

As you arrive:

1. Start up your computer and plug it in.
2. **Log into Angel** and go to CSSE 120.
Do the **Attendance Widget** – the PIN is on the board.
3. Go to the **Course Schedule** web page.
Open the **Slides** for today if you wish.
4. Checkout today's project:

Today:

Parameters and Robots

Session03_ParametersAndRobots

Functions – Parameters and Actual Arguments

Robots

- ❖ The Create robot – hardware
- ❖ The Create robot – software

Checkout today's project:

Session03_ParametersAndRobots

*Are you in the **Pydev** perspective? If not:*

Window ~ Open Perspective ~ Other then **Pydev**

Messed up views? If so:

Window ~ Reset Perspective

***Troubles getting
today's project? If so:***

*No **SVN repositories** view (tab)? If it is not there:*

Window ~ Show View ~ Other

then **SVN ~ SVN Repositories**

1. *In your **SVN repositories** view (tab), **expand your repository** (the top-level item) if not already expanded.*

- If no repository, perhaps you are in the wrong Workspace. Get help.

2. ***Right-click on today's project**, then select **Checkout**.*

*Press **OK** as needed. The project shows up in the*

Pydev Package Explorer

*to the right. Expand and browse the modules under **src** as desired.*

Pair Programming

3

- Working in pairs on a single computer
 - ▣ One person, the *driver*, uses the keyboard
 - ▣ The other person, the *navigator*, watches, thinks, and takes notes
- For hard (or new) problems, this technique
 - ▣ Reduces number of errors
 - ▣ Saves time in the long run
- Works best when partners have similar skill level
- If not, then student with most experience should navigate, while the other student drives.

Why functions?

- A function allows us to group together several statements and give them a name by which they may be invoked.
 - **Abstraction** (easier to remember the name than the code)
 - **Compactness** (avoids duplicate code)
 - **Flexibility / Power** (parameters allow variation)
- Example:

```
def complain(complaint):  
    print("Customer:", complaint)
```


Review: Parts of a Function Definition



*Defining a function
called "hello"*

```
def hello():  
    print("Hello")  
    print("I'd like to complain about this parrot")  
  

```



Indenting tells interpreter
that these lines are part of
the hello function



Blank line tells interpreter
that we're done defining
the hello function

Review: Defining vs. Invoking

- **Defining** a function **says** what the function should do
- **Calling** (**invoking**) a function **makes** that happen
 - Parentheses tell interpreter to invoke the function

```
hello()
```

```
Hello
```

```
I'd like to complain about this parrot
```

Review: Function with a Parameter

Parameter, information that comes INTO the function.
Use the parameter in the body of the function.

□ Definition:

```
def complain(complaint):  
    print("Customer: I purchased this parrot not half "  
          + "an hour ago from this very boutique")  
    print("Owner: Oh yes, the Norwegian Blue. "  
          + "What's wrong with it?")  
    print("Customer:", complaint)
```

□ Invocation: `complain("It's dead!")`

□ Prints:

```
Customer: I purchased this parrot not  
half an hour ago from this very boutique  
Owner: Oh yes, the Norwegian Blue. What's wrong with it?  
Customer: It's dead!
```

Parameter being used in
the body of the function.

Actual argument: the
parameter is set to this
value when this invocation
of the function executes

When a function is invoked (called), Python follows a four-step process:

1. Calling program pauses at the point of the call.
2. Formal parameters get assigned the values supplied by the actual arguments.
3. Body of the function is executed.
 - ▣ The function may *return* a value.
4. Control returns to the point in calling program just after where the function was called.
 - ▣ If the function returned a value, we capture it in a variable or use it directly.

```
from math import pi
```

2: deg = 45

```
def deg_to_rads(deg):
```

```
    rad = deg * pi / 180
```

```
    return rad
```

3

```
degrees = 45
```

```
radians = deg_to_rads(degrees)
```

```
print(degrees, radians)
```

1

4

Functions can (and often should) return values

- We've **written** functions that just do things

- `hello()`
- `complain(complaint)`

- We've **used** functions that **return** values

- `abs(-1)`
- `fn_root_1 = math.sqrt(b*b - 4*a*c)`

- Define a function that returns a value

```
def square(x):  
    return x * x
```

