20. Text Files & Energy in a Battery

Introduction:

The Pi-Pico has 264k on-chip SRAM. This allows you to

- Create a text file which controls the Pi-Pico's operation
- Write to a text file, saving your data

This lecture covers

- How to open and close text files
- Reading from text files
- String commands and parsing strings
- Reading a text file to play a tune
- Writing to text files, and
- Measuring the energy in a rechargeable battery

Rechargeable Batteries from Amazon: How much energy to they *really* have?

Opening & Closing Text Files

Opening a file: The general syntax to open a file in Python is:

```
file = open("File_Name", "Access_Mode")
```

Access Mode can take on several values:

Access mode	Function
"r"	Default mode Open a text file for reading. Pointer is placed at the start of the file. Results in an error if the file does not exist
"a"	Open a text file for appending. Pointer is placed at the end of the file. Creates a new file if it does not exist
"w"	Open a text file for write-only. Create a new file if one does not already exist. Clear out the contents of the existing file.
"x"	Create a new file Returns an error if the file already exists

The file can also be specified as a text file or a binary file (i.e. an image)

File Type	Function
"t"	Text file (default)
"b"	Binary file (image)

Closing a file: Once finished, files should always be closed

```
file.close()
```

Reading From a Text File

Text files are read as strings - regardless of whether the contents are actually numbers or text. When you read a text file, you can read some or all of the file

Command	Result
`Data = f.read(5)`	Read the next five characters into text string *Data*
`Data = f.readline()`	Read the next line into *Data*
`Data = f.readlines()`	Read the entire file into an array *Data*. Each line is stored in a different entry: Data[0], Data[1], etc
`Data = f.read()`	Read the entire file into a text string, *Data* Carriage returns and line feeds show up as /n/r

For example, assume a text file contains the following information:

readme.txt

```
Three rings for the Elven-kings under the sky
Seven for the Dwarf-lords in their halls of stone,
Nine for the Mortal Men doomed to die
```

This file can be read in its entirety

Program Window

```
f = open("readme.txt", "rt")
Data = f.read()
print(Data)
f.close
```

Shell

```
Three rings for the Elven-kings under the sky
Seven for the Dwarf-lords in their halls of stone,
Nine for the Mortal Men doomed to die

>>> Data
'Three rings for the Elven-kings under the sky\r\nSeven for
the Dward-lords in their halls of stone\r\nNone for the
Mortal Men doomed to die\r\n'
```

Note that

- The file is stored as a text string
- \r is a carriage return
- \n is a newline command

You can also read this file line by line

Program Window

```
f = open("readme.txt", "rt")
Data = f.readlines()
n = len(Data)
for i in range(0,n):
    print(i, Data[i])
f.close
```

Shell

```
0 Three rings for the Elven-kings under the sky

1 Seven for the Dwarf-lords in their halls of stone,

2 Nine for the Mortal Men doomed to die

>>> Data[0]
'Three rings for the Elven-kings under the sky\r\n'

>>> Data[1]
'Seven for the Dwarf-lords in their halls of stone,\r\n'

>>> Data[2]
'Nine for the Mortal Men doomed to die'
```

String Commands and Parsing Strings

One way to pass data to a Python program is through a text file. For example, the file could contain a list of numbers to graph or a list of music notes to play a song. Typically, the data in text files is separated by commas, spaces, or tabs. When reading the text file, the fields can thus be identified by looking for these characters.

To illustrate this, take the text file for the area of the Arctic covered in sea ice according to the National Sea and Ice Data Center (NSIDC):

```
# Arctic Sea Ice Extent
# https://nsdic.org/arcticseaicenews/sea-ice-tools/
1979  7.051  16.342
1980  7.667  16.041
1981  7.138  15.632
:
```

When this file is read into an array using the *readlines()* command, each line is read as a string. To pull out the separate columns, a subroutine *Parse()* is called. This subroutine

- Strips out any spaces at the beginning and end of the string (*strip()*)
- Replaces tabs and commas with spaces (*replace()*), and
- Removes multiple spaces (for-loop replacing double spaces with single spaces)

Once done, the number of fields (columns) can be found by counting the resulting number of spaces. The data for each columns is then found by

- Truncating the string from the start to the next space and
- Converting the truncated string to a floating point number

