



Reading From the Teensy/Arduino

READING THE TEENSY (OR ARDUINO) USING PYTHON

It was an improvement to run the Teensy using the Serial Monitor, but you need more to interface the experiment. You want a program that runs *on the laptop* that controls the Teensy and saves the data in a file. Python is the language you will use. I will send you the python code in the **CLASSWORK** section of the CoCalc project.

Installing python (if needed)

You will usually run the python program from the command line, so you have to have a python package installed on your computer. (python is shipped with Mac's but it is not usable for this purpose; it is part of the OS X system, so they do not want you to add modules.) I recommend the anaconda.com package. It is open source and free and does not have any advertising or extra programs with it.

- Go to the download link [here](#), download and install **Python 2.7** and the **64-bit** package.
- Start a command or terminal.
 - On windows press  (Windows key), type **command**, then click on the command or Anaconda command choice.
 - On a Mac press +<space>, (command key and the space bar), type **terminal** and double click on **terminal**.
 - On Linux, click start, then type **terminal** or type <control><alt>T.
- Install the **pyserial** module by typing **conda install pyserial** in the terminal.

Program Design

Whenever you write a program, you should think about the design of the program before you get seriously started. Here are the main design goals:

- Communicate with the Teensy. You have to:
 - find the USB port,
 - *write to*, that is send, commands to the Teensy, and
 - *read from*, that is listen to, the Teensy.
- Send setup commands like **RSENS 10000**.
- Send query commands and print the responses, like:

```
NAVG?
10000
```

- Open a data file.
- When the **data?** query is sent, save the data to a **.csv** file.

The ReadSCPI_18c.py Program

(Note: the program version in my naming system is **18c**. That means it is the third version I made in 2018.)

I will go over this code in detail.

The code below starts with a **#!** command. In some operating systems (Mac and Linux) you can change the file to executable and then run it with adding **python** in front of the command.

The next lines are

```
intro = \
```

.....

These lines are a long string I use to identify some basic information about this program, its name author, data, a description. Then I print this information with the line `print intro`. This is one way I use to make the program easy to read.

```
#!/usr/bin/env python
intro = \
.....
ReadSCPI_18b.py
ProfHuster
2018-02-05

This program collects data from an Arduino (or Arduino-clone) that uses
a SCPI-style command structure and saves it to a file.

Versions:
18c - Various mod's including saving CONF? query to the data file
18b - Flushes stdout
.....
print intro
```

The next two lines import modules (the python equivalent of libraries) into the code. The second line imports a python file `serialPorts.py` I wrote to help find the Arduino or Teensy.

```
import serial, io, re, sys, time
from serialPorts import serial_ports
```

Next I define some constants for the program.

Naming Data Files

From long experience, I found it best if your program automatically names data files for you. I define the string that will name the file with *SCPI*, then the *year-month-day-hourminute* and finally the *csv* suffix. This way, every time you run the program a new and unique data file is created. The `EOL` variable is the character that marks the end of a line of text. The `TIMEOUT` set how long the program will listen to the Arduino before it decides no more data is coming. Finally `DO_PRINT` helps debug the program. It is set to `False`, so less information is printed out.

```
# The next line defines the file name format
# Note that the format automatically include the
# year-month-date-hour-minute. This is incredibly useful to help
# keep data files orgnized. The alphabetical order of the files
# is also the chrononlogical order.
FILE_NAME_FORMAT = r"SCPI-%04d-%02d-%02d-%02d%02d.csv"
EOL = '\n'
TIMEOUT = 1.0
DO_PRINT = True # print everything out as you go
DO_PRINT = False
```

The next few lines of code gets the current time and uses the format string above to create the data file name. Then the program opens the file for writing with the `open` function call.

```
# make file name and open output data file
t = time.localtime()
fileName = FILE_NAME_FORMAT % (t.tm_year, t.tm_mon, t.tm_mday, \
    t.tm_hour, t.tm_min)
print 'Opening file "%s"' % (fileName)
fpData = open(fileName, 'w')
```

