Sense of Connection Workshop 2 - The Remote'ability

Using the Circuit Playground for a remote sense of connection. 2020.10.13

Alan Grover, Technical Assistant for Unravel The Code, MICA. he/him/his
Victor Ekanem, Individual Community Partner, MICA. he/him/his
Frederik De Bleser, Researcher/Professor in Digital Media, Sint Lucas Antwerpen, he/him/his
Nan Wang, Teacher, WdKA (http://nanwang.org/), she/her/hers

References

This document
>   https://github.com/paulmirel/telepresence_of_touch/archive/master.zip
>   Workshop_2_Documents/Sense of Connection Workshop 2 - The Remote'ability.pdf

Discord
>   https://discord.com/channels/760491586533589022/760491586533589026

Our wiki
>   http://ps.wdka.nl/digitalcraft/index.php/Unrvl2020

Circuit Playground devices
>   https://learn.adafruit.com/adafruit-circuit-playground-express
>   https://learn.adafruit.com/adafruit-circuit-playground-bluefruit

The "cp" interface to the circuit-playground:
>   https://circuitpython.readthedocs.io/projects/circuitplayground/en/latest/api.html

The full library:
>   https://circuitpython.readthedocs.io/en/latest/shared-bindings/index.html

Processing documentation:
>   https://processing.org/reference/

MQTT
>   http://mqtt.org

Prerequisites

1. Some kind of international English
2. Participation in Workshop 1, working Mu Editor, downloaded zip
3. Assignment to a 2 or 3 person group for Workshop 2
4. Logged into the UNRVL Discord. Introduce yourself to your workshop group.
    a. Camera, and microphone that works with Zoom *and* Discord.
    b. Use the introduction to check microphone/camera
5. Circuit Playground connected to your computer
6. (New) Download and install the Processing IDE, version 3
    a. https://processing.org/download/
    b. Check your version, we need version 3
    c. Run at least one example
7. (New) Download the "github zip" again, because I probably had to update something since Paul's workshop.
    a. https://github.com/paulmirel/telepresence_of_touch/archive/master.zip
8. Open *this* document (from the zip): Workshop_2_Documents/Sense of Connection Workshop 2 - The Remote'ability.pdf

Orientation

The workshop will be presented in an American Mid-Western English, with an interlude in whatever they call the English they teach in Belgium. Vic speaks an American East-Coast English. Nan also speaks English.

Zoom is where I'll be presenting the workshop. Use the "raise my hand" feature to interrupt and ask for clarifications. If something doesn't seem clear to you, it will probably help others to hear you ask about it. For example, I'm too familiar with terminology in programming, ask what words mean!

We'll be using Discord when you need help with each exercise. You should try asking your group members for help first. Then use the general channel on the UNRVL discord to ask for help. Vic, Frederik, (others?), and I will try to get to you.

I'm tired of typing out "circuit playground", I'll just use "CPX" or "CP".

Install MQTT lib

In processing, use the `Sketch:Import Library:Add Library` menu to add mqtt
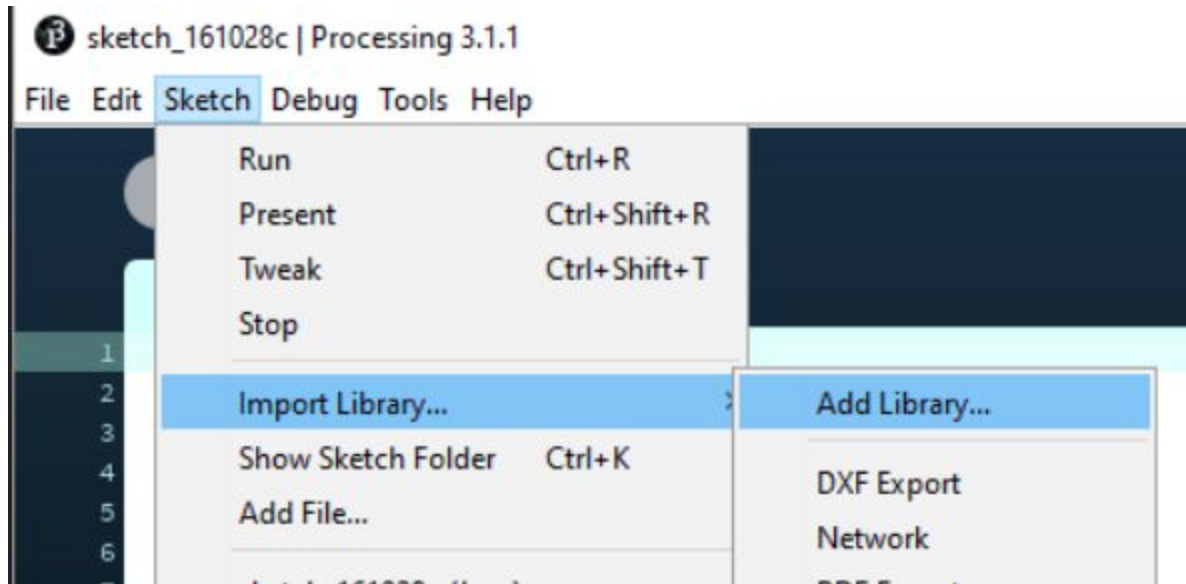
Then search for mqtt and install:

Interlude, The Metaphysics of Engineers

My favorite phrase, "Noooooo…".

