HELLO!

Welcome to this workshop.

What's the plan here?

# LET'S GET THE TOOLS..

nodejs.org

arduino.cc

# REFERENCE

https://nodejs.org/api

# DEMO TIME

Let's build a node.js application!

Everybody got some node.js working now?

Yay!

# INTERACTIVE INSTALLATIONS

**IN·TER·AC·TIVE**

influencing or having an effect on each other

# THE DATA MUST FLOW

Any distributed system relies on information moving from one place to another.

This applies at many physical scales and is an invisible fabric that ties our world together.

# NETWORKING

Servers, Clients, Addresses, and Ports

# SERVERS AND CLIENTS

Servers listen for connections.

Clients connect to servers.

Anything on a network can be a server, client, or both.

# IP ADDRESSES

IP addresses allow information to be be routed from one point to another.

192.168.1.172

# NETWORK PORTS

Ports allow for multiple simultaneous connections to exist without interference.

# NETWORK PORTS

- Like a radio channel, separates communications.
- A number between 0 and 65,535
- You can use any port that isn't already in use.
- In general, pick large numbers to be safe.

# ESTABLISHING A CONNECTION

- Server listens on a port.
- Client connects to server address / port.

# DEMO TIME

Let's make connections!

1. Make a server
2. Make a client
3. Say hello!

# HANDLING ERRORS

Servers:

- Ports might not be available.

Clients:

- Server isn't anwering.
- Server cannot be found.

Both:

- Connection might fail.

**YAY!**

Now we can make the whole internet.

(seriously)

# DEMO TIME: DOWNLOAD SOME DATA

# CROSS PLATFORMNESS

If you can get it on the network, it can communicate this way.

DEMO: connect from all the things.

And now...

A journey into abstraction

# DECIMAL, BINARY, AND HEXADECIMAL NUMBERS

The 'base' of a number system indicates the range of values each digit can represent.

## We typically use base 10

- Each digit can represent 10 unique values (0 to 9)
- Each 'place' is a power of 10
- 1, 10, 100, 1000, etc

Example: 12,825

| 10,000 | 1,000 | 100 | 10 | 1 |
|---|---|---|---|---|
| 1 | 2 | 8 | 2 | 5 |

Good so far?

Binary works exactly the same way, but uses base 2.

- Each digit can represent 2 unique values (0 to 1)
- Each 'place' is a power of 2
- 1, 2, 4, 8, 16, 32, etc

Example: 10110

| 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |

In base 10, this number is twenty-two.

Still good?

Hexadecimal works exactly the same way, but uses base 16.

- Each digit can represent 16 unique values (0 to F)
- Each 'place' is a power of 16
- 1, 16, 256, 4096, 65536, etc

Example: 0x3219

| 65,536 | 4,096 | 256 | 16 | 1 |
|---|---|---|---|---|
| 0 | 3 | 2 | 1 | 9 |

In base 10 this number is 12,825.

That's it!

Numbers are just numbers, we can display and think of them them any way we want.

12,825 = 0x3219 = 11001000011001

in base 5 it's 52251!

number bases are arbitrary

Why is binary used with computers?

- History and technological constraints
- Binary-stable systems are easeier to engineer (on/off)
- Early computers were based on mechanical switches
- a CPU is literally millions of tiny switches...
- ... controlled by other tiny switches.

## Why use hexadecimal?

- Complete range a digit (0 - F) fits evenly in four binary digits.
- (16 is an even power of 2)
- It's easy to move between binary and hexadecimal.

0x3219

| 3 | 2 | 1 | 9 |
|---|---|---|---|
| 0011 | 0010 | 0001 | 1001 |

## THE DATA

What is the "stuff" we're actually sending?

Wikipedia says:

> *an ordered and error-checked stream of octets*

octets are 8 binary digits of information, also known as a byte.

(binary digits are commonly referred to as "bits")

## THE DATA

What is the "stuff" we're actually sending?

answer: a series of one or more bytes

## THE HUMBLE BYTE

Bytes are the 1x1 lego block of the digital world.

The universal storage unit for data.

## THE HUMBLE BYTE

Some examples..

- Files on your hard drive.
- The RAM used by programs.
- The data sent over a USB cable.
- Any data flying around the internet.

## BUT WHAT *IS* A BYTE THOUGH?

- 8 binary digits (1's and 0's)
- Represent one of 256 possible values.
- Commonly written as hex pairs: 0C F3 82 7D A1
- *Anything beyond this depends on how the bytes are interpreted.*

**THAT TEXT WE JUST SENT?**

Not actually text, it was just a series of bytes.

Each byte has been arbitrarily assigned to a character.

ASCII - American Standard Code for Information Interchange

## JAVASCRIPT AND BINARY DATA

- Wasn't built with binary data in mind.
- Not a problem until recently.
- Bad for node.js and do-everything-in-a-browser

# DEMO: BYTES UNDER THE HOOD

Handy tools:

- Calculator app with "developer" mode
- Get a "hex editor"

# NODE.JS - BUFFER

Fast and convenient access to binary data.

Docs: nodejs.org/api/buffer.html

Node often relies on Buffer when reaching outside of the JavaScript environment.

- Networking
- File I/O
- Native libraries (USB, Serial port, etc)

# BACK TO NETWORKING!

We're connecting over a local network

Easy to find IP addresses.

Not in the same room?

Not in the same building?

Not in the same country?

## STABBING IN THE DARK

How will my clients find my server?

## REMEMBER: CLIENTS CONNECT TO THE SERVER

- Server doesn't care where the clients are.
- We only need one *findable* thing, the server address!

## POSSIBLE SOLUTIONS

- Have an IP Address that never changes.
- Register a URL for your server.
- Use some kind of service (DynDNS).

## THE DIY APPROACH

1. Server stashes its IP somewhere online.
2. Clients know where to look.
3. Fetch server IP then connect to it.

Lots of cheap/free ways to store data online.

# DEMO: DIY DNS

## LET'S REVIEW

- We can make servers
- We can connect clients
- Clients can exchange data
- Clients can find the server

## NEXT TIME

- ESP8266 joins the fray
- Hook up all the things!
- What do you want to make?