General Assembly

Command Line JS
+ Data Types

- Class Rules reminder
  - How did the break work?
- Thanks for Exit Tickets
- Joke: 2 hard problems in copm sci: Naming Things, Concurrency, Off By 1 Errors
  - Concurrency isn't a problem in JS (see: Event Loop from lesson 00)
  - We'll see why this lesson!

JS1

Last Lesson Recap

[Ask class]

- Using the Command Line
- Internet architecture
  - DNS / Client + Server model
- Git / GitHub
  - [on board] `init` / `clone` / `status` / `config` / `add` / `commit` / `push` / `log`
- [next] Blew everyone's mind...

JS1

Objectives

- Javascript on the command line
- Understand Data Types & Variables
- Create & Access Arrays
- Iterate & Manipulate Arrays

- Objects come in a later class

Command Line JS

Javascript on the command line

- JS isn't just in browser. Also in node!
- Installed node first lesson
- The good stuff!
- Uses Command Line
  - Swiss Army Knife
  - Been around forever
- Right of passage: Hello World
  - 1972, Brian Kernighan (author of the C language, which JS is based on)
- `node` to start
  - `console.log('Hello world')`
    - note the '`undefined`' means no further results (aka; a *return* value, learn that in later lesson)
  - `var x = 5`
  - `var y = 2`
  - console.log(x)
    - no quotes? Why?
  - x + y
  - y * 4
  - x = 9
  - x + y
  - Play for 5 minutes, see what you can discover
- Ctrl-c-c to exit

JS1

Data Types

Data Types

What is a Data Type?

- Computers need to store things
- Different types are stored differently
- [Anecdote]: ASM coding Nintendo DS - dealt with data types on the binary level!

Data Types

Data Type	Example
Strings	`"kittens"/'kittens'`
Integers	`42/1024`
Floats	`3.14/3.0`
Booleans	`true/false`
Arrays	`['Sam', 'Ash', 'Tony']`
Objects	`{age: 29}`

- [on board]
- Up next: Strings + Numbers (Integers / Floats)
- Later in lesson: Arrays
- Next lesson: Booleans
- Later lesson: Objects

Data Types

On the Command Line

- `Math.pow/Math.sqrt/Math.random`
- `0.1 + 0.2 === 0.30000000000000004`
- `.toString()` on everything
- `typeof()`

- `node`
- Numbers:
  - 1 + 2, 10 / 5, etc (from start of class)
  - Other number operators:
    - Math.pow, Math.sqrt, Math.random, Math.sqrt(-1) === NaN
  - IEE 754 format
    - 0.1 + 0.2
    - Math.floor
    - Number((0.1 + 0.2).toFixed(2))
- Strings
  - "Hello World"
  - `.toString()` on everything
    - Try it on different types
    - Wrap types in `( )` first
    - ({..}).toString() == `[object Object]`
- `typeof()`
- Further reading: http://www.2ality.com/2011/11/improving-typeof.html

Data Types

Variables

- Only dealt with *values* so far
- What do we *do* with those values?

Data Types

Variables

Write to a variable

```
var greeting = "hello world"
```

- Like a bucket
- Create bucket with `var`
- Has a name `greeting`
- Has a value of string type

Data Types

Variables

Reading from a variable

```
greeting
```

- Access it by writing out its name

Data Types

Variables

Changing a variable

```
var greeting = "hello world"
...
greeting = "g'day everyone"
```

- Change its value by using equals again, but not `var`!

Data Types

Variables

Give it a try:

```
var greeting = "hello world"
greeting
greeting = "g'day everyone"
greeting
```

- Try it in console
- Hand up when done

Data Types

Variables

Modifying in-place

```
var pi = 3.14
```

```
pi *= 2
```

- Can also modify numbers like so
- `//` is comment, JS ignores it, you don't have to write it out

Data Types

Variables

Modifying in-place

```
var pi = 3.14

pi *= 2 // Same as: pi = pi * 2

pi // 6.28
```

Data Types

Variables

Naming things is hard

```
var aNumber = 3.14
```

- Refer back to joke

Data Types

Variables

Naming things is hard

```
var pi = 3.14
```

```
var buttons = '↑ ↑ ↓ ↓ ← → ← → B A'
```

- Semantic naming
- Don't just say the type, or even the actions
- Name them obviously.

