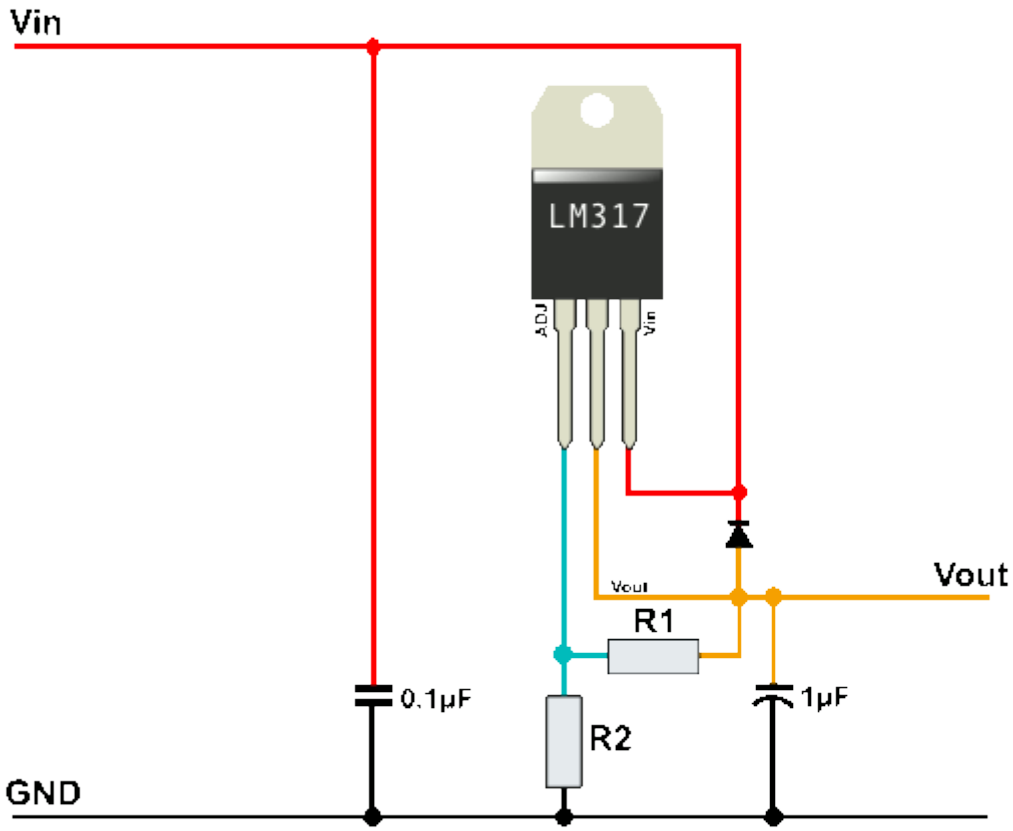
An LM317 Calculator and Python Deployment

One of the things I have been looking at with some trepidation is deployment of programs written in python as packages that can be run on different machines that don't necessarily have python installed. In my investigations the two tools that most people seem to use are ***cx_Freeze*** (https://anthony-tuininga.github.io/cx_Freeze/) and, for Windows, ***py2exe*** (http://www.py2exe.org/). Ultimately I am looking to package a Pygame / Tkinter project with a bunch of resources such as images, sound files, and text files. To get to grips with the deployment tools I decided to write a small *Tkinter* tool for calculating the resistance and voltage values for the LM317 voltage regulator. To make it a little more challenging I wanted to include a full colour image in the interface. What I ended up with was a nice little calculator that I thought would be worthwhile sharing.

I took the calculation formula and basic variable voltage regulator circuit diagram from the Texas Instrument LM317 datasheet (http://www.ti.com/lit/ds/symlink/lm317.pdf see page 10). The circuit diagram layout has been rejigged a little obviously, and some of the ripple control components removed.



The formula given in the datasheet is:

$$V_o = V_{ref}\left(1+\frac{R_2}{R_1}\right)+I_{adj}R_2$$

where:

- *Vo* is the output voltage
- Vref is 1.25V
- R1 and R2 are the two resistors shown on the diagram
- Iadj is the current through the ADJ leg of the regulator and is very small – typically 50µA.

Although the last term in the expression is very small, I decided to make the calculator a little bit more accurate by including it. This also meant my calculator required an iterator for calculating R2.