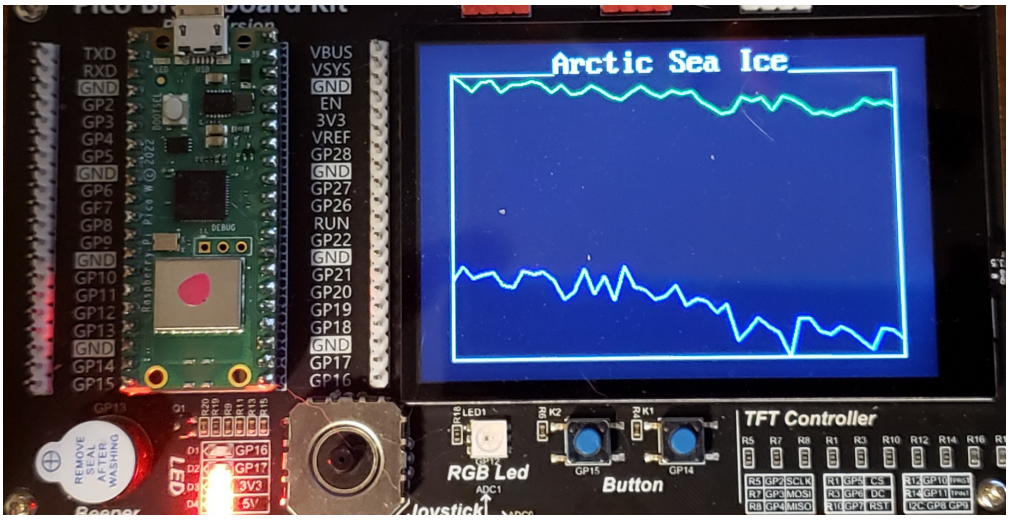
```
def Parse(X):
    X = X.strip()
    X = X.replace(',',' ')
    X = X.replace('\t',' ')
    for i in range(0,10):
       X = X.replace('  ',' ')
    ncol = X.count(' ') + 1
    Y = [0]*ncol

    for i in range(0,ncol):
        m = X.find(' ')
        if(m>0):
            Y[i] = float(X[0:m])
        else:
            Y[i] = float(X)
        X = X[(m+1):]
    return(Y)

Data = '1979  7.051  16.342'
Y = Parse(Data)
print(Y)
```

```
[1979.0, 7.051, 16.342]
```

To read a data file with three columns and plot the data on the TFT display,

- Each row is read into a vector of three numbers
- Each vector is appended to the previous results, producing an Nx3 matrix (where N is the number or data entries),
- Once complete the matrix is transposed, producing an 3xN matrix

Each column can then be pulled out and plotted using the *Plot()* routine from the LCD library.



Reading data from a text file (Arctic Sea Ice) and plotting the data on the TFT Display

```
import LCD
import matrix

def Parse(X):
    :

f = open("SeaIce.txt", "rt")
Data = f.readlines()
f.close()

n = len(Data)
Y = []
for i in range(0,n):
    Y.append(Parse(Data[i]))

Y = matrix.transpose(Y)

t = Y[0]
Icemin = Y[1]
Icemax = Y[2]

Navy = LCD.RGB(0,0,5)
White = LCD.RGB(100,100,100)
LCD.Init()
LCD.Clear(Navy)
LCD.Plot(Y[0],[Y[1],Y[2]])
LCD.Title('Arctic Sea Ice', White, Navy)
```

Routine for reading a text file and plotting the data on the TFT display

Example: Mario Brothers Tune

As a second example of that you can do with text fles, let's play the first four bars of the Mario Brothers theme. First, create a text file on the Pi-Pico which contains

- The note,
- The octave, and
- The duration of the note in 16th's of a beat:

file Mario_Bros.txt

```
E4,2
E4,2
E4,4
0,2
C4,2
E4,4
G4,4
0,4
G3,4
0,4
```

Next, write a routine which

- Strips out the note and duration, and
- Returns these as a string (note) and duration (integer)

Parse subroutine

```
def Parse(X):
    X = X.strip()
    X = X.replace(',',' ')
    X = X.replace('\t',' ')
    for i in range(0,10):
       X = X.replace('  ',' ')

    m = X.find(' ')
    Note = X[0:m]
    Dur = int(X[(m+1):])
    return([Note,Dur])

f = open("Mario_Bros.txt", "rt")
Data = f.readlines()
f.close()

n = len(Data)
Y = []
for i in range(0,n):
    Y.append(Parse(Data[i]))

print(Y)
```

Shell

```
[['E4',2], ['E4',2], ['E4',4], ['0',2], ['C4',2], ['E4',4],
['G4',4], ['0',4], ['G3',4], ['0',4]]
```

Nest, define a subroutine which returns the frequency of each note. From Wikipedia, the frequency of the zeroth ocrive for each music note is:

Note	C0	C#0	D0	E0	F0	F#0	G0	G#0	A0	A#0	B0
Hz	16.35	17.32	18.35	20.6	21.83	23.12	24.5	25.96	27.5	29.14	30.87

