is_pressed('shift'):
        Process what to do for shift button
    else:
        Process what to do for normal button press
```

See appendix for a list of keys supported by keyboard

Menu Widget

Create Menu

```
# Create Menu
menubar = Menu(root)
```

Create Menu Group

```
# Create File Menu
file = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='File', menu = file)
```

Create Menu Members

```
# Add members to file menu
file.add_command(label='Save Work', command=write_config_file)
file.add_command(label='Restore Work', command=read_config_file)
file.add_separator()
file.add_command(label='Quit', command=quit_prog)
```

Activate Menu

```
# display Menu
root.config(menu = menubar)
```

Disable Menu Entry

```
file.entryconfigure('Save Work', state=DISABLED)
```

Enable Menu Entry

```
file.entryconfigure('Save Work', state=NORMAL)
```

Sample Code

```
# Create Menu
menubar = Menu(root)

# Create File Menu
file = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='File', menu = file)

# Add members to file menu
file.add_command(label='Save Work', command=write_config_file)
file.add_command(label='Restore Work', command=read_config_file)
file.add_separator()
file.add_command(label='Quit', command=quit_prog)

# Create Edit Menu
edit = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Edit', menu = edit)

# Add Members to edit menu
edit.add_command(label='Clear Puzzle', command=clear_alphabet)
edit.add_command(label='Copy 2 Clipboard', command=to_clipboard)

# Create Tool Menu
tool = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Tools', menu = tool)

#add Members to Tool menu
tool.add_command(label='Crypto Helper', command=c_helper)
tool.add_command(label='Restart', command=my_restart)

# Add Help Menu
help_ = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Help', menu = help_)

# Add Members to Help menu
help_.add_command(label='Show Hint', command=show_hint, state=DISABLED)
help_.add_command(label='Show Answer', command=show_answer, state=DISABLED)
help_.add_command(label='Check Work', command=check_work, state=DISABLED)
help_.add_separator()
help_.add_command(label='Help', command=my_help)
help_.add_command(label='About', command=my_about)

# display Menu
root.config(menu = menubar)
```

Child Windows

Creating a Child Window

Creating a debug child window with quit button

```
def debug_quit():
    global window
    window.destroy()

def debug():
    global window
    window = Toplevel(height=220, width=260)
    window.title('title')
    windframe = ttk.Frame(window, padding='3', height=220, width=260)
    windframe.grid(column=1, row=2, sticky=(N, W, E, S))
    dbug_quit = ttk.Button(windframe, text='Quit',command=debug_quit)
    dbug_quit.grid(column=7, row=5)
    window.mainloop()
```

Creating Only One Child Window

This example shows creating a main window with
a label showing the a and b values and a button to
create a child window. The child window will NOT be created
more than once. Closing the child windows does NOT abort the program.
Closing the Main window will close BOTH windows. Pressing the buttons
on the child window will change the values of a and b and update them on
the main window.

```
from tkinter import *
import tkinter as tk
from tkinter import ttk

def setme():
    aa.set(6)
    bb.set(5)
    update_lbl()

def setme2():
    aa.set(5)
    bb.set(6)
    update_lbl()

def update_lbl():
    wstr = f'A={aa.get()}   B={bb.get()}'
    l.config(text=wstr)
```

```
def nw():
    global top
    try:
        if top.state() == 'normal':
            top.focus()
    except:
        top = Toplevel()
        top.title('Child Window')
        top.geometry("200x200")
        b = ttk.Button(top, text='Change', command=setme)
        b.grid(column=0, row=0)
        b2 = ttk.Button(top, text='Change2', command=setme2)
        b2.grid(column=1, row=0)
        top.mainloop()



root = Tk()
aa=IntVar()
bb=IntVar()
child_exists = IntVar()
child_exists.set(0)
aa.set(5)
bb.set(6)
root.title('Main Window')
root.geometry("200x200")
wstr = f'A={aa.get()}   B={bb.get()}'
l = tk.Label(root, text=wstr)
l.grid(column=0, row=0)
b3 = ttk.Button(root, text='New Window', command=nw)
b3.grid(column=1, row=0)

root.mainloop()
```

Features

Quitting a program

```
Use this function to quit a gui program. (Assuming you created the window with root
= Tk()

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

Process Close [X] Button on main window

```
Def on_closing():
    root.destroy()

root.protocol("WM_DELETE_WINDOW", on_closing)
```

Restart tkinter program

```
This program will restart the current tkinter program.

import sys
import os
from tkinter import Tk, Label, Button

def restart_program():
    """Restarts the current program.
    Note: this function does not return. Any cleanup action (like
    saving data) must be done before calling this function."""
    python = sys.executable
    os.execl(python, python, * sys.argv)

root = Tk()
Label(root, text="Hello World!").pack()
Button(root, text="Restart", command=restart_program).pack()

root.mainloop()
```

Tkinker Status Bar

```
There is NO statusbar widget in tkinter. Instead use a Label Widget
Use :< to left justify, :> to right justify and :^ to center in the width
Use the grid method to put the statusbar on the bottom row. Add all the columns to
come up with the columnspan= value, and make the sticky SW+SE to fill the bottom.
The actual number of characters in the label depends on the font used. Use trial
and error for width=value
Here is an example of the function to update the status bar and
the creation of the statusbar object itself.

def update_statusbar(num):
    str1 = "{:<30}{:>40}".format(stats[num],"File Scanner 1.0")
    statusbar.configure(text=str1)

statusbar=Label(root,text="", bd=1, relief=SUNKEN, anchor=W, width=40)
statusbar.grid(column=0, row=4, columnspan=3, sticky=SW+SE)
update_statusbar(0)
```

Alternative Status Bar

```
In this version of the status bar, the width of the status bar is calculated based
on the gui width (gui_wide). The string gap between the information and the time is
calculated automatically based
on the font specified for the status bar.  If you change the font or size of the
status bar your results will vary and you may change the / 7 to something else
depending on the size of the font you choose.

def update_statusbar():
    global status
    global alive_val, cnt
    if alive_val == alive.get():
        cnt += 1
    else:
        alive_val = alive.get()
        cnt = 0
    if cnt > 2 :
        term_rcv()

    rtc_time.set(time_to_str())
    stat_str = ('CLOSED', 'OPEN  ')

    sp = ' '
    str1 = f" RFComm Simulator | Version {prog_id['version']} | "
    str1 += f"{portv.get():5} is {stat_str[rcv_active.get()]} | "
    str2 = f'{mmath.current_time()} |{alive_val:3}'

    wk = sb_len - (len(str1) + len(str2))

    str1 = str1 + f'{sp:{wk}}' + str2

    statusbar.configure(text=str1)
    status = root.after(2000, update_statusbar)
```

In the gui code section

```
## ------------------------------------
##  Status Bar Widget
## ------------------------------------
fontstyle = tkFont.Font(family='Courier New', size=10)
sb_len = int(gui_wide / 8)
statusbar = Label(root, text="", relief=SUNKEN, anchor=W, width=sb_len,
font=fontstyle)
statusbar.grid(column=0, row=21, columnspan=4, sticky=SW+SE)
update_statusbar()
```

About Box

There is no mechanism for an about box, the recommendation is to use a messagebox
this example shows setting up the variables used by about box, the about box routine and
how to declare a label to act as a button to trigger the about box function

```
#  Includes
from tkinter import *
from tkinter import messagebox

#  Message box variables
progname = "ABB_Test_Shell"
title = 'ABB Bypass Test Manager'
version = '1.0'
date = "27 November 2017"
rev_date = "27 November 2017"
author = "Peter Hedlund"
description = 'This program will allow running of Microsoft Test ' +\
              'Manager on selected test or group of tests'
#  Message box creation
def about_box(event=None):
    messagebox.showinfo("About " + progname,
                        "Filename : " + progname + "\n" +
                        "Title : " + title + "\n" +
                        "Version : " + version + "\n" +
                        "Creation Date : " + date + "\n"
                        "Revision Date : " + rev_date + "\n"
                        "Author : " + author + "\n" +
                        "Description : " + description)
#  Clickable Label to activate message box
lab = Label(mainframe, text="About", relief=SUNKEN, anchor='center', width=8)
lab.grid(column=7, row=7)
lab.bind("<Button-1>", about_box)
```

==========

Note: Since I originally wrote this, I have come up with a better solution.
In the includes section.

Add

```
from my_lib.about import *
```

Remove

```
include tkMessageBox
```

Use the structure below instead of individual variables, and change the call from about_box to my_about_box. Change the binding to my_about_box.

```
prog_id = {'progname': 'N2_Test.py',
           'title': 'N2 Protocol Test GUI',
           'version': '1.1',
           'date': "12 January 2018",
           'rev_date': '25 January 2018',
           'author': "Peter Hedlund",
           'description': 'To perform testing of the N2 communications\n'
           ' protocol on the ABB Bypass unit.'}

def my_about_box(event=None):
    about_box(prog_id)
```

Help

```
Create a text file file called read.me and put in the information you want
displayed when you press
the help button.
The help button calls the my_help routine
inport help
def my_help()
    help.Dialog(mainframe, title =’You supply the title here”, filename = None)
None = use read.me otherwise you can put your own text file name here.
```

On Screen LED indicators

```
This section will describe how to use an led indicator on the screen that can be
used with the grid placement system. There is a file led.py that contains the code
on creating led’s.
To use what has been written:
```

```
import led

For a single led in your gui code area use

w, ind = led.make_led_1r(tab1, 'BUSY')
w.grid(column=1, row=0)

To turn the led on (the #dd4444 is the rgb code for red but you can use any code)
led.set_led_1r(w, ind, '#dd4444')


To turn if off (the #476042 is the rgb code for dark grey but you can use any code)
led.set_led_1r(w, ind, "#476042")
```

Text To / From the Clipboard

```
Part of the Tkinter library
root.clipboard_clear() #  Clear the clipboard
root.clipboard_append(“put me here”)   # append data to the clipboard
st = root.clipboard_get() # get whats in the clipboard to a string

Assuming st is a string with the value you want put into the clipboard
root.clipboard_clear()
root.clipboard_append(st)
root.update()
```

Canvas Widget

Creating a Canvas

```
Here is a sample widget. It was taken from a graphing program I wrote that has a
horizontal slider at the top of the screen

from tkinter import *
import tkinter as tk
from tkinter import ttkroot = tk.Tk()

root = tk.Tk()
root.geometry("1400x650")
root.geometry('%dx%d+%d+%d' % (1400, 850, 20,   20))

root.title("Graphing Accelerometer Data")

cv = tk.Canvas(root, height="550", width=1400, bg='white',
               scrollregion=(0, 0, MX_DATA, 550))
cv.grid(column=0, row=1, columnspan=10)
hbar = Scrollbar(root, orient=HORIZONTAL)
hbar.config(command=cv.xview)
hbar.grid(column=0, row=0, columnspan=10, sticky='ew')
cv['xscrollcommand'] = hbar.set
setup_screen()
root.mainloop()
```

Clearing a Canvas

cv.delete(all)

Drawing an Arc

To draw an arc you must understand the coordinates.

```
          90
          135       45
180               0
          225       315
               270
So to draw an arc that would look like a meter (open on the bottom)

cv.create_arc(700,100,1000,400,start=290, extent=320, width=2)

Note Start is staring point based on the drawing above and extent is the
number of degrees to swing the arc “Counter Clockwise” So keep in mind the compass
is not only off by 90 but it is also drawn in reverse.
```

Drawing a Line

Drawing a line from x1, y1, to x2 y2, line width and color

position 0, 0 is the upper left corner of the canvas.

```
cv.create_line(25, 25, 400, 25, width=1, fill='blue')
```

Drawing Text

Drawing from x,y the text the color in this case 200 printed at location 15,25 in blue

```
cv.create_text(15, 25, text='200', fill='blue')
```

Logging

Log Window – Creation

This creates a text widget to be used as a log window for output messages to the user.

