General Assembly

Objects + JSON

- Class Rules reminder
  - No screens
  - There are no stupid questions
  - Dinner during break
  - Don't interrupt someone else's question / answer
  - Be on time
- Thanks for Exit Tickets
- Joke: How do you think the unthinkable? With an itheberg.

JS1

Scope + Closures Recap

- Scope
- Scope Chains
- Closures

JS1

Objectives

- Understand Objects in JS
- Access & Assign Object values
- Iterate over Object values
- Understand Object Oriented Programming
- Apply prior knowledge for Data Privacy
- Understand JSON

JS1

Objects + JSON

- Objects are a data type
- Similar to Arrays; a collection of other data types
  - Instead of *elements*, it's *properties*
    - Each property is a key/value pair (key == name)
    - [vocab]: *key/value*
- keys mean: can name your data
- Used to structure and organise concepts in code
- [next...] Simplest object

Objects + JSON

Empty Object

```
var car = {}
```

- Simplest object, an *empty* object

Objects + JSON

Object Properties

- Objects are like Hash / Map / Dictionary from other languages
- Properties are how we get data into an object

Objects + JSON

Object Properties

```
var car = { wheels: 4, colour: 'red' }
```

- key/value pairs
  - colon to assign, comma to separate
- unordered (unlike arrays)
  - Array indexes of items don't change. Order stays the same
  - [on whiteboard]:
    - `var names = ['Ash', 'Kelly']` (value swapped == different array)
    - `car` (keys swapped == Same object)
- Easier to read on multiple lines

Objects + JSON

Object Properties

```
var car = {
  wheels: 4,
  colour: 'red'
}
```

- *values* are any data type
- *keys* are coerced to a string
    - ie; only use strings as keys

Objects + JSON

Object Properties

```
var car = {
  wheels: 4,
  colour: 'red',
  owners: ['Ash', 'Kelly']
}
```

- Values are any data type
  - Including arrays and other objects...

Objects + JSON

Object Properties

```
var car = {
  wheels: 4,
  colour: 'red',
  owners: ['Ash', 'Kelly'],
  size: {
    length: 4117,
    width: 1620,
    height: 1394
  }
}
```

- Actual size for '99 Hyundai Excel Sedan!

Objects + JSON

Accessing & Assigning Values

```
var car = {
  wheels: 4,
  colour: 'red'
}
```

- We've set values already during creation of the object

Objects + JSON

Accessing Values

```
var car = {
  wheels: 4,
  colour: 'red'
}

car.wheels // 4
```

```
car['wheels'] // 4
```

- Dot notation that we've already seen (`.forEach`, etc)
- Bracket notation
  - Same result as dot notation
  - Keys are strings, must use a string when using bracket notation

Objects + JSON

Assigning Values

```
var car = {
  wheels: 4,
  colour: 'red'
}
```

```
car.owners = ['Ash', 'Kelly']
```

```
car['size'] = {
  length: 4117,
  width: 1620,
  height: 1394
}
```

- Assignment with equals same as other data types
- Value can be any data type (number & string & array & object shown here)
- [on board]: `typeof(car.size) === 'object'`
- [ask class]: How would we access the `length` key?

Objects + JSON

Assigning Values

```
var car = {
  wheels: 4,
  colour: 'red'
}
car.owners = ['Ash', 'Kelly']

car.start = function() {
  console.log('Vvrrroom!')
}
```

- [ask class]: How can we execute `start`?
  - `car.start()`

Objects + JSON

Aside: Bracket Notation

```
car['size']
```

- Used when keys aren't valid variable names, so can't use dot notation
- [on board]: examples
  - eg; contains a hyphen
  - starts with a number
  - has a space
  - etc

Objects + JSON

Aside: Bracket Notation

```
var whichKey = 'size'
car[whichKey]
```

- Used to access a variable key
- Dot notation wont work (`car.whichKey` - `whichKey` isn't a key on the object)

Objects + JSON

Iterating

- Like arrays; can iterate

Objects + JSON

Iterating

```
var car = {
  wheels: 4,
  colour: 'red'
}
```

```
for (var key in car) {
  console.log(car[key])
}
```

```
// 4
// 'red'
```

- Similar to `for()` we saw earlier, but with objects
- [vocab]:
  - `for(var key in car)`
  - mdn.io/forin
- Has limitations (can accidentally pick up `.toString`, etc)

Objects + JSON

Iterating

```
var car = {
  wheels: 4,
  colour: 'red'
}

Object.keys(car)
```

```
// ['wheels', 'colour']
```

- Returns array of keys as strings
- [vocab]:
  - `Object.keys(car)`
  - mdn.io/object.keys
- Not so useful on its own, but it's just an array, can iterate on that!

Objects + JSON

Iterating

```
var car = {
  wheels: 4,
  colour: 'red'
}

var keys = Object.keys(car)
```

```
keys.forEach(function(key) {
  console.log(car[key])
}
```

- Using the `.forEach` from arrays we already learned!

Objects + JSON

Object Oriented Programming (OOP)

- Modeling the real world as objects
- (eg; the `car` we saw above)
- Can simplify our code into smaller managable chunks

Objects + JSON

Object Oriented Programming (OOP)

Nouns

Verbs

- When modeling, think about these
- Nouns are *things* / *state*
- Verbs are *actions*

Objects + JSON

Object Oriented Programming (OOP)

Nouns: *data*

Verbs: *methods / functions*

- Eg; `car`:
  - Nouns: `wheels`, `colour`, `owners`, etc
  - Verbse: `start`, `drive`, etc
- Exercise (20min):
  - In pairs
  - Come up with a few nouns & verbs based on the card topic
  - Then code that up in a `.js` file that runs with `node`

Objects + JSON

Getters & Setters

- So far; we've manually set all the values on the objects
- works, but has no limits
  - eg; age shouldn't be negative!
  - name should be a string, etc

Getters & Setters

Getters

```
var car = {
  wheels: 4,
  colour: 'red'
}
```

```
car.getColour = function() {
  return this.colour
}
```

- function value for `getColour`.
- ie; `getColour` is a method
- `this` refers to the object (in this case; `car`)
  - more `this` in later lesson
  - `this` `===` `car`, same result
- What's the point? Why not just `car.colour`?