"No" means "wait a minute while I think about that." (compare with "niet mogelijk") . Focus vs. divergence. Some do have a sense of humour.

Recapitulation

Using Paul's Step xxx code, Mu Editor, is your CPX working? Touch -> LED.
Workshop_1_Documents/Telepresence_of_Touch_W1_Local.pdf, page 1..2.
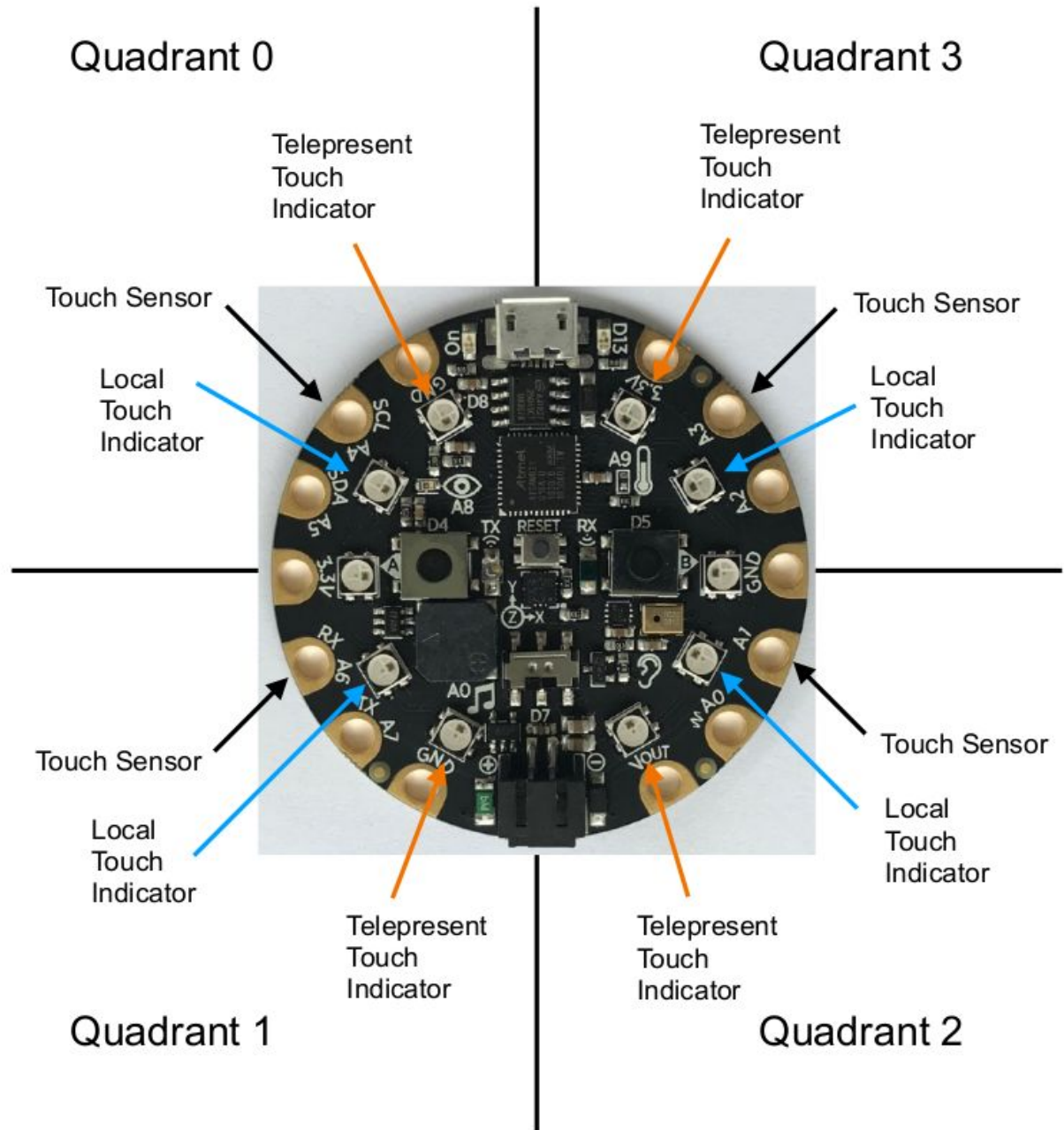
- Raise hand in zoom to clarify presentation
- On Discord, ask your group, and general, for help doing the exercise

Elaboration

More complex software!
Concepts: touch pads, neopixels (LEDs), serial pane, heartbeat.