This also includes a scroll bar to allow user to see previous log entries.

```
#  Text Widget with scrollbar
txt = Text(mainframe, width=50, height=24, fg='lightgreen', bg='black', padx=15,
pady=15)
txt.grid(column=3, row=1, columnspan=4, rowspan=15, sticky="nsew")
txt.insert(1.0, "Start log file\n")
txt.configure(state='disabled')

scrollb = ttk.Scrollbar(mainframe, command=txt.yview)
scrollb.grid(row=1, column=7, rowspan=15, sticky='nse')
txt['yscrollcommand'] = scrollb.set
```

Log Window – Clearing the window **

Unprotect the widget, delete contents of widget, protect widget

```
def clear_log():
    #  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')
```

Log Window – Save log to file

```
Save the text widget contents to a user defined file name.

def save_log():
    #  Get all text from text widget
    lines = txt.get("1.0", END).splitlines()
    #  Save as ???
    path = filedialog.asksaveasfilename(parent=mainframe)
    #  open file
    fo = open(path, 'w')
    #  iterate each line
    for line in lines:
        #  Write one line at a time to file
        fo.write(line + "\n")
    #  Close File
    fo.close()
```

Log Window – Write in different color **

This allows the user to write in a selected color to the text widget

note: Use separate functions for each color and different tag names.

```
def to_log_red(str1):
    # 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, str1)
    #  get ending text position
    ed = txt.index('insert')
    #  tag area bound by start and stop
    txt.tag_add("junk", st, ed)
    #  Change color of tagged area
    txt.tag_config("junk", 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')
** Note: Since creation of this document, the calls to writing to the log file have
changed, they have been moved into to_log.py. This includes to_log, to_log_red,
to_log_yellow, to_log_green, to_log_blue, to_log_cyan, to_log_white, to_log_violet
clear_log and save_log. To use, in the include area put
from to_log import *
and to call one of the functions, use
to_log_red(txt, 'text') where txt is the name of the Text widget
```

Log Window – Adding to the log **

Unprotect the widget, go to the end, add text, update other tasks, protect widget.

```
def to_log(str1):
    txt.configure(state='normal')
    txt.insert(END, str1)
    txt.see(END)
    txt.index(END)
    txt.update_idletasks()
    txt.configure(state='disabled')
```

Binding

Bind Return key to Entry Box

Have an entry widget that processes the entry when the user presses the <Return> key or presses the associated Calc Button.

```
#  if binding and calling from a button the function needs an event=None
def calc_a(event=None):
    calc_gen(fbus_cw, PARAM_3_1, "FBUS_CW_1 :\n")


#  Entry Widgets
en1 = ttk.Entry(mainframe, textvariable=fbus_cw)
en1.grid(column=1, row=1)
en1.bind('<Return>', calc_a)
#  Button Widgets
fbus_cw_b = ttk.Button(mainframe, text="Calc", command=calc_a)
fbus_cw_b.grid(column=2, row=1)
```

Binding a Key combo to all Widgets

```
Assuming you have a function about_box and a widget savelog

savelog.bind_all('<Control-Alt-KeyPress-r>', about_box)
will cause the about_box function to be called whenever
Ctrl-Alt-r are pressed.  Function being called must have (event=None) as its first
parameter.
```

Binding a Change in selected tab to a function

```
To have the function term_rcvr called whenever a tab has been changed
put this at the end of your tabcontrol section

tabControl = ttk.Notebook(mainframe)
tab1 = ttk.Frame(tabControl)
tabControl.add(tab1, text='Internal Loop')
tab2 = ttk.Frame(tabControl)
tabControl.add(tab2, text='Sim Android')
tab3 = ttk.Frame(tabControl)
tabControl.add(tab3, text='Sim Control')
tab4 = ttk.Frame(tabControl)
tabControl.add(tab4, text='Debug')

tabControl.grid(column=0, row=1, padx=5, pady=5)

tabControl.bind('<<NotebookTabChanged>>',term_rcvr)
```

MatPlotLib

Simple Plot

This program will plot the data from a csv file td.dat. It shows the minimum plot program.

```
from matplotlib import pyplot
import csv


def load_from_file(fname):
    """
    Get accelerometer x, y and z, course and speed from file (fname)
    """
    with open(fname, 'r') as csvfile:
        sr = csv.reader(csvfile, delimiter=',')
        amt = []; cnt=[]
        line_count = 0
        for row in sr:
            if row != []:
                # print(row)
                if line_count == 0:
                    print(f'Column names are {", ".join(row)}')
                else:
                    ##  print(f'x = {row[0]} y = {row[1]} z = {row[2]}  Speed =
{row[3]}')
                    cnt.append(row[1])
                    amt.append(int(row[2]))
                line_count += 1
        print(f'Processed {line_count} lines.')
        return cnt, amt


fname = 'td.dat'
ndx, amt = load_from_file(fname)

fig = pyplot.figure(f'TD Ameritrade : Data file {fname}')
fig.suptitle(f"Test _DEBUG_ = {_DEBUG_}")

axis= fig.add_axes([0.1,0.1,0.8,0.8])
pyplot.plot(ndx, amt)
axis.set_ylim([250000,270000])
pyplot.subplots_adjust(top=0.91, bottom=0.06, left=0.05, right=0.99,
                       hspace=0.25, wspace=0.35)
pyplot.grid(True)
pyplot.show()
```

Data File TD.Dat for example

```
11,8/17,261634,m
12,8/18,261563,tu
13,8/19,260430,w
14,8/20,260244,th
15,8/21,260167,f
16,8/24,262460,m
17,8/25,262727,tu
18,8/26,263515,w
19,8/27,263018,th
20,8/28,264665,f
21,8/31,263218,m
22,9/1,264572,tu
23,9/2,266857,w
24,9/3,261691,th
25,9/4,260911,f
27,9/8,257305,tu
28,9/9,260413,w
29,9/10,257725,th
30,9/11,258631,f
31,9/14,261096,m
32,9/15,262059,tu
33,9/16,262123,w
34,9/17,261399,th
35,9/18,259641,f
36,9/21,256070,m
37,9/22,256523,tu
38,9/23,252492,w
39,9/24,252541,th
40,9/25,254268,f
41,9/28,257397,m
42,9/29,257177,tu
43,9/30,258150,w
```

Plot Characteristics

Set Plot Figure Description

Create the fig variable

fig = pyplot.figure(f'Accelerometer vs GPS Speed : Data file {fname}')

Set Plot Figure Size

Sets the size of the figure

fig = pyplot.figure(f'Accelerometer vs GPS Speed : Data file {fname}',figsize=(18,7))

Set Plot X,Y position

After the fig variable is set … Position the figure at location 10x, 10y with the move_figure function

```
def move_figure(f, x, y):
    """Move figure's upper left corner to pixel (x, y)"""
    backend = matplotlib.get_backend()
    if backend == 'TkAgg':
        f.canvas.manager.window.wm_geometry("+%d+%d" % (x, y))
    elif backend == 'WXAgg':
        f.canvas.manager.window.SetPosition((x, y))
    else:
        f.canvas.manager.window.move(x, y)
move_figure(fig, 10, 10)
```

Set Plot Super title

```
After the figure variable is set
_DEBUG_ = 'T00021.dat'
fig.suptitle(f"Test Data Plot = {_DEBUG_}")
```

Set Grid Color

```
pyplot.grid(color='red')
```

Set Grid Color for One Line

```
ax = pyplot.subplot(212)
a = ax.get_ygridlines()
b = a[6]  # 6th horizontal grid line from the bottom
b.set_color('red')
```

SubPlots

```
Using the subplot command sets up the width, heights and instance of the subplots.
i.e. You want 8 plots arranged in 4 rows of 2 columns
[ PLOT 421]    [ PLOT 422]
[ PLOT 423]    [ PLOT 424]
[ PLOT 425]    [ PLOT 426]
[ PLOT 427]    [ PLOT 428]

pyplot.subplot(421)
pyplot.plot(x, y)
pyplot.subplot(422)
pyplot.plot(x, y1)
pyplot.subplot(423)
pyplot.plot(x, y2)
pyplot.subplot(424)
pyplot.plot(x, y3)
pyplot.subplot(425)
pyplot.plot(x, y4)
pyplot.subplot(426)
pyplot.plot(x, y5)
pyplot.subplot(427)
pyplot.plot(x, y6)
pyplot.subplot(428)
pyplot.plot(x, y7)
```

Set Plot Scale

Scales available ‘linear’, ‘log’, ‘symlog’, ‘lolog’

pyplot.yscale('linear')

Set Plot Limits (X and Y)

First assign a variable to the subplot

Then set the min/max of both x and y directions

```
 axis = pyplot.subplot(425)
 axis.set_ylim([lo_val, hi_val])
 axis.set_xlim([0, 1600])
```

Set Plot Grid

```
You can turn the grid on and off with the grid command
You can set the y (and /or) x tics with the yticks (xticks) command

 pyplot.grid(True)
# Set ticks to 15, 20, 25, 30, 35, 40 and 45
 pyplot.yticks([15,20,25,30,35,40,45])
```

Set Plot Legend

A legend needs a label for each data plotted (at the end of the plot command) and where to put the legend (in this example top left and since there are 8 legends I wanted them across the top hence ncol = 8

```
axis = pyplot.subplot(111)
axis.set_ylim([lo_val, hi_val])
axis.set_xlim([0, 1600])
pyplot.plot(c1, t1, linewidth=.75, label='C1 T1')
pyplot.plot(c2, t2, linewidth=.75, label='C2 T2')
pyplot.plot(c3, t3, linewidth=.75, label='C3 T3')
pyplot.plot(c4, t4, linewidth=.75, label='C4 T4')
pyplot.plot(c5, t5, linewidth=.75, label='C5 T5')
pyplot.plot(c6, t6, linewidth=.75, label='C6 T6')
pyplot.plot(c7, t7, linewidth=.75, label='C7 T7')
pyplot.plot(c8, t8, linewidth=.75, label='C8 T8')
pyplot.yscale('linear')
pyplot.title('C1 T1 - C8 T8')
pyplot.legend(loc='upper left', borderaxespad=0., ncol = 8)
pyplot.grid(True)
```

Set Plot Color

The color option in the plot command lets you set the line color.

```
pyplot.plot(c1, t1, color='black', label='C1 T1')
pyplot.plot(c2, t2, color='blue', label='C2 T2')
pyplot.plot(c3, t3, color='green', label='C3 T3')
pyplot.plot(c4, t4, color='red', label='C4 T4')
```