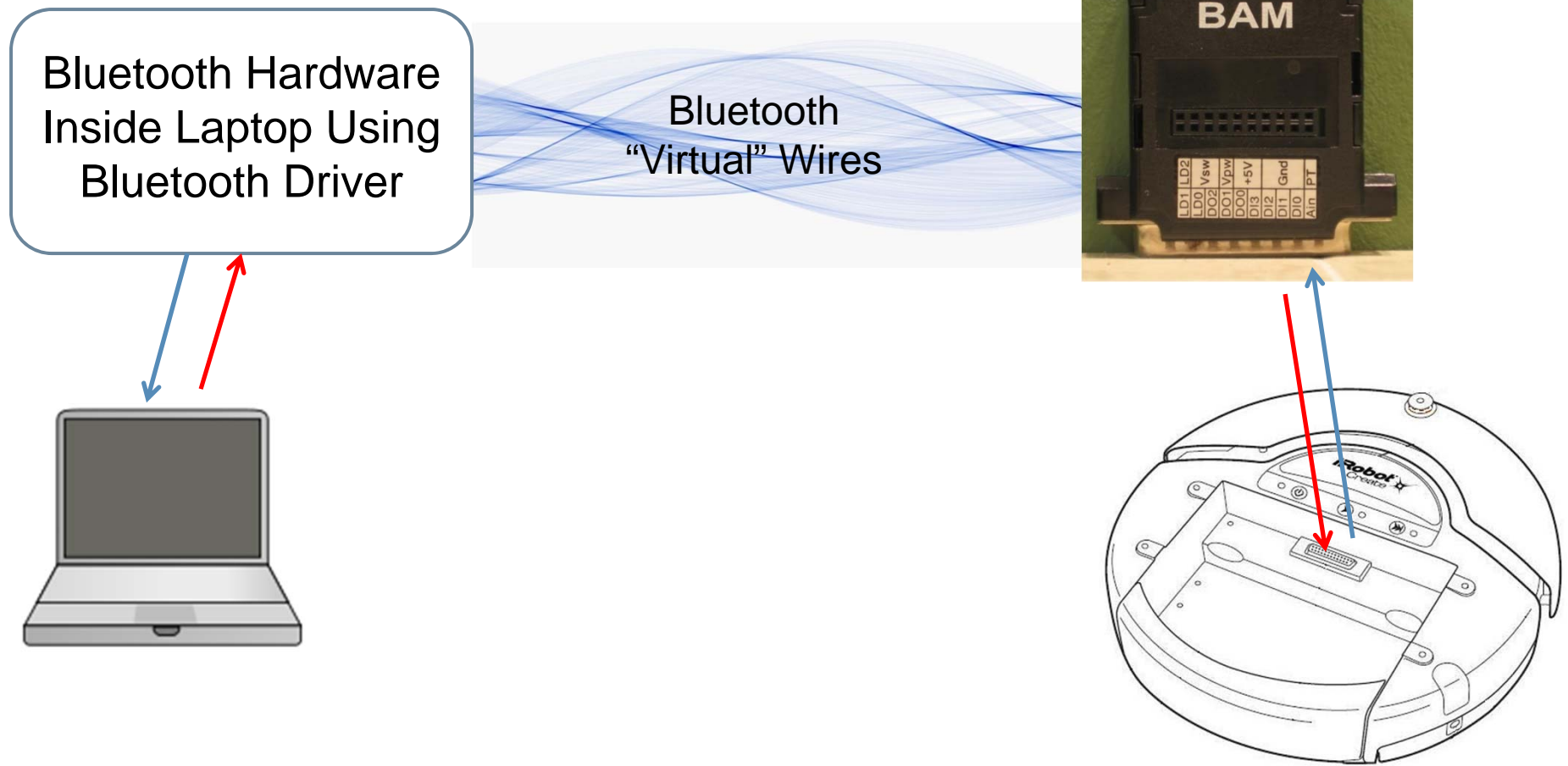
← return statement

Why might it be better to **return** than **print** when a function performs a calculation?

Parameters and actual arguments

- Do the exercises in `m1_parameters_and_parrots.py`

Wireless Bluetooth using the BAM!



BAM = Bluetooth Access Module

How to connect

- ONE partner, go to:

[http://www.rose-hulman.edu/class/csse/
resources/Robotics/ConnectingToTheCreateRobot.htm](http://www.rose-hulman.edu/class/csse/resources/Robotics/ConnectingToTheCreateRobot.htm)

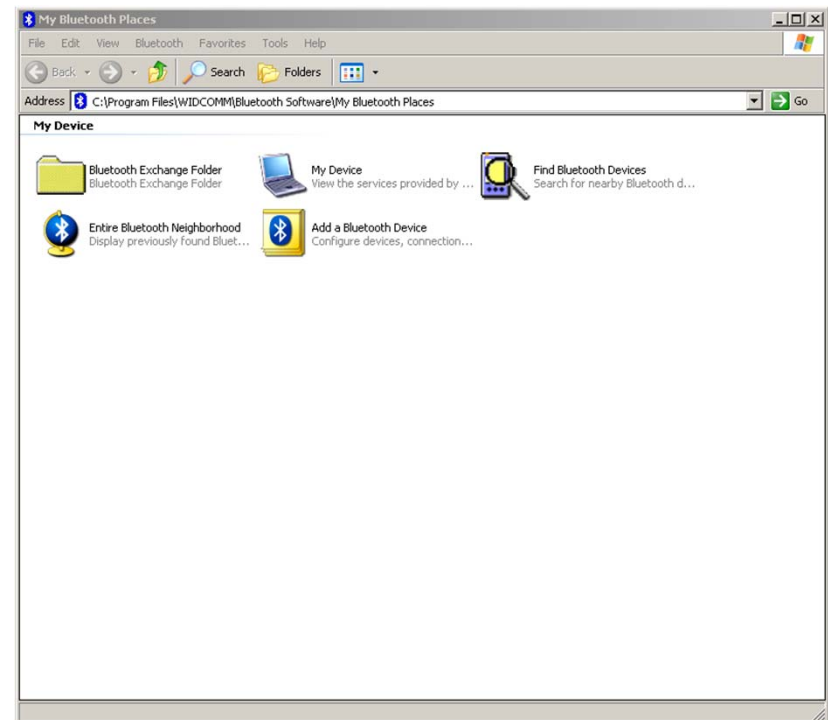
- Follow the directions there ***when your instructor tells you to do so.***
 - ▣ ***Once you have connected, shut down your robot***
- Meanwhile, let's talk about how Python can control the robot, then the PyCreate module that it uses to do so

Finding your Bluetooth device (pre-2009 laptops only)

- If you installed the Bluetooth driver from HW1 you should have a Bluetooth icon...

