Taking over the world with Scratch

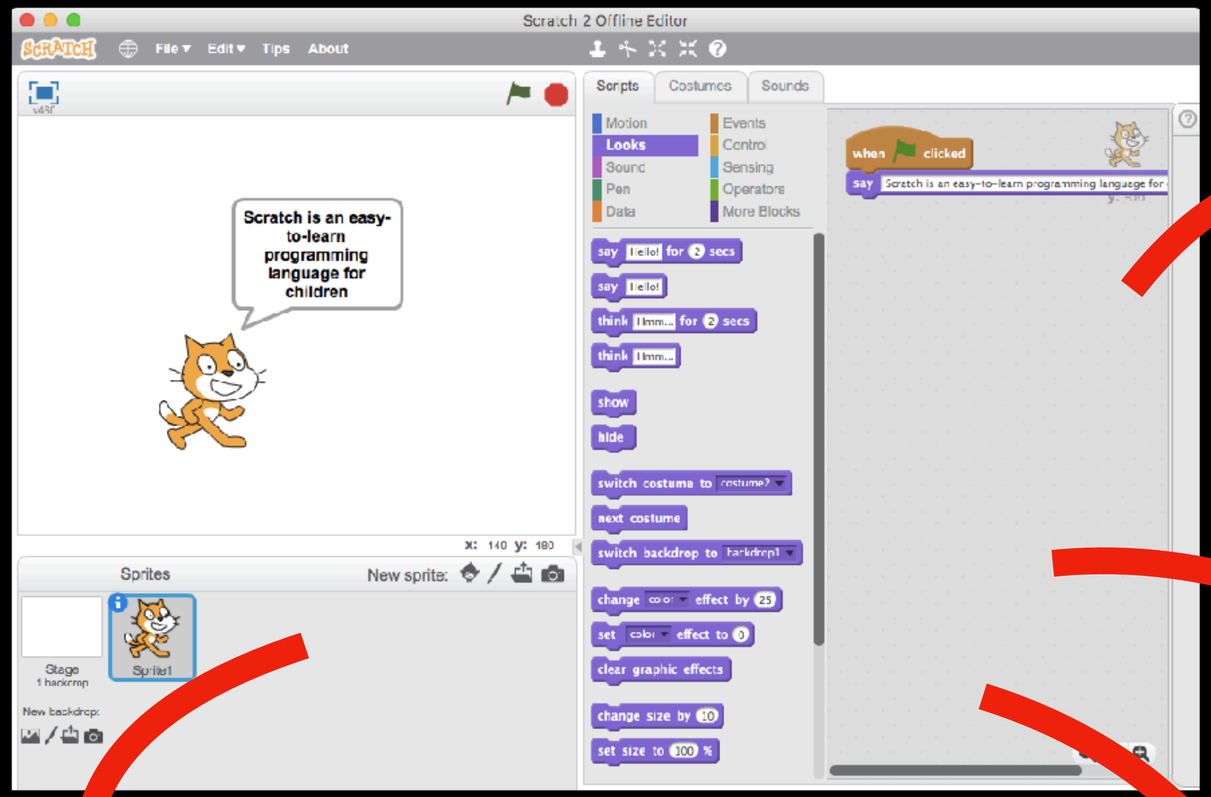
Kevin Sheldrake

EMFCamp2018

whoami

Work	**Play**
Researcher (hacking)	Making and breaking
Tool development	Conference speaker
Offensive cryptography	Arduino
Reverse engineering	Lego + Power Functions
Radio & RFID	Hypnotist and magician

EMFCamp2018



EMFCamp2018

Legal Note: Printing this to hardcopy prevents the electronic edition from smelling like burning paper. We'll be printing a few thousand of our own, but we also insist that you print it by laserjet or typewriter самиздат, giving it away to friends and strangers. Sneak it into a food delivery rack at your local dive bar, or hide it between two books on the shelves of your university library.

Reprints: Bitrot will burn libraries with merciless indignity that even Pets Dot Com didn't deserve. Please mirror—don't merely link!—`pocorgtfo18.pdf` and our other issues far and wide, so our articles can help fight the coming flame deluge. Not running one of our own, we like the following mirrors.

Let’s talk about Scratch!

EMFCamp2018

Scratch

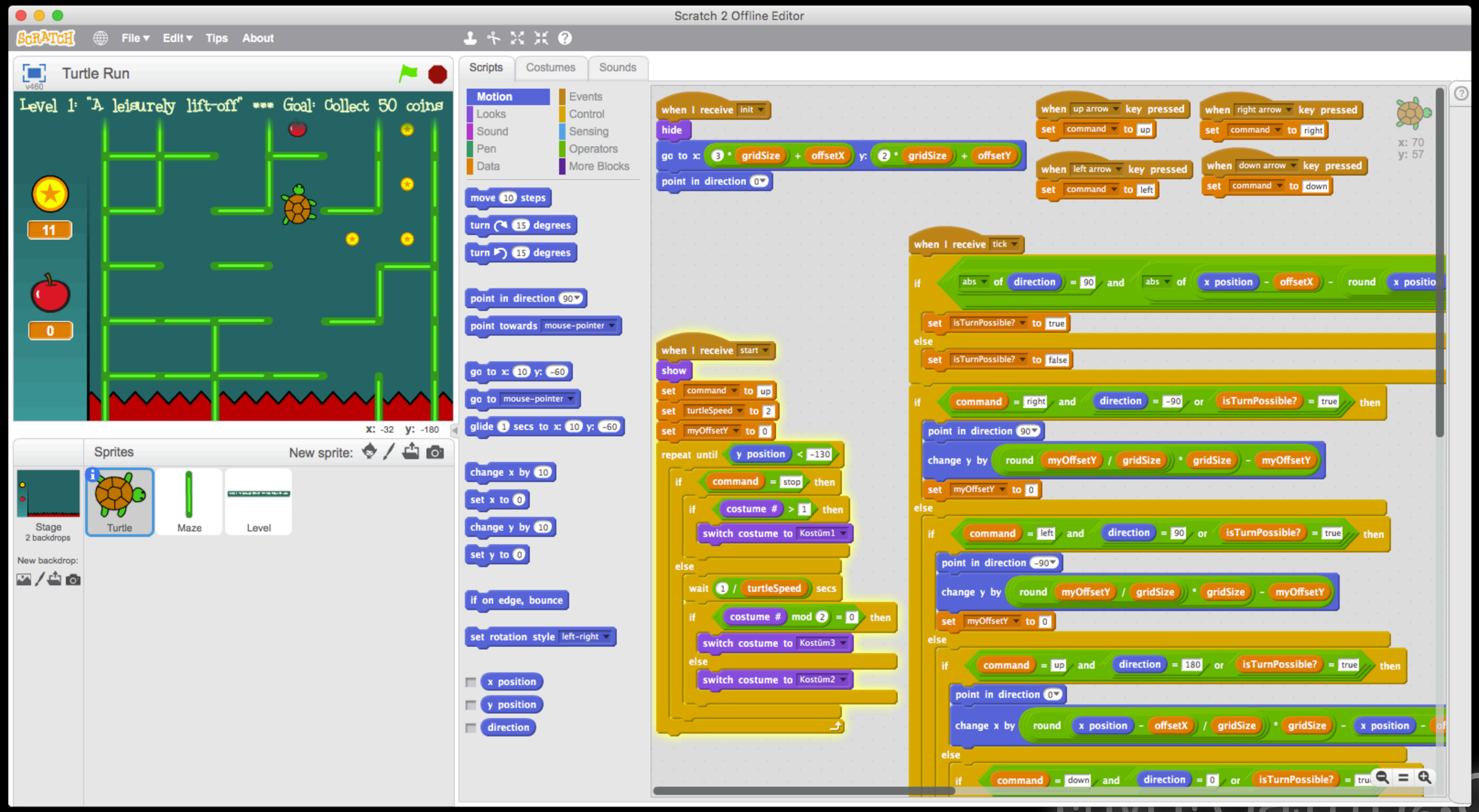
- Developed by the Lifelong Kindergarten group at the MIT Media Lab in 2003
  - Written in squeak/smalltalk
- Scratch 2 was released in 2013 and included custom blocks
  - Written in Flash/Adobe Air
- Scratch 3 is in development
  - HTML5/Javascript