- The Circuit Playground Bluefruit requires some extra files in `your-computer/CIRCUITPY/lib`, so leave them (from Paul's workshop)
- Make sure you don't have old code in lib/ ! No mqtt_serial.py, no every.py
  - Clean up!
- Copy contents of `telepresence_of_touch/circuit-playground/telepresence_of_touch_code_remote`
  to
  `your-computer/CIRCUITPY`
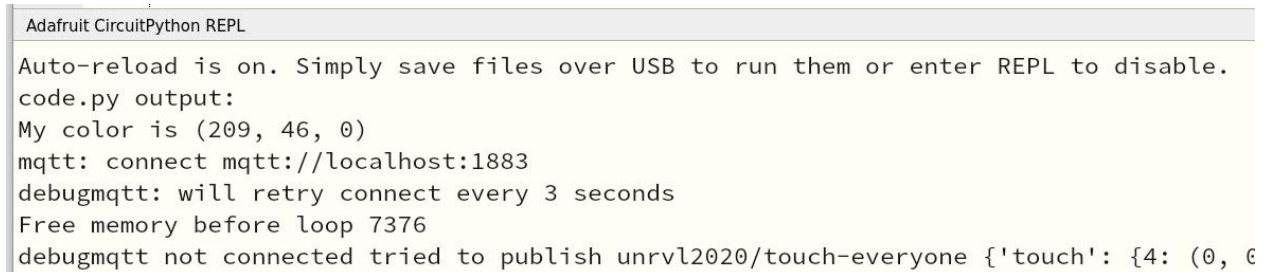 (I don't have tree view :)
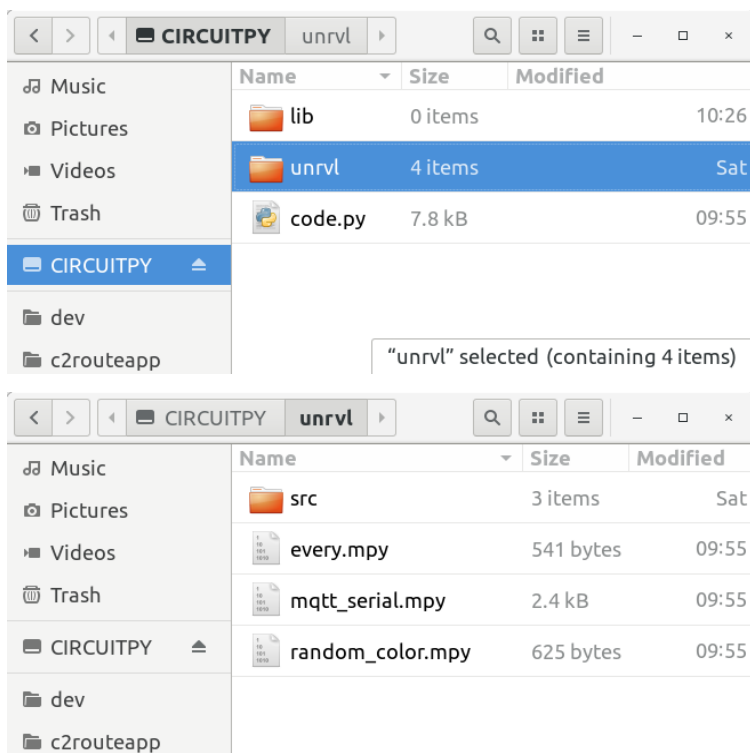
- Using Mu Editor, load your-computer/CIRCUITPY/code.py
  - Do the "control-c", "control-d" thing in the "serial" pane:

  Should show a "serial" pane at the bottom of the Mu Editor (note the "Adafruit CircuitPython REPL"):

```
11  while True:
12        if cp.touch_A4 :
13            if not LastTouched[ 0 ]: # if the touch i
Adafruit CircuitPython REPL
```

```
                                                    Adafruit  ⚙
```

- More output in serial:

```
Adafruit CircuitPython REPL

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
My color is (209, 46, 0)
mqtt: connect mqtt://localhost:1883
debugmqtt: will retry connect every 3 seconds
Free memory before loop 7376
debugmqtt not connected tried to publish unrvl2020/touch-everyone {'touch': {4: (0, 0
```

- "out of memory"
  - `MemoryError: memory allocation failed, allocating 513 bytes`
  - press the tiny little button ("reset"), try again
- On the CP, the big LED near the USB connector should light up in some color.
- The tiny red LED on the other side of the USB connector ("D13") should blink slowly. This is the "heartbeat".
- Touching the pads should light up LEDs. Note that I "moved" the LEDs a bit.
- Pictures in this document are out of date! "touch-everyone" vs "touch/everyone"...