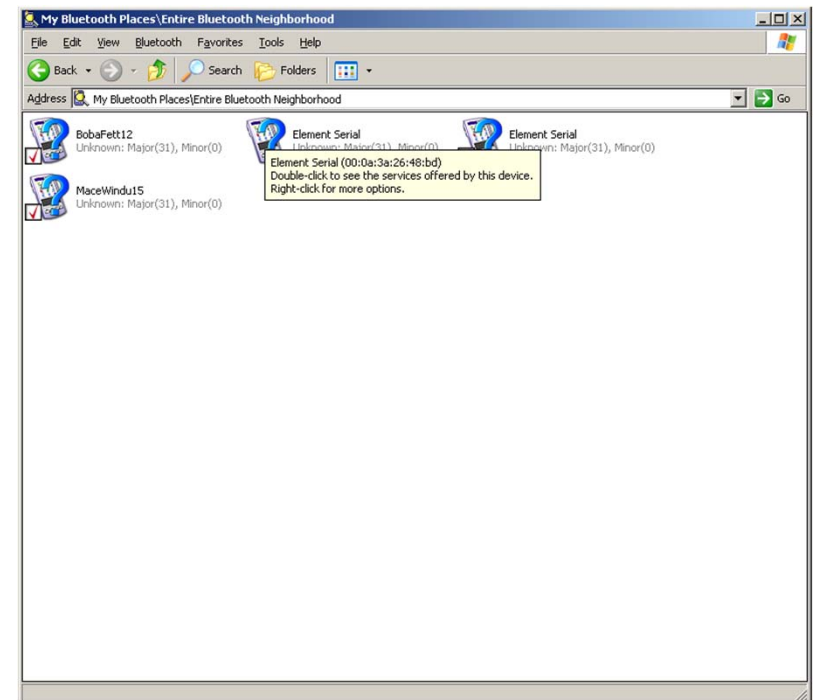
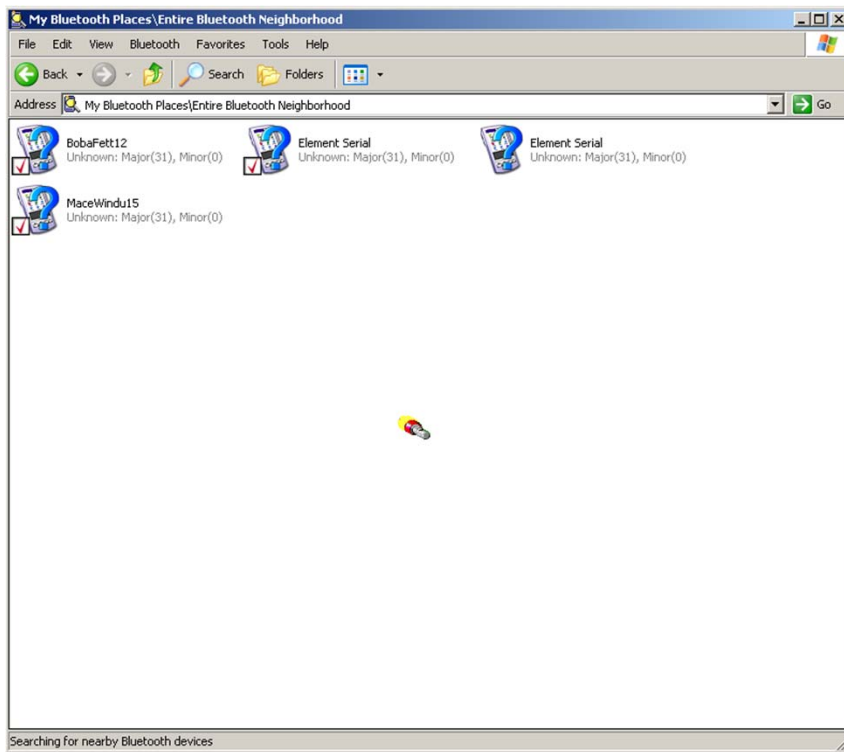


- Double click on that icon to bring up “My Bluetooth Places”
- Turn on Robot
- Double click on “Find Bluetooth Devices”



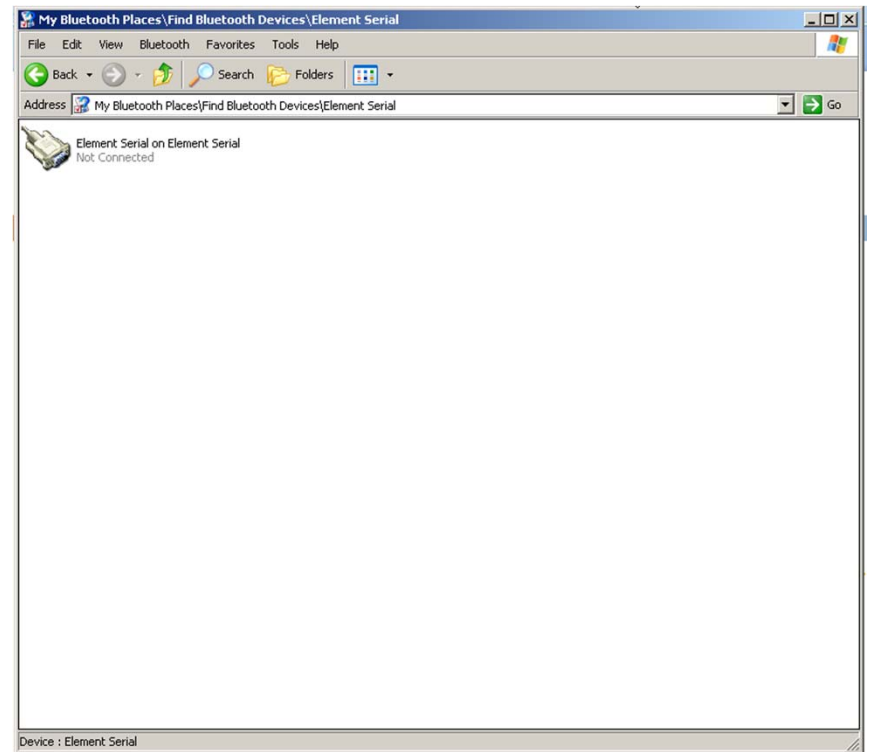
Finding YOUR iRobot Create BAM

- Let your computer find all devices
- Hover your mouse over an “Element Serial”
- Find the number that matches the one printed on your BAM