EMFCamp2018

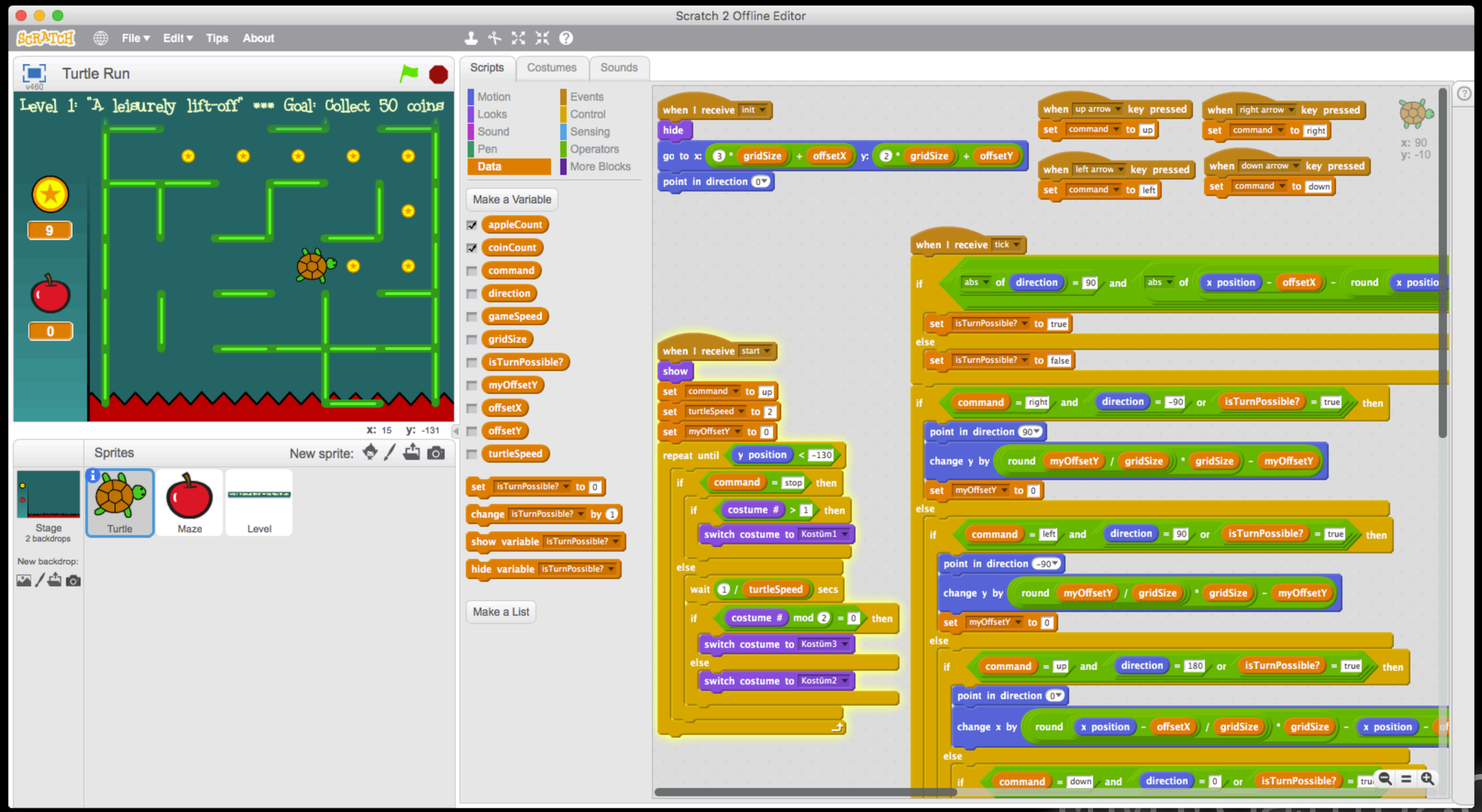
Scratch

- Typically online activity - http://scratch.mit.edu
- Offline versions of Scratch 1.4 and 2 are available
- Scratch 2 offline version permits
  - 'Experimental HTTP Extensions'
    - Robot control, for example

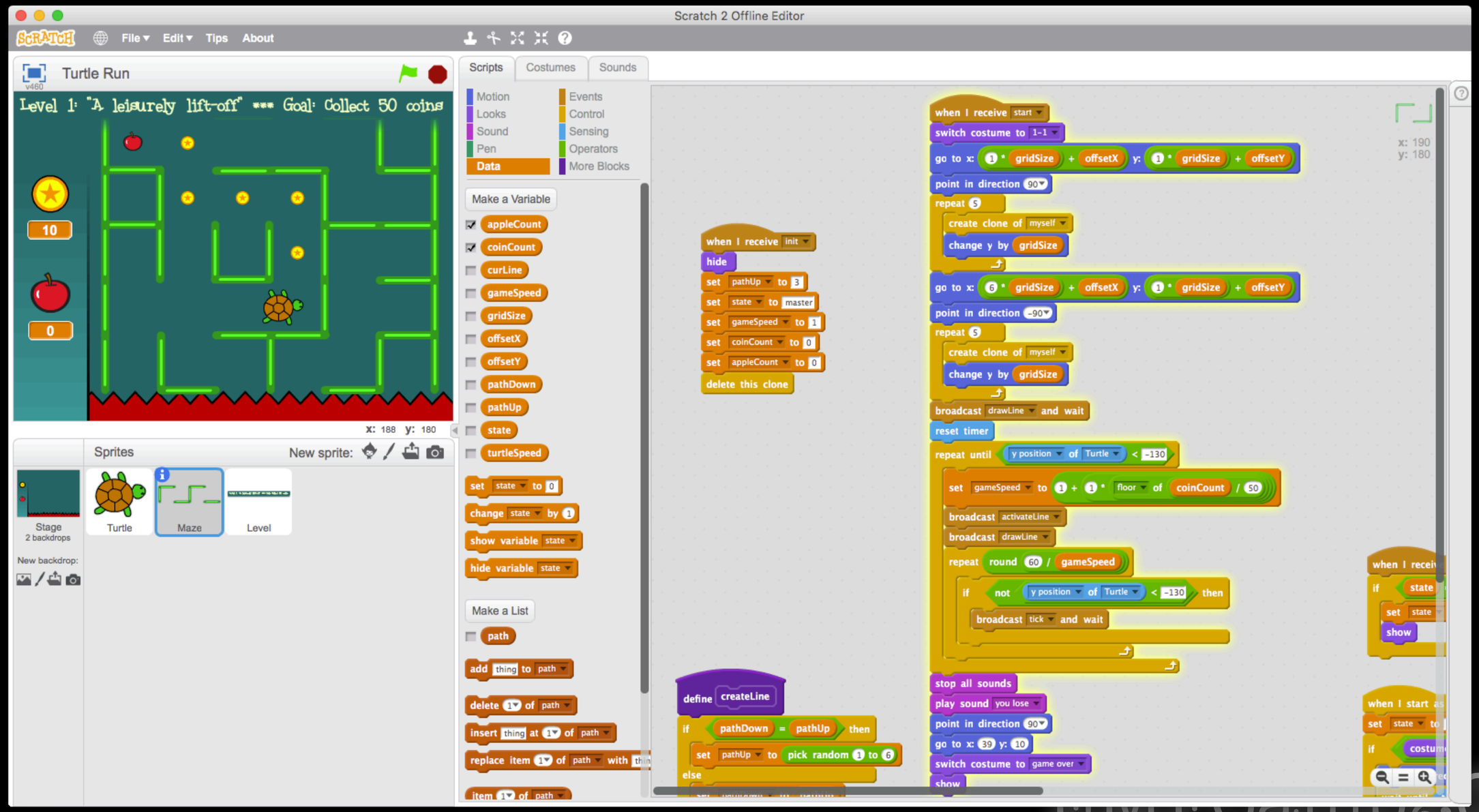
Scratch code



Scratch code



Scratch code



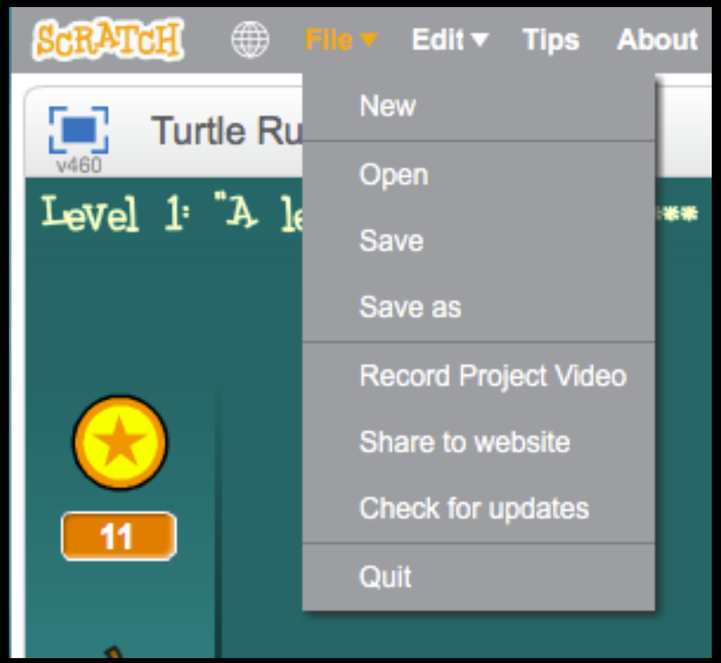
Scratch elements

- Variables:
  - global
  - local to sprite / stage
- Blocks:
  - procedures
  - not functions, e.g. no return values

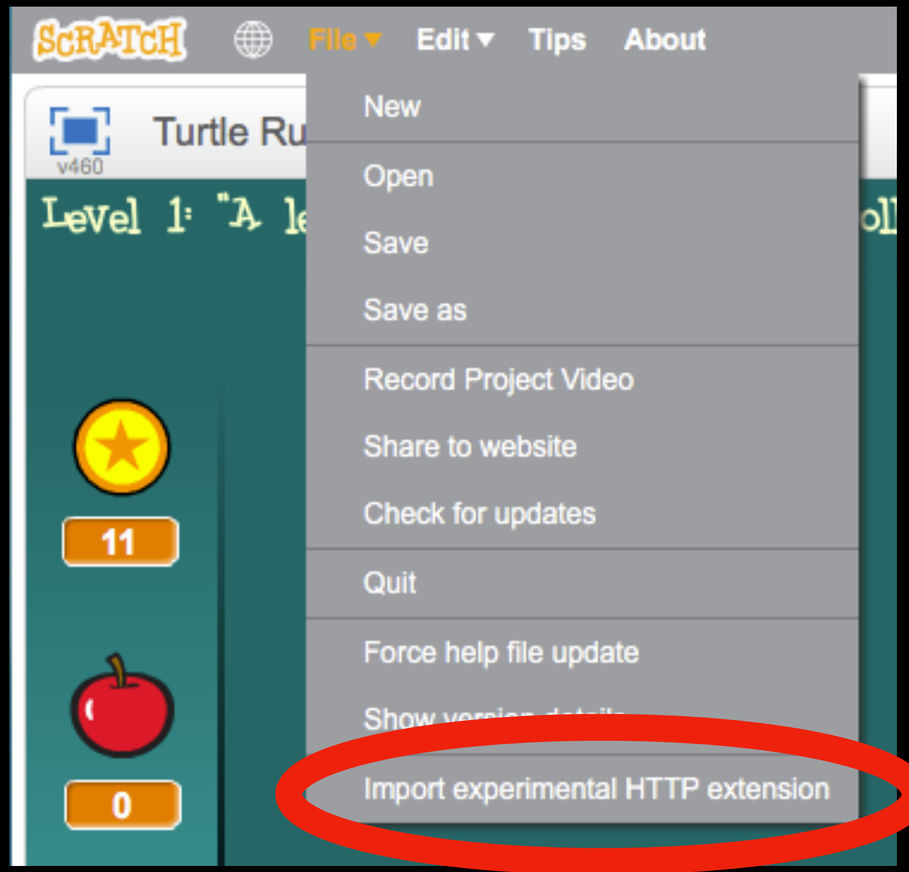
Experimental HTTP Extensions?