Set Plot Line Thickness

The linewidth option in the plot command sets the line thickness

```
pyplot.plot(c1, t1, linewidth=.75, label='C1 T1')
pyplot.plot(c2, t2, linewidth=.25, label='C2 T2')
pyplot.plot(c3, t3, linewidth=.50, label='C3 T3')
pyplot.plot(c4, t4, linewidth=1.0, label='C4 T4')
```

Full Plot Example

```
import math
import matplotlib
import matplotlib.pyplot as pyplot
import datetime
#  from scrap_data3 import *
import csv
def move_figure(f, x, y):
    """Move figure's upper left corner to pixel (x, y)"""
    backend = matplotlib.get_backend()
    if backend == 'TkAgg':
        f.canvas.manager.window.wm_geometry("+%d+%d" % (x, y))
    elif backend == 'WXAgg':
        f.canvas.manager.window.SetPosition((x, y))
    else:
        # This works for QT and GTK
        # You can also use window.setGeometry
        f.canvas.manager.window.move(x, y)
        # pyplot.show()

def load_temp_from_file(fname):
    with open(fname, 'r') as csvfile:
        sr = csv.reader(csvfile, delimiter=',')
        c = []; t=[];
        line_count = 0
        for row in sr:
            if row != []:
                wk = row[0]
                if wk.find('File_Name:') == -1:
                    if line_count == 0:
                        print(f'Column names are {", ".join(row)}')
                    else:
                        c.append(float(row[0]))
                        st = row[0]
                        st = st[6:]
                        t.append(float(row[9]))
                    line_count += 1
        print(f'Processed {line_count} lines.')
        return c, t

def about(val, goal, rng):
    if val >= goal -rng and val <= goal + rng:
        return True
    else:
        return False

plot_type = 8
##  -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

#  Reset All lists
t1 = []; t2 = []; t3 = []; t4 = []
t5 = []; t6 = []; t7 = []; t8 = []
c1 = []; c2 = []; c3 = []; c4 = []
c5 = []; c6 = []; c7 = []; c8 = []
```

```
# get temperature data and counts
c1, t1 = load_temp_from_file('collect/t00020.dat')
c2, t2 = load_temp_from_file('collect/t00021.dat')
c3, t3 = load_temp_from_file('collect/t00022.dat')
c4, t4 = load_temp_from_file('collect/t00023.dat')
c5, t5 = load_temp_from_file('collect/t00024.dat')
c6, t6 = load_temp_from_file('collect/t00025.dat')
c7, t7 = load_temp_from_file('collect/t00026.dat')
c8, t8 = load_temp_from_file('collect/t00027.dat')

fname = 'T00020 - T00027'
fig = pyplot.figure(f'Accelerometer/Gyro Temperature : Data file
{fname}',figsize=(18,9))

move_figure(fig, 10, 10)

lo_val = 15
hi_val = 45

if plot_type == 8:
    fig.suptitle(f"Test Data Plot = {_DEBUG_}")

    axis = pyplot.subplot(421)
    axis.set_ylim([lo_val, hi_val])
    axis.set_xlim([0, 1600])
    pyplot.plot(c1, t1, linewidth=.75)
    pyplot.yscale('linear')
    pyplot.title('C1 T1')
    pyplot.yticks([15,20,25,30,35,40,45])
    pyplot.grid(True)

    axis = pyplot.subplot(423)
    axis.set_ylim([lo_val, hi_val])
    axis.set_xlim([0, 1600])
    pyplot.plot(c2, t2, linewidth=.75)
    pyplot.yscale('linear')
    pyplot.title('C2 T2')
    pyplot.grid(True)

    axis = pyplot.subplot(425)
    axis.set_ylim([lo_val, hi_val])
    axis.set_xlim([0, 1600])
    pyplot.plot(c3, t3, color='red', linewidth=.75)
    pyplot.yscale('linear')
    pyplot.title('C3 T3')
    pyplot.grid(True)

    axis = pyplot.subplot(427)
    axis.set_ylim([lo_val, hi_val])
    axis.set_xlim([0, 1600])
    pyplot.plot(c4, t4, color='springgreen', linewidth = .5)
    pyplot.yscale('linear')
    pyplot.title('C4 T4')
    pyplot.grid(True)

    axis = pyplot.subplot(422)
    axis.set_ylim([lo_val, hi_val])
```

```
    axis.set_xlim([0, 1600])
    pyplot.plot(c5, t5, color='black', linewidth = .5)
    pyplot.yscale('linear')
    pyplot.title('C5 T5')
    pyplot.grid(True)

    axis = pyplot.subplot(424)
    axis.set_ylim([lo_val, hi_val])
    axis.set_xlim([0, 1600])
    pyplot.plot(c6, t6, color='black', linewidth = .5)
    pyplot.yscale('linear')
    pyplot.title('C6 T6')
    pyplot.grid(True)

    axis = pyplot.subplot(426)
    axis.set_ylim([lo_val, hi_val])
    axis.set_xlim([0, 1600])
    pyplot.plot(c7, t7, color='black', linewidth = .5)
    pyplot.yscale('linear')
    pyplot.title('C7 T7')
    pyplot.grid(True)

    axis = pyplot.subplot(428)
    axis.set_ylim([lo_val, hi_val])
    axis.set_xlim([0, 1600])
    pyplot.plot(c8, t8, color='black', linewidth = .5)
    pyplot.yscale('linear')
    pyplot.title('C8 T8')
    pyplot.grid(True)
else:
    fig.suptitle('Comparing Accel/Gyro Temperatures from Tests T00020 - T00027')
    axis = pyplot.subplot(111)
    axis.set_ylim([lo_val, hi_val])
    axis.set_xlim([0, 1600])
    pyplot.plot(c1, t1, linewidth=.75, label='C1 T1')
    pyplot.plot(c2, t2, linewidth=.75, label='C2 T2')
    pyplot.plot(c3, t3, linewidth=.75, label='C3 T3')
    pyplot.plot(c4, t4, linewidth=.75, label='C4 T4')
    pyplot.plot(c5, t5, linewidth=.75, label='C5 T5')
    pyplot.plot(c6, t6, linewidth=.75, label='C6 T6')
    pyplot.plot(c7, t7, linewidth=.75, label='C7 T7')
    pyplot.plot(c8, t8, linewidth=.75, label='C8 T8')
    pyplot.yscale('linear')
    pyplot.title('C1 T1 - C8 T8')
    pyplot.legend(loc='upper left', borderaxespad=0., ncol = 8)
    pyplot.grid(True)

fig.tight_layout()

pyplot.subplots_adjust(top=0.91, bottom=0.06, left=0.05, right=0.99,
                       hspace=0.25, wspace=0.35)

#  mng = pyplot.get_current_fig_manager()
#  mng.full_screen_toggle(/)

pyplot.show()
```

Full Plot Example with Tkinter Gui

The example below is a working program that no only contains the matplotlib plot window but 3 buttons, 2 live labels, 1 fixed label, 1 Combobox, 3 buttons and a slider.