To convert to a different octave, multiply the note by 2**Octave

Freq subroutine

```
def Freq(a):
    n = len(a)
    Note = a[0:n-1]
    Octave = a[n-1]
    Hz = 0
    if(Note == 'C'):
        Hz = 16.35
    elif(Note == 'C#'):
        Hz = 17.32
    elif(Note == 'D'):
        Hz = 18.35
    elif(Note == 'E'):
        Hz = 20.60
    elif(Note == 'F'):
        Hz = 21.83
    elif(Note == 'F#'):
        Hz = 23.12
    elif(Note == 'G'):
        Hz = 24.50
    elif(Note == 'G#'):
        Hz = 25.96
    elif(Note == 'A'):
        Hz = 27.50
    elif(Note == 'A#'):
        Hz = 29.14
    elif(Note == 'B'):
        Hz = 30.87
    if(Hz > 0):
        Hz = Hz * (2 ** int(Octave))
    return(Hz)

print('A3 = ', Freq('A3'), ' Hz')
print('D4 = ', Freq('D4'), ' Hz')
print('G#5 = ', Freq('G#5'), ' Hz')
```

shell

```
A3 = 220.0 Hz
D4 = 293.6 Hz
G#5 = 830.72 Hz
```

Finally, reuse the *Play(Hz, Dur)* subroutine from before to play

- A note at frequency *Hz*
- For a duration of *Dur* / 16 seconds

With these three routines, you can read and play a text file. (This is more impressive the a video)

```
from time import sleep_ms
from machine import Pin, PWM

Spkr = PWM(Pin(18))
Spkr.freq(100)
Spkr.duty_u16(0)

def Parse(X):
   :

def Freq(Y):
   :

def Play(Hz, Eighths):
    if(Hz > 0):
        Spkr.freq(round(Hz))
        Spkr.duty_u16(32768)
    else:
        Spkr.duty_u16(0)
    sleep_ms(75 * Eighths - 50)
    Spkr.duty_u16(0)
    sleep_ms(50)

f = open("Mario_Bros.txt", "rt")
Data = f.readlines()
f.close()

n = len(Data)
Y = []
for i in range(0,n):
    Y.append(Parse(Data[i]))

for i in range(0,n):
    Hz = Freq(Y[i])
    Dur = Y[i][1]
    print(i, Hz, Dur)
    Play(Hz, Dur)
```

Writing to a Text File

In addition to reading from a text file, the Pi-Pico can create and write to a text file. This is useful if you want to save your data for later use with Matlab or other routines. For example, the following section will look at recording into a text file the voltage of a rechargeable battery as it discharges across a 10 Ohm resistor.

Before trying to measure the energy in a rechargeable battery, let's first look at how to save data to your Pi-Pico board. The basic program to open a text file, write to it, the close the text file is as follows:

```
file1 = open("readme.txt", "w")
print('File Opened')

for i in range(0,6):
    file1.write(str(i))
    file1.write("x")
    file1.write(str(i))
    file1.write("\n")

file1.close()
print('File Closed')
```

After running this program, file readme.txt contains the following:

file readme.txt

```
0x0
1x1
2x2
3x3
4x4
5x5
```

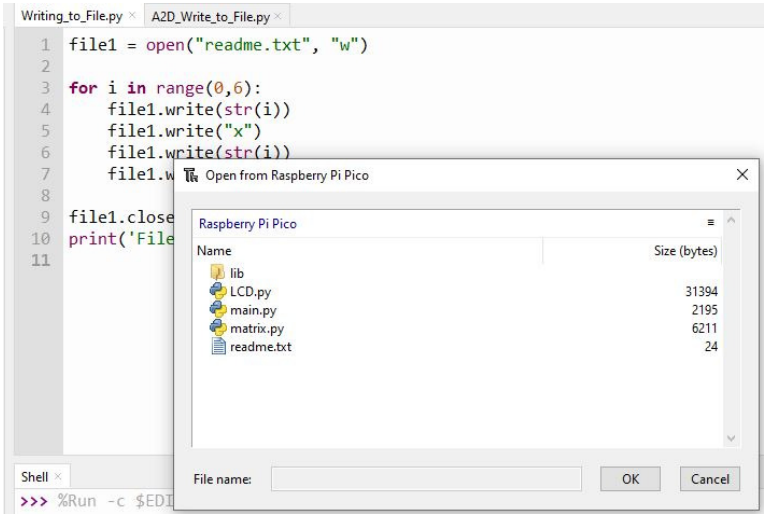
The way this files works is:

- file1 is opened as a write-only file
- When writing to a text file, only strings are written. All data needs to be converted to a string to write to the file.
- Subsequent writes will be appended to the current line.
- To start a new line, you need to write \n

From Thonny, you can open the file *readme.txt* to see the contents. The contents can then be copied to Notepad, Matlab, or wherever you want to place the results.