So here are the formula implemented in the python code;
In the case where resistances R1 and R2 are known the output voltage is calculated by;

$$V_o = V_{ref}\left(1+\frac{R_2}{R_1}\right)+I_{adj}R_2$$

In the case where the target voltage Vo and resistance R1 are known, R2 can be calculated with;

$$R_2 = R_1\left(\frac{V_o - I_{adj}R_2}{V_{ref}} - 1\right)$$

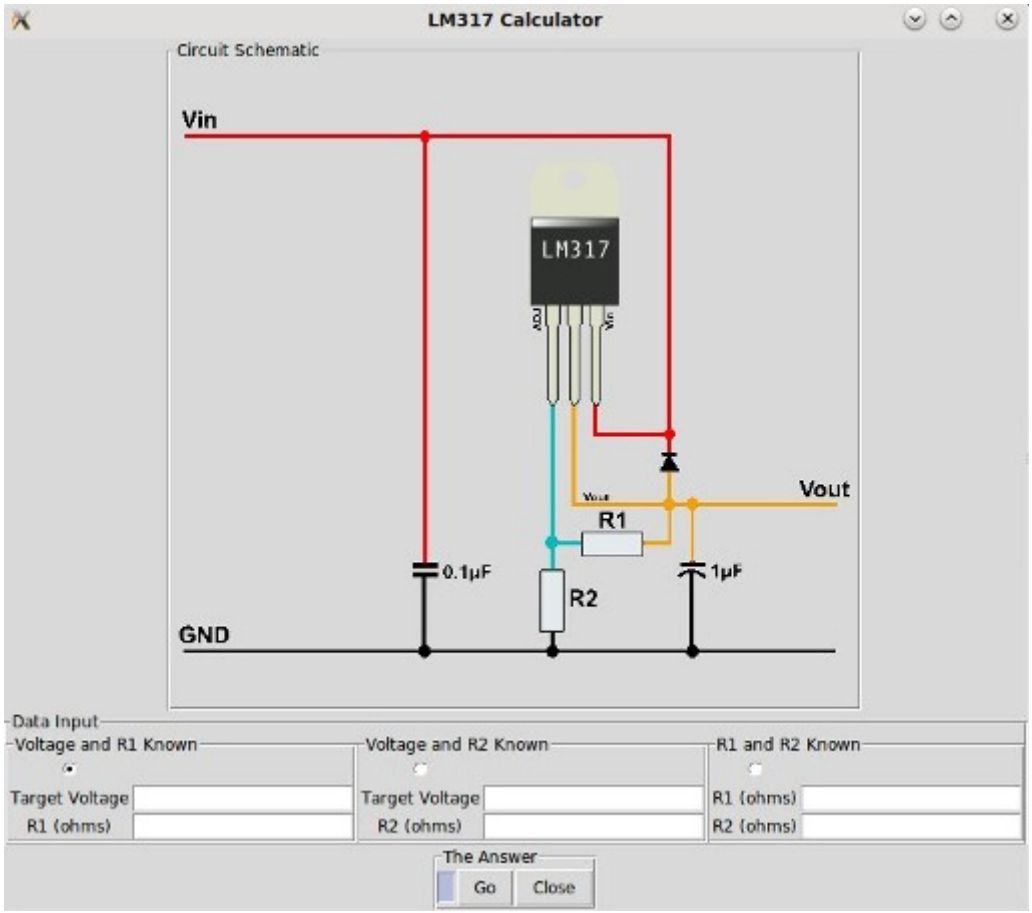
R2 appears on both sides and so needed to be iterated to find it. This converges very quickly and typically took about two or three rounds to settle on a number within 1% of the final value.

In the case where the target voltage Vo and resistance R2 are known, R1 can be calculated using;

$$R_1 = \frac{R_2 V_{ref}}{V_o - I_{adj}R_2 - V_{ref}}$$

The Python Code

The Tkinter interface was relatively straight forward to produce with *Frames* to guide the layout. The nifty bit was learning of the existence of the PhotoImage class in Tkinter which allows any .gif image to be used where the standard (and very boring) Tkinter images are used. You will notice when you run this that the radio button follows where you are actively entering data. This was achieved by binding the mouse click events to an inelegant bunch of functions that updated the variable controlling which radio button was selected.



One quirk is that the values you might have used in some fields don't clear if you move onto entering values into another section of the calculator. Not that this is a problem because it means you can click on the radio

button for that section and reactivate those values.

Quirks aside, the script was really built to test deployment python code.

