COURSEWARE

Professional Skills		
Agile Fundamentals		
Jira		
Git		
Databases Introduction		
Java Beginner		
Maven		
Testing (Foundation)		
Java Intermediate		
HTML		
CSS		
Javascript		
0	What is JavaScript	
0	Getting started with JS	
0	Variables	
0	Data types	
0	ASI	
0	Strict mode	
0	Iteration	
0	Conditionals with Truthy / Falsey	
0	Objects, Arrays + JSON	
0	Structuring JS Code	
0	Destructuring	
0	Scope	
0	Functions, function expressions and arrow functions	
0	The ECMAScript 6 Specification	
0	OOP in JavaScript	
0	Best Practices	
0	Closures	
0	Callbacks and Promises	
0	Cookies	
0	Hoisting	
0	Prototypes	
0	Query Parameters	

Higher Order Functions

OOP in JavaScript

Contents

- Overview
- <u>Tutorial</u>
 - Encapsulation
 - Classes
 - Inheritance
 - Extends and Super()
- Exercises

Overview

The basic idea of **Object Orientated Programming** (OOP) is that we use objects to model things we want to represent inside our programs and/or provide a simple way to access functionality that would otherwise be difficult to make sense of.

Objects can contain data and code which represents the thing you are trying to model. Object data can be **encapsulated** (stored neatly) inside a **namespace** (object package), making it easy to structure and access.

Tutorial

Encapsulation

Encapsulation protects data from uncontrolled access; it is important that 'private properties' aren't accessed directly. Instead we must use 'getters' or 'accessors' to retrieve the value of a property and 'setters' or 'modifiers' to change the value of a property.

This allows the class to decide if the value is **permissible**.

It is good practise to declare private variables with a prefix of an underscore _ . Although this is purely convention there is no notion of private scope for properties in JavaScript yet.

```
constructor(wheels, speed, power){
    this._wheels = wheels;
    this._speed = speed;
    this._power = power;
}
```

Classes

Classes are the blue-prints used to instantiate objects.

They contain constructors which are used to initialise an object upon creation, as well as functions that operate on the data the class contains.

Instances of classes contain the data and functionality defined in the class. Lets have a look at an example...

_		
0	Web Storage	
0	DOM Manipulation	
0	Handling Events and Timed Events	
0	Asynchronous Programming	
0	HTTP-Requests	
0	XMLHttpRequests	
0	Fetch API	
Spring Boot		
Selenium		
Sonarqube		
Advanced Testing (Theory)		
Cucumber		
MongoDB		
Express		
NodeJS		
React		
Express-Testing		
Networking		
Security		
Cloud Fundamentals		
AWS Foundations		
AWS Intermediate		
Lin	ux	
DevOps		
Jenkins Introduction		
Jer	ikins Pipeline	
Markdown		

IDE Cheatsheet

```
class Pet {
  constructor(name) {
   this._name = name;
    this._eaten = false;
 hungry() {
   if (!this._eaten) {
      console.log(`${this._name} is hungry`);
   } else {
      console.log(`${this._name} has eaten!`);
   }
 }
  eating() {
   console.log(`${this._name} is eating!`);
    this._eaten = true;
 }
  speak() {
    console.log(`${this._name} says hello`);
 }
}
let pup = new Pet("Pup-eroni");
pup.hungry(); // Pup-eroni is hungry
pup.eating(); // Pup-eroni is eating
pup.speak(); // Pup-eroni says hello
pup.hungry(); // Pup-eroni has eaten!
```

- 1. In the above code, we have a class with a constructor which sets the name and sets the default vaulue of eaten = false.
- 2. The three functions inside the class retrieves the values of the variables in the class.
- 3. In the eating() function we update the value of eaten = true.
- 4. Outside of the class, we initialise a new object to the class using the new Pet() function call.
- 5. We can now access all of the functions and variables using the dot notation e.g. pup.hungry();

We can create multiple object instances by calling the constructor of the class. This will create a new 'copy' using the class as a template.

In cases where we want specialist properties and types we can use Inheritance...

Inheritance

Inheritance refers to an object being able to inherit methods and properties from a parent object. Yet still add unique features which are only available to it's own class. This way we don't need to duplicate code multiple times.

Inheritance and encapsulation are important as together they allow to build applications using re-usable code which will be maintainable and scalable.

Extends and Super()

We use the extends keyword to identify a child of a class. It creates a parent-child like relationship. We can inherit all of the properties and functions within the parent class and use it within the subclass.

We must use the super() function to pass the variables we wnt to use before using this.

Take a look at the example below:

```
class Dog extends Pet {
  constructor(name, tricks) {
    super(name); // Calls the super constructor and assigns the attribute.
    this._tricks = tricks;
  }
  play() {
    console.log(`${this._name} wants to show you his tricks!`);
 tricks() {
    console.log(`${this._name} can do a ${this._tricks}`);
  }
  speak() {
   super.speak();
    console.log(`WOOOOOOOF`); // Overrides parent function call();
 }
}
let doggy = new Dog("Bark Wahlberg", "Backflip");
doggy.hungry();
doggy.speak();
```

Exercises

- 1. Create a Person class with the properties:
 - Name
 - Gender
 - Age
 - Interests
 - Bio
 - Greeting
- 2. Create a new Teacher class which inherit methods from the Person Class, add a super call()
- 3. Create a new Student class which interhis methods from the Person Class, add a super call()
- 4. Instantiate multiple teacher and student objects and call methods from the respective classes.