Pick a Color, and a "Me"

We'll want some way of identifying your CP when we communicate remotely. Something like your initials,and a color.
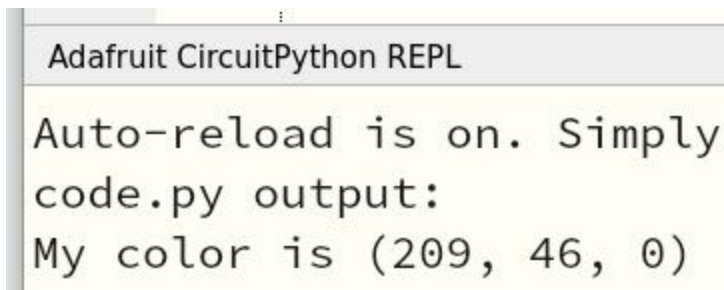
1. Edit "Me".

Pick some short nickname for yourself, like your initials (I'm "awg"). Let's stick with Latin characters (specifically, ASCII) for now (it will accept unicode, like hanzi, but won't display them usefully in Mu Edit's serial pane).

```
27  # your initials
28  Me='ANON'  # EDIT ME with your initials c
29
30  MyColor = (40,5,10) # random, or edit th
```

- Edit and save.
- It should show in the serial console
- Use Discord to help each other, and to discuss discoveries.
- Tell your group what your "initials" are.

2. Edit "MyColor"

Everytime you save the code.py, or push the CP reset button, or reconnect the CP, you'll get a random color.

```
Adafruit CircuitPython REPL

Auto-reload is on. Simply
code.py output:
My color is (209, 46, 0)
```

It is a "RGB" color, but won't look quite like a screen RGB.
- Keep hitting control-d till you get a color you like, or just start experimenting/guessing:
- Edit code.py, change `MyColor` till you like it (you only need to edit `MyColor)`

```
29
30  MyColor = None # random, or edit this
31  LocalColor = None # will be MyColor ur
32  RemoteColor = None # will be MyColor u
33
```

like:

```
30  MyColor = (40,5,10) # r
```
<- don't use that color!

- Save, and it should run it, and change to your color.
- Write down your color!
- Tell your group your color!

Interlude, The Metaphysics of Programming

Programs are for people, not computers. It's a craft.

Every bit of a program is a hypothesis: "Like this? Is this what I mean?". A program is contingent.

Evil is a technical term in programming, it means you will cause pain and suffering to yourself and others if you do that.

An Overview Of The Code

- Intro comment
- "import"
- Constants & Globals
- `def setup()`
- `def loop()`
- `def update_touch(mqtt_message):`
  - Looks almost like Paul's code
- `def handle_mqtt_message(topic, mqtt_message):`
- `lib/every.py`
- `lib/mqtt_serial.py`
-

"Every", in other words, "non-blocking" sleep

Remember "evil"? sleep() is evil. It's like stealing toys in kindergarten: it makes the other kids cry, and nobody can have any fun. What we need is something like "every 3 seconds, do this…".

So, I wrote "every", though other people have had the same idea.
- Import

```
19  from every import Every
```

- * Globals/Constant

```
44  ## HeartBeat
45  # We're using "Every":
46  # will be true "every n seconds", non-blocking time.sleep()
47  # to "flash" the built-in LED
48  HeartBeat = Every(3)
```

- Loop

```
84  def loop():
85      global FirstTime
86
87      # blink the plain LED (next to usb) slowly to indicate that we are running
88      if HeartBeat():
89          #print("HeartBeat free mermory", gc.mem_free(), time.monotonic())
90          cp.red_led = not cp.red_led # clever "toggle", aka "blink"
```

"Every Heartbeat…"
Look for other uses of "Every" in this code.
- four things "at once": heartbeat, touch, sending message, receiving a message

Interlude, Losing

https://dwarffortresswiki.org/index.php?title=DF2014:Fun&redirect=no I didn't say it was a good sense of humor.
AKA, Serial Port Contention