The code was written in Python 2.7. If you want to run it in Python 3.X just change the

```
import Tkinter as tk
```

to

```
import tkinter as tk
```

and it will work for you.

You can download the python code and resources here:
http://www.techmonkeybusiness.com/Code/LM317Calcv3.zip

The Code Listing

```
# LM317Calcv3.py
# This is a tKinter based calculator to calculate the resistances and output
voltages when using
# LM317 voltage regulators.  All calculations based on those in the Texas
Instruments LM317 datasheet.
# This will be used to testing deployment methods in Linux and Windows

# V0 sucessfully outputs correct answers to the terminal
# V1 Outputs to the Answer box
# V2 Puts a colour image in the Top section using the tkinter PhotoImage class
# V3 Says to hell with it, lets do this properly and include the Iadj term as
well for more accuracy

# The tool was written in python 2.7
# Program by Hamish Trolove - www techmonkeybusiness.com
# You are free to distribute it but cannot charge for it.  Creative Common 4.0
by-nc-sa type thing.

# No responsibility taken for use of this tool.  After all it was written to
test python code
# deployment methods rather than make the ultimate LM317 calculator.


import Tkinter as tk
import os



class Calculator:

    InitialPath = os.getcwd()

    def __init__(self):
        Homepath = Calculator.InitialPath

        self.CalcWin = tk.Tk()
        self.CalcWin.wm_title("LM317 Calculator" )   #Allows us to add text to
the Tkinter window title

        Imagefile = os.path.join(Homepath,"resources","LM317Circuit.gif")
        CircuitPic = tk.PhotoImage(file=Imagefile)
```

```
        self.Area1 = tk.LabelFrame(self.CalcWin, text="Circuit Schematic")  #
Frame to contain the Circuit Image
        self.Area1.pack()

        self.Area2 = tk.LabelFrame(self.CalcWin, text="Data Input")  # Frame to
contain the Circuit Image
        self.Area2.pack()

        self.SubArea1 = tk.LabelFrame(self.Area2, text="Voltage and R1 Known")
# Frame to contain the Output Voltage and R1
        self.SubArea1.grid(row =0 , column =0)

        self.SubArea2 = tk.LabelFrame(self.Area2, text="Voltage and R2 Known")
# Frame to contain the Output voltage and R2
        self.SubArea2.grid(row =0 , column =1)

        self.SubArea3 = tk.LabelFrame(self.Area2, text="R1 and R2 Known")  #
Frame to contain the data for r1 and r2
        self.SubArea3.grid(row =0 , column =2)

        self.Area3 = tk.LabelFrame(self.CalcWin, text="The Answer")  # Frame to
contain the Calculation output
        self.Area3.pack()

        #The Circuit diagram box
        self.ImageLabel = tk.Label(self.Area1,  image = CircuitPic ).pack()

        #Radio button arrangement
        self.CalcTypeSel = tk.IntVar()
        self.CalcTypeSel.set(1) #Preset the radio button selection
        self.CalcTypeVR1 = tk.Radiobutton(self.SubArea1, variable =
self.CalcTypeSel,  value = 1).grid(row=0, column=0)
        self.CalcTypeVR2 = tk.Radiobutton(self.SubArea2, variable =
self.CalcTypeSel,  value = 2).grid(row=0, column=0)
        self.CalcTypeR1R2 = tk.Radiobutton(self.SubArea3, variable =
self.CalcTypeSel,  value = 3).grid(row=0, column=0)



        #Entry boxes for data input target voltage and R1 values
        self.VoltsAOutVal = tk.StringVar()
        self.VoltALabel = tk.Label(self.SubArea1,  text = "Target
Voltage").grid(row = 1,  column = 0)
        self.VoltAEntry = tk.Entry(self.SubArea1, textvariable =
self.VoltsAOutVal)
        self.VoltAEntry.bind("<Button-1>",self.UpdateRadioPoint1)  #Have the
radio button follow action
        self.VoltAEntry.grid(row = 1, column = 1)

        self.R1AVal = tk.StringVar()
        self.R1ALabel = tk.Label(self.SubArea1,  text = "R1 (ohms)").grid(row =
2,  column = 0)
        self.R1AEntry = tk.Entry(self.SubArea1, textvariable = self.R1AVal)
        self.R1AEntry.bind("<Button-1>",self.UpdateRadioPoint1)  #Have the radio
button follow action
        self.R1AEntry.grid(row = 2, column = 1)

        #Entry boxes for Data input target voltage and R2 values
        self.VoltsBOutVal = tk.StringVar()
```

```
        self.VoltBLabel = tk.Label(self.SubArea2,  text = "Target
Voltage").grid(row = 1,  column = 0)
