Communicating with Serial Devices from Python

Why would you need to communicate with a serial device?

- they provide a connection to some external device
  - output/control
  - input/sensors

Characteristics (very crudely) of such devices

- relatively slow data rates
- sequential communications
- no guarantee of error checking or correction
  - that can exist, or you may have to design it in

Case study: How to talk to an Arduino that is used to control the motor that opens and closes the roof of the shed adjacent to the 31inch dome.

Serial Messages

Things to be aware of:

- Serial messages are device dependent, often have some common characteristics:
  - typically character or line driven
  - messages USUALLY end with \r, \n or \r\n (and this is how you'll reference those in Python)
  - usually have a consistent structure (but you'll need to read the device manual, or devise it yourself)
    - GPS Set Timing Mode (X=0,1 or 3): #07,X\r\n
  - Computer serial interfaces usually support

Decision(s):

- command/response
  - e.g. send OPEN command, receive ACK
- listen to constant broadcast
  - device sends status messages at intervals
    - e.g. GPS units constantly broadcast

Arduino use patterns

- Often setup to run autonomously
  - I/O is through the hardware interfaces (DIO pins, AIO pins, PWM etc)
- BUT they do support a serial interface for use beyond the initial programming of the Arduino
  - The simplest access is provided under the Arduino programming environment (useful for development & testing)
- When programming the Arduino, if you plan to make use of Serial communications, you will need the Serial library.

Arduino Serial Initialization

```
//---------------------------------------------------------------------
// One time setup operations
//  Set the output and input DIO lines
//  Set up serial communications
void setup() {
  pinMode(openOPin,  OUTPUT);  // init digital pin for OPEN for output
  pinMode(closeOPin, OUTPUT);  // init digital pin for CLOSE for output

  pinMode(openIPin,  INPUT);   // init digital pin for OPENED limit input
  pinMode(closeIPin, INPUT);   // init digital pin for CLOSED limit input
// or maybe use INPUT_PULLUP if inverted input

  Serial.begin(9600);     // opens serial port, sets data rate to 9600 bps
  while (!Serial) {
    ;  // wait for the serial port to connect
  }
  Serial.println ("Roof Control - Serial Port Connected");
  helppg();
  Serial.print (" > ");
}
```

The Computer Side of the Serial Link

1) OPEN - Establish a connection to a serial device

- figure out the serial device port - this is OS & device dependent
  - Windows - COM1, COM2 etc
  - MacOSX - /dev/cu.XXXX - e.g. /dev/cu.usbmodem1234, etc
  - Linux - /dev/ttyUSB0, /dev/ttyASM0, etc
- The PySerial package provides tools to help
  - The serial.tools.list_ports subpackage:
    - import serial.tools.list_ports as stl
      j = stl.comports()
- set the baud rate, data bits, parity, stopbits (e.g. 9600, 8, none, 1)
- Open the device:
  - self.ars_io = serial.Serial(port=port, baudrate=baud,
    bytesize=dbits, parity=parity,
    stopbits=stopbits)

Actual Communications

2) RECEIVER
- needs to listen on the port for incoming data
- when/how it listens depends on the device characteristics

2) SENDER
- construct and send message. If command/response, receiver will need to expect a response.

3) UI (cli, tk, PyQt, etc)

4) Logging options

5) CLOSE

Python GIL
Global Interpreter Lock

The Python Global Interpreter Lock (GIL):

- The CPython implementation (which is the most common) implements this.
- Effectively means that only ONE thread can execute python bytecode at a time.
- Important because your program may want to do multiple things at once (listen to the device, listen to you through the UI, display status, etc).
- For slow devices like these, with UIs, in practice the GIL is usually not an issue.
- Can construct several threads (each an operational path), and the interpreter will effectively time-slice, switching between them during the waits within each.
- If you have CPU intensive processes, this becomes a problem because you may have a process that almost never waits (so all the other will).

Overall Structure

Primary Classes, methods etc, Functions

- provides direct control from the standard Python interpreter

Overlaid with a GUI that imports that the classes and functions