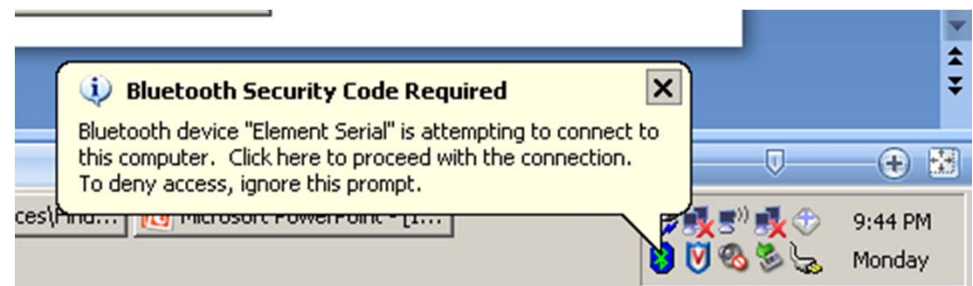
Connect to Element Serial

- Double click on the “Element Serial on Element Serial” to start the Bluetooth pairing process



Entering the Bluetooth security code

- ❑ You should have a window pop up that says “You need to enter a super top secret security code for this Bluetooth Device
- ❑ Click in that window
- ❑ (if you are too slow and the window goes away, click on the, now green, Bluetooth symbol directly)



This slide is Top Secret

- The top secret code for this Bluetooth device is the number...

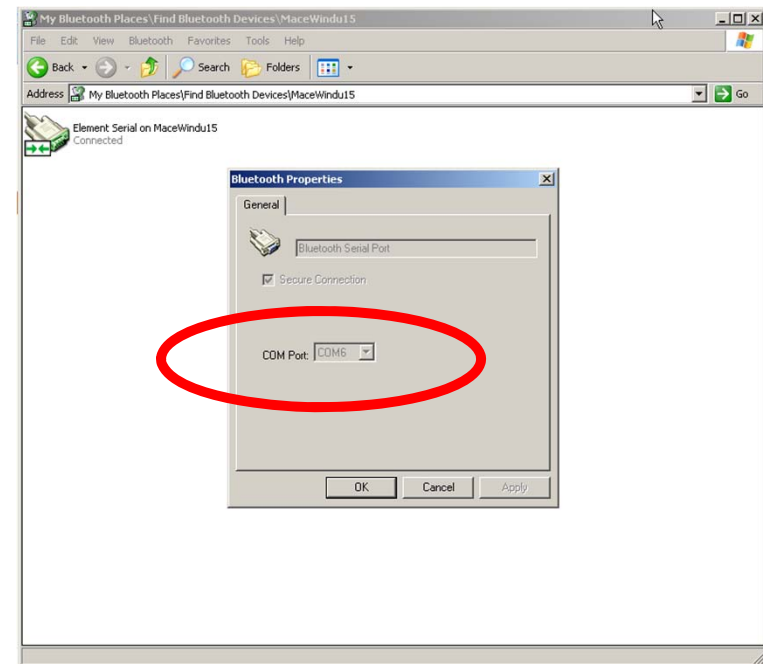
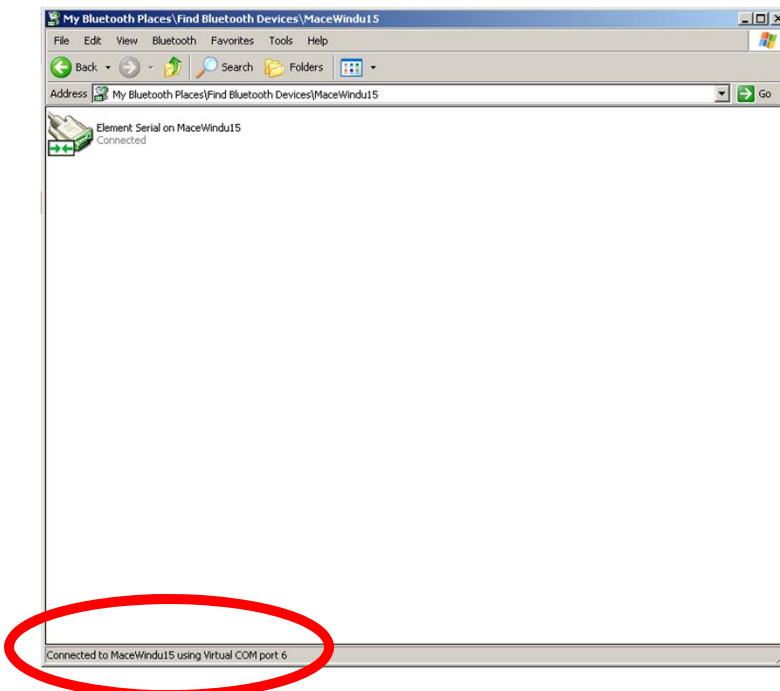
0000

Turns out every Bluetooth device that doesn't care about security is just four zeroes



