**JavaScript/Object Oriented JS**

Pieces of code **objects**, better known as Classes in most OOP programming languages and **Functions** in JavaScript.

We use objects as building blocks for our applications.

Building applications with objects allows us to adopt some valuable techniques, namely,

* **Inheritance** (objects can inherit features from other objects),
* **Polymorphism** (objects can share the same interface—how they are accessed and used—while their underlying implementation of the interface may differ), and
* **Encapsulation** (each object is responsible for specific tasks).

Inheritance and Encapsulation since only these two concepts apply to OOP in JavaScript. (To easily reuse code and abstract functionalities into specialized objects)

**Encapsulation** refers to enclosing all the functionalities of an object within that object so that the object’s internal workings (its methods and properties) are hidden from the rest of the application. This allows us to abstract or localize specific set of functionalities on objects.

**Inheritance** refers to an object being able to inherit methods and properties from a parent object.

Object Creation patterns (**Encapsulation**) and Code Reuse patterns (**Inheritance**)

**Modern JavaScript development Techniques**

New features:

arrow functions, module import/export capabilities, de-structuring, template strings, let and const, generators, promises.

**Alternatives:** TypeScript, CoffeeScript, PureScript, Elm

JavaScript Frameworks: ReactJS, Angular2, Vuejs etc.

ES6, jQuery, AJAX

**UI Componentization and Modularization**

Modularity is the key to creating a flexible design system. For a system to be modular, it must have interchangeable parts. These parts are components. In Web terms, a component is just a generic term for any pre-defined object that you intend to use across multiple pages (you might also refer to these by another name, like widgets or modules). In order for components to be reusable, they must be standardized in appearance and function. This ensures that each component will render reliably regardless of the context it's used in. Think of components as Legos: interchangeable building blocks you can assemble into pages.

**Component**-**Based Architecture**. ... The primary objective of **component**-**based architecture** is to ensure **component** reusability. A **component** encapsulates functionality and behaviors of a software element into a reusable and self-deployable binary unit.

**Asynchronous request handling Techniques**

**Synchronous**: A synchronous **request** blocks the client until operation completes. In such case, javascript engine of the browser is blocked.

**Asynchronous** An **asynchronous request** doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also.

* An asynchronous XMLHttpRequest, you receive a callback when the data has been received. This lets the browser continue to work as normal while your request is being handled.

**Promises**(introduced in ES2015/ES6) are a great way to handle async callbacks, because they give us a standardised way of **chaining actions** and **handling errors.**

If you want to use **await** at the top-level of your JavaScript program, a neat trick is to use an async “Immediately Executed Function Expression” (IIFE). Sounds complicated, but it isn’t really.

**REST and MVC**

REST means “Representational State Transfer” and API means Application Programming Interface.

The application makes a request to the API and the API from the database provides the required data to the application in the form of JSON or Xml, and the application then displays that data to the user.

API in MVC is not much different from MVC Web apps, as API provides JSON or XML data to the user and Web app provides a view to the user.

**Cross-browser Compatibility issue**

Five Major Browser Compatibility Issues

**Incorrect DOCTYPE**

* The DOCTYPE is something included in the first line of the HTML.

#### ****Absence of CSS Reset****

* The way the pages are rendered is different for each browser and can prevent the web page from displaying properly. Using a CSS reset stylesheet on a web page is one way to ensure that all the browsers render the content using the same criteria.

#### ****No Valid HTML/CSS****

* HTML and CSS are interpreted differently by most browsers and can occur due to a missing <div> in the code.

#### ****Older Browser Detection Scripts****

* Most of the compatibility issues can also occur if the browser is running an older version of the JavaScript.

#### ****Vendor Specific CSS Styles****

* Most of the new browsers often hide their latest CSS functionality behind vendor-specific CSS style.
* So it is best to ensure that the code works perfectly in all the browsers by including an unprefixed version beside the prefixed versions.