Processing (running/serial) and Mu-Editor (serial) ~30

There shall be only one.
The processing code I wrote uses the CP serial usb port. So does the Mu-Editor serial pane.
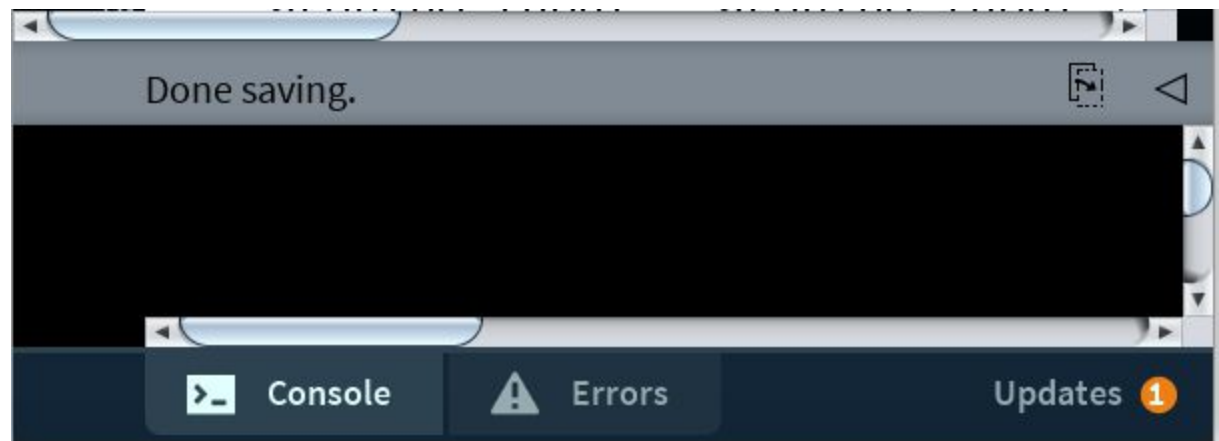Apparently this is far too difficult of a concept for computers in 2020. So:

- Close the Mu-Editor serial pane

- Open Processing
    - Open the "console" pane

      click the triangle to open:

    - Load
      telepresence_of_touch/processing/serial_mqtt_gateway/serial_mqtt_gateway.pd
      e
    - Run
    - Nice graphic-design, Alan.
    - Stuff in Processing's Console (which is the "serial" output). Looks familiar, partly.

```
> Auto-reload is on. Simply save files over USB to run them or enter REPL to
disable.
> code.py output:
> My color is (0, 141, 114)
> debugmqtt: will retry connect every 3 seconds
>> mqtt: connect mqtt://f557f2ed:e8f599a52aae3773@broker.shiftr.io
Will connect 'mqtt://f557f2ed:e8f599a52aae3773@broker.shiftr.io'
## finished connect
> Free memory before loop 7696
>> mqtt: subscribe unrvl2020/touch-everyone
> debugmqtt connected
>> mqtt: publish unrvl2020/touch-everyone {'message': 'hello', 'from': 'ANON'}
MQTT Received: unrvl2020/touch-everyone (string)  : {'message': 'hello',
'from': 'ANON'}
```

- When you save in Mu-Editor, switch to Processing and look at *its* serial console.
  - The Processing sketch has to be running.

Touch Someone Else

Finally.

1. The Processing sketch should be running...
2. Should see "connected" message `mqtt: connect mqtt://f557f2ed:e8f599a52aae3773@broker.shiftr.io`

`Started`

`/dev/ttyACM0 /dev/ttyS0 /dev/ttyS1 /dev/ttyS2 /dev/ttyS3 /dev/ttyS4 /dev/ttyS5 /dev/ttyS6 /dev/ttyS7 /dev/ttyS8 /dev/ttyS9 /dev/ttyS10 /dev/ttyS11 /dev/ttyS12 /dev/ttyS13 /dev/ttyS14 /dev/ttyS15 /dev/ttyS16 /dev/ttyS17 /dev/ttyS18 /dev/ttyS19 /dev/ttyS20 /dev/ttyS21 /dev/ttyS22 /dev/ttyS23 /dev/ttyS24 /dev/ttyS25 /dev/ttyS26 /dev/ttyS27 /dev/ttyS28 /dev/ttyS29 /dev/ttyS30 /dev/ttyS31`

`Connected to /dev/ttyACM0`

`> Traceback (most recent call last):`

`> File "code.py", line 251, in <module>`

`> File "code.py", line 107, in loop`

`> File "code.py", line 162, in update_touch`

`> File "neopixel.py", line 175, in __setitem__`

`> File "neopixel.py", line 232, in show`

`> File "neopixel.py", line 232, in <listcomp>`

`> KeyboardInterrupt:`

`> soft reboot`

`> Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.`

`> code.py output:`

`> My color is (0, 80, 175)`

`> debugmqtt: will retry connect every 3 seconds`

`>> mqtt: connect mqtt://f557f2ed:e8f599a52aae3773@broker.shiftr.io`

`Will connect 'mqtt://f557f2ed:e8f599a52aae3773@broker.shiftr.io'`

`## finished connect`

`> Free memory before loop 7200`

`> debugmqtt not connected tried to publish unrvl2020/touch/everyone {'touch': {4: (0, 0, 0), 1: (0, 0, 0), 2: (0, 0, 0), 3: (0, 0, 0)}, 'from': 'ANON'}`

`MQTT Connected, listening`

`> GOT 'mqtt: connected'`

`>> mqtt: subscribe unrvl2020/touch/everyone`

`> debugmqtt connected`

`>> mqtt: publish unrvl2020/touch/everyone {'message': 'hello', 'from': 'ANON'}`

`MQTT Received: unrvl2020/touch/everyone (string)  : {'message': 'hello', 'from': 'ANON'}`