        self.VoltBEntry = tk.Entry(self.SubArea2, textvariable =
self.VoltsBOutVal)
        self.VoltBEntry.bind("<Button-1>",self.UpdateRadioPoint2)  #Have the
radio button follow action
        self.VoltBEntry.grid(row = 1, column = 1)

        self.R2BVal = tk.StringVar()
        self.R2BLabel = tk.Label(self.SubArea2,  text = "R2 (ohms)").grid(row =
2,  column = 0)
        self.R2BEntry = tk.Entry(self.SubArea2, textvariable = self.R2BVal)
        self.R2BEntry.bind("<Button-1>",self.UpdateRadioPoint2)  #Have the radio
button follow action
        self.R2BEntry.grid(row = 2, column = 1)

        #Entry Boxes for Data input R1 and R2
        self.R1CVal = tk.StringVar()
        self.R1CLabel = tk.Label(self.SubArea3,  text = "R1 (ohms)").grid(row =
1,  column = 0)
        self.R1CEntry = tk.Entry(self.SubArea3, textvariable = self.R1CVal)
        self.R1CEntry.bind("<Button-1>",self.UpdateRadioPoint3)  #Have the radio
button follow action
        self.R1CEntry.grid(row = 1, column = 1)

        self.R2CVal = tk.StringVar()
        self.R2CLabel = tk.Label(self.SubArea3,  text = "R2 (ohms)").grid(row =
2,  column = 0)
        self.R2CEntry = tk.Entry(self.SubArea3, textvariable = self.R2CVal)
        self.R2CEntry.bind("<Button-1>",self.UpdateRadioPoint3)  #Have the radio
button follow action
        self.R2CEntry.grid(row = 2, column = 1)

        #Calculation output
        self.CalcOutNote = tk.StringVar()
        self.CalcOutMess = tk.Message(self.Area3,  textvariable  =
self.CalcOutNote,  bg = "#b6bcdb",  relief = tk.SUNKEN, width = 420).grid(row =
0,  column = 0)
        self.CalcGoButtn = tk.Button(self.Area3, text="Go")
        self.CalcGoButtn.bind("<Button-1>",self.DoCalc)   #The number being
passed is the ID for this dialogue
        self.CalcGoButtn.bind("<Return>",self.DoCalc)
        self.CalcGoButtn.grid(row = 0,  column = 1)

        self.CalcCancel = tk.Button(self.Area3, command=self.CloseDown)
        self.CalcCancel["text"]= "Close"
        self.CalcCancel.grid(row = 0, column = 2)



        self.CalcWin.mainloop()  #Keep this going until we quit out.

    def DoCalc(self,event=None):
        CalcID = self.CalcTypeSel.get()
        Vref = 1.25
        Iadj = 50*10**-6

        #Gather data
        if CalcID == 1:
            Volts = self.VoltsAOutVal.get()
```

```
            Res1 = self.R1AVal.get()
            #The following prevents the scrift from failing if junk text is
added to the inputs
            try:
                V = float(Volts)
                R1 = float(Res1)
            except:
                return

            #For R2 we need to iterate
            R2Old = 20  #Iterator startoff values
            R2 = 100
            R2DiffRat = 3

            while R2DiffRat > 0.01:
                R2Old = R2
                R2 = R1*((V-Iadj*R2Old)/Vref - 1)
                R2DiffRat = abs(R2 - R2Old)/R2

            Textreturn = "The Value of R2 is: {:.0f} Ohms".format(R2)
            self.CalcOutNote.set(Textreturn)


        elif CalcID ==2:
            Volts = self.VoltsBOutVal.get()
            Res2 =self.R2BVal.get()
            try:
                V = float(Volts)
                R2 = float(Res2)
            except:
                return

            R1 = R2*Vref/(V-Iadj*R2-Vref)
            Textreturn = "The Value of R1 is: {:.0f} Ohms".format(R1)
            self.CalcOutNote.set(Textreturn)


        else:
            Res1 = self.R1CVal.get()
            Res2 = self.R2CVal.get()

            try:
                R1 = float(Res1)
                R2 = float(Res2)
            except:
                return
            V = Vref*(1+R2/R1)+Iadj*R2
            Textreturn = "The Output Voltage is: {:.2f} V".format(V)
            self.CalcOutNote.set(Textreturn)


    def CloseDown(self):
        self.CalcWin.destroy()


    def UpdateRadioPoint1(self, event = None):
         self.CalcTypeSel.set(1)  #Put the R1 V area

    def UpdateRadioPoint2(self, event = None):
         self.CalcTypeSel.set(2)  #Put the R2 V area
```

```
    def UpdateRadioPoint3(self, event = None):
         self.CalcTypeSel.set(3)  #Put the R1 R2 area



LMCalc = Calculator()    #Open the Tkinter window here.
```

