DEV ACADEMY
TE KURA HANGARAU
O AOTEAROA

Node.js & JS (variables, functions, objects)

Week 1, Day 1

Agenda for Today

- Node.js - JavaScript *not* in the browser
- Revision:
  - Variables
  - Functions
- Objects & Arrays - `` `.` `` and `` `[]` `` notation
- Running `` `.js` `` challenges on Bootcamp
  - When to ask for help

Node.js - JavaScript *not* in the browser

- Node.js is an open-source JavaScript *runtime* environment that allows us to run JavaScript on our own computers (not just in the browser)
- First introduced in 2009 by Ryan Dahl, built on top of Google's V8 JavaScript engine
- You've been using it in Foundations to run tests
  - `` `npm test` `` to run tests
  - `` `npm install` `` to install packages
- Has many of the same functions/features as browser JavaScript, but...
  - is missing some (the DOM, `` `window` ``, `` `alert` ``)
  - has some extras (`` `fs` ``, `` `path` ``, `` `process` ``, `` `require` ``, `` `module.exports` ``)

Node.js - the REPL `Demo`

- REPL stands for Read-Eval-Print-Loop
- Run `node` in your terminal to start the Node.js REPL

```
$ node
Welcome to Node.js v18.17.0.
Type ".help" for more information.
> 1 + 1
2
> console.log('hello world')
hello world
```

Helpful tip: you can exit the REPL with `Ctrl` + `C` twice

Node.js - running a file

Demo

- You can also run a `.js` file with `node`
- Create a file called `hello.js` with the following contents:

```
console.log('hello class ☀')
```

- Run it with `node hello.js`

```
$ node hello.js
hello class ☀
```

Variables `Revision`

Three ways to define: `var`, `let`, `const`

`let`

- Values can be *reassigned*

```js
let string = 'hello'
console.log(string) // hello

string = 'goodbye'
console.log(string) // goodbye
```

`const`

- Values cannot be *reassigned*

```js
const string = 'hello'
console.log(string) // hello

// Uncaught TypeError: Assignment to constant variable.
string = 'goodbye'
```

Variables `Revision`

Three ways to define: `var`, `let`, `const`

`let`

- Values can be *reassigned*

`const`

- Values cannot be *reassigned*
- Does not prevent *mutation*

```
const object = {
  colour: 'red'
}
console.log(object) // { colour: 'red' }

object.colour = 'blue'
console.log(object) // { colour: 'blue' }
```

Objects & Arrays `Revision`

Objects and arrays are *mutable* data types - they can be changed after they are created

... even if they are a `const`

```
const teachers = ['Scooter', 'Kermit', 'Gonzo']

teachers // ['Scooter', 'Kermit', 'Gonzo']

teachers[0] = 'Piggy'
teachers // ['Piggy', 'Kermit', 'Gonzo']

teachers.push('Fozzie')
teachers // ['Piggy', 'Kermit', 'Gonzo', 'Fozzie']
```

Objects & Arrays

Revision

Objects and arrays are *mutable* data types - they can be changed after they are created

... even if they are a `const`

```
const waterBottle = {
  colour: 'Olive Green',
  size: '709mL',
  contents: 'Water'
}

waterBottle // { colour: 'Olive Green', size: '709mL', contents: 'Water' }

waterBottle.contents = 'Coffee'

waterBottle // { colour: 'Olive Green', size: '709mL', contents: 'Coffee' }
```

Functions

Revision

Functions are reusable blocks of code

```javascript
function myFunction(a, b) {
  console.log(a, b)

  // ... do something

  return 'return value'
}

// invoke the function
const result = myFunction('hello', 'world')
```

Anatomy:

- **Name:** `myFunction`
- **Parameters:** `a`, `b`
- **Return value:** `'return value'`
- **Side effects:** `console.log(a, b)`
- **Invocation:** `myFunction('hello', 'world')`
- **Arguments:** `'hello'`, `'world'`

Object notation

There are two ways of indexing/accessing properties on an object:

- `` `.` `` notation
- `` `[]` `` notation

```javascript
const object = {
  style: 'fancy',
  colour: 'red',
  'favourite food': 'pizza'
}

console.log(object.style) // fancy
console.log(object['colour']) // red
const key = 'favourite food'
console.log(object[key]) // pizza
```

`.` notation

```
const object = {
  style: 'fancy',
  colour: 'red',
  'favourite food': 'pizza'
}

object.style // fancy
object.colour // red
object.favourite food // Uncaught SyntaxError
```

- Most common way of accessing properties on an object
- Cannot be used if the property name contains spaces or special characters
- Cannot be used to access properties dynamically (*what you see is what you get*)

`[]` notation

```
const object = {
  style: 'fancy',
  colour: 'red',
  'favourite food': 'pizza'
}

object['style'] // fancy
object['favourite food'] // pizza
let key = 'colour'
object[key] // red
key = 'style'
object[key] // fancy
```

- Can be used to access properties with spaces or special characters
- Can be used to access properties dynamically (based on a variable)

Mutability

Primer

We will cover more about mutability in the another lesson, but for now:

- Objects and arrays are *mutable* data types

Example: water bottle

```javascript
const waterBottle = {
  colour: 'Olive Green',
  size: '709mL',
  contents: 'Water'
}

function fillBottleWithCoffee(bottle) {
  bottle.contents = 'Coffee'
}

waterBottle // { colour: 'Olive Green', size: '709mL', contents: 'Water' }

fillBottleWithCoffee(waterBottle)

waterBottle // { colour: 'Olive Green', size: '709mL', contents: 'Coffee' }
```

Mutability

Primer

```
const originalBottle = {
  colour: 'Olive Green',
  size: '709mL',
  contents: 'Water'
}

function fillANewBottleWithCoffee(bottle) {
  const newBottle = {
    ...bottle,
    contents: 'Coffee'
  }
  return newBottle
}

originalBottle // { colour: 'Olive Green', size: '709mL', contents: 'Water' }

const newBottle = fillANewBottleWithCoffee(originalBottle)

originalBottle // { colour: 'Olive Green', size: '709mL', contents: 'Water' }
newBottle // { colour: 'Olive Green', size: '709mL', contents: 'Coffee' }
```

Spread operator

Demo

The spread operator (`[...value]`, `{ ...value }`) can be used to *shallow copy* the contents of one array or object into a new array or object

```
const numbers = [1, 2, 3]

const copyOfNumbers = [...numbers]

copyOfNumbers // [1, 2, 3]

numbers.push(4)

numbers // [1, 2, 3, 4]
copyOfNumbers // [1, 2, 3]
```

```
const originalBottle = {
  colour: 'Olive Green',
  size: '709mL',
  contents: 'Water'
}

function fillANewBottleWithCoffee(bottle) {
  const newBottle = {
    ...bottle,
    contents: 'Coffee'
  }
  return newBottle
}

const newBottle = fillANewBottleWithCoffee(originalB

originalBottle // { colour: 'Olive Green', size: '70
newBottle // { colour: 'Olive Green', size: '709mL',
```

Review Questions

1. What is the difference between `let` and `const`?
2. What is the difference between `.` and `[]` notation?
3. What is the difference between a function's *parameters* and *arguments*?
4. What is the difference between a function's *return value* and *side effects*?
5. Given the following function, what is its return value?

```js
function logGreeting(greeting) {
  console.log(greeting)
}
```

6. How do you access the `favourite snack` property on the following object?

```js
const person = {
  name: 'Gerald',
  'favourite snack': 'toast'
}
```

Running `.js` challenges on Bootcamp

- Follow instructions step by step
- Often a message at the bottom describes what being done looks like
- It's okay to run out of time! You can always revisit it later
- Stretch section for extra practice e.g. testing

When to ask for help?

- If you're stuck on something for more than 15 minutes
- Constructive problem solving is good within reason
- Asking questions unlocks your true learning potential
- Don't be shy! We're here to get you unstuck