`> GOT 'mqtt: message {"payload":"{'message': 'hello', 'from': 'ANON'}","topic":"unrvl2020/touch/everyone"}'`

3. Touch your CP
4. WOW LIGHTS. It's chaos.
5. Watch the Processing console

MQTT

The "Internet Of Things" protocol. Protocol just means "the way of doing something". In programming, it is a very strict set of rules for the doing.

https://shiftr.io/unrvl2020/touch
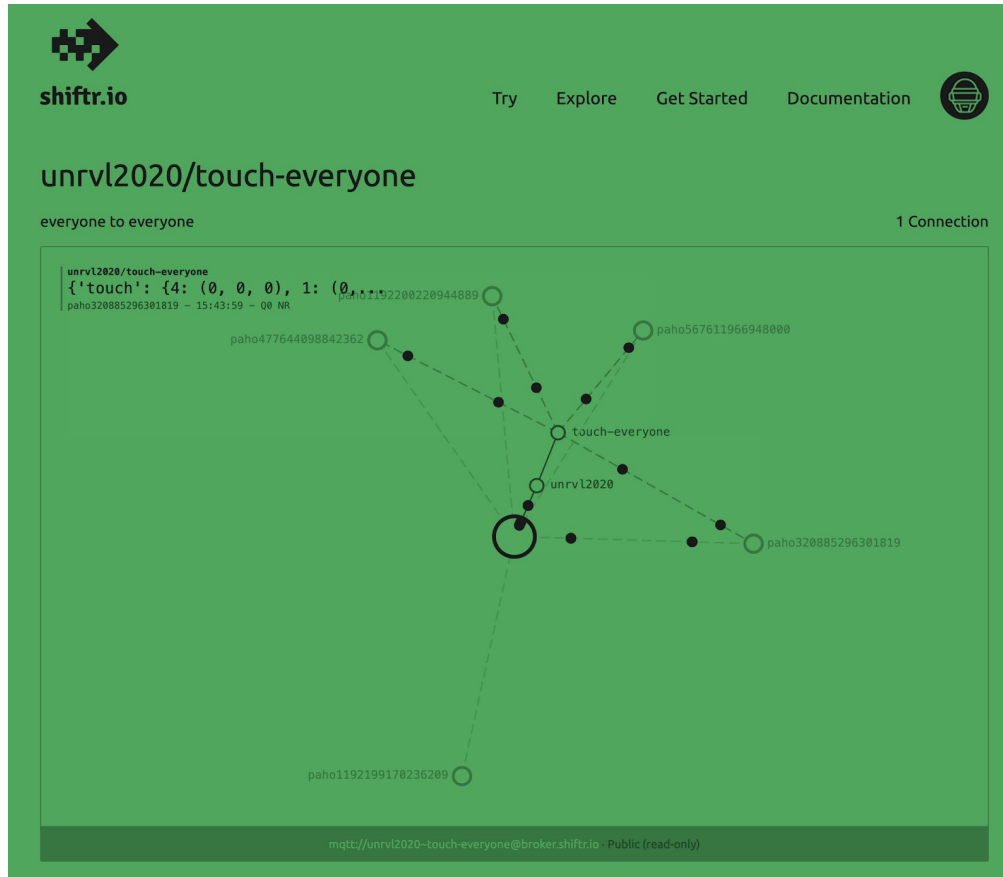
MQTT is like chat rooms for robots. It's called a publish/subscribe model. You "publish" a message to a topic, and you "subscribe" to a topic so you can get messages. But, MQTT does not say anything about the messages. Your "smart" lightbulb uses one kind of message, and your "smart" doorbell uses another. We have to define the message.

MQTT requires a server to host the topics (chat-rooms). Sounds like Discord. And, servers have rules.

We are using the server shiftr.io and the topic "unrvl2020/touch". You can have "nested" topics, maybe "unrvl2020/remote/shake/fish-group".