Note: To open readme.txt, from Thonny,

- Click on File Open
- Select Raspberry Pi Pico
- Select readme.txt



After writing to a file, the file on the Pi-Piico board can be opened using Thonny

Once you can write to a file, you can also write voltages as read by the A/D as well. In the following code,

- The three A/D inputs are read every 100ms
- These readings are displayed on the console,
- They are saved to a file *readme.txt*
- Then after ten reads, the file is closed

```
import machine
import time

a2d0 = machine.ADC(0)
a2d1 = machine.ADC(1)
a2d2 = machine.ADC(2)

kV = 3.3 / 65535

file1 = open("readme.txt", "w")

for i in range(0,10):

# A/D reads take 100us
    V0 = a2d0.read_u16() * kV
    V1 = a2d1.read_u16() * kV
    V2 = a2d2.read_u16() * kV

# write to file takes 1770us
    file1.write(str('{: 4.0f}'.format(i) + " ")
    file1.write(str('{: 7.3f}'.format(V0) + " ")
    file1.write(str('{: 7.3f}'.format(V1) + " ")
    file1.write(str('{: 7.3f}'.format(V2) + " ")

    time.sleep(0.1)

file1.close()
```

The readme.txt file then contains the following:

```
0   1.3925   1.4231   0.0556
1   1.3893   1.4215   0.0548
2   1.3869   1.4231   0.0556
3   1.3901   1.4231   0.0548
:
```

Using the *ticks_us()* function from time, the execution time of this program can be found:

- 100us: A/D reads
- 1.77ms: File write

This implies that 500Hz (2ms) is about the fastest you can read the A/D inputs and write to a text files.

Energy in a Battery: Hardware

Next, to measure the energy in a rechargeable battery,

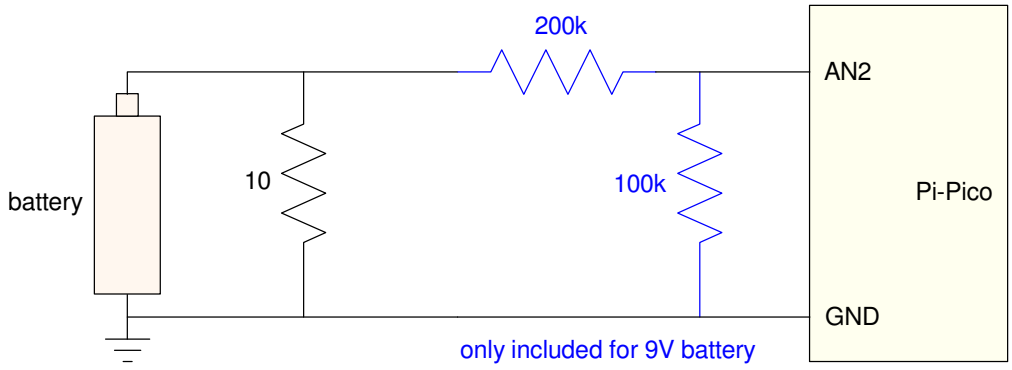
- Connect the battery to a 10 Ohm resistor, and
- Measure the voltage with a Pi-Pico

If the battery is a 9V battery, also add a divide-by-three voltage divider to get the voltage into the range of 0V - 3.3V.

This results in the current and power being dissipated being:

$$I = \frac{V}{R}$$

$$P = \frac{V^2}{R}$$



Hardware for measuring the energy in a rechargeable battery
Note: the resistor needs to be sized for 225mW (1.5V) or 8.1W (9V)

With this, the expected time to discharge a battery can be calculated. The manufacturer claims for these batteries are:

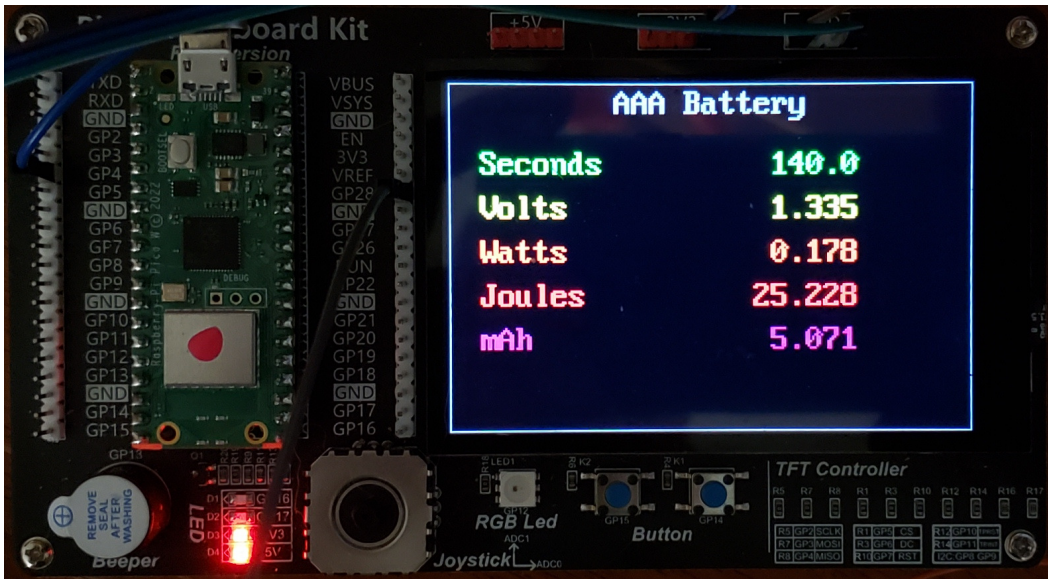
Battery Type	Voltage	mAh	mA @ R	Hours
AAA	1.5V	750	150	5.00h
AA	1.5V	2,400	150	16.0h
9V	9.0V	600	90	6.67h