These lines allow you to write comments about the program in the data file. You should include information like which LED, resistor, or device you are taking data on.

```
# Let user put in comments
while True:
    comment = raw_input("Enter a comment (return to end): ")
    if len(comment.strip()) == 0:
        break
    fpData.write("# " + comment.strip()+'\n')
fpData.flush()
```

This code uses my python module to find and list USB devices. Once you identify your Arduino, enter that number to open a serial connection with it. The `serial.Serial` function opens the connection, set the speed as **115299** bits per second, and sets the timeout. The last section of code here reads data from the Arduino with `ser.readline()`. If it reads a line of zero length, that means the serial connection timed out.

```
# First get the name of and open the serial port
ports = serial_ports()
print "The available ports are:"
for (i, port) in enumerate(ports):
    print "%d, %s" % (i, port)
iPort = int(raw_input("Enter serial port number: "))
#select serial port, baudrate (default = 9600), timeout (default = 1)
ser = serial.Serial(ports[iPort], 115200, timeout=TIMEOUT)

# wait for Arduino to boot and print any output
time.sleep(TIMEOUT)

# This code is used later also
# Read a line form Arduino
line = ser.readline()
# While line has some length. If length is zero, there is no more to read
while len(line) > 0:
    if DO_PRINT: print ">%s<" % (line.strip())
    line = ser.readline()
```

The code in the section below is the main command loop. First it prints out a prompt, then the loop starts. The code `while True` will loop forever unless the while loop is broken out of. Next is some code to see if the first character of the command is a `q`. If it is, the while loop is broken out of and the program will end. The code `command[0].lower()` makes sure the first character of the string `command` is converted to a lower case for comparison to the character `q`. So even if the user enters `QUIT`, the program still ends. Finally the end of line character is added to the command before it is sent to the Arduino.

```
print "Enter command line (help? for help)"
print "Enter 'q' to quit"
# Loop until a break statement
while True:
    # Get line to write
    command = raw_input(": ")
    if DO_PRINT: print "Command:%s:" % (command.strip())
    # Check if a q for quit was entered
    if len(command)>0 and command[0].lower() == 'q':
        break
    # Add a newline before sending command to Arduino
    command += EOL
```

Here the code see if a `data?` Command was entered. The module `re` does searches and replaces for strings and text. Here the `re.search` method is looking for the data query string in the command. The flag tells the search to ignore the case, so upper and lower cases all match. I want the Teensy configuration to be

printed in the data file, so I send the `conf?` command and write the output in the data file. However I don't want the configuration data to get confused with the numerical data, so I use another `re` command to substitute the `#` character at the beginning of the line. Finally I send the command.

```
# If it is a data command, send config query and write response to the data
file
if re.search(r'data?', command, flags=re.I):
    ser.write('conf?\n')
    line = ser.readline()
    while len(line) > 0:
        if DO_PRINT: print ">%s<" % (line.strip())
        sys.stdout.flush()
        re.sub(r'^', r'# ', line)
        line = re.sub(r'$', r'', re.sub(r'<', '', re.sub(r'>', r'',
re.sub(r'^', r'# ', line))))
        fpData.write(line)
        fpData.flush()
        line = ser.readline()
    ser.write(command)
```

This is the loop for the data query. It reads lines from the Teensy and writes them to the data file. When it reads a line of length zero, the data query loop ends.

```
# look for the data command. If found, save to the file
if re.search(r'data?', command, flags=re.I):
    # Read and print first line
    line = ser.readline()
    if DO_PRINT: print ">%s<" % (line.strip())
    # Now save the data output
    line = ser.readline()
    while len(line) > 0:
        if DO_PRINT: print ">%s<" % (line.strip())
        sys.stdout.flush()
        fpData.write(line.strip()+'\n')
        fpData.flush()
        line = ser.readline()
    if not DO_PRINT: print "Done"
else:
    # skip one line
    line = ser.readline()
    for line in ser.readlines():
        lineStrip = line.strip()
        print ">%s<" % (lineStrip)
ser.close()
fpData.close()
```

I know there are a lot of details in the code, but please take some time to look it over.