UNRVL - Sense of Connection
System Diagram [REMOTE]
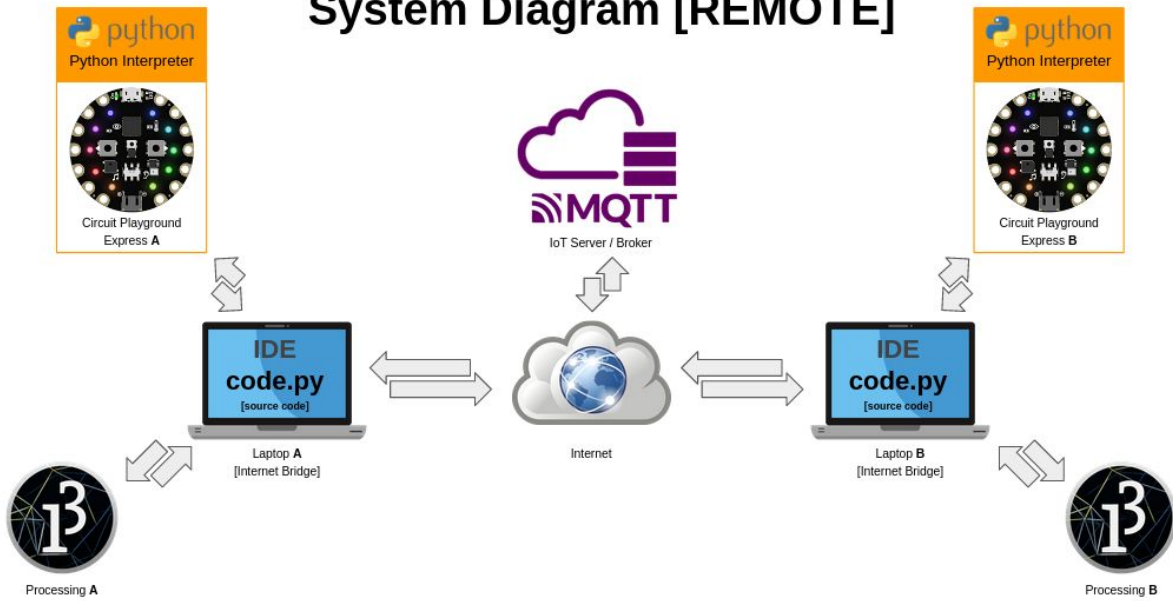
Interlude, Protocols with Frederik

Protocols are an **agreement** between two or more parties on the kind of language they will use to communicate. Just like in formal protocols, or etiquette, protocols tell you how to behave under certain conditions.

Saying hello with a handshake

To communicate over the internet, computers use a protocol called TCP/IP. To start communicating, two machines first perform a handshake*: a process of sending messages and replying to the other's messages to establish a connection:

1. Alice sends Bob a SYN message (for "synchronize") with a sequence number.
2. Bob receives the SYN message and replies with a SYN-ACK message ("synchronize-acknowledgement") with their own sequence number, and Alice's sequence number + 1.
3. Alice replies with a ACK ("acknowledgement") message with Bob's sequence number + 1. Now they can start communicating.

Protocols are arbitrary

As long as there is agreement on the messages you can invent any kind of protocol. For example, to illuminate a LED light on somebody else's Circuit Playground's device, you might use any of the following messages:

- **LED ON 3** — Turn on LED at position number 3
- **ILLUMINATE** — Turn on a LED. We don't allow control over which LED is turned on.
- **LO3** — "Led On 3" but shorter.
- **X Æ A-12** — Actually the name of Elon Musk's son, but you could use it here as well.

All of these perform direct control over the LED. But maybe that's too specific. You could also invent a protocol that conveys meaning. E.g. the message might be "**I MISS YOU**". Turning on one or more LEDs, indicating that you're thinking of the other person, is just a byproduct.

Q Think of creative ways you might send messages across? What can you convey? What can you hide?

Protocols are a form of code

Protocols can also be seen as a form of encryption code. If you don't understand the rules of how to speak the protocol, you're not part of the in-crowd, and can't participate. Caesar's Cipher was a simple encryption technique that requires knowledge of the protocol to be able to be

deciphered. It is a simple substitution cipher that replaces every letter with another letter, shifting it a fixed number of positions down the alphabet.
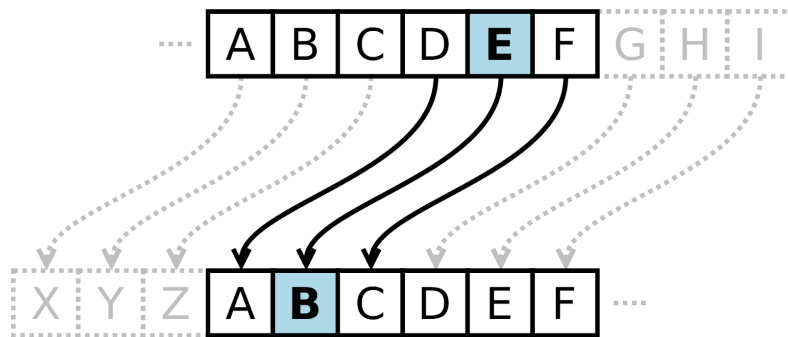
Best Practices for Protocols

A nice way to get started is to use a "phrase" system, meaning it consists of a **verb** and a **noun**. The verb describes the type of action you want to perform: turn on a LED, play a tune, measure temperature, ... The noun describes the data that is affected. The meaning of the noun depends on the context sensitive. For example:

Verb	Noun	Description
LEDON	3	Turn LED 3 on
LEDOFF	3	Turn LED 3 off
TONE	440	Play a tone at 440hz
SOUND	1	Play sound file 1.wav

This system of control was invented for the Apollo Guidance Computer's user interface DSKY. Note that you don't have to follow this system strictly. For example, in this system, there is no way for the TONE command to indicate the length of the tone. Feel free to adapt the protocol to your needs.