So, this experiment might take some time... This is one reason to write the data to a file: the program can be run over night and the results can then be obtained from the text file stored on the Pi-Pico.

Energy in a Battery: Software

1000+ data points should be plenty, so set the sampling rate to 60 seconds (one data point is collected every minute). Once the voltage drops below 0.5V, the file is written and then the program stops. (see Bison Academy for a complete program listing). Since it takes a while for the battery to discharge, a running result is displayed every second including

- Time in seconds
- Current voltage of the battery as it discharges across the 10 Ohm load
- The power being dissipated across the 10 Ohm load
- The total Joules the battery has output so far, and
- The total current in mAh the battery has output:



Running display for the battery test.
Once the voltage drops below 0.5V, the data is written to a file and the program ends.

An abbreviated routine is as follows. The full program is available on Bison Academy under lecture #20.

```
from machine import Pin, ADC, Timer

a2d2 = ADC(0)
kV = 3.3 / 65535
flag = 1
T = 60

def tick(timer):
    global flag
    flag = 1

Time = Timer()
Time.init(freq = 1/T, mode=Timer.PERIODIC, callback = tick)

Beeper = Pin(13,Pin.OUT)

file1 = open("Battery_Test.txt", "w")
Volts = 9

time = mA = mAh = Watts = Joules = 0

while(Volts > 0.5):
    while(flag == 0):
        pass
    flag = 0
    Volts = (a2d2.read_u16() * kV)
    mA = Volts / 10 * 1000
    mAh += mA * T / 3600
    Watts = ( Volts ** 2 ) / 10
    Joules += Watts * T

    file1.write(str('{: 7.0f}'.format(time)))
    file1.write(str('{: 7.4f}'.format(Volts)))
    file1.write(str('{: 7.4f}'.format(mAh)))
    file1.write(str('{: 7.4f}'.format(Joules)))
    file1.write("\n")
    print(Volts)
    time += T

file1.close()

print('Done')
```

Rechargeable AAA Battery

- Rated Energy: 750mAh
- Work in progress

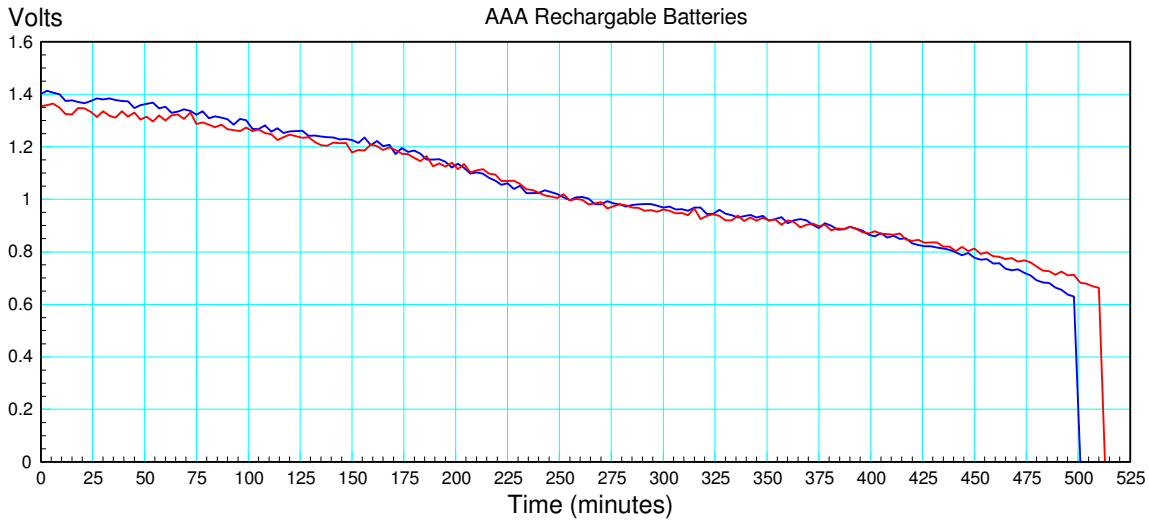
To illustrate writing data to a text file, let's determine