The majority of the program is contained in the class FigureFrame. The program will let the user browse for a file to graph, select the graph style and allow (depending on style) adjustment.

```
#region  Include Files
import numpy as np
import tkinter as tk
from tkinter import ttk
import csv
import math
import os
from tkinter.filedialog import askopenfilename

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.backends.backend_tkagg import NavigationToolbar2Tk
from matplotlib.figure import Figure
#endregion Include Files

#region Constants Lists
gr_type = ['Accel VS Speed', 'Gyro Vs Speed', 'Wheels VS speed',
           'Wheels & Avg Wheel vs Speed','Accel & calc Speed vs Speed',
           'Accel Calc Speed vs Speed', 'Wheel avg z, calc spd, gps spd']
search_lbl = ['Xaz', 'Yaz', 'Zaz', 'Gx', 'Gy', 'Gz', 'LW', 'RW',
              'Bk', 'Gps_Speed']

#endregion Constant Lists

#region Functions and classes
def load(fn):
    v0 = []; v1 = []; v2 = []; v3 = []; v4 = []
    v5 = []; v6 = []; v7 = []; v8 = []; v9 = []
    """
    Search for different labels based on graph type
    """
    with open(fn, 'r') as csvfile:
        sr = csv.reader(csvfile, delimiter=',')
        line_count = 0
        for row in sr:
            if row != []:
                wk = row[0]
                if wk.find('File_Name:') == -1:
                    if line_count == 0:
                        # print(f'Column names are {", ".join(row)}')
                        a = row.index(search_lbl[0])
                        b = row.index(search_lbl[1])
                        c = row.index(search_lbl[2])
                        d = row.index(search_lbl[3])
                        e = row.index(search_lbl[4])
                        f = row.index(search_lbl[5])
```

```
                        g = row.index(search_lbl[6])
                        h = row.index(search_lbl[7])
                        i = row.index(search_lbl[8])
                        j = row.index(search_lbl[9])
                    else:
                        v0.append(float(row[a]) )
                        v1.append(float(row[b]) )
                        v2.append(float(row[c]) )
                        v3.append(float(row[d]) )
                        v4.append(float(row[e]) )
                        v5.append(float(row[f]) )
                        v6.append(float(row[g]) )
                        v7.append(float(row[h]) )
                        v8.append(float(row[i]) )
                        v9.append(float(row[j]) )
                        # row[9] is temperature
                    line_count += 1

        print(f'Processed {line_count} lines from {fn}.')
    return v0, v1, v2, v3, v4, v5, v6, v7, v8, v9


def _quit():
    win.quit()     # stops mainloop
    win.destroy()  # this is necessary on Windows to prevent
                    # Fatal Python Error: PyEval_RestoreThread: NULL tstate

class FigureFrame(tk.Frame):
    fsel = 0
    csel = 0
    w_fname = ''
    short_nm = ''
    old_cb2 = 0
    scale = 2.0
    old_scale = 100000

    def __init__(self,master=None,**kwargs):
        """ Here is where the gui is created """
        super().__init__(master,**kwargs)
        self.fig = Figure(figsize=(19, 9), dpi=100)
        self.canvas = FigureCanvasTkAgg(self.fig, master=self)
        x = 1
        tk.Button(self,text=" Browse ",command=self.browse).grid(row=0,
                  column=x,sticky="e")
        x += 1
        tk.Label(self, text='', width=20, relief=tk.RIDGE).grid(
                 row=0, column=x, sticky='w')
        x += 1
        tk.Label(self, text='Graph Type: ').grid(row=0, column=x, sticky='e')
        x += 1
        self.cb2 = ttk.Combobox(self, value=gr_type, state='readonly',
                                width=30)
        self.cb2.current(self.csel)
        self.cb2.grid(row=0, column=x, sticky='w')
        x += 1
        self.scl = ttk.Scale(self, value=self.scale, length=200, from_=0,
                             orient=tk.HORIZONTAL, to=4.0, command=self.round)
```

```
    self.scl.grid(row=0, column=x, sticky='e')
    x += 1
    tk.Label(self, text=self.scale, width=10, relief=tk.RIDGE).grid(
             row=0, column=x, sticky='w')
    x += 1
    tk.Button(self,text="  RE-PLOT  ",command=self.plot_next).grid(row=0,
              column=x,sticky="w")
    x += 1
    tk.Button(self,text=" [ QUIT ] ",command=_quit).grid(row=0,column=x,
              sticky="w")
    graph_widget = self.canvas.get_tk_widget()
    graph_widget.grid(row=1, column=0, columnspan=15, sticky = 'nsew')
    self.after(2000, self.cng)


def browse(self):
    path = askopenfilename(filetypes=(('Dat Files', '*.dat'),
                                      ('Tst Files', '*.tst'),
                                      ('All Files', '*.*')))
    if path == '':
        self.short_nm = ''
        self.w_fname = ''
        return
    self.w_fname = path
    self.short_nm = os.path.basename(path)
    tk.Label(self, text=self.short_nm, width=20, relief=tk.RIDGE).grid(
             row=0, column=2, sticky='w')
    self.plot_next()


def round(self, pos):
    val = float(pos)
    val2 = int(val * 100)
    val = val2/100
    fs = '{:2.3}'.format(val)
    tk.Label(self, text=fs, width=10, relief=tk.RIDGE).grid(
             row=0, column=6, sticky='w')


def cng(self):
    x = self.cb2.current()
    if self.short_nm != '':
        if x != self.old_cb2:
            self.old_cb2 = x
            if x == 5:
                self.scl.config(from_=0, to=5, value=2.5)
                self.round(2.5)
            elif x == 6:
                self.scl.config(from_=-2, to=5, value=0.0)
                self.round(self.scl.get())
            self.plot_next()
    else:
        self.old_cb2 = x

    self.after(500, self.cng)
```

```
def get_baseline(self,val):
    sm = 0
    for x in range(1,11):
        sm += val[x]
    return sm/10


def plot_next(self):
    self.fig.clear()
    x = self.cb2.current()
    if self.short_nm == '':
        return
    subttl = f"{gr_type[x]} Plot For {self.short_nm} Test Runs"
    self.fig.suptitle(subttl)
    m = []; n=[]; o=[]; p=[]; q = []
    ax,ay,az,gx,gy,gz,wl,wr,bk,spd = load(self.w_fname)
    if x == 0:
        m = ax; n = ay; o = az; q = spd
        lbl = ['Acc x','Acc y', 'Acc z','','Gps Speed']
    elif x == 1:
        m = gx; n = gy; o = gz; q = spd
        lbl = ['Gyro x','Gyro y', 'Gyro z','','Gps Speed']
    elif x == 2:
        m = wl; n = wr; o = bk; q = spd
        lbl = ['Left WS','Right WS', 'Brake','','Gps Speed']
    elif x == 3:
        m = wl; n = wr; o = bk; q = spd
        lbl = ['Left WS','Right WS', 'Brake','Avg Whl','Gps Speed']
        for w in range(0, len(ax)):
            p.append((m[w] + n[w]) / 2)
    elif x == 4:
        m = ax; n = ay; o = az; q = spd
        lbl = ['Acc x','Acc y', 'Acc z','Calc Spd','Gps Speed']
        p.append(0)
        for w in range(1, len(ax)):
            if spd[w] == 0:
                p.append(0)
            else:
                p.append(p[w-1] + az[w] + (az[1]/2))
    elif x == 5:
        m = ax; n = ay; o = az; q = spd
        lbl = ['Acc x','Acc y', 'Acc z','Calc Spd','Gps Speed']
        p.append(0)
        sc = float(self.scl.get())
        bl = self.get_baseline(az)
        ofs = bl * sc
        for w in range(1, len(ax)):
            z = az[w]
            # simple noise filter
            if (z > (az[1] + ofs)) or (z < (az[1] - ofs)):
                p.append(p[w-1] + az[w] + az[1])
            else:
                p.append(p[w-1] + 0)
    elif x == 6:
        n = ay; o = az; q = spd
        lbl = ['Wheel Avg Spd','Acc y','Acc z','Calc Spd','Gps Speed']
        sc = float(self.scl.get())
```

```
            bl = self.get_baseline(az)
            ofs = bl * sc
            for w in range(0, len(ax)):
                m.append((wl[w] + wr[w]) / 2)

            p.append(0)
            for w in range(1, len(ax)):
                z = az[w]
                # simple noise filter
                if (z > (az[1] + ofs)) or (z < (az[1] - ofs)):
                    p.append(p[w-1] + az[w] + az[1])
                else:
                    p.append(p[w-1] + 0)
        else:   #  Default setting Filtered Accel vs Speed
            m = ax; n = ay; o = az; q = spd
            lbl = ['Acc x','Acc y', 'Acc z','','Gps Speed']

        self.fig.add_subplot(511,title=lbl[0]).plot(m, linewidth=.6)
        self.fig.add_subplot(512,title=lbl[1]).plot(n, linewidth=.6)
        self.fig.add_subplot(513,title=lbl[2]).plot(o, linewidth=.6)
        if len(p) > 0:
            self.fig.add_subplot(514,title=lbl[3]).plot(
                                p, color='green', linewidth=.6)
        self.fig.add_subplot(515,title=lbl[4]).plot(q, color='red',
         linewidth=.6)
        self.fig.subplots_adjust(top=0.91, bottom=0.06, left=0.05, right=0.95,
                        hspace=0.25, wspace=0.35)

        self.canvas.draw_idle()
#endregion Functions and classes


win = tk.Tk()
win.title('Data Analsys D1905 -Speed-')
# win.geometry('+{}+{}'.format(5,10))
win.geometry('+5+10')
fnames = []
start_num = 28
end_num =70
# for x in range(start_num, end_num+1):
#     fnames.append('T000{:2d}.dat'.format(x))
f = FigureFrame(win)
f.grid(row=0,column=0)

win.mainloop()
```

Tutorials

Converting C Structures to Python

C Structure

Name	Type	Size (bytes)
Record_Num	Long Int	4
Date_Time	char[14\	14
Generator Power	Float	4
Seat Occupied	Boolean	1
Checksum	Unsigned char	1

Python

```
import struct
RecNm = ('Record Num', 'Date Time', 'Gen Power', 'Seat Stat', 'cksum')
rec_size = 24
struct_str = '=I 14s f B B'
s = struct.Struct(struct_str)

def get_packet(rec):
    fp = open(df, 'rb')
    fp.seek(rec * rec_size)
    st = fp.read(rec_size)
    return st

def calc_csum(rec):
    cs = 0
    for x in range(0, rec_size-1):
        cs = cs + rec[x]
        cs = cs % 256
    return cs

def show_record(rec):
    for x in range(0, len(rec)):
        print(f'{record[x]:>5}:{rec[x]}')

# Read records 5 through 14
for x in range(5, 15):
    st = get_packet(x)    # Read Record
    cs = calc_csum(st)    # Calculate checksum
    raw = struct.unpack(sstr, st) # convert bin data to record
    #  Print record #, calculated checksum and stored checksum
    print(f'Record {x}  Calc Checksum {cs}  Stored Checksum {raw[len(raw) - 1]}')

#Shows last record
show_record(raw)
```

Sample Menu Widget

```
The following code creates a menubar at the top of the window
(the following functions are assumed to be in existence
quit_prog, my_clear_log, my_save_log, my_help, my_about)


root = Tk()

# Creating Menubar
menubar = Menu(root)
file = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='File', menu = file)
file.add_separator()
file.add_command(label='Exit', command=quit_prog)

edit = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Edit', menu = edit)
edit.add_command(label='Clear Log', command=my_clear_log)
edit.add_command(label='Save Log', command=my_save_log)

help_ = Menu(menubar, tearoff = 0)
menubar.add_cascade(label ='Help', menu = help_)
help_.add_command(label='Help', command=my_help)
help_.add_command(label='About', command=my_about)

# display Menu
root.config(menu = menubar)
```

Using a Matrix to fill out GUI

When dealing with multiple buttons, labels, radio buttons, etc, etc, making a change can be involved since every instance of the involved widgets needs to be changed. I am going to show you a method for organizing your gui widgets that leds itself to rapid editing and reduces over all code. In this sample I will use buttons and labels

```
lbl1=StringVar()
lbl2=StringVar()
lbl1.set('')
lbl2.set('')
ctrl = (('BTN', 0, 0, 'Start',     start_rout),
      ('BTN', 0, 1, 'Stop',     stop_rout),
      ('BTN', 0, 2, 'Abort',   abort_rout),
      ('BTN', 0, 5, 'Quit',     quit_prog),
      ('LBL', 0, 3,  lbl1),
      ('LBL', 0, 4, lbl2),
for x in range(0, len(ctrl)):
      if ctrl[x][0] == 'BTN':
            bt = ttk.Button(mainframe, text=ctrl[x][3], command=ctrl[x][4])
            bt.grid(column=ctrl[x][1], row=ctrl[x][2], padx=5, pady=5)
      elif ctrl[x][0] == 'LBL'
            lb = ttk.Label(mainframe, textvariable=ctrl[x][3])
            lb.grid(column=ctrl[x][1], row=ctrl[x][2], padx=5, pady=5)
```

With this way of thinking, you can add additional buttons or labels with only 1 line of code for each instead of 2 lines and you get the advantage of having all properties in one array. The layout of the array can be whatever you desire, I have the first column be widget type, second column, third row, fourth text and in the case of the buttons fifth is the function to call when pressed. If for some reason you wanted to add width to the labels, place it in the fifth position and if there is a label that you don't want to specify the width enter None in that field.

Simple State Machine

This state machine uses tkinter and the if elif else statements as well as the root.after timer.
The purpose of this sample is to show HOW to make a state machine. It reads from a script file and outputs it to the terminal. The file mc_script.scr is a text file.

```
from tkinter import *
from tkinter import ttk

wfn = 'mc_script.scr'

def state_mach():
    state = sm_state.get()  # get into local variables
    x = fp_pos.get()
    if state == 0:  # Idle State -- Do nothing
        pass
    elif state == 1:  # Start State machine, read first line
        fp = open(wfn, 'rt')
        cl = fp.readline()
        x = fp.tell()
        print(cl[:-1])  # for print() eliminate \n at end of the string
        fp.close()
        state += 1
    elif state == 2:  # wait a bit
        state += 1
    elif state == 3:  # Read subsuquent lines until done
        fp = open(wfn, 'rt')
        fp.seek(x)
        cl = fp.readline()
        if cl != '':
            x = fp.tell()
            print(cl[:-1])  # for print() eliminate \n at end of the string
            fp.close()
            state -= 1
        else:   # safety exit Similar to default in c
            state = 0
    sm_state.set(state)  # restore variables
    fp_pos.set(x)
    root.after(500, state_mach)

def start_sm():
    if sm_state.get() == 0:  # make sure we are not running
        sm_state.set(1)

root = Tk()
sm_state = IntVar()
fp_pos = IntVar()
mainframe = ttk.Frame(root, padding='3', height=200, width=230)
mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
mainframe.grid_propagate(0)

btn = ttk.Button(mainframe, text='Start SM', command=start_sm)
btn.grid(row=0, column=0)
state_mach()
root.mainloop()
```

Debugging your code

There are several ways to debug code, Using the Ide's debugger, printing to the terminal, output to a file, etc. The Ide's debugger works fine until the bug your working on only happens after a while.