Watch the Order!

To send a message, a lot is taking place: your Circuit Playground talks to Processing, which sends out a message over MQTT, which gets routed through multiple connection points, distributed to other listeners, back to Processing, to somebody else's Circuit Playground. In the process, messages might be delayed or not get delivered at all. MQTT actually has some mechanisms built in to guarantee delivery over reliable networks, but order can not be guaranteed.

So if you want to play a tune on somebody else's Circuit Playground, it might not make sense to send it note for note, as the order or timing might mess up:

Sent	Received
TONE 659 0.2 TONE 622 0.2 TONE 659 0.2 TONE 622 0.2 TONE 659 0.2 TONE 494 0.2 TONE 587 0.2 TONE 523 0.2 TONE 440 0.4	TONE 659 0.2 TONE 622 0.2 TONE 622 0.2 TONE 659 0.2 TONE 659 0.2 TONE 494 0.2 TONE 587 0.2 TONE 440 0.4 TONE 523 0.2

What you could do is send a longer string of notes at once, as a single message:

Sent	Received
TONE 659 0.2 622 0.2 622 0.2 659 0.2 494 0.2 659 0.2 587 0.2 440 0.4 523 0.2	TONE 659 0.2 622 0.2 622 0.2 659 0.2 494 0.2 659 0.2 587 0.2 440 0.4 523 0.2

Questions about protocol

- What is your protocol about?
- What level of protocol are you talking about? Lights, or sequences of commands, or feelings, or, ...
- Is your protocol secret or public? Do you want others to participate?

Touch Differently

Let's tame the chaos.

1. Pick a sub-topic for your group.
   a. Boring would be "unrvl2020/touch/mica-1". That's what an engineer would do, so I guess it is ok.
   b. **Must** start with "unrvl2020/touch/". maybe "unrvl2020/touch/ice-fishing"
   c. Letters, Numbers, and Dashes. American centric: letters/numbers!
   d. sub-sub-sub topics are allowed "unrvl2020/touch/witty/fish/sayings"
   e. Typos. Hmm. Interesting.
2. Edit code.py, change "MQTTPublishTo" to your topic

```
36  # Send touch messages to topic:
37  MQTTPublishTo = 'unrvl2020/touch-everyone' # # "unrvl2020/shake-group1"
38  # Listen for touch messages in topic(s):
39  MQTTSubscribe = [
40      MQTTPublishTo,
41      # more than 1...
42      ]
```

change to something like:

```
33
34  # Send touch messages to topic:
35  MQTTPublishTo = 'unrvl2020/touch/ice-fishing' #
36  # Listen for touch messages in topic(s):
```

3. Save.
4. Touch!
5. Go do it. Coordinate on Discord. Less chaos.
   a. talk on discord, test with each other
6. You can subscribe to more than one topic!

Don't Touch

The CP code has to react to the message, and decide what to do. Remember Paul's touch->LED code (`def update_touch(mqtt_message):`)?

There's something like that for "if mqtt message says 0, then light LED 2".
```
def handle_mqtt_message(topic, mqtt_message):
```
But, it's written like a more experienced programmer. Oh oh.

Message Structure

```
>> mqtt: publish unrvl2020/touch/everyone {'touch': {4: (0, 124, 131), 1: (0,
0, 0), 2: (0, 0, 0), 3: (0, 0, 0)}, 'from': 'ANON'}
```

The "topic" is not part of the message.

`{`	a list of `key : value` ("dictionary" or "dict" or "hash")
`'touch': {`	The "touch" portion of the message, has more key:value parts. We have to look for and react to it. Cf. ~line 196: `message["touch"]`
`4: (0, 124, 131),`	The "4" part of "touch", its value is a RGB color ("list")

`1: (0, 0, 0),`
 `2: (0, 0, 0),`
 `3: (0, 0, 0)`
	The other parts (quadrants) of "touch". That's right, order doesn't matter.

`},`	End of "touch" part.
'from': 'awg'	The "from" part. That's "Me".
`}`	End of message

We can construct a message about shaking, temperature, sound, or other things. And then we can react to it.

Different kinds of messages could go to different topics. We could only react to certain messages if the "from" is one we choose.

LocalColor vs RemoteColor
cp.pixels.brightness
public mqtt servers/topics

Inspiration

(aka Homework)

Our technologies:
- circuit playground express, and circuit playground bluefruit
- Neopixels, "capacitive touch"
- circuitpython programming language (a version of python)
- Mu Editor (python code editor and CP serial console)
- MQTT IoT protocol
- shiftr.io
- Processing IDE

Find 1 more tool for our technologies. iphone? android? programming? more portable?

Try out Frederik's examples:
      telepresence_of_touch/circuit-playground/telepresence_of_touch_extra/