- The voltage output,
- The total mAh output, and
- The total energy output

of a rechargeable AAA battery as it discharges across a 10 Ohm resistor. The procedure of the experiment was to:

- First, charge all three AAA batteries until they were fully charged (green LED came on).
- Each battery was then placed into a battery holder, which had a 10 Ohm resistor shorting the leads
- A Python program was then run, measuring the voltage every second for computations, and saving the data to a file every 60 seconds
- The program ends (data no longer collected) once the voltage drops below 0.5V

The results for a AAA rechargeable battery with a rating of 750mAh was as follows:



Voltage of rechargeable AAA batteries discharging across a 10 Ohm resistor

Battery	mAh	Joules
1	879.53	3486.54
2	886.80	3444.23
3	880.86	3547.31

Measured energy in three rechargeable AAA batteries

From this data, you can compute

- The 90% confidence interval for the mAh and Joules in any given AAA battery, and
- The probability that a given battery will meet the 750mAh specification.

This requires using a student-t distribution (a topic covered in statistics and embedded systems).

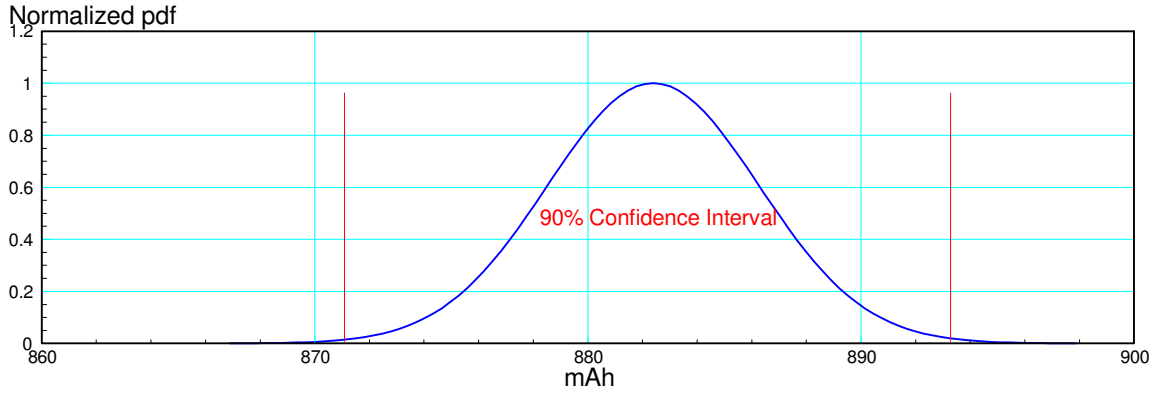
Start with the mean and standard deviation of the data (from Matlab): (mAh are used here, but you could do the same thing with the energy in Joules):

```
>> mAh = [879.531, 886.804, 880.86];
```

The mean and standard deviation for this data is:

```
>> X = mean(mAh)
X =  882.3983

>> S = std(mAh)
S =   3.8729
```

The mean and standard deviation tell you the probability density function (pdf) for this battery. It's going to be a normal distribution (almost everything is) which looks similar to the following



pdf for the energy in a AAA recharge battery based upon measurements

The 90% confidence interval is a two-sided test (you have two tails). Each tail should have an area of 5% for the middle to have an area of 90%. From StatTrek.com, the t-score for

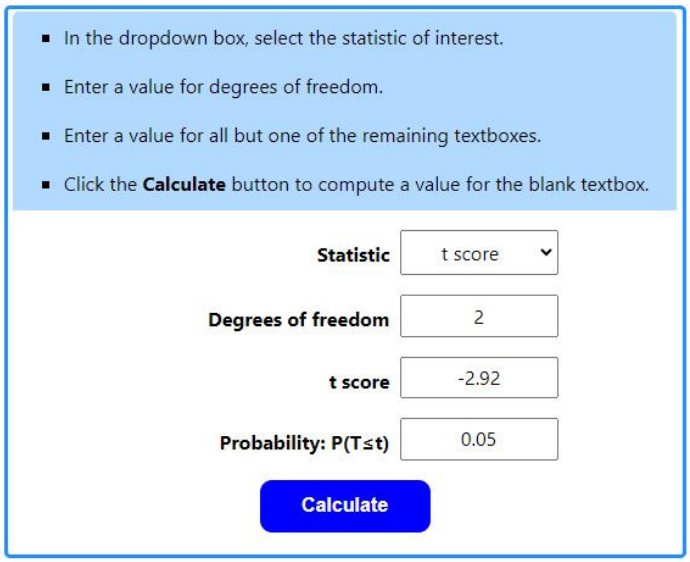
- 5% tails (90% confidence interval) and
- Two degrees of freedom (degrees of freedom is sample size minus one)

is 2.920.