You may find you need different debugging techniques depending on what problem your looking into.

In my example, there are 4 ways to output your debug info, 1. The terminal (print command), 2. On the screen of the program itself, 3 to a file, 4. don't do anything with it. (debug_destination)

There is also the code area to debug this can be selected by adding with the variable (debug_extra)

debug_destination and debug_extra are read from a config file at the start of the program Tkinter is needed for this.

The file handling is done with a library I created called file_log and the screen routines are handled with my library scrn_log.

The skeleton is shown below setting up the constants and variables, read_config is not show as I have examples of reading config files elsewhere.

There is a skeleton Tkinter gui program outlined

I kept the ability to change the two debug variables out of my program and they are only changed in the ini file after exiting the program. This is a personal choice for the application I was debugging you requirements may be different than mine. As is said this is a suggestion. Any type of output device can be used for this style of debugging, Serial Port, led's attached to gpio, even using a tone out the speaker

```
import mylib.scrn_log as log              # Screen log library
import mylib.file_log as my_log           # file log library

dbg_none = 0                              # define debug_destinations
dbg_term = 1
dbg_scrn = 2
dgb_file = 3

# generic reporting module
def reporting(dstr):
    if debug_destination.get() == dbg_term:
        print(dstr)
    elif debug_destination.get() == dbg_scrn:
        scrn.log_scrn(dstr + '\n')
    elif debug_destination.get() == dbg_file:
        my_fl.log(dstr + '\n')

# This routine only gets executed if bit 1 is set in debug_extra
def ins_report(rl, x, by):
    if (debug_extra.get() & 1) == 1:
        z1 = instrument.cts
        z2 = instrument.dsr
        st = f'rl = {rl:<6} x = {x:<6}  '
        st += f'Bytes {by:4}  CTS = {z1}  DSR = {z2}'
        reporting(st)

root = Tk()
# Create Varsiables
debug_extra = IntVar()
debug_destination = IntVar()

# Set Default Values
debug_extra.set(0)
debug_destination.set(0)

read_config()

# make the gui
root.resizable(0, 0)
root.title(prog_id['title'])
mainframe = ttk.Frame(root, padding='3', height=600, width=830)
mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
mainframe.grid_propagate(0)
root.geometry(f'+{savex.get()}+{savey.get()}')
root.protocol("WM_DELETE_WINDOW", quit_prog)

# buttons and gui widgets go here

#setup the screen logging
scrn = log.scrn_log(mainframe, 57, 34, 'lightgreen', 'black', 2, 0, 6, 20 )

#setup the file logging if necessary
if debug_destination.get() == dbg_file:
    my_fl = my_log.file_log('test_log.txt')

root.mainloop()
```

Timer Issues in Python

Quitting a timer at the wrong time can cause a crash.
Running multiple timers (root.after,some_time, a_function) can cause problems when trying to stop the timer and exiting the program without errors.

```
class tmr:
      tn = ''

tm1 = tmr
tm2 = tmr

def timer1():
      do_some_work
      tm1.tn = root.after(1000, timer1)

def timer2():
  do_other_work
      tm2.tn = root.after(2000, timer2)

def start_tmr():
  timer1()
      timer2()

def quit_prog():
      root.after_cancel(tm1.tn)
      root.after_cancel(tm2.tn)
      root.destroy()
```

In this example quit prog will only cancel the timers if they are in idle, waiting for the timer to trip.
If either timer is doing work then the quit_prog is called, only the idle timer will be cancelled. The other timer will cause an error.

While the after_cancel is fine for only one timer, the more timers you have the more problems you will run into with management.

Proposal (Not perfect but a start)
Use a status for each timer, that way you can be sure all timers are off before terminating the program.

```
class tmr:
      tn = ''
      st = 0

tm1 = tmr
tm2 = tmr

def timer1():
      do_work
      if tmr1.st == 2
            tm1.st = 3
            tm1.tn=''
            return
      tm1.tn = root.after(1000, timer1)
```

```
def timer2():
      do_other_work
      if tmr2.st == 2
            tm2.st = 3
            tm2.tn=''
            return
      tm2.tn = root.after(1000, timer2)

def start_tmr():
      if tm1.ts == 0:
            tm1.ts = 1
            timer1()
      if tm2.ts == 0:
            tm2.ts = 1
            timer2()

def stop_tmr():
      if tm1.ts == 1:
            tm1.ts = 2
      if tm2.ts == 1:
            tm2.ts = 2

def quit_prog():
      if tm1.ts == 1:
            tm1.ts = 2
      else:
        tm1.ts = 3
      if tm2.ts == 1:
            tm2.ts = 2
      else:
        tm2.ts = 3
      if (tm1.ts == 3) and (tm2.ts == 3):
            root.destroy
      root.after(200, quit_prog()
```

This method uses a status to get current timer status and is setup to allow any existing timers to time out before exiting the program.

Data Integrity (checksum and crc)

This tutorial show how to calculate checksum 8 and checksum 16 as well as crc 16 and crc 16 CCITT

pkt is an array of bytes

```
def checksum_8(pkt):
    crc = 0
    for x in range(0, len(pkt)):
        crc += pkt[x]
    crc %= 256
    return crc

def checksum_16(pkt):
    crc = 0
    for x in range(0, len(pkt)):
        crc += pkt[x]
    crc %= 65536
    return crc

def crc16_CCITT(data : bytearray, offset , length):
    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

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 crc_16( st, crc):
    """Given a bunary string and starting CRC, Calc a final CRC-16 """
    for ch in st:
        crc = (crc >> 8) ^ table[(crc ^ ch) & 0xFF]
    return crc
```

Test Data

```
data = (0x02, 0x02, 0x5B, 0xD8, 0x80, 0x39, 0xF3, 0x4D, 0x98, 0x5D, 0x03)
data = bytes(data)
data2 = (0x02, 0x04, 0x5B, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
    0x00, 0x00, 0x00, 0x5D, 0x03)
data2 = bytes(data2)
```

Examples

Function Call	**Result**
checksum_8(data)	0x28
checksum_8(data2)	0xC3
checksum_16(data)	0x0428
checksum_16(data2)	0x00C3
crc_16(data, INITIAL_DF1)	0x1d47
crc_16(data2, INITIAL_DF1)	0x9DD9
crc16_CCITT(data, 0, length(my_data))	0x5984
crc16_CCITT(data2, 0, length(my_data))	0x7871

Simple setup menu (using Toplevel)

This code creates a simple setup menu to show how to create your own. The code displays a dummy window, pressing the setup button causes the setup gui to appear. The only option is to save current window position so the next time the program is run it returns to this position.

```
# pylint: disable=unused-wildcard-import, method-hidden
#
# #-----------------------------------------------------------------------------
# Name:        setup_demo
# Purpose:
#
# Author:      PHedlund
# Created:     07/29/2021
# Copyright:   (c) PHedlund 2021
#-------------------------------------------------------------------------------

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


##  ------------------------------------------------------------------------
##  Constants
##  ------------------------------------------------------------------------
fn = os.path.dirname(sys.argv[0]) + '\\' + 'setup_demo.ini'
_debug_ = 0

Author = 'Peter Hedlund'
Version = '1.0'
Release_Date = '07/22/2021'
Update_Date = ''

##  ------------------------------------------------------------------------
##  Functions
##  ------------------------------------------------------------------------
def save_config():
    config = ConfigParser()
    if remember.get() == 1:
        xx, yy = get_xy_pos()
        if _debug_ == 1:
            print(f'{xx}, {yy}')
        x_pos.set(xx)
        y_pos.set(yy)
    config['position'] = {'remember' : remember.get(),
```

```
                          'x_pos'    : x_pos.get(),
                          'y_pos'    : y_pos.get() }
    with open(fn, 'w') as configfile:
        config.write(configfile)


def load_config():
    config = ConfigParser()
    config.read(fn)
    remember.set(config['position']['remember'])
    x_pos.set(config['position']['x_pos'])
    y_pos.set(config['position']['y_pos'])


def get_xy_pos():
    rg = root.geometry()
    xi = rg.find('+')
    yi = rg.find('+', xi+1)
    x = int(rg[xi+1:yi])
    y = int(rg[yi+1:len(rg)])
    return x, y


##  ---------------------------------------------------------------------------
##  Controls
##  ---------------------------------------------------------------------------
def setup_quit():
    global top
    top.destroy()


def setup_save():
    global top
    save_config()
    top.destroy()


def setup_window():
    global top
    try:
        if top.state() == 'normal':
            top.focus()
    except:
        top = Toplevel()
        top.title('Setting')
        x,y = get_xy_pos()
        top.geometry(f'190x270+{x+20}+{y+20}')
        tk.Label(top, text='  ').grid(column=3, row=x+2)
        rem = ttk.Checkbutton(top, variable=remember, text='Save x,y')
        rem.grid(column=0, row=x+3, columnspan=3)
        tk.Label(top, text='  ').grid(column=0, row=x+4)

        tk.Label(top, text=' ').grid(column=4, row=x+5)
        qu = tk.Button(top, text='  Cancel  ', command=setup_quit)
        qu.grid(column=2,row=x+5, columnspan=3, pady=3)
        qu2 = tk.Button(top, text='  SAVE  ', command=setup_save)
```

```
        qu2.grid(column=0,row=x+5, columnspan=3, pady=3)


def quit_prog():
    root.destroy()


##  ---------------------------------------------------------------------
##
##  GUI Program Starts Here
##
##  ---------------------------------------------------------------------
root = tk.Tk()
disp_mode = StringVar()
remember = IntVar()
x_pos = IntVar()
y_pos = IntVar()
disp_mode.set('')
x_pos.set(10)
y_pos.set(10)
bg_a = 'white'
bg_b = 'black'

if os.path.exists(fn) == False:
    save_config()
load_config()
if remember.get() == 0:
    x_pos.set(10)
    y_pos.set(10)

root.geometry('%dx%d+%d+%d' % (245, 150, x_pos.get(), y_pos.get()))
root.title("PyClock")
s1 = ttk.Style()
s1.configure('red.TButton', background='Red')

if _debug_ == 1:
    print(disp_mode.get())

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

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

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

stop2_b = tk.Button(root, text=' Settings ', command= setup_window)
stop2_b.grid(column=0, row=5, padx=1)

quit = ttk.Button(root, text=' Quit ', command=quit_prog, style='red.TButton')
quit.grid(column=2, row=5, padx=1)

root.mainloop()
```

APPENDIX

A: General Notes

1. Keep definitions of similar widgets together with a comment, at the start of the group of widgets.
2. Add docstring to the beginning of your code to give general info.
3. To Run python programs from editor (synwrite) Create a button that calls the Python interpeter

    Use python.exe if you want live debug info to the output box of the editor

    Use pythonw.exe if you want the debug info AFTER the program finishes to the output window.