Getters & Setters

Getters

```
var car = {
  wheels: 4
}

car.getColour = function() {
  return this.colour || 'blue'
}
```

- Logical operator is an expression
- Defaults `colour` to `'blue'`.
  - Note - does *not* assign it, `car.colour` is still `undefined`

Getters & Setters

Setters

```
var car = {
  wheels: 4,
  colour: 'red'
}
```

```
car.setColour = function(newColour) {
  this.colour = newColour
}
```

- Again, why not just do `car.colour = whatever`?
- because we want to make sure it's a string

Getters & Setters

Setters

```
var car = {
  wheels: 4,
  colour: 'red'
}
car.setColour = function(newColour) {
  if (typeof(newColour) !== 'string') {
    console.error('The car colour must be a string!')
    return
  }
  this.colour = newColour
}
```

- console.error() similar to console.log() but... an error!
- early return to stop executing the rest of the function
- Exercise Part 2 [15min]:
    - Add getters / setters to your objects in node

Object Oriented Programming

Data Privacy

- We have `setColour`, but can still assign to `car.colour` - what's the point?
- Need a way to make `car.colour` private
  - Avoid accidentally changing it to a number
  - you can be more certain about the behaviour of your code
- No such thing as private object data in JS
- But; outside variables can't access local scope variables!

```
function makeCar() {
  return {
    wheels: 4,
    colour: 'red'
  }
}
var car = makeCar()
car.colour // 'red'
```

- Another type of factory function
  - aka: "The Factory Pattern"
  - Would you like to know more? https://www.youtube.com/watch?v=ImwrezYhw4w
- Very similar to before
- But now we're DRY / can run `makeCar()` multiple times

```
function makeCar(numberOfWheels, carColour) {
  return {
    wheels: numberOfWheels,
    colour: carColour
  }
}
var car = makeCar(4, 'red')
car.colour // 'red'
```

- Using parameters to set the initial values

```
function makeCar(numberOfWheels, carColour) {
  return {
    wheels: numberOfWheels,
    colour: carColour,
    getColour: function() {
      return this.colour || 'blue'
    }
  }
}
var car = makeCar(4, 'red')
car.colour // 'red'
car.getColour() // 'red'
```

- But can also still do `car.colour`

```
function makeCar(numberOfWheels, carColour) {
  return {
    wheels: numberOfWheels,
    colour: carColour,
    getColour: function() {
      return this.colour || 'blue'
    },
    setColour: function(newColour) {
      if (typeof(newColour) !== 'string') {
        console.error('The car colour must be a string!')
        return
      }
      this.colour = newColour
    }
  }
}
var car = makeCar(4, 'red')
car.setColour('green')
```

- This is our entire object put together
- But still no privacy on `car.colour`

```
function makeCar(numberOfWheels, carColour) {
  return {
    wheels: numberOfWheels,

    getColour: function() {
      return carColour || 'blue'
    },
    setColour: function(newColour) {
      if (typeof(newColour) !== 'string') {
        console.error('The car colour must be a string!')
        return
      }
      carColour = newColour
    }
  }
}
var car = makeCar(4, 'red')
car.setColour('green')
```

- Take advantage of scope & closures!
- Full privacy on the colour now!
- Could do same for `wheels`
- Exercise Part 3 [15min]:
  - factory function & private data for your solution

Objects + JSON

JSON

- JavaScript Object Notation
- A way to convert Objects to strings
- [on board]: http://json.org/

Objects + JSON

JSON

```
var car = {
  wheels: 4, // Four wheels!
  colour: 'red'
}
```

```
{
  "wheels": 4,
  "colour": "red"
}
```

- JS on top, JSON on bottom
- Differences:
  - no variable in json - it's purely the object only
  - keys wrapped in double quotes (single quotes *not* allowed)
  - strings wrapped in double quotes too
  - No comments!
- Other diffs:
  - No functions either
- [next...]: How do you convert between the two?

Objects + JSON

JS -> JSON

```
var car = {
  wheels: 4,
  colour: 'red'
}
JSON.stringify(car)
// '{"wheels":4,"colour":"red"}'
```

- [vocab]: `JSON.stringify` / mdn.io/json.stringify
- Notice output is a string
- Makes it language independent (can be parsed in any language)
    - eg; C#, Java, Go, Haskel, etc

Objects + JSON

JSON -> JS

```
JSON.parse('{"wheels":4,"colour":"red"}')
// {
//   wheels: 4,
//   colour: 'red'
// }
```

- This is how you read it in JS
- [vocab]: `JSON.parse` / mdn.io/json.parse
- Because it's so easy to use, it's great for sharing & storing data over the internet

JS1

Objectives

- Revisit each of the objectives on board
- No Homework. Continue with Sticky Notes exercise

JS1

Next Lesson

Intro to DOM + jQuery

- No class on Monday!
- Next class is Wednesday!

JS1

Questions?

JS1

Exit Tickets

http://ga.co/js1syd

- [share in Slack]

General Assembly

JS1

- Resources
  - Factory functions: https://www.youtube.com/watch?v=ImwrezYhw4w
  - JSON spec: http://json.org/