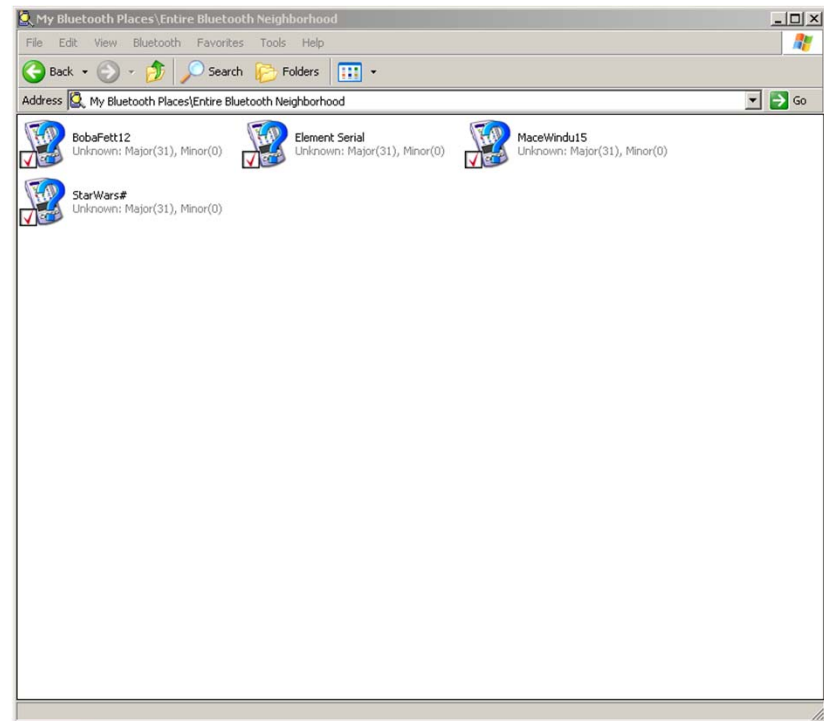
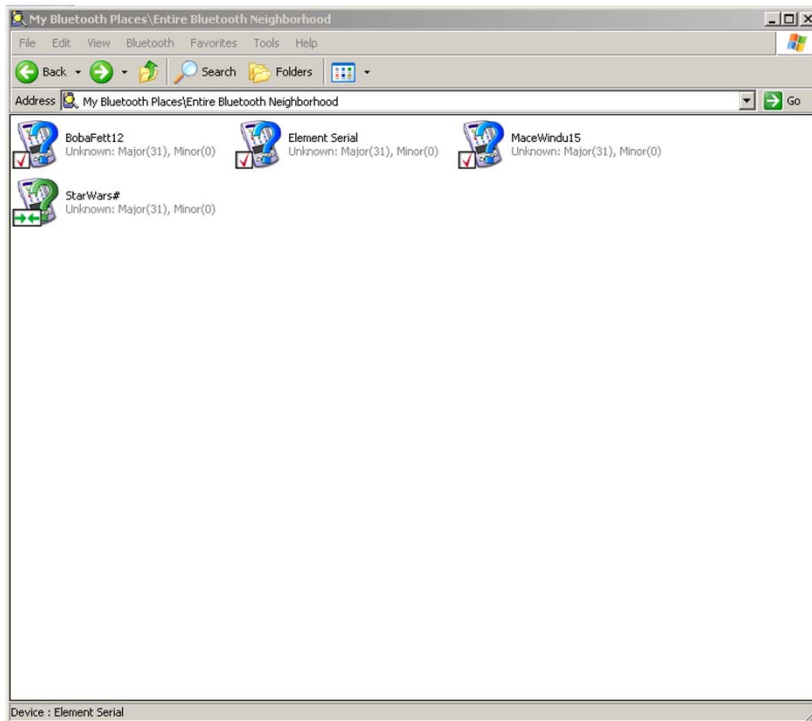
Remember the COM port

- You should now be connected to the BAM!
 - ▣ It's just like two virtual wires Rx and Tx for serial communication to the iRobot Create
- You need to know the COM port #
 - ▣ It's at the very bottom of the window (mine is COM port 6)
- (if you are too slow you can always right-click and open the Bluetooth Properties)