B: Compile to Single EXE file

Use tool nuitka

```
Compile decoder.py to decoder.exe where decoder.py is a windows not console program
For A Windows Program
From a dos shell type
c:\user\python27\scripts\nuitka –recurse-all --windows-disable-console decoder.py
or create a batch file
n_build.bat
c:\user\python27\scripts\nuitka ^
      --recurse-all ^
      --windows-disable-console decoder.py
And run the batch file.On Screen LED's

For a dos console program
From a dos shell type
c:\user\python27\scripts\nuitka –recurse-all decoder.py
or create a batch file
n_build.bat
c:\user\python27\scripts\nuitka ^
      --recurse-all ^
            decoder.py
And run the batch file.
```

Use py2exe and NSIS Installer

To be created

Use Pyinstaller to create a standalone program

```
Create a batch file in your project directory (we will use crypto.py as our target)
python is installed at c:\user\python37_32
pyinstaller was installed with pip install pyinstaller from the c:\user\
python37_32\scripts directory

create a batch file make_exe.bat
```

```
PATH=%PATH%;c:\user\python37_32\scripts
pyinstaller crypto.py -F -w
rem optional
copy puzzle.dat dist
copy answer.dat dist
```

```
run make_exe
Under the dist directory will be your executable.

Warning: This method is not yet working with matplotlib program.
This may be due to the antivirus program on my test computer.
```

C: Python Packages (Libraries)

Format of a Package

This tutorial show how to create and use a package named mylib with the files, about.py, help.py and led.py. This tutor assumes python is located in c:\user\python27\ and the 3 files already exist.

1. Create a directory **c:\user\python27\Lib\mylib**
2. Copy help.py c:\user\python27\Lib\mylib
3. copy about.py c:\user\python27\Lib\mylib
4. copy led.py c:\user\python27\Lib\mylib
5. Create an empty file called __init__.py in the c:\user\python27\Lib\mylib directory.

Use of a package

```
To use this library
OLD Python 2.7
import help
import about
import led

led.set_led_1r(w, ind, led_on_color)
about.about_box(prog_id)
help.Dialog(mainframe, title='N2_Test Help File', filename='read.me')

NEW Python 3.7
import mylib.help
import mylib.about
import mylib.led

mylib.led.set_led_1r(w, ind, led_on_color)
mylib.about.about_box(prog_id)
mylib.help.Dialog(mainframe, title='N2_Test Help File', filename='read.me')
```

Using Doc Strings in a Package

```
The Doc String consists of text enclosed in three quotes on either end
“”” This is a doc string “””
Af far as Packages are concerned, Inserting a doc string before any code or
includes, makes that the doc string for the package
Any doc strings in a function make that functions doc string.
As an example here is the led_d.py file in the mylib directory
(c:\user\python37-37\Lib\mylib\led_d.py)
(working code removed for clarity) to demonstrate the Doc String
```

```
"""
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.
"""
 from tkinter import *

LIB_VERSION = "1.0.0"

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

def get_full_lib_version():
    """Returns version information and library name."""

def make_led_r(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"""

def make_led_s(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"""

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."""

def get_led(c, led):
    """Returns the status of the led."""
```

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
```

Contents of a package

```
The module led_d will be looked at
In Idle
>>> dir(mylib.led_d)
['ACTIVE', 'ALL', 'ANCHOR', 'ARC', 'BASELINE', 'BEVEL', 'BOTH', 'BOTTOM', 'BROWSE',
'BUTT', 'BaseWidget', 'BitmapImage', 'BooleanVar', 'Button', 'CASCADE', 'CENTER',
'CHAR', 'CHECKBUTTON', 'CHORD', 'COMMAND', 'CURRENT', 'CallWrapper', 'Canvas',
'Checkbutton', 'DISABLED', 'DOTBOX', 'DoubleVar', 'E', 'END', 'EW', 'EXCEPTION',
'EXTENDED', 'Entry', 'Event', 'EventType', 'FALSE', 'FIRST', 'FLAT', 'Frame',
'GROOVE', 'Grid', 'HIDDEN', 'HORIZONTAL', 'INSERT', 'INSIDE', 'Image', 'IntVar',
'LAST', 'LEFT', 'LIB_VERSION', 'Label', 'LabelFrame', 'Listbox', 'MITER', 'MOVETO',
'MULTIPLE', 'Menu', 'Menubutton', 'Message', 'Misc', 'N', 'NE', 'NO', 'NONE',
'NORMAL', 'NS', 'NSEW', 'NUMERIC', 'NW', 'NoDefaultRoot', 'OFF', 'ON', 'OUTSIDE',
'OptionMenu', 'PAGES', 'PIESLICE', 'PROJECTING', 'Pack', 'PanedWindow',
'PhotoImage', 'Place', 'RADIOBUTTON', 'RAISED', 'READABLE', 'RIDGE', 'RIGHT',
'ROUND', 'Radiobutton', 'S', 'SCROLL', 'SE', 'SEL', 'SEL_FIRST', 'SEL_LAST',
'SEPARATOR', 'SINGLE', 'SOLID', 'SUNKEN', 'SW', 'Scale', 'Scrollbar', 'Spinbox',
'StringVar', 'TOP', 'TRUE', 'Tcl', 'TclError', 'TclVersion', 'Text', 'Tk',
'TkVersion', 'Toplevel', 'UNDERLINE', 'UNITS', 'VERTICAL', 'Variable', 'W', 'WORD',
'WRITABLE', 'Widget', 'Wm', 'X', 'XView', 'Y', 'YES', 'YView', '__builtins__',
'__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', 'constants', 'enum', 'get_full_lib_version', 'get_led',
'get_lib_version', 'getboolean', 'getdouble', 'getint', 'image_names',
'image_types', 'mainloop', 'make_led_r', 'make_led_s', 're', 'set_led', 'sys',
'wantobjects']
>>>

The user functions in this module are

get_full_lib_version
get_led
get_lib_version
make_led_r
make_led_s
set_led
```

D: Exceptions List

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
+-- StopIteration
+-- StopAsyncIteration
+-- ArithmeticError
| +-- FloatingPointError
| +-- OverflowError
| +-- ZeroDivisionError
+-- AssertionError
+-- AttributeError
+-- BufferError
+-- EOFError
+-- ImportError
+-- LookupError
| +-- IndexError
| +-- KeyError
+-- MemoryError
+-- NameError
| +-- UnboundLocalError
+-- OSError
| +-- BlockingIOError
| +-- ChildProcessError
| +-- ConnectionError
| | +-- BrokenPipeError
| | +-- ConnectionAbortedError
| | +-- ConnectionRefusedError
| | +-- ConnectionResetError
| +-- FileExistsError
| +-- FileNotFoundError
| +-- InterruptedError
| +-- IsADirectoryError
| +-- NotADirectoryError
| +-- PermissionError
| +-- ProcessLookupError
| +-- TimeoutError
+-- ReferenceError
+-- RuntimeError
| +-- NotImplementedError
| +-- RecursionError
+-- SyntaxError
| +-- IndentationError
| +-- TabError
+-- SystemError
+-- TypeError
+-- ValueError
| +-- UnicodeError
| +-- UnicodeDecodeError
| +-- UnicodeEncodeError
| +-- UnicodeTranslateError
+-- Warning
+-- DeprecationWarning
```

```
+-- PendingDeprecationWarning
+-- RuntimeWarning
+-- SyntaxWarning
+-- UserWarning
+-- FutureWarning
+-- ImportWarning
+-- UnicodeWarning
+-- BytesWarning
+-- ResourceWarning
```

E: Format Specifiers

Print Format Specifiers

You can format numbers using the format specifier given below:

Type	Meaning
d	Decimal integer
c	Corresponding Unicode character
b	Binary format
o	Octal format
x	Hexadecimal format (lower case)
X	Hexadecimal format (upper case)
n	Same as 'd'. Except it uses current locale setting for number separator
e	Exponential notation. (lowercase e)
E	Exponential notation (uppercase E)
f	Displays fixed point number (Default: 6) {:6.3f} shows 6 char max 3 dec pt
F	Same as 'f'. Except displays 'inf' as 'INF' and 'nan' as 'NAN'
g	General format. Rounds number to p significant digits. (Default precision: 6)
G	Same as 'g'. Except switches to 'E' if the number is large.
%	Percentage. Multiples by 100 and puts % at the end.

```
Note See: Simple Use of String Format
Examples
   print("{} : {}".format(10, 20))
   Outputs    10 : 20
   print("{1} : {0}".format(10, 20))
   Outputs    20 : 10
   print("{1:02X} : {0:02X}  {6:3f}".format(10, 20, 234.333456))
   Outputs    14 : 0A  234.333
   print(" 0x{0:02x}  0x{0:02X}  0b{0:08b}".format(10))
   Outputs    0x0a  0x0A  0b00001010
```

Number formatting with alignment

```
The operators <, ^, > and = are used for alignment when assigned a certain width to
the numbers.
Number formatting with alignment
```

Type	Meaning
<	Left aligned to the remaining space
^	Center aligned to the remaining space
>	Right aligned to the remaining space
=	Forces the signed (+) (-) to the leftmost position

Format Characters for struct.pack and struct.unpack

Format characters have the following meaning; the conversion between C and Python values should be obvious given their types. The ‘Standard size’ column refers to the size of the packed value in bytes when using standard size; that is, when the format string starts with one of `'<'`, `'>'`, `'!'` or `'='`. When using native size, the size of the packed value is platform-dependent.

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	bytes of length 1	1	
b	signed char	integer	1	(1), (2)
B	unsigned char	integer	1	(2)
?	_Bool	bool	1	(1)
h	short	integer	2	(2)
H	unsigned short	integer	2	(2)
i	int	integer	4	(2)
I	unsigned int	integer	4	(2)
l	long	integer	4	(2)
L	unsigned long	integer	4	(2)
q	long long	integer	8	(2)
Q	unsigned long long	integer	8	(2)
n	ssize_t	integer		(3)
N	size_t	integer		(3)
e	(6)	float	2	(4)
f	float	float	4	(4)
d	double	float	8	(4)
s	char[]	bytes		
p	char[]	bytes		
P	void *	integer		(5)

Notes:

1. The '?' conversion code corresponds to the _Bool type defined by C99. If this type is not available, it is simulated using a char. In standard mode, it is always represented by one byte.
2. When attempting to pack a non-integer using any of the integer conversion codes, if the non-integer has a `__index__()` method then that method is called to convert the argument to an integer before packing.
   Changed in version 3.2: Use of the `__index__()` method for non-integers is new in 3.2.
3. The 'n' and 'N' conversion codes are only available for the native size (selected as the default or with the '@' byte order character). For the standard size, you can use whichever of the other integer formats fits your application.
4. For the 'f', 'd' and 'e' conversion codes, the packed representation uses the IEEE 754 binary32, binary64 or binary16 format (for 'f', 'd' or 'e' respectively), regardless of the floating-point format used by the platform.