Translation: If you go left and right of the mean by 2.92 standard deviations, you'll capture 90% of the population.

```
>> X + 2.920*S
ans =  893.7071

>> X - 2.920*S
ans =  871.0896
```

Translation: 90% of the AAA batteries should be in the range of (871.09, 893.71) mAh.

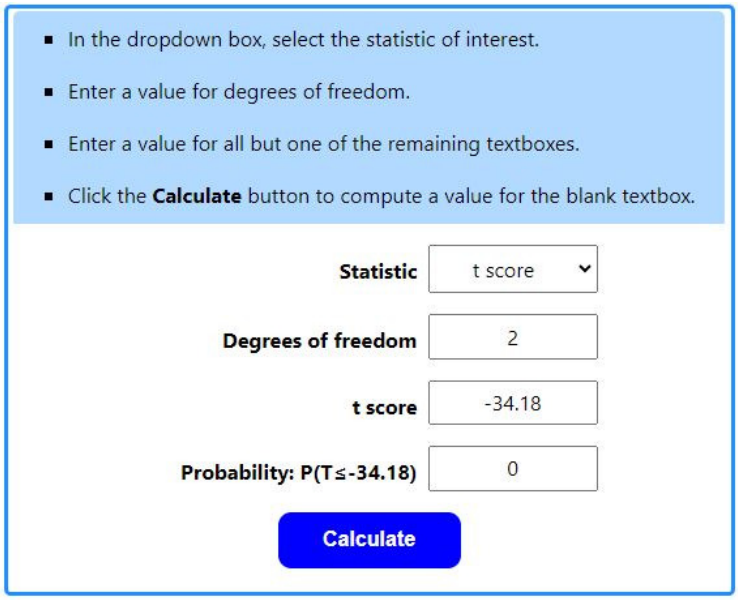


The t-score for 5% tails and a sample size of two (1 dof) is 6.314
from www.StatTrek.com

The probability that a given battery has at least 750mAh is a single-sided t-test. The t-score for 750mA is

```
>> t = (750 - X) / S
t =   -34.1863
```

From StatTrek, this corresponds to a tail of 0% (or less than 0.005%, rounded to 0%) - meaning at least 99.95% of AAA rechargeable batteries should meet specs. That's kind of surprising - most manufacturers are generous in their claims. This claim appears to be actually true (!).

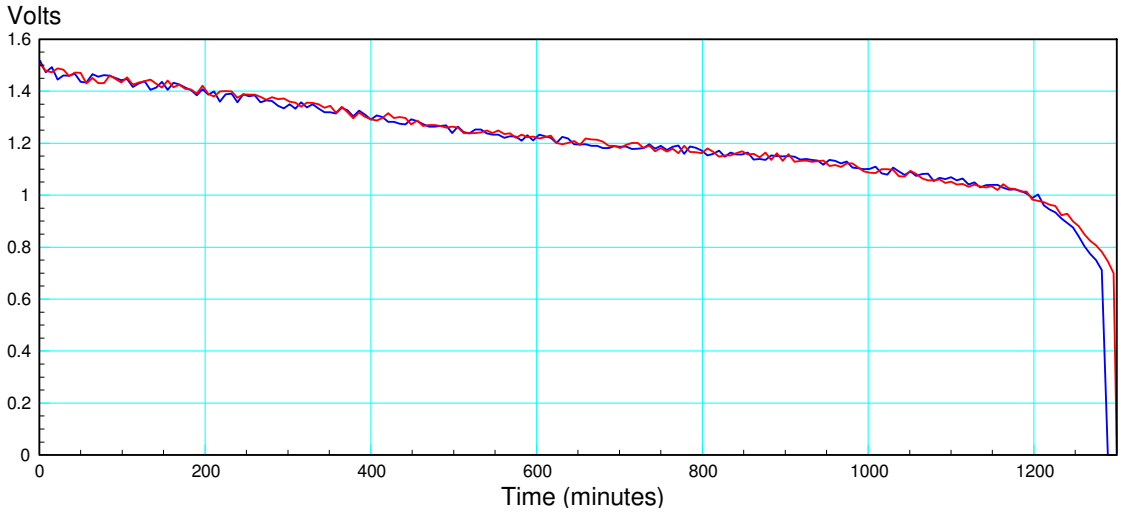


From this data, there is less than a 0.005% chance (rounded to 0%) any given AAA battery will have less than 750mAh

More data would give better results - but you can still get meaningful results with just a sample size of three.

Rechargeable AA Battery

Next, let's look at rechargeable AA batteries. These are rated at 2400mAh. Proceeding as before, the voltage across the battery as it discharges across a 10 Ohm resistor was measured for two AA batteries. The resulting voltage vs. time is as follows:



Voltage across a rechargeable AA battery, discharging across a 10 Ohm resistor

The resulting mAh and energy in Joules was:

Battery	mAh	Joules
1 (blue)	2,596.1	11,512.723
2 (red)	2,623.5	11,632.354

Following the same procedure as before, the statistics for a AA battery are as follows. Note that with a sample size of two, the t-score for 5% tails increases to 6.314 (vs. 2.92 in the previous case).

```
>> mAh = [2596.1, 2623.5];
>> X = mean(mAh)
X =  2.6098e+003

>> S = std(mAh)
S =   19.3747

>> X + 6.314*S
ans =  2.7321e+003

>> X - 6.314*S
ans =  2.4875e+003

>> t = (2400 - X) / S
t =  -10.8285
```

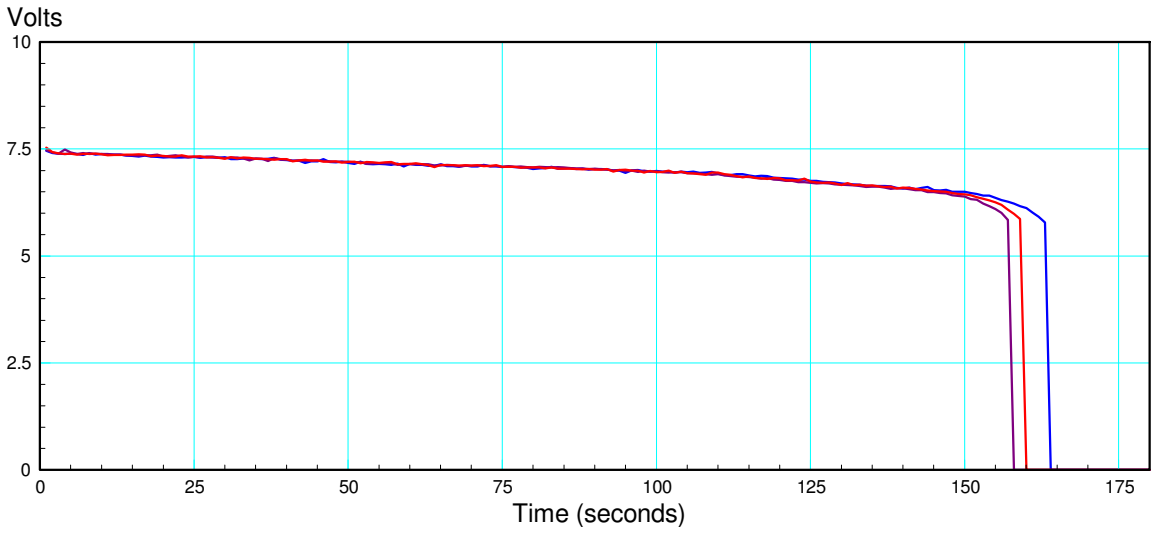
From StatTrek the tail has an area of 2.9%

- The 90% confidence interval for the energy in a rechargeable AA battery is (2,487.5, 2,732.1)mAh
- Only 2.9% of these batteries should have less than rated energy (2400mAh)

Again, a larger sample size would give better results.

Rechargeable 9V Battery

Finally, let's determine the energy in a rechargeable 9V battery which is rated at 600mAh. Discharging across a 47 Ohm resistor results in the following voltage vs. time:



Voltages of 9V rechargeable battery driving a 47 Ohm load

The resulting energy in three batteries were:

Battery	mAh	Joules
1	402.354	10,128.095
2	388.744	9,809.798
3	393.570	9,924.400

This provides

- A 90% confidence interval of (374.74, 415.04) mAh
- A 99.9995% chance that a given battery has less than the rated 600mAh of energy.

In this case, it looks like the manufacturer was a little generous in its claims.

Summary

Python is able to read from and write to text files fairly easily. With this, you can

- Plot data you recorded earlier,
- Play different tunes by saving data to a given text files, and
- Save data when you collect it for later analysis.

References

Pi-Pico and MicroPython

- https://github.com/geeekpi/pico_breakboard_kit
- https://micropython.org/download/RPI_PICO/
- https://learn.pimoroni.com/article/getting-started-with-pico
- https://www.w3schools.com/python/default.asp
- https://docs.micropython.org/en/latest/pyboard/tutorial/index.html
- https://docs.micropython.org/en/latest/library/index.html
- https://www.fredscave.com/02-about.html

Pi-Pico Breadboard Kit

- https://wiki.52pi.com/index.php?title=EP-0172

Other

- https://docs.sunfounder.com/projects/sensorkit-v2-pi/en/latest/
- https://electrocredible.com/raspberry-pi-pico-external-interrupts-button-micropython/
- https://peppe8o.com/adding-external-modules-to-micropython-with-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-esp32-esp8266-micropython/