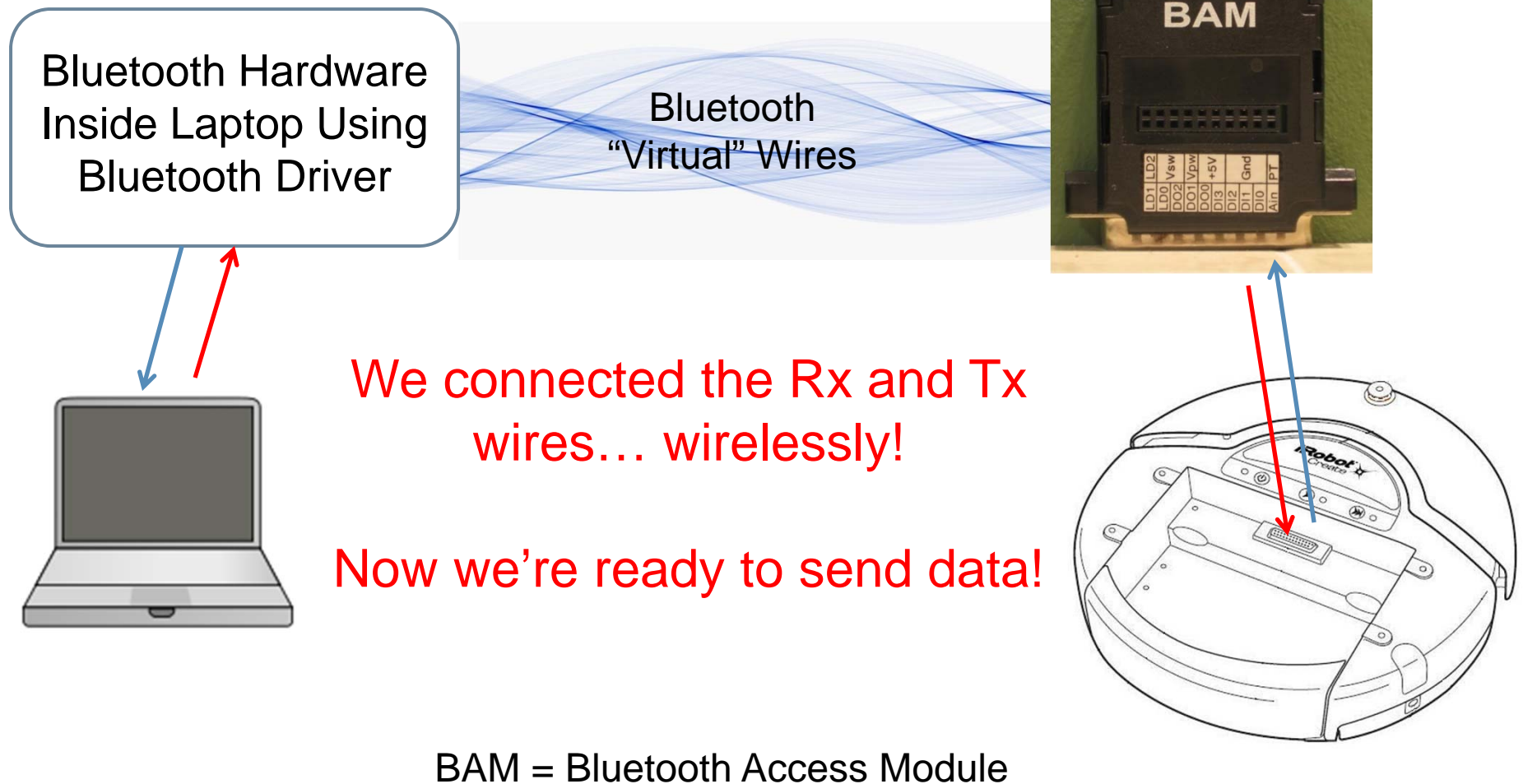


(optional) Renaming your Element Serial

- If you want you can rename your “Element Serial” to a name that is easier to find in your Bluetooth Neighborhood. I called this one “StarWars#”
- This only effects your computer, it doesn't change anything on the BAM
- Then next time you “Find Devices” it's easier to spot in the list



What did we just do?

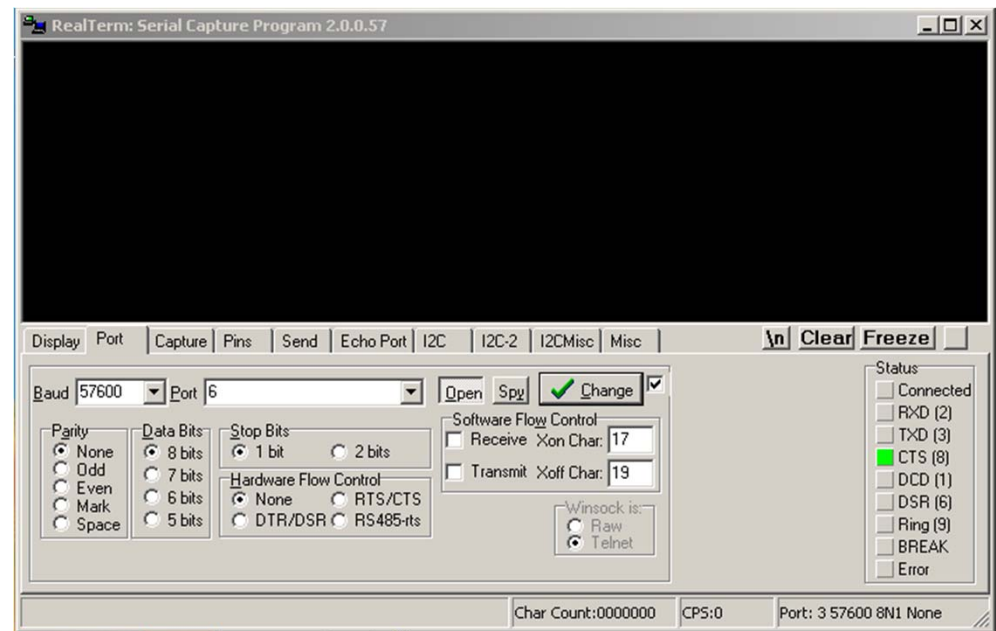


Communication Protocol

- iRobot sets the rules for communication
 - ▣ iRobot store website <http://store.irobot.com>
- Learn and practice the UART commands
 - ▣ Click on Educational... then Manuals
 - Owner's Guide
 - Open Interface Specifications
 - RealTerm
- Let's start with RealTerm
 - ▣ Sends UART messages over a COM port

RealTerm demo

- Open RealTerm
- Port tab
- Set Port to your COM port #
- Set Baud Rate to 57600 bits/second
- Click the Change button to make it actually happen

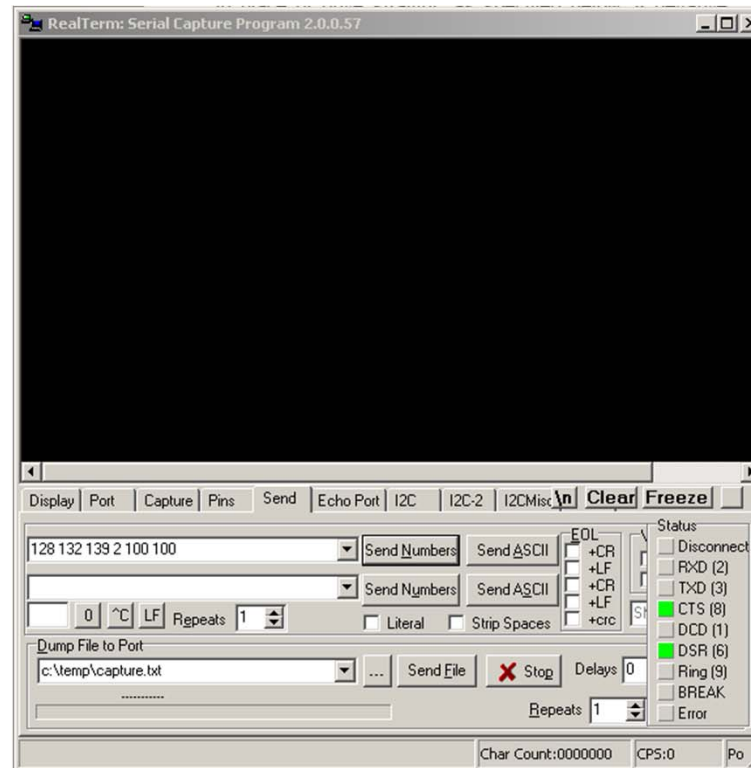


Send an LED command

- Turn on the Play LED with an amber Power LED
- Type 128 132 139 2 100 100

- Click “Send numbers”

- Watch!



LEDs **Opcode: 139** **Data Bytes: 3**

This command controls the LEDs on Create. The state of the Play and Advance LEDs is specified by two bits in the first data byte. The power LED is specified by two data bytes: one for the color and the other for the intensity.

- Serial sequence: [139] [LED Bits] [Power Color] [Power Intensity]
- Available in modes: Safe or Full
- Changes mode to: No Change
- LEDs data byte 1: LED Bits (0 – 10)

Advance and Play use green LEDs. 0 = off, 1 = on

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-----|-----|-----|-----|-----|---------|-----|------|-----|
| LED | n/a | n/a | n/a | n/a | Advance | n/a | Play | n/a |

Power uses a bicolor (red/green) LED. The intensity and color of this LED can be controlled with 8-bit resolution.

- LEDs data byte 2: Power LED Color (0 – 255)
0 = green, 255 = red. Intermediate values are intermediate colors (orange, yellow, etc).
- LEDs data byte 3: Power LED Intensity (0 – 255)
0 = off, 255 = full intensity. Intermediate values are intermediate intensities.

Example:

To turn on the Advance LED and light the Power LED green at half intensity, send the serial byte sequence [139] [8] [0] [128].

Digital Sensor Readings

- Let's request bumper and cliff sensor data

Bumps and Wheel Drops Packet ID: 7 Data Bytes: 1 unsigned

The state of the bumper (0 = no bump, 1 = bump) and wheel drop sensors (0 = wheel raised, 1 = wheel dropped) are sent as individual bits.

Range: 0 - 31

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------|-----|-----|-----|------------------|----------------|-----------------|-----------|------------|
| Sensor | n/a | n/a | n/a | Wheeldrop Caster | Wheeldrop Left | Wheeldrop Right | Bump Left | Bump Right |

Wall Packet ID: 8 Data Bytes: 1 unsigned

The state of the wall sensor is sent as a 1 bit value (0 = no wall, 1 = wall seen).

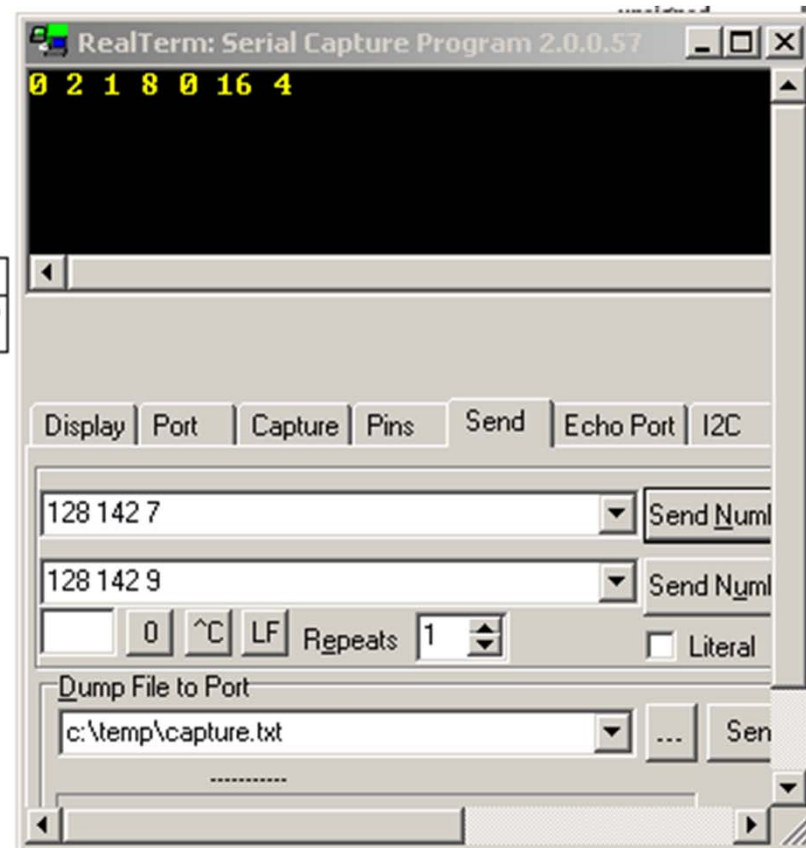
Range: 0 - 1

Cliff Left Packet ID: 9 Data Bytes: 1 unsigned

The state of the cliff sensor on the left side of Create is sent as a 1 bit value (0 = no cliff, 1 = cliff).

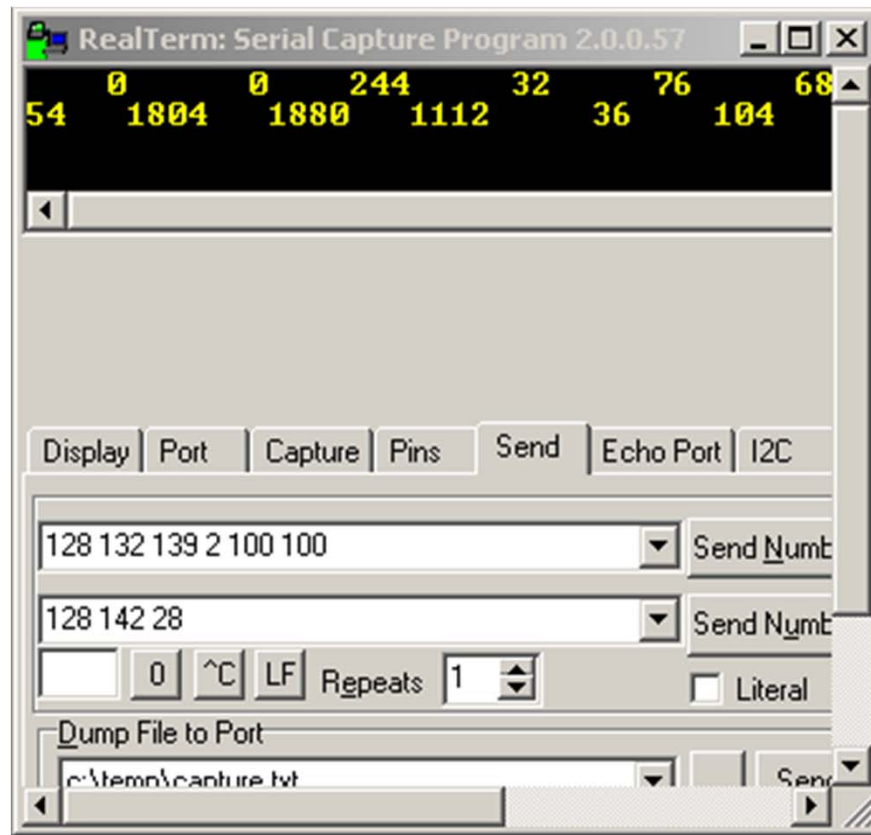
Range: 0 - 1

Digital
Version
Here



Analog Sensor Readings

- Request an analog value



Wall Signal Packet ID: 27 Data Bytes: 2
unsigned

The strength of the wall sensor's signal is returned as an unsigned 16-bit value, high byte first.

Range: 0-4095

Cliff Left Signal Packet ID: 28 Data Bytes: 2
unsigned

The strength of the left cliff sensor's signal is returned as an unsigned 16-bit value, high byte first.

Range: 0-4095

Cliff Front Left Signal Packet ID: 29 Data Bytes: 2
unsigned

The strength of the front left cliff sensor's signal is returned as an unsigned 16-bit value, high byte first.

Range: 0-4095

Sending commands in Python

- ❑ Open IDLE
- ❑ Import the serial Python library
- ❑ Make sure you are connected via Bluetooth
- ❑ Open a Serial port Object called tty

```
>>> from serial import *
>>> dir()
['EIGHTBITS', 'FIVEBITS', 'FileLike', 'MS_CTS_ON', 'MS_DSR_ON', 'MS_RING_ON', 'MS_RLSD_ON', 'PARITY_
EVEN', 'PARITY_MARK', 'PARITY_NAMES', 'PARITY_NONE', 'PARITY_ODD', 'PARITY_SPACE', 'SEVENBITS', 'SIX
BITS', 'STOPBITS_ONE', 'STOPBITS_TWO', 'Serial', 'SerialBase', 'SerialException', 'SerialTimeoutExce
ption', 'VERSION', 'XOFF', 'XON', '__builtins__', '__doc__', '__name__', 'device', 'os', 'portNotOpe
nError', 'serial', 'serialutil', 'serialwin32', 'sys', 'win32con', 'win32event', 'win32file', 'write
TimeoutError']
>>> tty = Serial(port=5,baudrate=57600,timeout=0.01)
```

Note: The serial module is zero based not 1 based so COM 6 is port 5 (sorry)

Sending commands in Python

- Send commands one by one in Python instead of RealTerm (kind of a brute force method)

```
>>> from serial import *
>>> tty = Serial(port=5, baudrate=57600, timeout=0.01)
>>> tty.write(chr(128))
>>> tty.write(chr(132))
>>> tty.write(chr(139))
>>> tty.write(chr(2))
>>> tty.write(chr(100))
>>> tty.write(chr(100))
>>>
```

Note: The serial module is zero based not 1 based so COM 6 is port 5 (sorry)

Make a function to setPlayLED

- We could make an LED function

```
>>> from serial import *
>>> tty = Serial(port=5, baudrate=57600, timeout=0.01)
>>> def setPlayLED(serialObject):
    serialObject.write(chr(128))
    serialObject.write(chr(132))
    serialObject.write(chr(139))
    serialObject.write(chr(2))
    serialObject.write(chr(100))
    serialObject.write(chr(100))

>>> setPlayLED(tty)
>>> tty.close()
```

When you are finished, close the COM port connection.

Using create.py

□ So much better! So much easier!

```
>>> from create import *
>>> robot = Create(6)
pycreate version 2.0
PORT is 6
Serial port did open on iRobot Create...
Putting the robot into safe mode...
>>> robot.setLEDs(200,255,1,1)
>>> robot.shutdown()
>>> |
```

While we are getting Bluetooth connections to work, examine the PyCreate handout. Determine how to make the robot:

- Construct a Create and initiate a connection
- Close a connection (shutdown)
- Go forward -- Turn -- Play a song

Your homework will involve these activities!

A PyCreate example

- `from create import *`
- `# Initiate a connection to the robot.`
`# Use the port for YOUR robot.`
- `robot = Create(9)`
- `for k in range(31, 128, 10):`
- `print "Note ", k`
- `robot.playNote(k, 16)`
- `time.sleep(0.5) # To give the note`
- `# time to play`
- `# Go forward for a couple of seconds`
- `robot.go(10)`
- `# Go forward 20 cm`
- `robot.go(10)`
- `robot.waitDistance(20)`
- `robot.stop()`
- `# Spin 180 degrees`
- `robot.go(0, 30)`
- `robot.waitAngle(180)`
- `robot.stop()`
- `# Disconnect`
- `robot.shutdown()`

Rest of Session

- ***Work on today's homework***

- Ask questions as needed!

- **Sources of help after class:**

- **Assistants in the CSSE lab**

- And other times as well (see link on the course home page)

- **Email**

`csse120-staff@rose-hulman.edu`

- You get faster response from the above than from just your instructor

Moench F-217

7 to 9 p.m.

Sundays thru Thursdays