5. The 'P' format character is only available for the native byte ordering (selected as the default or with the '@' byte order character). The byte order character '=' chooses to use little- or big-endian ordering based on the host system. The struct module does not interpret this as native ordering, so the 'P' format is not available.
6. The IEEE 754 binary16 "half precision" type was introduced in the 2008 revision of the IEEE 754 standard. It has a sign bit, a 5-bit exponent and 11-bit precision (with 10 bits explicitly stored), and can represent numbers between approximately 6.1e-05 and 6.5e+04 at full precision. This type is not widely supported by C compilers: on a typical machine, an unsigned short can be used for storage, but not for math operations. See the Wikipedia page on the half-precision floating-point format for more information.

Strftime Format specifiers

```
strftime(format[, tuple]) -> string

Convert a time tuple to a string according to a format specification.
See the library reference manual for formatting codes. When the time tuple
is not present, current time as returned by localtime() is used.

Commonly used format codes:
```

%Y	Year with century as a decimal number.
%m	Month as a decimal number [01,12].
%d	Day of the month as a decimal number [01,31].
%H	Hour (24-hour clock) as a decimal number [00,23].
%M	Minute as a decimal number [00,59].
%S	Second as a decimal number [00,61].
%z	Time zone offset from UTC.
%a	Locale's abbreviated weekday name.
%A	Locale's full weekday name.
%b	Locale's abbreviated month name.
%B	Locale's full month name.
%c	Locale's appropriate date and time representation.
%I	Hour (12-hour clock) as a decimal number [01,12].
%p	Locale's equivalent of either AM or PM.

```
Other codes may be available on your platform.  See documentation for
the C library strftime function.
```

local time structure

```
tm = time.localtime()
```

Name	Contains	Name	Contains	Name	Contains
tm.tm_year	Year	tm.tm_mon	Month	tm.tm_mday	Day of month
tm.tm_hour	Hour of day	tm.tm_min	Minute	tm.tm_sec	Seconds
tm.tm_wday	Day of week	tm.tm_yday	Day of year	tm.tm_isdst	Daylight sav

F: Keys supported by keyboard module.

'backspace'	'execute'	f	z	'/'	‘f20’	'volume down'
'tab'	'print screen'	g	'left windows'	‘f1’	‘f21’	'volume up'
'clear'	'insert'	h	'right windows'	‘f2’	‘f22’	next track'
'enter'	'delete'	i	'applications'	‘f3’	‘f23’	'previous track’
'shift'	'help'	j	'sleep'	‘f4’	‘f24’	'stop media'
'ctrl'	0	k	0	‘f5’	'num lock'	'play/pause media'
'alt'	1	l	1	‘f6’	'scroll lock'	'start mail'
'pause'	2	m	2	‘f7’	‘left shift’	'select media'
'caps lock'	3	n	3	‘f8’	‘right shift’	'start application 1'
'spacebar'	4	o	4	‘f9’	'left ctrl'	'start application 2'
'page up',	5	p	5	‘f10’	‘right ctrl'	
'page down'	6	q	6	‘f11’	‘left menu’	
'end'	7	r	7	‘f12’	‘right menu’	
'home'	8	s	8	‘f13’	'browser back'	
'left'	9	t	9	‘f14’	'browser forward'	
'up'	a	u	*	‘f15’	'browser refresh'	
'right'	b	v	+	‘f16’	'browser stop’	
'down'	c	w	'separator'	‘f17’	'browser search key'	
'select'	d	x	‘-’	‘f18’	'browser favorites'	
'print'	e	y	‘decimal'	‘f19’	'browser start and home’	

G: Standard Colors, Styles and Escape Codes

Standard Color Definitions

"black"	"cyan"	"light blue"	"magenta"	"white"
"blue"	"gray"	"light gray"	"red"	"yellow"
"brown"	"green"	"light green"	"violet"	

System Colors Definitions

SystemActiveBorder	SystemButtonText	SystemInactiveCaptionText
SystemActiveCaption	SystemCaptionText	SystemMenu
SystemAppWorkspace	SystemDisabledText	SystemMenuText
SystemBackground	SystemHighlight	SystemScrollbar
SystemButtonFace	SystemHighlightText	SystemWindow
SystemButtonHighlight	SystemInactiveBorder	SystemWindowFrame
SystemButtonShadow	SystemInactiveCaption	SystemWindowText

Escape Character Codes

\'	Single Quote		\N	Name (unicode)
\"	Double Quote		\ooo	Octal Character
\\	Backslash		\r	Carrage return
\a	Bell Character		\t	Tab
\b	Backspace		\uxxxx	16 bit unicode char
\f	Formfeed		\Uxxxxxxxx	32 bit unicode char
\n	Newline		\xhh	8 bit hex character

Box Character Codes

┌	\u255c	┬	\u252c	┐	\u2510	╔	\u2554	╦	\u2556	╗	\u2557
├	\u251c	┼	\u253c	┤	\u2524	╠	\u2560	╬	\u256c	╣	\u2563
└	\u2514	┴	\u2534	┘	\u2518	╚	\u255a	╩	\u2569	╝	\u255d
│	\u2502	─	\u2500			═	\u2551	═	\u2550		
╓	\u2553	╥	\u2565	╖	\u2556	╒	\u2552	╤	\u2564	╕	\u2555
╟	\u255f	╫	\u256b	╢	\u2562	╞	\u255e	╪	\u256a	╡	\u2561
╙	\u2559	╨	\u2568	╜	\u255c	╘	\u2558	╧	\u2567	╛	\u255b

Style Themes

```
>>>from tkinter import ttk
>>>s=ttk.Style()
>>>s.theme_names()
```

'winnative'	'alt'	'classic'	'xpnative'
'clam'	'default'	'vista'	

Ansi Escape codes

```
# Note : make sure the following code snippets is in your code to
# enable ansi escape codes.

from colorama import init
init()
```

Normal Foreground		**Normal Background**		**Cursor position**	
Black	'\033[30m'	Black	'\033[40m'	UP n lines	'\x1b[nA'
Red	'\033[31m'	Red	'\033[41m'	Down n lines	'\x1b[nB'
Green	'\033[32m'	Green	'\033[42m'	Fwd n chars	'\x1b[nC'
Yellow	'\033[33m'	Yellow	'\033[43m'	Bkwd n chars	'\x1b[nD'
Blue	'\033[34m'	Blue	'\033[44m'	Beg next n line	'\x1b[nF'
Magenta	'\033[35m'	Magenta	'\033[45m'	Pos n current line	'\x1b[nG'
Cyan	'\033[36m'	Cyan	'\033[46m'	Move to x,y	'\x1b[x,yH'
White	'\033[37m'	White	'\033[47m'	Erase disp n (0-3)	'\x1b[nJ'
Bright Foreground		**Bright Background**		**Cursor position**	
Black	'\033[90m'	Black	'\033[100m'	Erase in line n	'\x1b[nK'
Red	'\033[91m'	Red	'\033[101m'	Scroll up n	'\x1b[nS'
Green	'\033[92m'	Green	'\033[102m'	Scroll dn n	'\x1b[nT'
Yellow	'\033[93m'	Yellow	'\033[103m'	Save Pos	'\x1b[ns'
Blue	'\033[94m'	Blue	'\033[104m'	Restore Pos	'\x1b[nt'
Magenta	'\033[95m'	Magenta	'\033[105m'	Show cursor	'\x1b[?25h'
Cyan	'\033[96m'	Cyan	'\033[106m'	Hide Cursor	'\x1b[?25l'
White	'\033[97m'	White	'\033[107m'		
NORMAL Color	'\x1b[0m'	Bold	'\x1b[1m'		
Italic	'\x1b[2m'	Underline	'\x1b[3m'		

H: Operators

Arithmetic	**Adjust**	**Binary**
Addition +	Increment +=	A=0011 0001
Subtraction -	Decrement -=	B=1001 0011
Multiplication *	Multiply *=	And B&A 0001 0001
Division /	Divide /=	Or B|A 1011 0011
Modulus %	Exponent **=	Xor B^A 1010 0010
Exponents **	Floor //=	Not ~A 1100 1110
Floor Division //		A << 2 1100 0100
Comparison	**Logical**	A >> 2 0000 1100
Equal ==	A = true	
Not Equal !=	B = false	
Greater than >	A and B false	
Less than <	A or B true	
Greater then or equal >=	not(a and b) true	
Less than or equal <=	A xor B true	

I: Math Library

Most popular math functions
Import math

Function	**Description**
	PYTHON MATH FUNCTIONS
math.e	Returns Euler's Number 2.71828
math.pi	Return Pi Value 3.14
Math.inf	Positive infinity
Math.nan	Return Not A Number as output
divmod(t,d)	Modulus divide, returns 2 values int(t/d) and t%d
Fabs(x)	Absolute value of x
fmod(x,y)	Calculates the module
gcd(x,y)	Returns greatest common divisor of x and y
isclose(x,y)	Returns true if x and y are close to each other otherwise false
isinf(x)	True if infinity(Pos or neg) False otherwise
isnan(x)	True if x id a Number or false if not (Not A Number)
round(x)	Rounds x to nearest integer
trunc(x)	Removes decimal value from x and returns integer only.
	PYTHON POWER AND LOG FUNCTIONS
exp(x)	Calculates power of E where E is Euler's nuber 2.71828

pow(x)	Power of x
sqrt(x)	Square Root of x
	PYTHON TRIG FUNCTIONS
acos(x)	Arc Cosine
asin(x)	Arc Sine
atan(x)	Arc Tangent
atan2(y,x)	Returns angle (in rad) from x axix to y,x point
cos(x)	Cosine
sin(x)	Sine
tan(x)	Tangent
	PYTHON ANGULAR FUNCTIONS
degrees(x)	Converts from radians to degrees
radians(x)	Convert from degrees to radians

J: Mouse Events

Tkinter uses so-called event sequences for allowing the user to define which events, both specific and general, he or she wants to bind to handlers. It is the first argument "event" of the bind method. The event sequence is given as a string, using the following syntax:

<modifier-type-detail>

The type field is the essential part of an event specifier, whereas the "modifier" and "detail" fields are not obligatory and are left out in many cases. They are used to provide additional information for the chosen "type". The event "type" describes the kind of event to be bound, e.g. actions like mouse clicks, key presses or the widget got the input focus.

Event	Description
<Button>	A mouse button is pressed with the mouse pointer over the widget. The detail part specifies which button, e.g. The left mouse button is defined by the event <Button-1>, the middle button by <Button-2>, and the rightmost mouse button by <Button-3>. <Button-4> defines the scroll up event on mice with wheel support and and <Button-5> the scroll down. If you press down a mouse button over a widget and keep it pressed, Tkinter will automatically "grab" the mouse pointer. Further mouse events like Motion and Release events will be sent to the current widget, even if the mouse is moved outside the current widget. The current position, relative to the widget, of the mouse pointer is provided in the x and y members of the event object passed to the callback. You can use ButtonPress instead of Button, or even leave it out completely: , , and <1> are all synonyms.
<Motion>	The mouse is moved with a mouse button being held down. To specify the left, middle or right mouse button use <B1-Motion>, <B2-Motion> and <B3-Motion> respectively. The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback, i.e. event.x, event.y
<ButtonRelease>	Event, if a button is released. To specify the left, middle or right mouse button use <ButtonRelease-1>, <ButtonRelease-2>, and <ButtonRelease-3> respectively. The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback, i.e. event.x, event.y
<Double-Button>	Similar to the Button event, see above, but the button is double clicked instead of a single click. To specify the left, middle or right mouse button use <Double-Button-1>, <Double-Button-2>, and <Double-Button-3> respectively. You can use Double or Triple as prefixes. Note that if you bind to both a single click (<Button-1>) and a double click (<Double-Button-1>), both bindings will be called.
<Enter>	The mouse pointer entered the widget. Attention: This doesn't mean that the user pressed the Enter key!. <Return> is used for this purpose.
<Leave>	The mouse pointer left the widget.
<FocusIn>	Keyboard focus was moved to this widget, or to a child of this widget.