Deployment

So that's the LM317 calculator coding. To deploy the code I used ***cx_Freeze*** for the Linux version and ***py2exe*** for the Windows version.

Linux using cx_Freeze

I spent a long time fighting with *cx_Freeze*. It would compile nicely, but the compiled executable would always come back with a missing "`__setup__`" module error. Huh? Not one of mine. This was even happening on the simple demonstration examples included with *cx_Freeze*. Anyway, in the end I found that this was a problem with the Linux *cx_Freeze* and the development community appeared to have solved it about 11 days beforehand. So following the advice of ***klensy*** on the cx_Freeze issues discussion I removed my existing version of *cx_Freeze* and pulled a patched version from Github.

```
pip uninstall cx_Freeze
pip install git+https://github.com/anthony-tuininga/cx_Freeze.git@v5.x
```

After copying the resources folder into the built binary folder, I tried the freshly cx_Freezed (*cx_Frozen*?) output and it worked perfectly.

You can find the standalone executable as a zipped up file here. LM317Calcv3-Linux.zip

The script used to create this was:

```
# cxFreeze_setupLM317Calcv3p2.py
# A script to guide cxFreeze to build an executable
# on a linux 32 bit installation.

# The source file is a calculator incorporating a diagram
# a Tkinter interface

#Use the following command to run it.
# python cxFreeze_setupLM317Calcv3p2.py build

#Once it is done copy the resources folder into the newly created output
directory.


import cx_Freeze
from cx_Freeze import setup, Executable
import os
import sys

base = None
if sys.platform == 'win32':
    base = 'Win32GUI'

Initialdir = os.getcwd()

executables = [cx_Freeze.Executable("LM317Calcv3.py", base=base)]
Imagefile = os.path.join(Initialdir,"resources","LM317Circuit.gif")
```

```
cx_Freeze.setup(
    name="LM317 Calculator",
    options={"build_exe": {"packages":["Tkinter","os"],
                           "include_files":[Imagefile]}},
    executables = executables

    )
```

To build it, I used the following command in the terminal while in the directory where the files were.

```
python cxFreeze_setupLM317Calcv3p2.py build
```

where the " `cxFreeze_setupLM317Calcv3p2.py`" is just the name of the setup file above.

Windows using py2exe

Deploying on windows was relatively straightforward (Once my old *Windows XP* machine had started and I had installed ***Python 2.7*** and ***py2exe***. Luckily ***py2exe*** is distributed as a windows installer. Had it required Pip, I would have been sunk because that machine has no internet connection. So this was made on a Windows XP 32-bit machine. How it performs on a Windows 10 or other Microsoft abominations I have no idea.

Like *cx_Freeze* I needed to write a setup script. I pretty much just pinched the example one from the *py2exe* library. *Py2exe* is smart enough to figure everything else out.

```
#setupp2e.py
from distutils.core import setup
import py2exe

#Run this with the command "python setupp2e.py py2exe"
#Copy the resources directory across to the output folder once it has been
created

executables = "LM317Calcv3.py"

setup(
    version = "3",
    description = "LM317 Calculator",
    name = "LM317 Calculator",

    # targets to build
    console = [executables],
    )
```

I navigated to the working directory in the command line thing and used the following command to start the process.

```
python setupp2e.py py2exe
```

where `setupp2e.py` was just the name of my setup script.

So that churned it around for a bit and produced a new directly called `dist`. I copied the resources folder with the image into it and ran it. It worked fine. It also worked under *Wine* although was pretty slow to start when compared to the Linux native version.

So here is the Windows executable. LM317Calcv3-Windows.zip

Just unzip this where you like and look for the ***LM317Calcv3.exe*** file.