EMFCamp2018

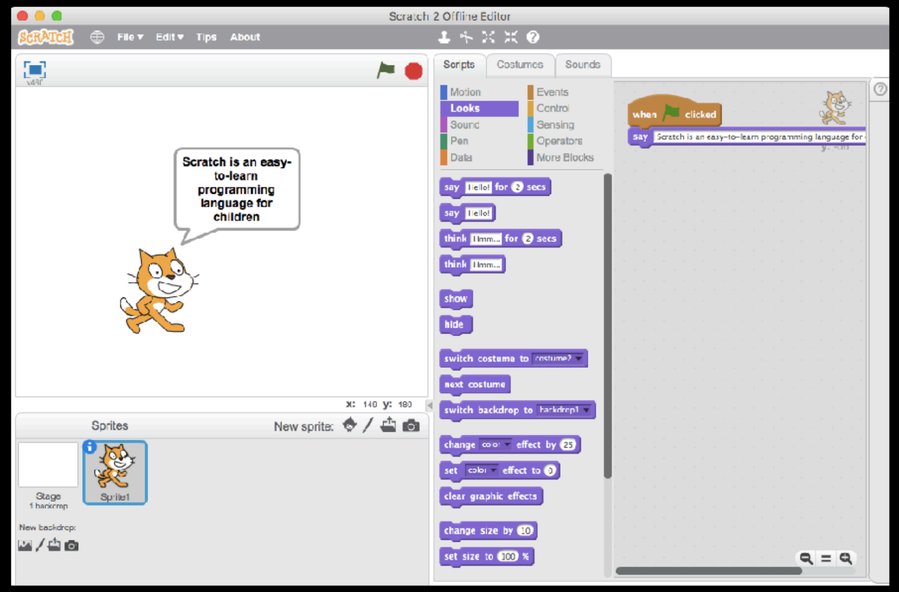
Standard File menu



Shift-click File menu



EMFCamp2018



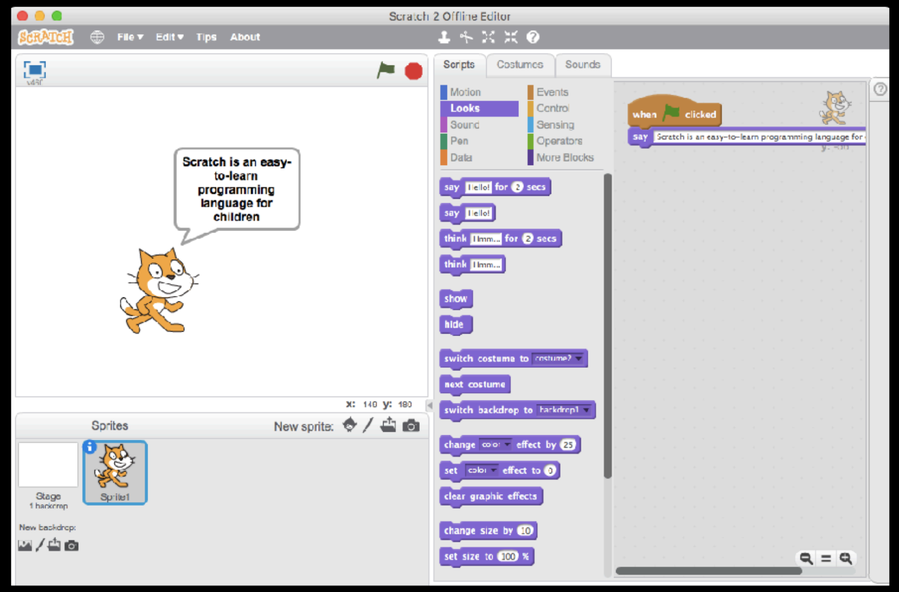
Import Scratch extension (.s2e file)

Invoke procedures

Poll for variables (30/s)

Web server on localhost

EMFCamp2018



Invoke procedures

Procedures

Modify variables

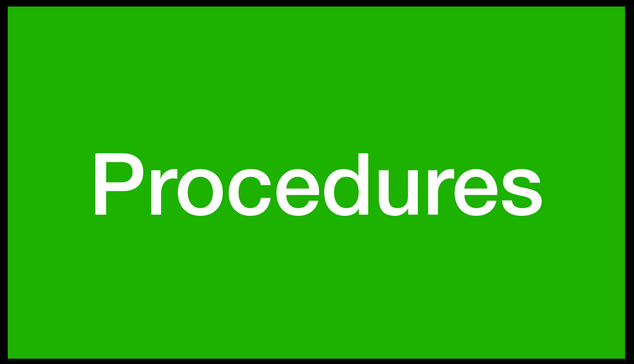
Exposed variables

Read variables (poll 30/s)

EMFCamp2018

Extension example

- Robot arm:
  - Procedures:
    - Move motor 1 n degrees
    - Move motor 2 n degrees
  - Variables:
    - Motor 1 limit switch
    - Motor 2 limit switch



What if…?

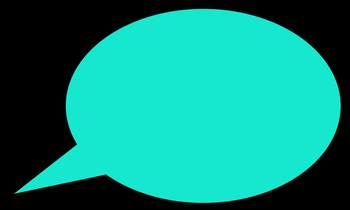
- Procedures:
  - Play MIDI note on channel for given time
  - MIDI ‘note on’ event
  - MIDI ‘note off’ event
  - MIDI controller change
  - MIDI pitch bend change

Procedures

Exposed variables

EMFCamp2018

What if...?



- Procedures:
  - ‘Say’ message with specific voice

Procedures

Exposed variables

What if...?

- Procedures:
  - Lego motor blue/red forwards/ backwards
  - Lego motor blue/red speed -7..0..7
  - Lego motor blue/red stop
- Variables:
  - motors ready?

Procedures

Exposed variables

EMFCamp2018

What if...?

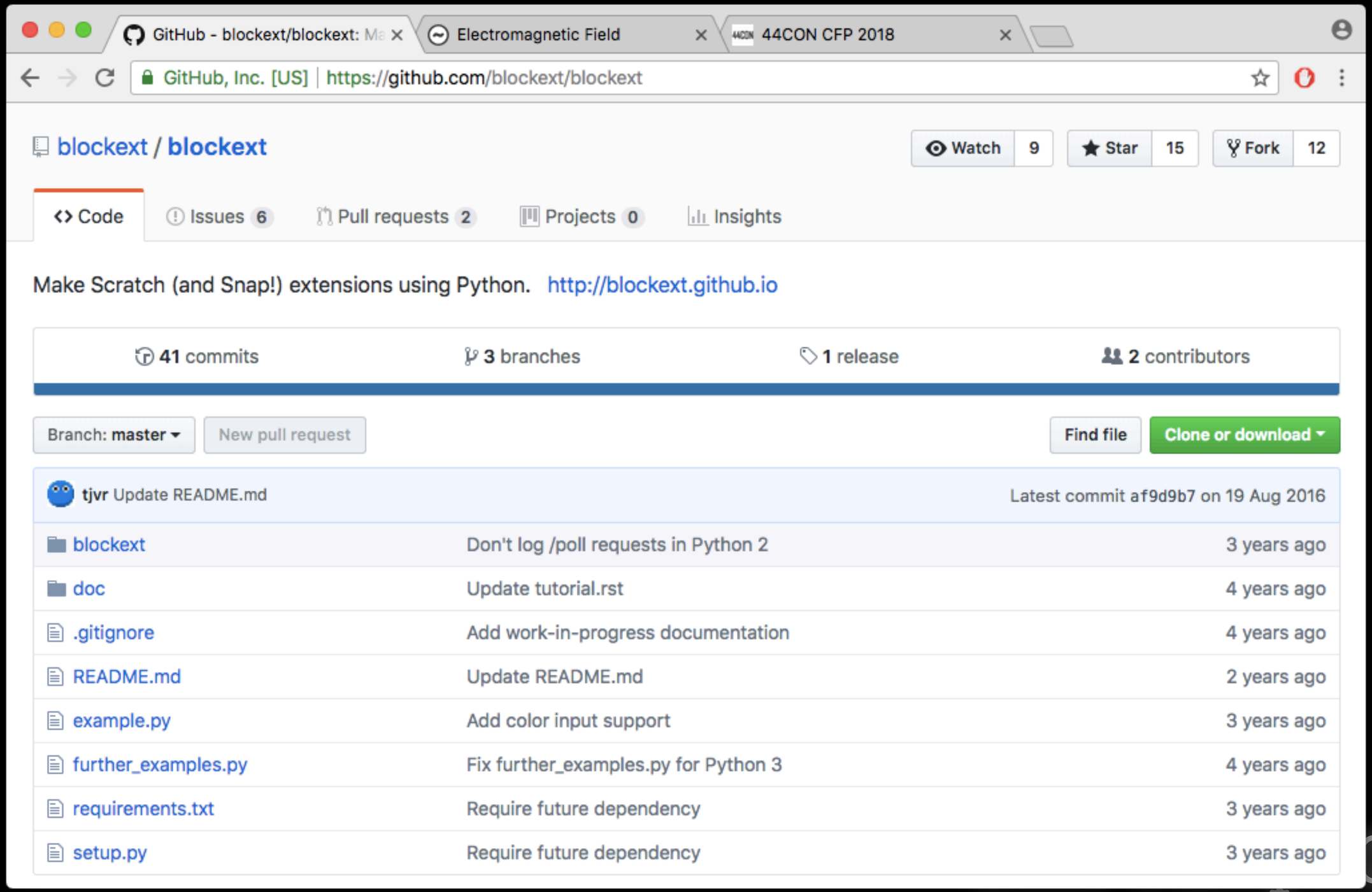
- Procedures:
  - Open TCP socket
  - Write data to socket
  - Read data from socket
- Variables
  - Socket state
  - Last data buffer read from socket

Procedures

Exposed variables

blockext
python module

EMFCamp2018


GitHub - blockext/blockext: Ma
Electromagnetic Field
44CON CFP 2018
GitHub, Inc. [US] | https://github.com/blockext/blockext
blockext / blockext
Watch 9
Star 15
Fork 12
Code
Issues 6
Pull requests 2
Projects 0
Insights
Make Scratch (and Snap!) extensions using Python. http://blockext.github.io
41 commits
3 branches
1 release
2 contributors
Branch: master
New pull request
Find file
Clone or download
tjvr Update README.md
Latest commit af9d9b7 on 19 Aug 2016
blockext Don't log /poll requests in Python 2 3 years ago
doc Update tutorial.rst 4 years ago
.gitignore Add work-in-progress documentation 4 years ago
README.md Update README.md 2 years ago
example.py Add color input support 3 years ago
further_examples.py Fix further_examples.py for Python 3 4 years ago
requirements.txt Require future dependency 3 years ago
setup.py Require future dependency 3 years ago

Tutorial

To add an extension block to Scratch, you need to do two things. First, you write the Python code that implements the block's functionality. Then, you need to tell Scratch about the block, so it knows how to use it.

In Blockext, those two steps are separate:

- First, you define a class. This is an ordinary Python class where each method contains the code for an extension block.
- Next, you create a `Descriptor` object. This stores information about the extension. It tells Scratch how to connect to the extension, and the list of blocks that Scratch can use.

Finally, you bring them together in an `Extension` object.

Let's see an example!

```
from blockext import *

class Tutorial:
    def __init__(self):
        self.light = False

    def do_toggle_light(self, times):
        for i in range(times):
            self.light = not self.light
```

```
#!/usr/bin/python
from blockext import *

class MyClass:
    <variables>
    <procedures>
descriptor = Descriptor(name, port, blocks, menus)

extension = Extension(MyClass, descriptor)
if __name__ == '__main__':
    extension.run_forever(debug=True)
```

EMFCamp2018

‘Say’ example

‘Say’ example

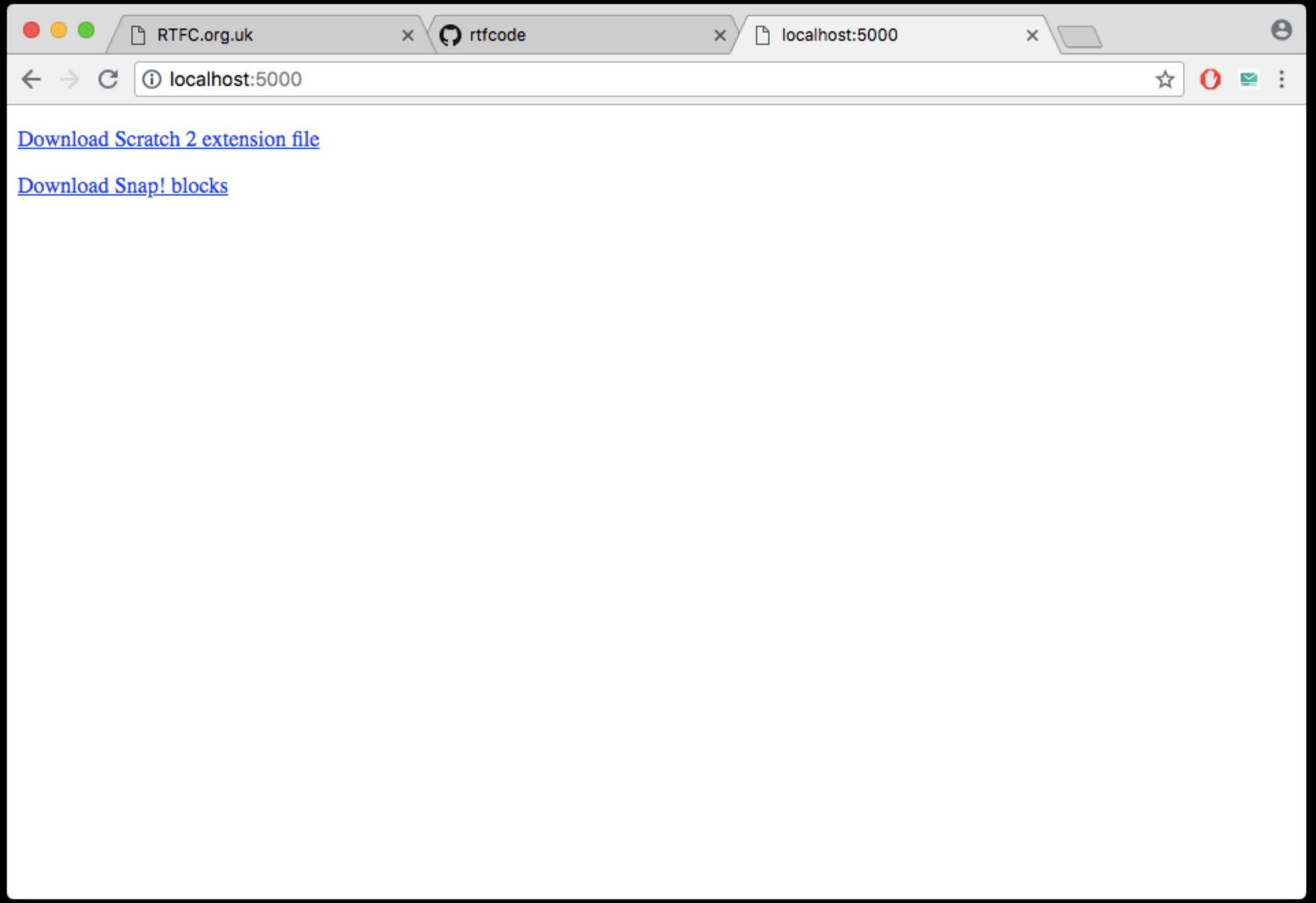
```
class SSay:


    def say(self, statement, voice):
        if voice == "":
            voice = "Alex"
        os.system("say -v " + voice + " " + statement)
```

‘Say’ example

```
descriptor = Descriptor(
    name = "Scratch Say",
    port = 5000,
    blocks = [
        Block('say', 'command', 'say %s with voice %m.voices’,
            defaults=[“hello", "Alex"]),
    ],
    menus = dict(
        voices = subprocess.check_output('say -v ? | grep en_ | cut -d" " -f1’,
            shell=True).split()
    ),
)
```

‘Say’ example

```
#!/usr/bin/python

from blockext import *
import os

class SSay:

    def say(self, statement, voice):
        if voice == '':
            voice = "Alex"
        os.system("say -v " + voice + " " +
statement)

descriptor = Descriptor(
    name = "Scratch Say",
    port = 5000,
    blocks = [
        Block('say', ‘command',
            'say %s with voice %m.voices’,
            defaults=["hello", "Alex"]),
    ],
    menus = dict(
        voices = subprocess.check_output(
            'say -v ? | grep en_ | cut -d" " -f1’,
            shell=True).split()
    ),
)

extension = Extension(SSay, descriptor)
if __name__ == '__main__':
    extension.run_forever(debug=True)
```



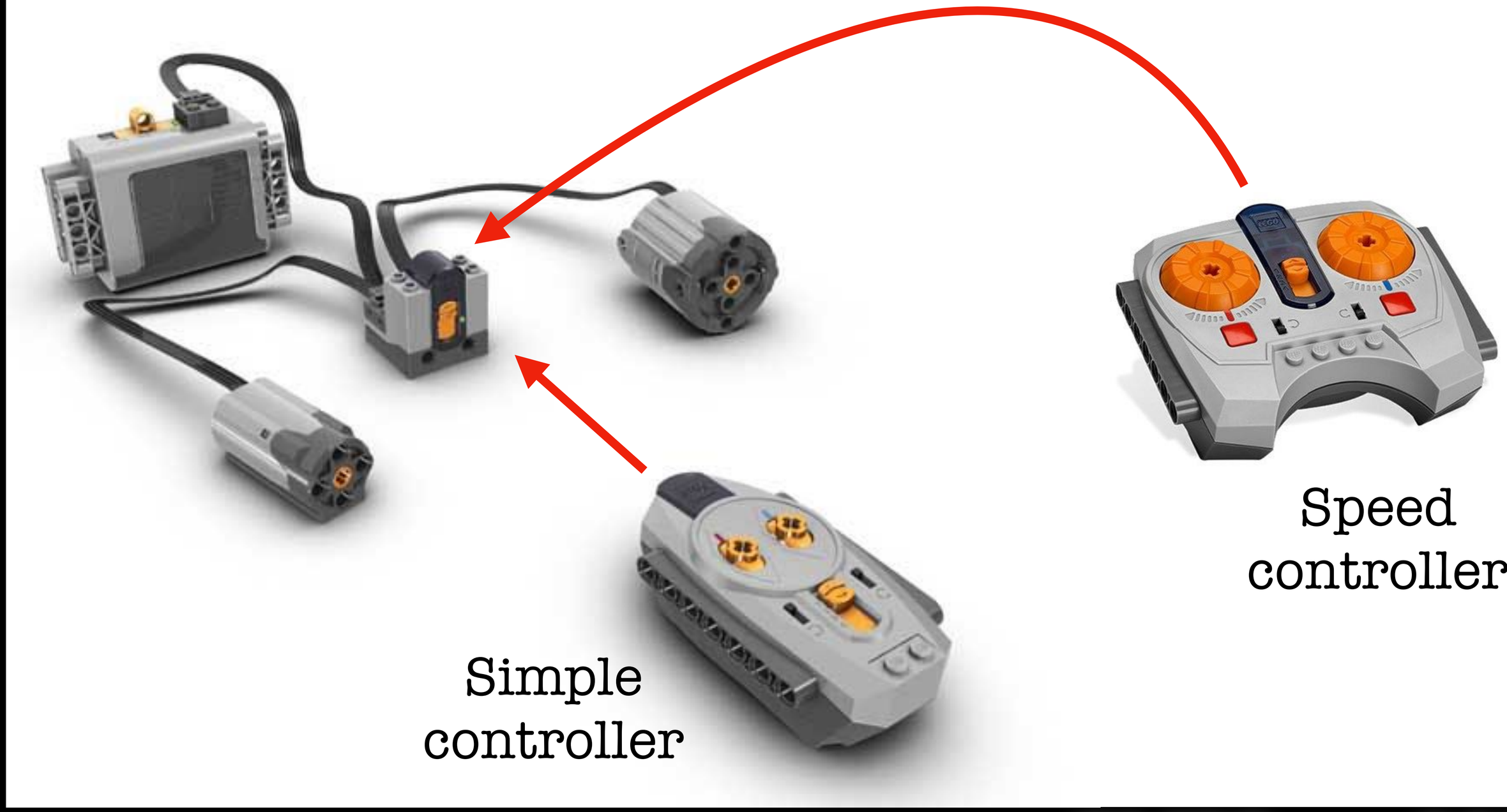
EMFCamp2018

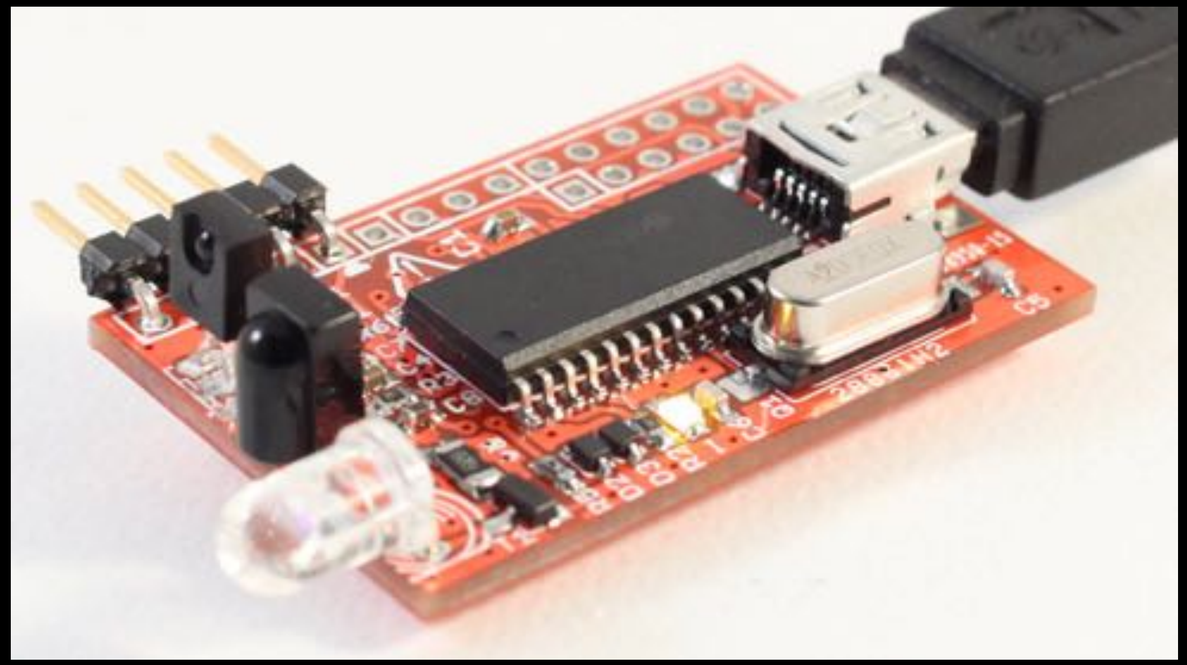
‘Say’ demo

EMFCamp2018

How do Lego Power Functions work?

EMFCamp2018


Speed
controller
Simple
controller



- IRToy
- Dangerous Prototypes
- Transmit / Receive infrared
- USB interface to computer

Sample with IRToy

```
#!/bin/python2

import serial
import sys


p = serial.Serial('/dev/ttyACM0',
baudrate=115200, timeout=2.0)
if not p:
    print 'Failed to open serial port'
    sys.exit(1)
```

```
p.write(b'\x00\x00\x00\x00\x00')
p.write(b's')
version = p.read(3)
#print version

data = p.read(512)
while data:
    sys.stdout.write(data)
    data = p.read(512)


p.close()
```

EMFCamp2018

Red button Up, Channel 1

```
00000000  00 06 00 32 00 07 00 1b  00 06 00 0e 00 05 00 0f  |...2............|
00000010  00 05 00 0f 00 05 00 0f  00 05 00 0f 00 05 00 0f  |................|
00000020  00 05 00 1d 00 05 00 0f  00 05 00 0f 00 05 00 0f  |................|
00000030  00 05 00 1d 00 05 00 0f  00 05 00 1d 00 05 00 1d  |................|
00000040  00 05 00 1c 00 05 0b d8  00 07 00 32 00 07 00 1b  |...........2....|
00000050  00 07 00 0d 00 06 00 0d  00 05 00 0f 00 06 00 0d  |................|
00000060  00 06 00 0e 00 06 00 0e  00 06 00 1b 00 06 00 0e  |................|
00000070  00 06 00 0d 00 06 00 0e  00 05 00 1d 00 06 00 0e  |................|
00000080  00 06 00 1b 00 06 00 1b  00 06 00 1a 00 07 0b b9  |................|
00000090  00 07 00 32 00 07 00 1b  00 05 00 0f 00 05 00 0f  |...2............|
000000a0  00 05 00 0f 00 05 00 0f  00 05 00 0f 00 05 00 0f  |................|
000000b0  00 05 00 1d 00 05 00 0f  00 05 00 0f 00 05 00 0f  |................|
000000c0  00 05 00 1d 00 05 00 0f  00 05 00 1d 00 05 00 1d  |................|
000000d0  00 05 00 1c 00 05 12 ba  00 07 00 32 00 07 00 1b  |...........2....|
000000e0  00 07 00 0d 00 05 00 0f  00 05 00 0f 00 05 00 0f  |................|
000000f0  00 05 00 0f 00 05 00 0f  00 05 00 1d 00 05 00 0f  |................|
```

EMFCamp2018

Parsing the raw IR data

```
#!/bin/python2

import sys

if len(sys.argv) < 2:
    print 'extract.py file'
    print 'extracts every fourth
byte from input stream'
    print 'to a maximum of 18 bytes'
    sys.exit(0)

f = open(sys.argv[1], "rb")
```

```
for j in range(0,8):
    data = f.read(4)
    for i in range(0,16):
        data = f.read(4)
       if ord(data[3]) < 0x14:
            print "0",
        else:
            print "1",
    data = f.read(4)
    print
```

EMFCamp2018

Parsed raw data

1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 - Red **Up** Channel 1

1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 - Red **Up** Channel 1

1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 - Red **Up** Channel 1

1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 - Red **Up** Channel 1

0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 - Red **Off** Channel 1

0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 - Red **Off** Channel 1

0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 - Red **Off** Channel 1

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 - no data

Parsed raw data

```
1000  0001  0001  0111  - Red Up Channel 1
1000  0001  0001  0111  - Red Up Channel 1
1000  0001  0001  0111  - Red Up Channel 1
1000  0001  0001  0111  - Red Up Channel 1
0000  0001  0000  1110  - Red Off Channel 1
0000  0001  0000  1110  - Red Off Channel 1
0000  0001  0000  1110  - Red Off Channel 1
0000  0000  0000  0000  - no data
```

Decoding the data - simple

1000 0001 0001 0111 - Red **Up** Channel 1

- **Nibble 1**
- First 2 bits are 10 if 1 button pushed and 00 otherwise
- Second 2 bits are channel number, 00 to 11 = 0-3
- Channels are numbered 0-3 in protocol, and 1-4 on equipment

Decoding the data - simple

`1000 **0001** 0001 0111` - Red **Up** Channel 1

- **Nibble 2**
- Is always 0001 in the simple protocol

Decoding the data - simple

1000 0001 **0001** 0111 - Red **Up** Channel 1

- **Nibble 3**
- First bit is Blue Down state, 0 = off, 1 = on
- Second bit is Blue Up state
- Third bit is Red Down state
- Fourth bit is Red Up state

Decoding the data - simple

1000 0001 0001 **0111** - Red **Up** Channel 1

- **Nibble 4**
- Checksum
- Nibble1 XOR Nibble2 XOR Nibble3 XOR 1111
- 1000 XOR 0001 XOR 0001 XOR 1111 = **0111**
- Nibble1 XOR Nibble2 XOR Nibble3 XOR Nibble4 = 1111

Decoding the data - simple

0000 0001 0000 1110 - Red **Off** Channel 1

- First nibble - No buttons pushed, channel 00
- Second nibble - Always 0001
- Third nibble - All buttons in off state
- Fourth nibble - Checksum

Speed controller raw data

```
1000  0110  0100  0101  - Red inc channel 1
1000  0110  0100  0101  - Red inc channel 1
1000  0110  0100  0101  - Red inc channel 1
0000  0110  0100  1101  - Red inc channel 1
0000  0110  0100  1101  - Red inc channel 1
0000  0110  0100  1101  - Red inc channel 1
1000  0110  0100  0101  - Red inc channel 1
```

Decoding the data - speed

1000 0110 0100 0101 - Red inc channel 1

- **Nibble 1**
- First bit is change indicator; flips on new message, 1-0-1-0-1-0-1-0. Allows for message redundancy
- Second bit is always 0
- Last 2 bits are channel number, 00 to 11 = 0-3
- Channels are numbered 0-3 in protocol, and 1-4 on equipment

Decoding the data - speed

1000 **0110** 0100 0101 - Red inc channel 1

- **Nibble 2**
- First two bits are always 01
- Third bit is speed/stop - 1 for speed message and 0 for stop message
- Fourth bit is output colour - 1 for blue and 0 for red

Decoding the data - speed

1000 0110 **0100** 0101 - Red inc channel 1

- **Nibble 3**
- First two bits are speed/stop indicator - 01 for speed change and 10 for stop message
- Third bit is always 0
- Fourth bit is controller direction - 0 for clockwise and 1 for anti-clockwise

Decoding the data - speed

1000 0110 0100 **0101** - Red inc channel 1

- **Nibble 4**
- Checksum
- Nibble1 XOR Nibble2 XOR Nibble3 XOR 1111
- 1000 XOR 0110 XOR 0100 XOR 1111 = **0101**
- Nibble1 XOR Nibble2 XOR Nibble3 XOR Nibble4 = 1111

Decoding the data - speed

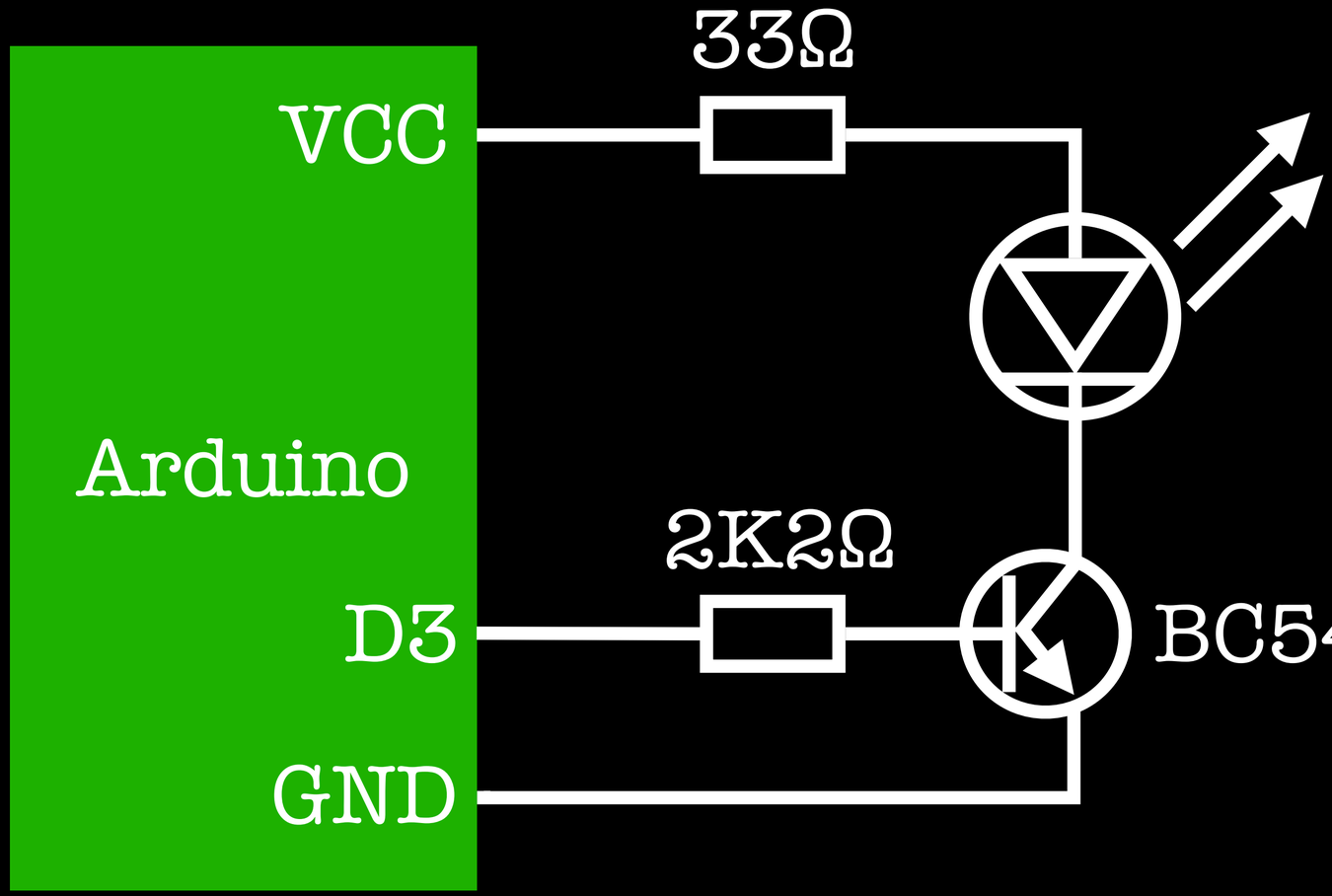
1 0 0 0 0 1 1 0 0 1 0 0 0 1 0 1 - Red inc channel 1

- First nibble - Change indicator 1, channel 00
- Second nibble - Speed change, red output
- Third nibble - Speed change, clockwise
- Fourth nibble - Checksum

Transmitting
Infrared messages

EMFCamp2018

Arduino transmitter



‘High power’
infrared LED
5mm, 940nm
<=100mA

EMFCamp2018

Arduino transmitter

- Single-byte protocol over serial
- Converts to 16bit Lego Power message
- Transmits using IRremote library

csBlueChip's SerialControl library

- https://github.com/csBlueChip/SerialControl
- Provides a serial message library that can be easily plugged into your code
- Human-readable/interaction interface
- Define commands and variable parameters
- Calls specified functions directly

Protocol choices

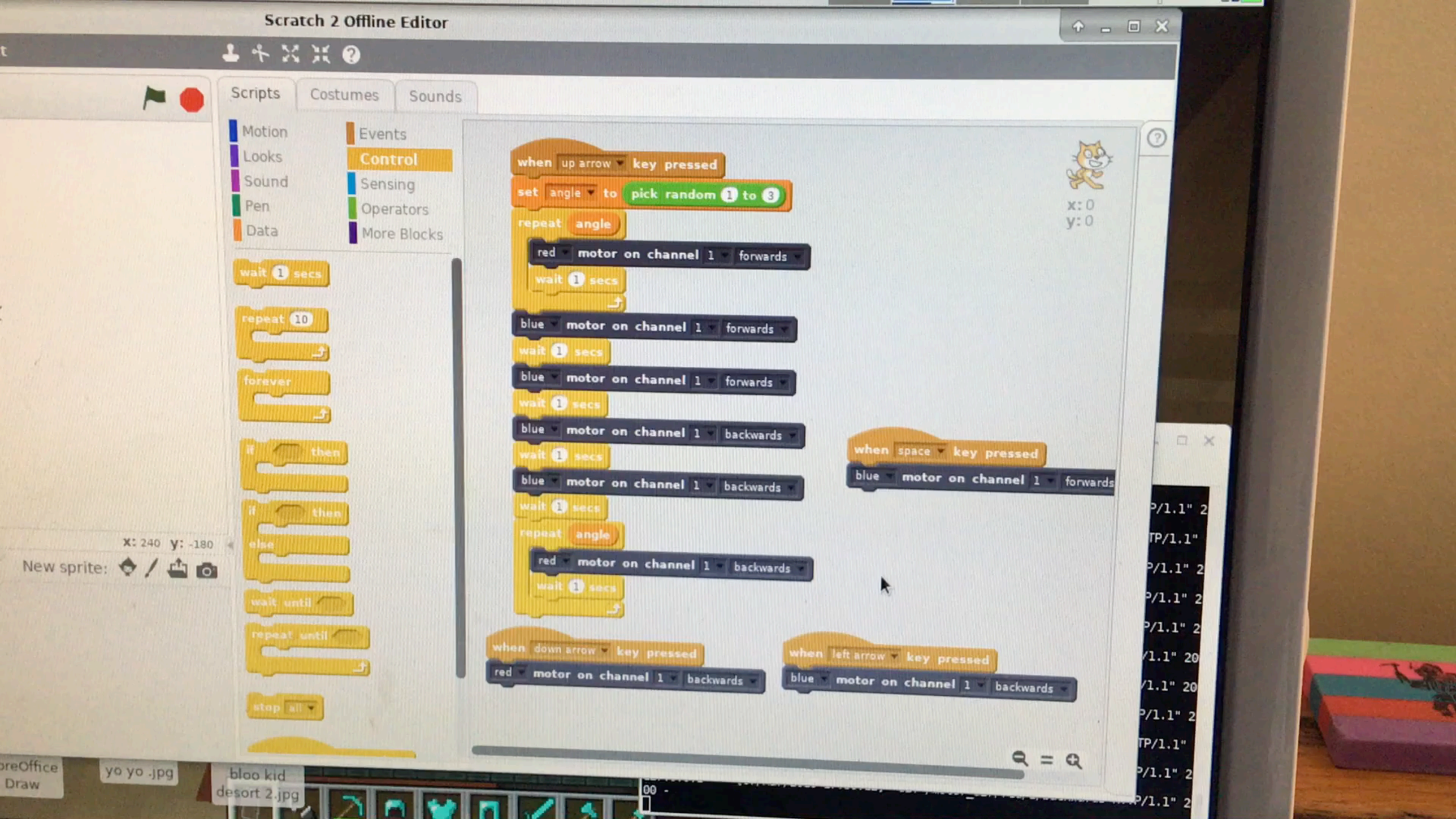
	Single-byte protocol	SerialControl-based protocol	Home-grown multi-byte protocol
Complexity	Simple	Simple	Complex
Flexibility	Huge	Huge	Varies
Maintainability	Easy	Easy	Tricky
Reliability	High	High	Low-medium

Single-byte protocol

- Not used
- Colour (red=1, blue=0)
- Channel (0-3)
- Direction for simple proto (01=up, 10=down) or 11
- Direction for speed proto (01=clock, 10=anti, 00=stop) or 11

Demo Lego Power

EMFCamp2018


Scratch 2 Offline Editor
Scripts
Costumes
Sounds
Motion
Looks
Sound
Pen
Data
Events
Control
Sensing
Operators
More Blocks
when up arrow key pressed
set angle to pick random 1 to 3
repeat angle
red motor on channel 1 forwards
wait 1 secs
blue motor on channel 1 forwards
wait 1 secs
blue motor on channel 1 forwards
wait 1 secs
blue motor on channel 1 backwards
wait 1 secs
blue motor on channel 1 backwards
wait 1 secs
repeat angle
red motor on channel 1 backwards
wait 1 secs
when space key pressed
blue motor on channel 1 forwards
when down arrow key pressed
red motor on channel 1 backwards
when left arrow key pressed
blue motor on channel 1 backwards
x: 0
y: 0
x: 240 y: -180
New sprite:
wait 1 secs
repeat 10
forever
if then
if then else
wait until
repeat until
stop all
yo yo .jpg
bloo kid desort 2.jpg

A CARD GAME
KITTENS
POLICE


EXPLODING KITTENS
A CARD GAME
EXPLODING KITTENS
POLICE


POLICE

Demo Lego Power

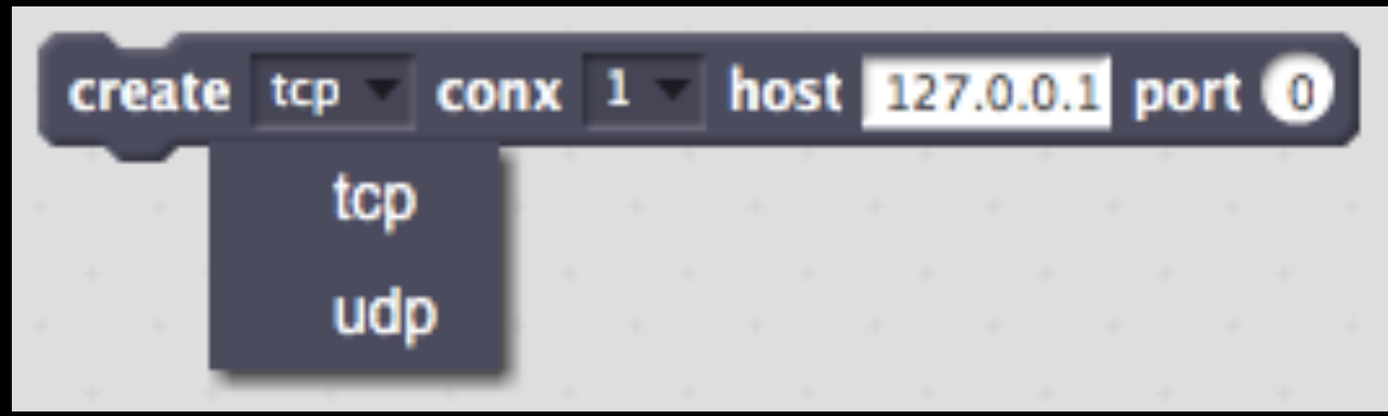
EMFCamp2018

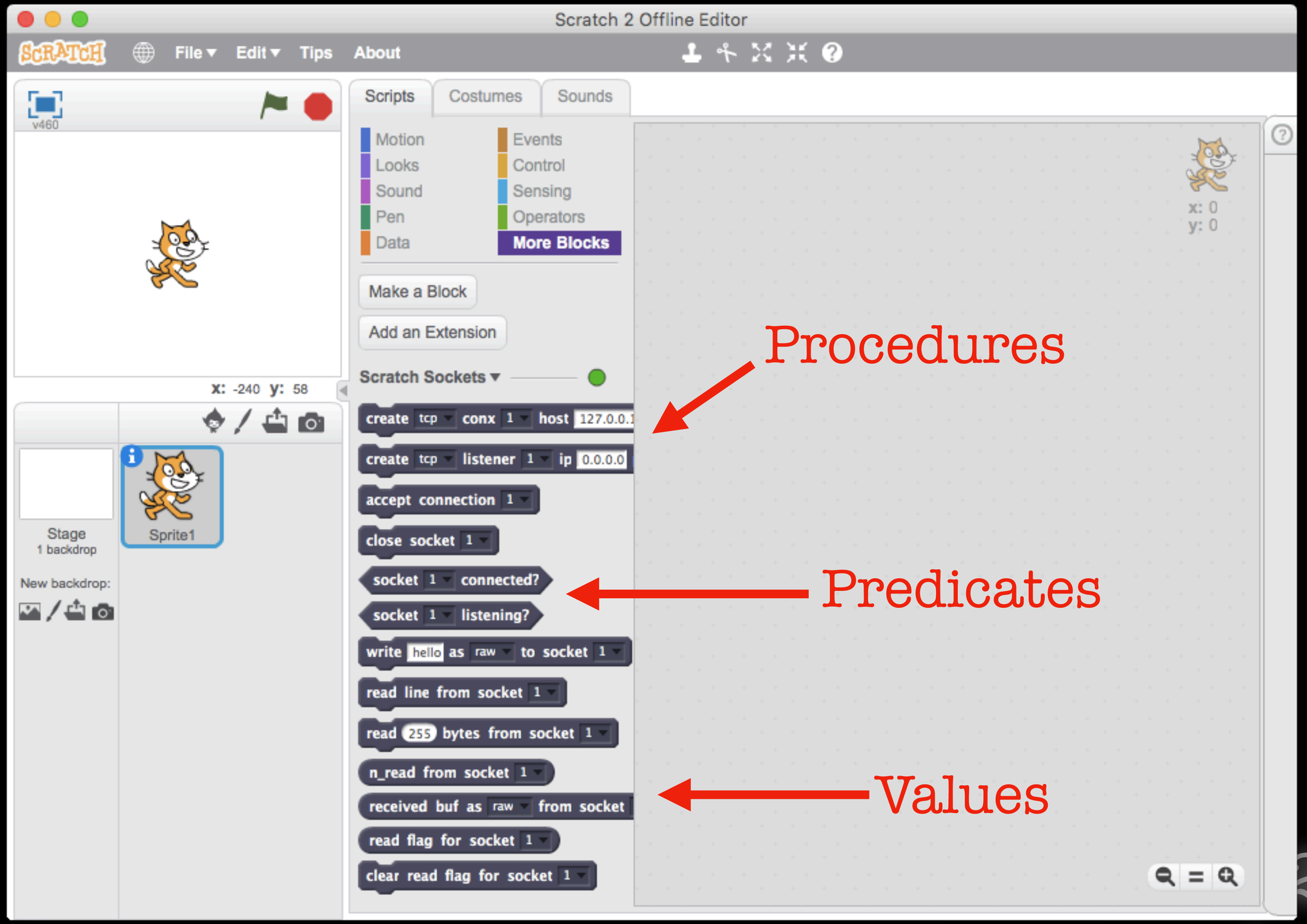
Haxoring with Scratch

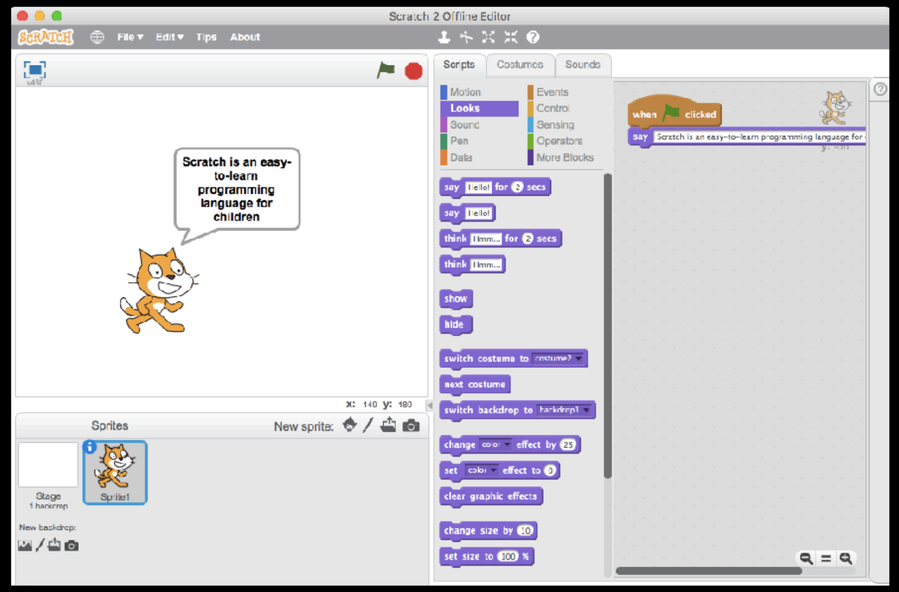
EMFCamp2018

```
class SSocket:
    def __init__(self):
    def _on_reset(self):
    def create_socket(self, proto, sock, host, port):
    def write_socket(self, data, type, sock):
    def recv_socket(self, length, sock):
    def readline_socket(self, sock):
    def n_read(self, sock):
    def readbuf(self, type, sock):
    def close_socket(self, sock):
```

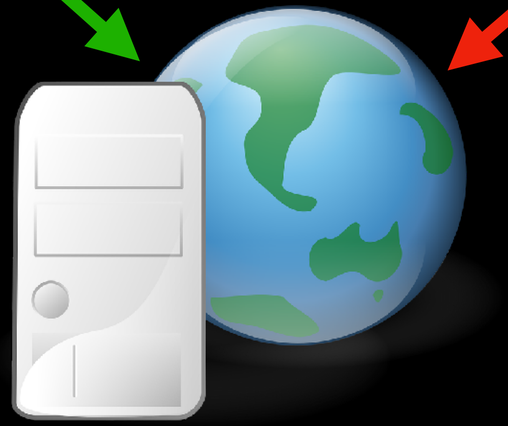
```
descriptor = Descriptor(
    name = "Scratch Sockets",
    port = 5000,
    blocks = [
        Block('create_socket',
            'command',
            'create %m.proto conx %m.sockno host %s port %n',
            defaults=["tcp", 1, "127.0.0.1", 0]  ),
        ...
    ],
```




Scratch 2 Offline Editor
File
Edit
Tips
About
Scripts
Costumes
Sounds
Motion
Looks
Sound
Pen
Data
Events
Control
Sensing
Operators
More Blocks
Make a Block
Add an Extension
Scratch Sockets
create tcp conx 1 host 127.0.0.1
create tcp listener 1 ip 0.0.0.0
accept connection 1
close socket 1
socket 1 connected?
socket 1 listening?
write hello as raw to socket 1
read line from socket 1
read 255 bytes from socket 1
n_read from socket 1
received buf as raw from socket
read flag for socket 1
clear read flag for socket 1
x: -240 y: 58
x: 0
y: 0
Stage
1 backdrop
Sprite1
New backdrop:
Procedures
Predicates
Values
2018