<FocusOut>	Keyboard focus was moved from this widget to another widget.
<Return>	The user pressed the Enter key. You can bind to virtually all keys on the keyboard: The special keys are Cancel (the Break key), BackSpace, Tab, Return(the Enter key), Shift_L (any Shift key), Control_L (any Control key), Alt_L (any Alt key), Pause, Caps_Lock, Escape, Prior (Page Up), Next (Page Down), End, Home, Left, Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, Num_Lock, and Scroll_Lock.
<Key>	The user pressed any key. The key is provided in the char member of the event object passed to the callback (this is an empty string for special keys).
a	The user typed an "a" key. Most printable characters can be used as is. The exceptions are space (<space>) and less than (<less>). Note that 1 is a keyboard binding, while <1> is a button binding.
<Shift-Up>	The user pressed the Up arrow, while holding the Shift key pressed. You can use prefixes like Alt, Shift, and Control.
<Configure>	The size of the widget changed. The new size is provided in the width and height attributes of the event object passed to the callback. On some platforms, it can mean that the location changed.

K: String Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
just()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value

Method	Description
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Note: All string methods returns new values. They do not change the original string.

L: Lists

Creation

```
L = [1,2,3,4,5,9,8,7,6]
>>>print L
returns: [1,2,3,4,5,9,8,7,6]
```

Accessing / Indexing

```
L[0]  = returns '1'
L.index(9) = returns 5 (works with string lists as well.
```

Slicing

```
L[1:4]  = returns [2,3,4]
L[2:]   = returns [3,4,5,9,8,7,6]
L[:2]   = returns [1,2]
L[-1]   = returns 6
L[1:-1] = returns [2,3,4,5,9,8,7]
```

Convert tupil to List

```
List(seq)
```

Keyword "in" - can be used to test if an item is in a list

```
if 'red' in L:
    print "list contains", 'red'
```

For-in statement - makes it easy to loop over the items in a list

```
for item in L:
    print item
```

List Methods

Starting with the below list for all functions

```
ls = [1,2,3,4,5,6,7,8,9,10]
b = [10,11,12,13]
```

Method	Usage	Result
Appending	ls.append(11)	[1,2,3,4,5,6,7,8,9,10,11]
Count val in list	ls.count(5)	1 (5 is only in the list 1 time)
Extend	ls.extend(b)	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 11, 12, 13]
Index val in list	ls.index(5)	4 (5 is in the 4th position in the list from 0)
Insert	ls.insert(1,12)	[1, 12, 2, 3, 4, 5, 6, 7, 8, 9, 10] insert 12 at pos 1
inLength	len(l)	10
Maximum Val	A = max(ls)	10
Minimum Val	A = min(ls)	1
pop item	ls.pop(5)	Returns 6, ls = [1, 2, 3, 4, 5, 7, 8, 9, 10]
pop last	ls.pop()	Returns 10, l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Remove	ls.remove(2)	[1, 3, 4, 5, 6, 7, 8, 9, 10] removes 2 from list
Reverse list order	ls.reverse()	[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Sorting	c = sorted(ls)	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

M: Sample GUI Template in Python

This is the template I would recommend for your python program. It helps organize sections of your code. This is by no means the be all and end all of templates but merely a guide to help with code consistency. This program will run as is and it creates a simple gui with a entry box and a quit button.

This layout was setup based on a program that required the gui to adapt based on settings in the configuration file. Hence after the root-TK(), the variable declaration, default values and read configuration before creating the gui. I would also recommend using regions if your editor supports them to help with code organization and clarity when focusing on a particular area.

```
#-------------------------------------------------------------------------------
# Name:        module1
# Purpose:
#
# Author:      Your Name goes here
#
# Created:     Todays Date
# Copyright:   (c) Your name  This Year
#-------------------------------------------------------------------------------
##  ------------------------------------------------------------------------
##  Imports
##  ------------------------------------------------------------------------
from tkinter import *
from tkinter import ttk

##  ------------------------------------------------------------------------
##  TODO Lists Goes Here
##  ------------------------------------------------------------------------

##  ------------------------------------------------------------------------
##  Constants
##  ------------------------------------------------------------------------

##  ------------------------------------------------------------------------
##  Functions
##  ------------------------------------------------------------------------

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


def quit_prog():
    root.destroy()


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

```
root = Tk()

# Declare variables of type StringVar and Intvar here
works = StringVar()

# Define defaults for variables
works.set(‘’)
# load config file
# Create GUI
#  prevent window resizing.
root.resizable(0, 0)
root.title('')
mainframe = ttk.Frame(root, padding='3', height=120, width=180)
mainframe.grid(column=1, row=2, sticky=(N, W, E, S))
mainframe.grid_propagate(0)

# define any styles here

# Create Notebooks

# Create Notebook Tabs

# Create Menus

# Create Widgets

lbl = ttk.Label(mainframe, text='Test Label')
lbl.grid(column=0, row=0)
ans = ttk.Entry(mainframe, textvariable=works)
ans.grid(column=0, row=1)
quitProg = ttk.Button(mainframe, text='Done', command=quit_prog)
quitProg.grid(column=0, row=2)

# Create Status Bar

# File Logging setup

# Re-occuring functions

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

root.mainloop()
```

N: Using Editors

Using Sublime Text with Python

Sublime Text version of Snippets

Doc String

```
<snippet>
    <content><![CDATA[
"""
Title:   ${1}
Author:  ${2}
Created: ${3}
Purpose: ${4}
"""
${0}
]]></content>
    <scope>source.python</scope>
    <description> Doc String </description>
</snippet>
```

Add Button Widget Snippet

```
<snippet>
    <content><![CDATA[
${1} = ttk.Button(mainframe, text='${2}',command={${3})
${1}.grid(column=${4}, row=${5})
${0}
]]></content>
    <scope>source.python</scope>
    <description> Add button widget </description>
</snippet>
Add Entry Widget
<snippet>
    <content><![CDATA[
${1} = ttk.Entry(mainframe, textvariable=${2})
${1}.grid(column=${3}, row=${4})
${0}
]]></content>
    <scope>source.python</scope>
    <description> Add button widget </description>
</snippet>
```

Add Label Widget

```
<snippet>
    <content><![CDATA[
${1} = ttk.Label(mainframe, text='${2}')
${1}.grid(column=${3}, row=${4} sticky='w')
${0}
]]></content>
    <scope>source.python</scope>
    <description> Add button widget </description>
</snippet>
```

Using Visual Studio Code

Disable Warnings in Visual Studio Code

```
To disable warning in a file in Visual Studio code add this line to to top of your
code
(The Problem tab at the bottom of the screen)
For example to disable the warning
Unused import enum from wildcard import pylint(unused-wildcard-import)

Add this line to the top of the file

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

Recommended Extensions for Visual Studio Code

Extension Name	**Author**
Bookmarks	Alessandro Fragnani
Overtype	Adam Maras
Print Code	Nobuhito
Project Manager	Alessandro Fragnani
Python	Microsoft
Python for VSCode	Thomas Haakon Townsend
Python Snippets	Ferhat Yalcin
Seperators	Alessandro Fragnani
Todo Tree	Gruntfuggly

O: Recommendations

Add On Libraries

pySerial : Serial Port Support
https://www.github.com/pyserial/pyserial

minimalModbus : Modbus Support
https://pypi.python.org/pypi/MinimalModbus

configobj : Read/Write config files (ini)
http://www.voidspace.org.uk/python/configobj.html

keyboard : read keyboard keys and status

auto-py-to-exe : Front end for pyinstaller
https://pypi.org/project/auto-py-to-exe/

matplotlib : Graphic plotting libraries includes numpi
https://matplotlib.org/

pygame : Libraries for wrighting games in python.
https://www.pygame.org/news

Editors

pyscripter
https://sourceforge.net/projects/pyscripter

Sublime Text
https://www.sublimetext.com/

Notepad++
https://notepad-plus-plus.org

PyCharm (community)
https://www.jetbrains.com/pycharm/download/#section=windows

Visual Studio Code
https://code.visualstudio.com

Programs

Python : The Python Language
https://www.python.org/

PyWin32 : Windows 32 bit extension
https://sourceforge.net/projects/pywin32/files/pywin32/

NSIS Installer
https://sourceforge.net/projects/nsis/

pyinstaller works for python 3.7
https://www.pyinstaller.org/

PYTHON 2.7 AND BELOW ONLY

Nuitka : Convert .py to single .exe file
http://nuitka.net/

py2exe (not working for python 3.7
http://www.py2exe.org/

P: Musical Note to Frequency Conversion Chart

Note	Frequency	Note	Frequency	Note	Frequency
A0		A3	110	A6	880
B0		B3	123	B6	988
C0	16	C3	131	C6	1047
D0	18	D3	147	D6	1175
E0	21	E3	165	E6	1319
F0	22	F3	175	F6	1397
G0	25	G3	196	G6	1568
A1	28	A4	220	A7	1760
B1	31	B4	247	B7	1976
C1	33	C4	262	C7	2093
D1	37	D4	294	D7	2349
E1	41	E4	330	E7	2637
F1	44	F4	349	F7	2794
G1	49	G4	392	G7	3136
A2	55	A5	440	A8	3520
B2	62	B5	494	B8	3951
C2	65	C5	523	C8	4186
D2	73	D5	587	D8	4699
E2	82	E5	659	E8	5274
F2	87	F5	698	F8	5588
G2	98	G5	784	G8	6272

Conversion chart from letter note to frequency (Hz). Middle C on the piano keyboard is C_4 at 262 Hz, and the highest note on the piano is C_8 at 4186 Hz. Hearing is typically tested between C_4 and an octave above the highest note on the piano keyboard. A common notation is to have both the note and the frequency together, as A[440], which is also A_5. To get the semitone frequency, multiply the note below it by $\sqrt[12]{2}$ or 1.0595. Note. From The Acoustical Foundations of Music (p. 153), by J. Backus, 1977, New York: W. W. Norton & Company, Inc. Copyright 1977 by W. W. Norton & Company, Inc. Adapted with permission.

Duration
Whole Note = 4 beats
Half Note = 2 beats
Quarter Note = 1 beat
Eight Note = 1/2 beat
Sixteenth Note = 1/4 beat

Q: ASCII Table

ASCII characters 0 to 127

Code 0 to 31 (and # 127) are non-printing, mostly obsolete control characters that affect how text is processed. There are 95 printable characters.

To print one, press the ALT key (hold it down) and type the decimal number.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	∑
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Notes: