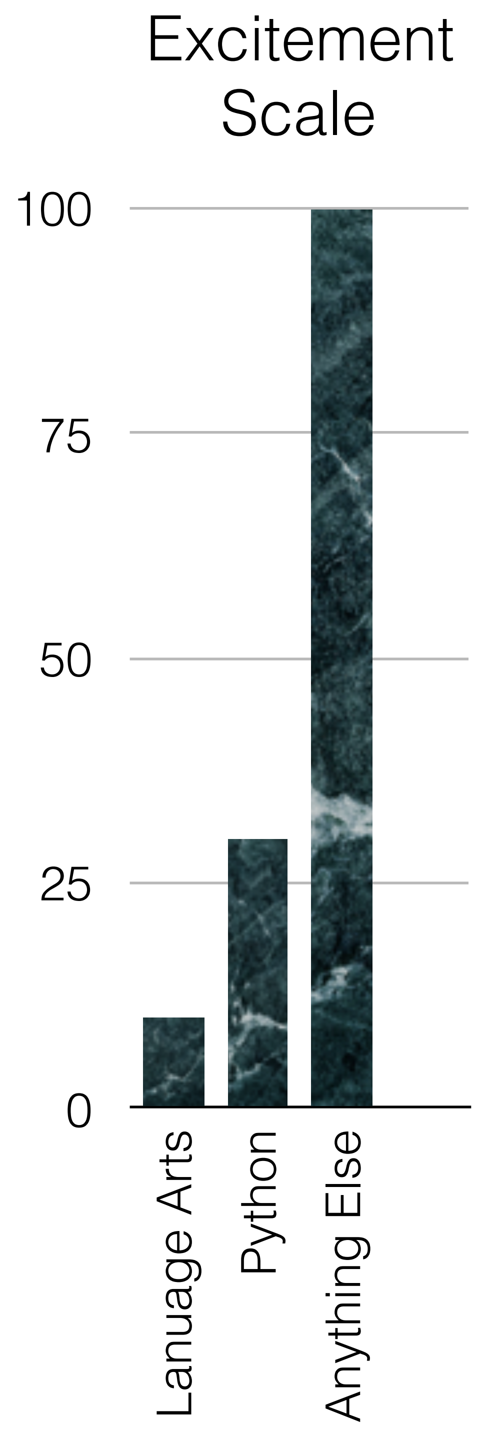
Abstract Syntax Trees
Raspberry Pi

https://github.com/sumsted/mempy_20160321.git

March 21, 2016
Scott Umsted

Kids in Python class are rarely excited about programming



But kids like Python class a lot more than other classes, especially on a Sunday afternoon

Things kids like more than school on a Sunday afternoon

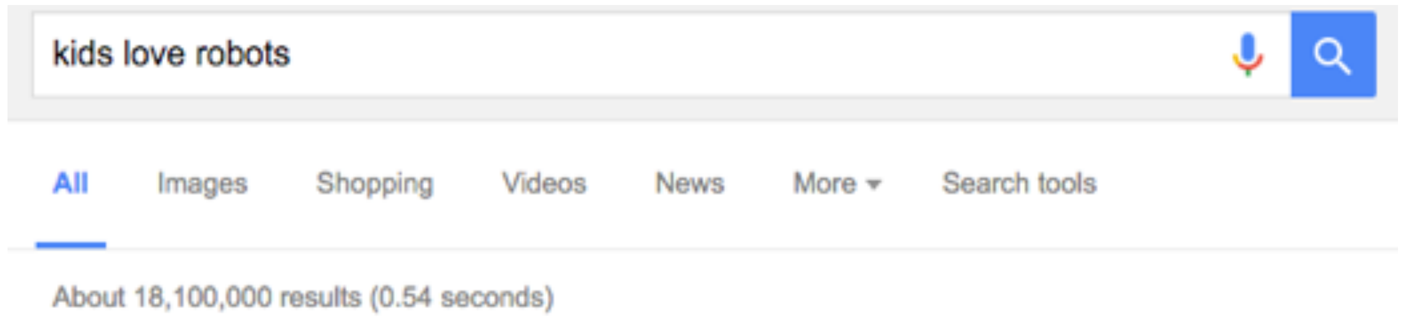
Kids like making things crash into one another

They like blinky lights

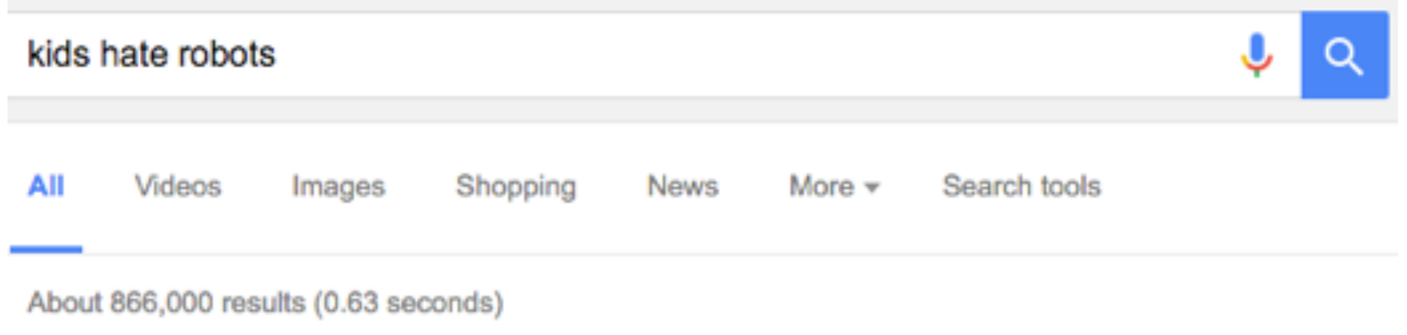
And they like Minecraft

I like the regular show

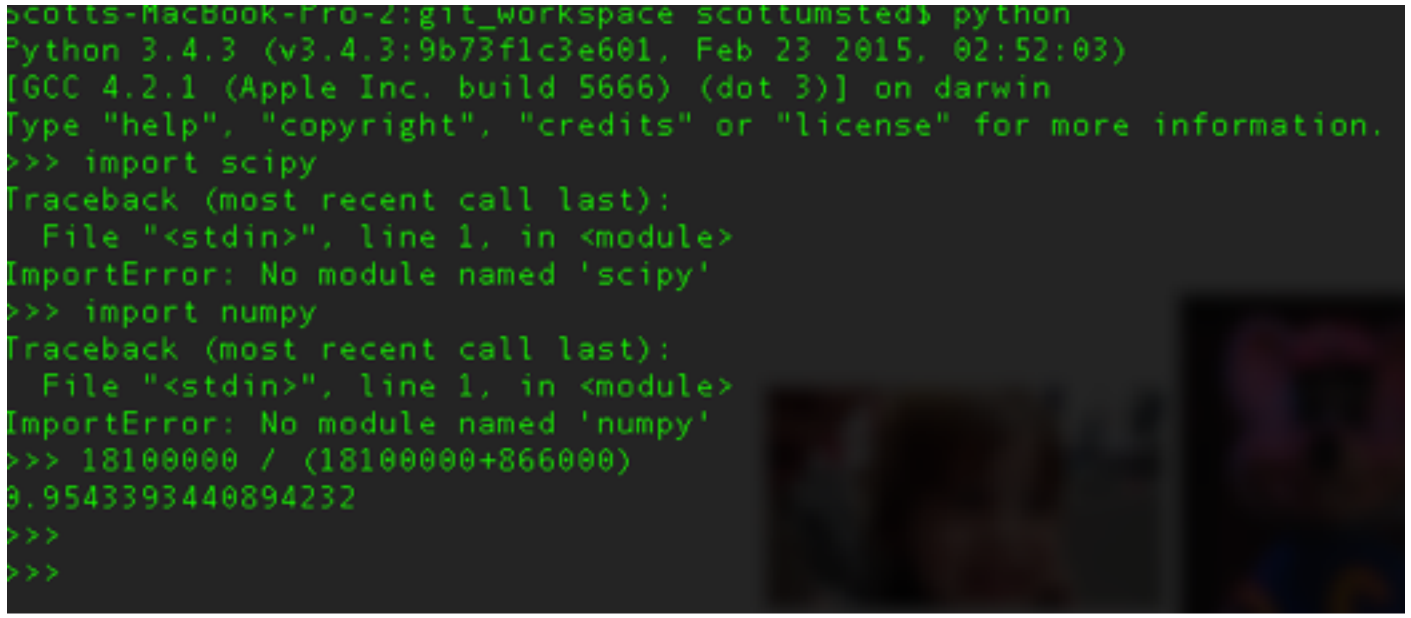
Scientifically provable that 95% of kids love robots


kids love robots

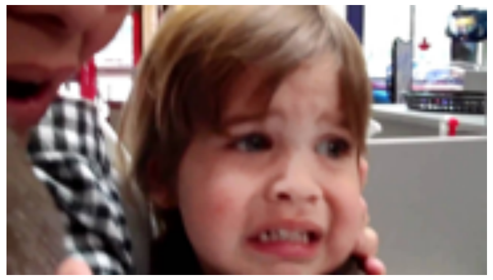
About 18,100,000 results (0.54 seconds)


kids hate robots

About 866,000 results (0.63 seconds)


```
Scotts-MacBook-Pro-2:git_workspace scottumsted$ python
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import scipy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'scipy'
>>> import numpy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'numpy'
>>> 18100000 / (18100000+866000)
0.9543393440894232
>>>
>>>
```

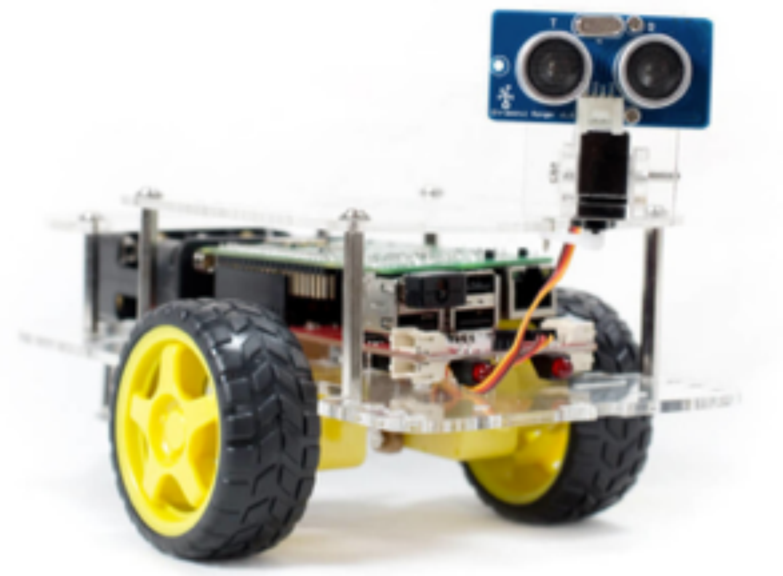
The other 5% live in a constant state of fear

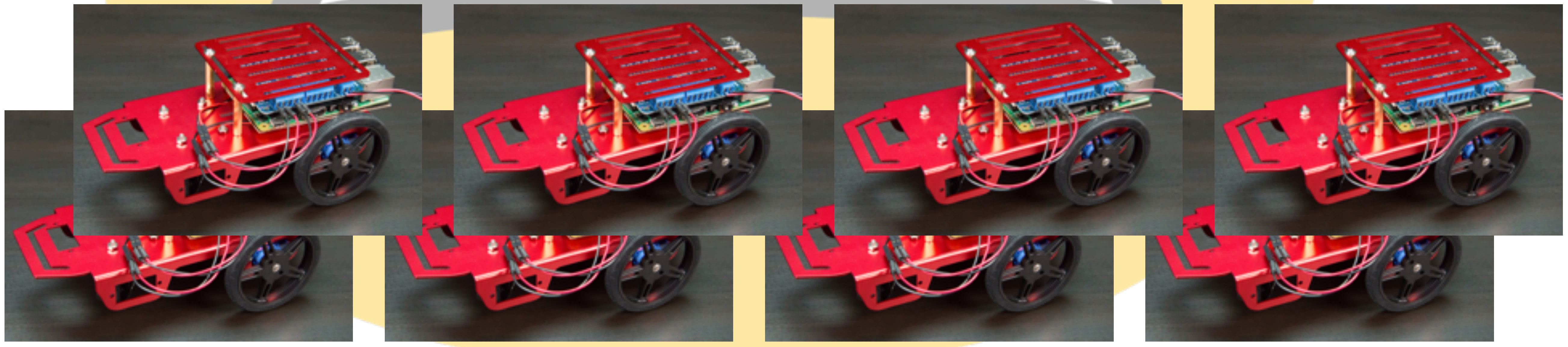


Today we have one…

But one day we will have an army!

You have been warned!

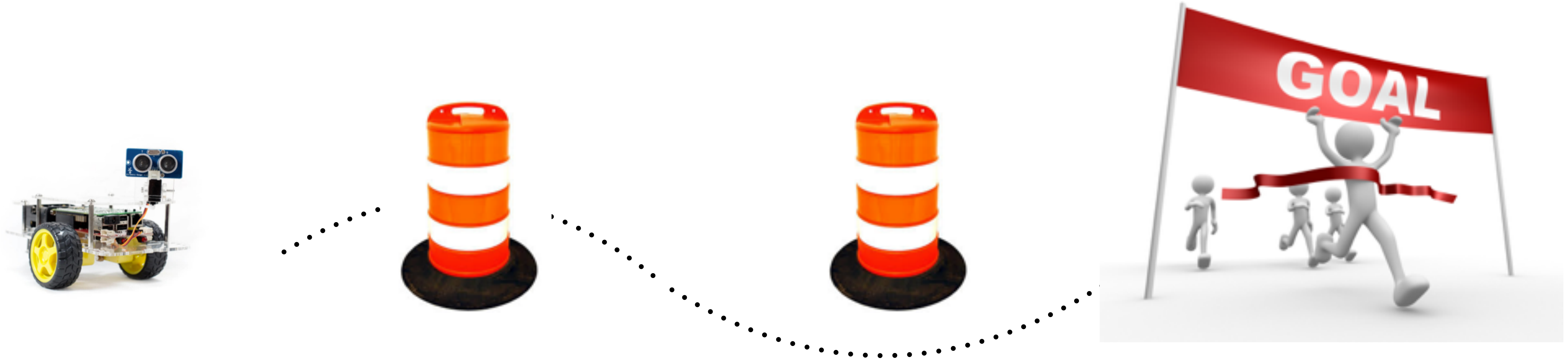

Gopigo - Dexter Industries



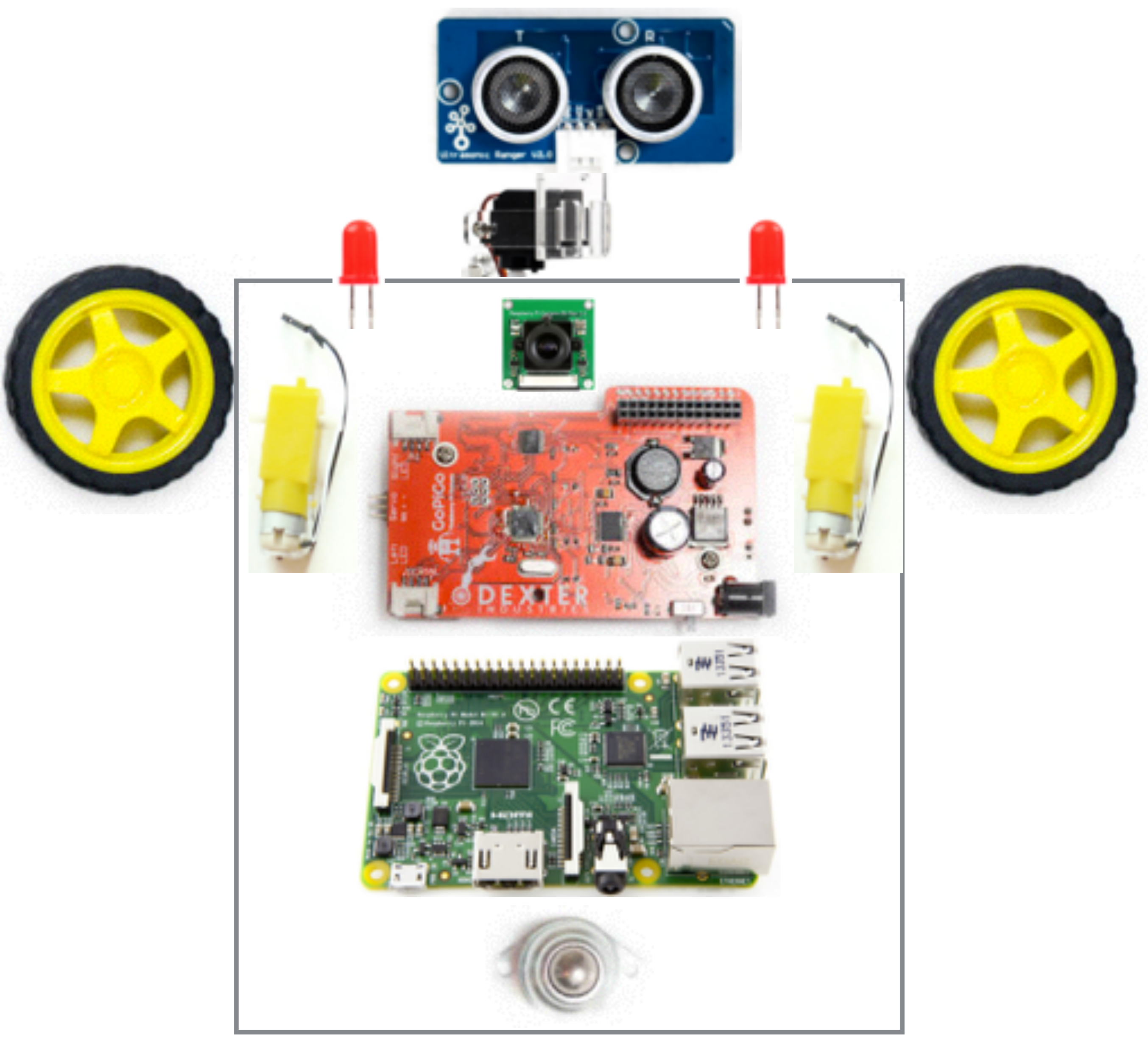
Big Problem

How do you get a room full of students to share one robot, in only one hour?

abstract syntax trees and lots of patience



how gopigo works



gopigo.py - 50 commands

```
#Move the GoPiGo forward without PID
def motor_fwd():
        return write_i2c_block(address,motor_fwd_cmd+[0,0,0])

#Move GoPiGo back
def bwd():
        return write_i2c_block(address,motor_bwd_cmd+[0,0,0])

#Move GoPiGo back without PID control
def motor_bwd():
        return write_i2c_block(address,motor_bwd_cmd+[0,0,0])

#Turn GoPiGo Left slow (one motor off, better control)
def left():
        return write_i2c_block(address,left_cmd+[0,0,0])

#Rotate GoPiGo left in same position (both motors moving in the opposite direction)
def left_rot():
        return write_i2c_block(address,left_rot_cmd+[0,0,0])

#Turn GoPiGo right slow (one motor off, better control)
def right():
        return write_i2c_block(address,right_cmd+[0,0,0])

#Rotate GoPiGo right in same position both motors moving in the opposite direction)
def right_rot():
        return write_i2c_block(address,right_rot_cmd+[0,0,0])

#Stop the GoPiGo
def stop():
        return write_i2c_block(address,stop_cmd+[0,0,0])

#Increase the speed
def increase_speed():
        return write_i2c_block(address,ispd_cmd+[0,0,0])

#Decrease the speed
def decrease_speed():
        return write_i2c_block(address,dspd_cmd+[0,0,0])
```