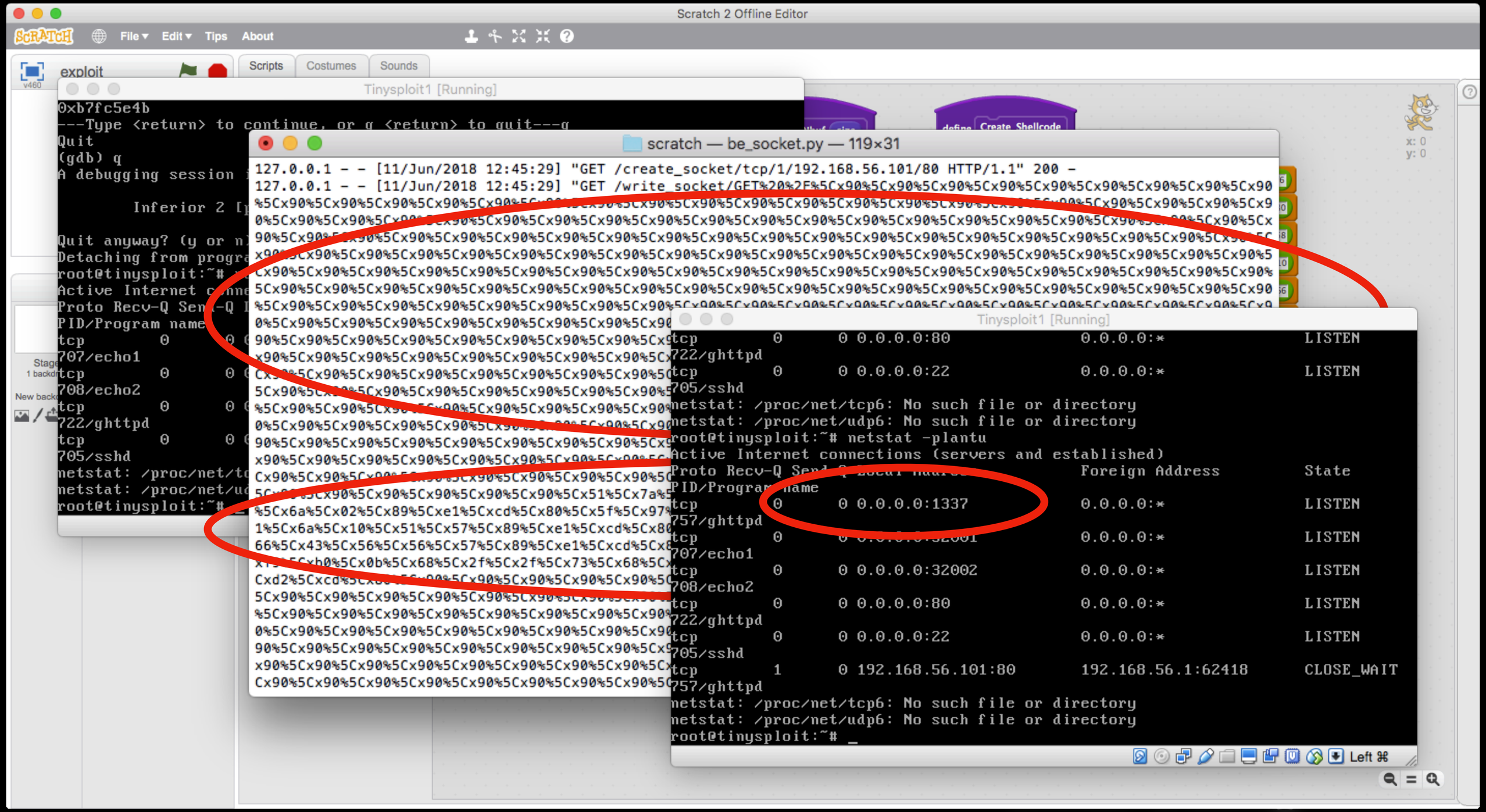


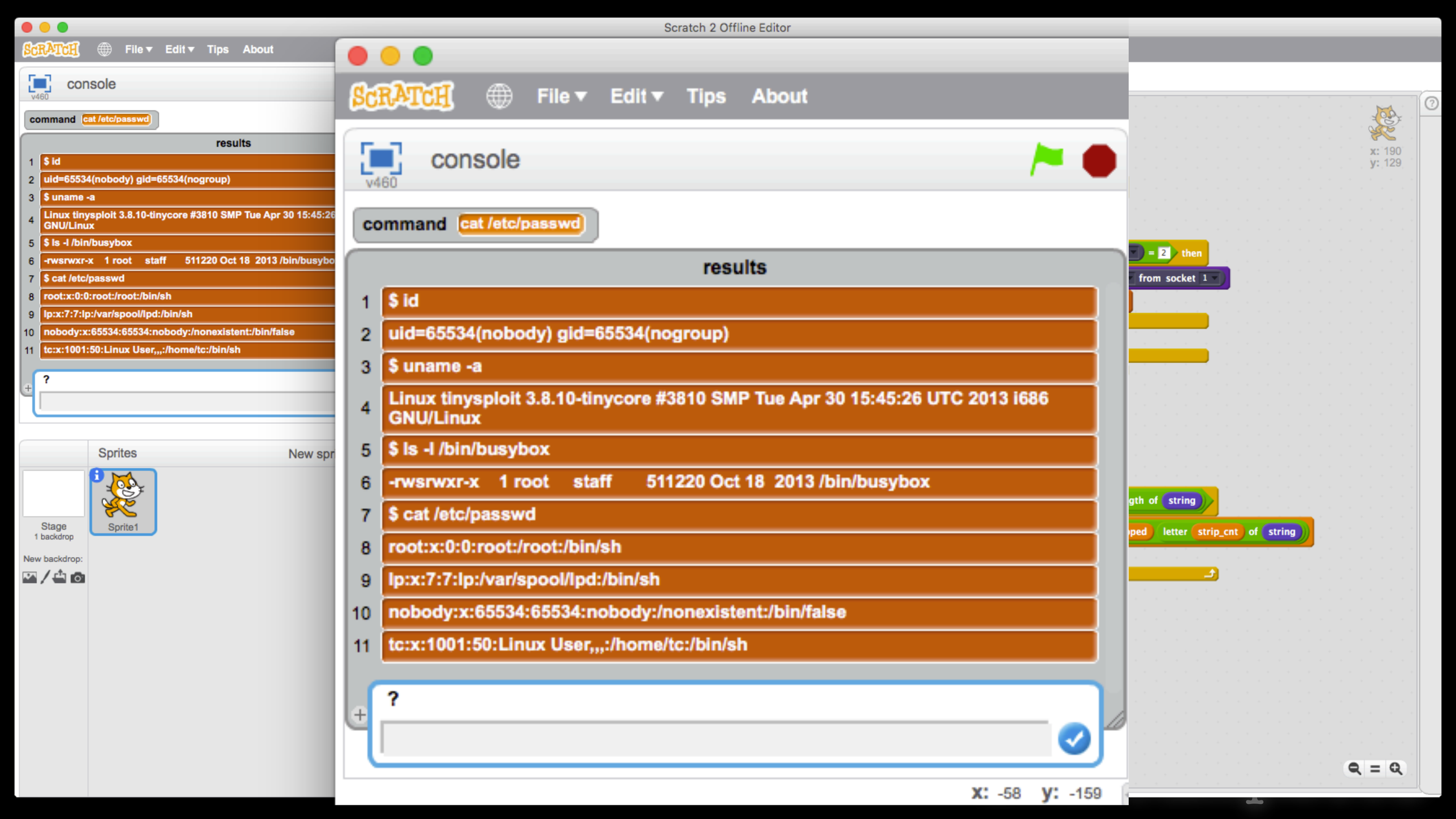
Web server on ‘tinysploit’ virtual machine



Web server on localhost (blockext)

EMFCamp2018


Scratch 2 Offline Editor
File
Edit
Tips
About
exploit
Scripts
Costumes
Sounds
Tinysploit1 [Running]
0xb7fc5e4b
---Type <return> to continue, or q <return> to quit---q
Quit
(gdb) q
A debugging session
Inferior 2 [
Quit anyway? (y or n
Detaching from progra
root@tinysploit:~#
Active Internet c
Proto Recv-Q Sen -Q
PID/Program name
tcp 0 0
707/echo1
tcp 0 0
708/echo2
tcp 0 0
722/ghttpd
tcp 0 0
705/sshd
netstat: /proc/net/tc
netstat: /proc/net/uc
root@tinysploit:~#
scratch — be_socket.py — 119×31
127.0.0.1 - - [11/Jun/2018 12:45:29] "GET /create_socket/tcp/1/192.168.56.101/80 HTTP/1.1" 200 -
127.0.0.1 - - [11/Jun/2018 12:45:29] "GET /write_socket/GET%20%2F%5Cx90%5Cx90%5Cx90%5Cx90%5Cx90%5Cx90%5Cx90%5Cx90%5Cx90%5Cx90%5Cx90
Tinysploit1 [Running]
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN
722/ghttpd
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN
705/sshd
netstat: /proc/net/tcp6: No such file or directory
netstat: /proc/net/udp6: No such file or directory
root@tinysploit:~# netstat -plantu
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
PID/Program name
tcp 0 0 0.0.0.0:1337 0.0.0.0:* LISTEN
757/ghttpd
tcp 0 0 0.0.0.0:32001 0.0.0.0:* LISTEN
707/echo1
tcp 0 0 0.0.0.0:32002 0.0.0.0:* LISTEN
708/echo2
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN
722/ghttpd
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN
705/sshd
tcp 1 0 192.168.56.101:80 192.168.56.1:62418 CLOSE_WAIT
757/ghttpd
netstat: /proc/net/tcp6: No such file or directory
netstat: /proc/net/udp6: No such file or directory
root@tinysploit:~#
Left ⌘


Scratch 2 Offline Editor
SCRATCH
File
Edit
Tips
About
console
v460
command cat /etc/passwd
results
1 $ id
2 uid=65534(nobody) gid=65534(nogroup)
3 $ uname -a
4 Linux tinysploit 3.8.10-tinycore #3810 SMP Tue Apr 30 15:45:26 UTC 2013 i686 GNU/Linux
5 $ ls -l /bin/busybox
6 -rwsrwxr-x 1 root staff 511220 Oct 18 2013 /bin/busybox
7 $ cat /etc/passwd
8 root:x:0:0:root:/root:/bin/sh
9 lp:x:7:7:lp:/var/spool/lpd:/bin/sh
10 nobody:x:65534:65534:nobody:/nonexistent:/bin/false
11 tc:x:1001:50:Linux User,,,:/home/tc:/bin/sh
?
Sprites
New sprite
Stage
1 backdrop
Sprite1
New backdrop:
x: -58 y: -159
x: 190
y: 129
then
from socket 1
length of string
letter strip_cnt of string

Conclusions

EMFCamp2018

Conclusions

- Scratch is an ‘interesting’ language
  - No functions, only procedures
- Extensions can provide power and reach
  - Written in python
  - Trivial to simple in coding terms
- Makes Scratch more fun!
- Pathway from Scratch to Python and beyond...

rtfcode@gmail.com

@kevsheldrake

Any questions?

https://github.com/rtfcode

EMFCamp2018