Data Types

Variables

Naming things is hard

```
var pi = 3.14
```

```
var konamiCode = '↑ ↑ ↓ ↓ ← → ← → B A'
```

- It's ok for variable names to be long


15 min

Data Types

Arrays

- Another type; a *collection* of other types
- variables = buckets; arrays = series of buckets
- [use board]:
  - square brackets
  - value == bucket
  - Each value called "*element*"
  - Great for *ordering* elements

Data Types

Arrays

```
var friends = ['Sam', 'Ash', 'Tony']
```

```
friends[0]
```

- [on board] Alternatively; `var friends = new Array('Sam', ..)`
  - But don't use that
- Access: `friends[0]`
- 0 index, last at `lengh - 1`
  - `friends[0]` / `friends[2]`
  - Refer to joke: Off by 1 errors.
    - Human brains can't cope :/
- Can think of Strings as arrays of characters
  - `var friend = 'Kelly' friend[1]`

Data Types

Arrays

```
var friends = ['Sam', 'Ash', 'Tony']
friends[0]
friends.length
```

- length is 1 more than highest index

Data Types

Arrays

```
var pets = []

pets[0] = 'dog'
pets[1] = 'cat'
pets.length // 2
```

```

pets[100] = 'fish'
```

```
pets.length // 101
```

- Walk through
- [ask clsss] what `[]` is
- [ask class] what `pets.length` is
- [ask class] what `pets[2]` is

Data Types

Array Methods

- Different things you can do with arrays
- Don't worry if you don't remember them all, mdn has it all documented
- [on board]: Write each
    - [later] will ask to define each

Array Methods

.toString()

```
var friends = ['Sam', 'Ash', 'Tony']
friends.toString()
```

```
// Sam,Ash,Tony
```

- Saw earlier in lesson
- Gives string representation *of each element*
- Pretty boring for arrays of strings

Array Methods

.toString()

```
var foo = [{age: 29}, {age: 36}]
foo.toString()
```

```
// [object Object],[object Object]
```

- Our old friend `object Object`
- Calls `.toString()` on each element

Array Methods

.join()

```
var foo = ['Sam', 'Ash', 'Tony']
foo.join()
```

```
// Sam,Ash,Tony
```

```
foo.join(' and ')
```

```
// Sam and Ash and Tony
```

- Looks same as `.toString()`, but can change delimiter

Array Methods

.indexOf()

```
var friends = ['Sam', 'Ash', 'Tony']
friends.indexOf('Tony')
```

```
// 2
```

```
friends.indexOf('Kelly')
```

```
// -1
```

- Can't say 0, because that item already exists

Array Methods

.pop() & .push()

```
var friends = ['Sam', 'Ash', 'Tony']
friends.pop()
```

```
// Tony
```

```
friends
// [ 'Sam', 'Ash' ]
```

- Removes and returns element

Array Methods

.pop() & .push()

```
// [ 'Sam', 'Ash' ]
```

```
friends.push('Kelly')
```

```
// 3
```

```
friends
// [ 'Sam', 'Ash', 'Kelly' ]
```

- Adds element

Array Methods

.shift() & .unshift()

```
var friends = ['Sam', 'Ash', 'Tony']
friends.shift()
```

```
// Sam
```

```
friends
// [ 'Ash', 'Tony' ]
```

- Removes and returns *first* element

Array Methods

.shift() & .unshift()

```
// [ 'Ash', 'Tony' ]
```

```
friends.unshift('Kelly')
```

```
// 3
```

```
friends
// [ 'Kelly', 'Ash', 'Tony' ]
```

- Adds element *to start*

Array Methods

.reverse()

```
// [ 'Kelly', 'Ash', 'Tony' ]
```

```
friends.reverse()
```

```
// [ 'Tony', 'Ash', 'Kelly' ]
```

```
friends
```

```
// [ 'Tony', 'Ash', 'Kelly' ]
```

