Understand key JavaScript concepts

Aims

- Understand scope in JS (var, let, const)
- Functions
- Strings
- Arrays
- Describe the prototypical nature of all JavaScript-based inheritance
- Closure scope

Variable scope and hoisting in JS (var, let, const)

Problems with var:

- Scope
- Hoisting
- Redeclarion

``let`` vs ``const``

- Rule of thumb to prefer const over let
- const can’t be reassigned
- Note the difference for Objects and Arrays

Functions

- Multi-paradigm nature of JS
- Functions passed as arguments
- Functions assigned as values in an object
- (Fat) Arrow Functions

Strings

- We can use different quotes to declare a string (single, double and backtick)
- Backtick defined strings can have interpolated values declared with `` `${}` ``.
- Strings are immutable but you can access characters
- Helper methods can transform strings and pass values back.

Arrays

- Define with `` `[]` ``.
- Key methods (.join(), .map(), .filter(), .reduce(), .forEach())
- Destructuring

Prototypical Inheritance

Inheritance with JS is achieved with a chain of prototypes. These approaches have evolved significantly over time.

The three common approaches to creating a prototypal chain:

- functional
- constructor functions
- class-syntax constructors

For the purposes of these examples, we will be using a Wolf and Dog taxonomy, where a Wolf is a prototype of a Dog.

Prototypical Inheritance (Functional)

```javascript
const wolf = {
  howl: function() { console.log(`${this.name} awoooooo`)}
}

const dog = Object.create(wolf, {
  woof: {value: function() {console.log(`${this.name} woof`)}}
})

const rufus = Object.create(dog, {
  name: {value: 'Rufus the dog'}
})

rufus.woof()
rufus.howl()
```

Prototypical Inheritance (Constructor function)

```javascript
function Wolf(name) {
  this.name = name;
}

Wolf.prototype.howl = function() {
  console.log(`${this.name} awoooooooo`)
}

function Dog(name) {
  Wolf.call(this, `${name} the dog`)
}

Object.setPrototypeOf(Dog.prototype, Wolf.prototype)

Dog.prototype.woof = function() {
  console.log(`${this.name} woof`)
}

const rufus = new Dog('Rufus')

rufus.woof()
rufus.howl()
```

Prototypal Inheritance (Class-Syntax Constructors)

```javascript
class Wolf {
  constructor(name) {
    this.name = name
  }
  howl() {
    console.log(`${this.name} awoooooooo`)
  }
}

class Dog extends Wolf {
  constructor(name) {
    super(`${name} the dog`)
  }
  woof() {
    console.log(`${this.name} woof`)
  }
}

const rufus = new Dog('Rufus')

rufus.woof()
rufus.howl()
```