code must run on gopigo

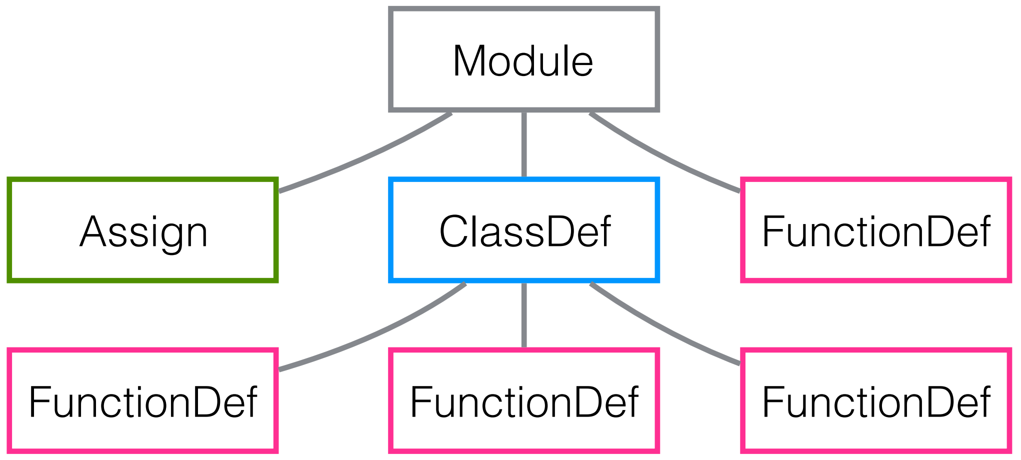
I could code a client and server, but ugh

what’s an abstract syntax tree

An abstract syntax tree is how Python dissects your code into Python grammar.

```python
SOME_CONSTANT = 4


class MyClass():
    def __init__(self, some_number):
        self._some_number = some_number
        self._a_name = None

    def do_something(self, some_other_number, a_name):
        self._a_name = a_name
        return self._some_number + some_other_number

    def greet(self):
        return 'Hello ' + self._a_name


def start():
    my_object = MyClass(123)
    print(my_object.do_something(321, 'Bob'))
    print(my_object.greet())


if __name__ == '__main__':
    start()
```

```
Module(
    body=[
        Assign(
            targets=[
                Name(
                    id='SOME_CONSTANT',
                    ctx=Store()
                )
            ],
            value=Num(n=4)
        ),
        ClassDef(
            name='MyClass',
            bases=[],
            body=[
                FunctionDef(
                    name='__init__',
                    args=arguments(args=[
                        Name(id='self',
                             ctx=Param()),
                        Name(
                            id='some_number',
                            ctx=Param())],
                                vararg=None,
                                kwarg=None,
                                defaults=[]),
```



evaluating gopigo.py

```
#Write I2C block
def write_i2c_block(address,block):
      try:
            op=bus.write_i2c_block_data(address,1,block)
            time.sleep(.005)
            return op
      except IOError:
            if debug:
                  print "IOError"
            return -1
      return 1

#Write a byte to the GoPiGo
def writeNumber(value):
      try:
            bus.write_byte(address, value)
            time.sleep(.005)
      except IOError:
            if debug:
                  print "IOError"
            return -1
      return 1

#Read a byte from the GoPiGo
def readByte():
      try:
            number = bus.read_byte(address)
            time.sleep(.005)
      except IOError:
            if debug:
                  print "IOError"
            return -1
      return number

#Control Motor 1
def motor1(direction,speed):
      return write_i2c_block(address,m1_cmd+[direction,speed,0])

#Control Motor 2
def motor2(direction,speed):
      return write_i2c_block(address,m2_cmd+[direction,speed,0])
```

```
{'name': 'write_i2c_block', 'arguments': ['address', 'block']}
{'name': 'writeNumber', 'arguments': ['value']}
{'name': 'readByte', 'arguments': []}
{'name': 'motor1', 'arguments': ['direction', 'speed']}
{'name': 'motor2', 'arguments': ['direction', 'speed']}
{'name': 'fwd', 'arguments': []}
{'name': 'motor_fwd', 'arguments': []}
{'name': 'bwd', 'arguments': []}
{'name': 'motor_bwd', 'arguments': []}
{'name': 'left', 'arguments': []}
{'name': 'left_rot', 'arguments': []}
{'name': 'right', 'arguments': []}
{'name': 'right_rot', 'arguments': []}
{'name': 'stop', 'arguments': []}
{'name': 'increase_speed', 'arguments': []}
{'name': 'decrease_speed', 'arguments': []}
{'name': 'trim_test', 'arguments': ['value']}
{'name': 'trim_read', 'arguments': []}
{'name': 'trim_write', 'arguments': ['value']}
{'name': 'digitalRead', 'arguments': ['pin']}
{'name': 'digitalWrite', 'arguments': ['pin', 'value']}
{'name': 'pinMode', 'arguments': ['pin', 'mode']}
{'name': 'analogRead', 'arguments': ['pin']}
{'name': 'analogWrite', 'arguments': ['pin', 'value']}
{'name': 'volt', 'arguments': []}
```

generateapi.py

student laptop

Using ast automatically generate server, client and test code.

Students import client code to run their controllers remotely.

student_code.py

```
import time
import gopigo_client as gopigo

gopigo.set_speed(120)
gopigo.set_right_speed(125)
gopigo.fwd()
time.sleep(5)
gopigo.stop()
```

gopigo_client.py

```
import requests

host = 'http://0.0.0.0:8080'


def g(action, *args):
    url = host+'/'+action
    for arg in args:
        url += '/'+str(arg)
    r = requests.get(url)
    return r.json()


def set_right_speed(speed):
    return g('set_right_speed', speed)['return_value']


def set_speed(speed):
    return g('set_speed', speed)['return_value']
```

gopigo

gopigo_server.py

```
from bottle import get, request, response, run
import gopigo


@get('/set_right_speed/<speed>')
@handle_padded
def set_right_speed(kargs):
    r = {'return_value':
          gopigo.set_right_speed(int(kargs['speed']))}
    return r


@get('/set_speed/<speed>')
@handle_padded
def set_speed(kargs):
    r = {'return_value':
          gopigo.set_speed(int(kargs['speed']))}
    return r


run(host='0.0.0.0', port=8080, debug=True)
```

gopigo.py

```
#Set speed of the right motor
#      arg:
#            speed-> 0-255
def set_right_speed(speed):
      if speed >255:
            speed =255
      elif speed <0:
            speed =0
      return write_i2c_block(address,
            set_right_speed_cmd+[speed,0,0])

#Set speed of the both motors
#      arg:
#            speed-> 0-255
def set_speed(speed):
      if speed >255:
            speed =255
      elif speed <0:
            speed =0
      set_left_speed(speed)
      time.sleep(.1)
      set_right_speed(speed)
```

https://github.com/sumsted/mempy_20160321.git

gopigo robot
wifi: gopu   raspberry
http://192.168.42.1:8080

gopigo_client.py - generated
robot.py - wrapper for students
solution.py - student code

---

sense hat
wifi: gopi   raspberry
sense_hat_server
http://192.168.1.110:8088

sense_hat_client.py - generated
rainbow.py - paint a rainbow
text_scroll.py - scroll text
examine sense_hat_client for calls

---

hack_sense
http://192.168.1.110:8080

guess.py - client api
exercise.py - student code
linear_search.py - solution
binary_search.py - solution

---

mempi
minecraft
wifi: gopi   raspberry
192.168.1.111

mcpi - minecraft python library
server.py - rpi address
shapes.py - ascii art
simplemine.py - wrapper, student code
woolcolors.py - enum for colors

- Gyroscope, accelerometer, and magnetometer sensor
- Temperature and humidity sensor
- Barometric pressure sensor
- 8×8 RGB LED display
- Mini joystick

```
from sense_hat import SenseHat
sense = SenseHat()

temperature = sense.temperature
sense.show_message("Temperature is %d" % temperature)
```

```
#from sense_hat import SenseHat
import sense_hat_client as sense

#sense = SenseHat()

temperature = sense.temperature
sense.show_message("Temperature is %d" % temperature,
       .1, (0, 255, 0), (255, 0, 255))
```