- Returns the reversed array
- *And* saves it
- All of these are "in-place" - they change the actual array, called *mutation*.
- Other methods do not mutate, they return a copy

Array Methods

Pair & Share

- [Think/Pair/Share]: What each of those methods do
- 5 min

Data Types

Array Iteration

Array Iteration

`while`

Array Iteration

`while`

Instead of this

```
console.log(0)
console.log(1)
console.log(2)
console.log(3)
console.log(4)
console.log(5)
// etc
```

Array Iteration

`while`

We can iterate

```
var i = 0
while (i < 10) {
  console.log(i)
  i++
}
```

- Is a counter, starts at 0, goes up by 1 (++), while it's still less than 10.
- i could be used as array index!
- [Ask class] Expected output?

Array Iteration

`while`

We can iterate

```
                              // 0
var i = 0                     // 1
while (i < 10) {              // 2
  console.log(i)              // 3
  i++                         // 4
}                             // ...
```

Array Iteration

`while`

We can iterate *over an array*

```
var friends = ['Tony', 'Ash', 'Kelly']
var i = 0
while (i < friends.length) {
  console.log(friends[i])
  i++
}
```

- Awesome! But, cumbersome
- Lots of lines

Array Iteration

`for`

Same as `while`, but compact

```
for(var i = 0; i < 10; i++) {
  console.log(i)
}
```

Array Iteration

`for`

Same as while, but compact

```
var friends = ['Tony', 'Ash', 'Kelly']
for(var i = 0; i < friends.length; i++) {
  console.log(friends[i])
}
```

- Awesome!
- But still kinda complicated and annoying

Array Iteration

`.forEach()`

Another Array method

```
var friends = ['Tony', 'Ash', 'Kelly']
for(var i = 0; i < friends.length; i++
  console.log(friends[i])
}
```

- Given the for loop, we can rewrite it with `.forEach()`

Array Iteration

`.forEach()`

Another Array method

```
var friends = ['Tony', 'Ash', 'Kelly']
for(var i = 0; i < friends.length; i++
  console.log(friends[i])
}
```

```
var friends = ['Tony', 'Ash', 'Kell
friends.forEach(function(name) {
  console.log(name)
})
```

- [on board]: .forEach()
- Give it a function
- Which receives a `name`
- Applied to *every* element in Array
- Functions are:
  - A small, reusable piece of code
  - Defined with the `function` keyword
- Ensure the braces match up

Array Iteration

.every()

```
var ages = [21, 90, 15, 35]
```

```
ages.every(function(age) {
  return age >= 18
})
```

```
// false
```

- Checking for adults
- `age >= 18` returns `true` or `false`
- Looks for any that don't match. Then returns `false`. Or; `true` if they all match

Array Iteration

.some()

```
var ages = [21, 90, 15, 35]
```

```
ages.some(function(age) {
  return age >= 18
})
```

```
// true
```

- Visits *every* element
- At least one has to be `true`
- Looks for any that don't match. Then returns `true`. Or; `false` if none match

Array Iteration

.filter()

```
var ages = [21, 90, 15, 35]
```

```
ages.filter(function(age) {
  return age >= 18
})
```

```
// [ 21, 90, 35 ]
```

```
ages
```

```
// [ 21, 90, 15, 35 ]
```

- Visits *every* element
- `age >= 18` returns `true` or `false`
- Removes elements that are `false`
- Is *not* in-place. It does *not* mutate. Returns a new array

Data Types

Arrays

http://mdn.io/array

- [on board] write it
- MDN is *great*!
- There is a lot you can do with arrays
- Collections are important in programming

Data Types

Homework

http://bit.ly/array-practice

```
$ node array-practice.js
```

Extended Homework

http://bit.ly/lesson-02-homework

- Save that file as `array-practice.js`
- Run with `node`

JS1

Objectives

- Revisit each of the objectives on board

JS1

Next Lesson

- TODO

JS1

Questions?

- Further learning:
  - TODO

JS1

Exit Tickets

http://ga.co/js1syd

- [share in Slack]

General Assembly

JS1