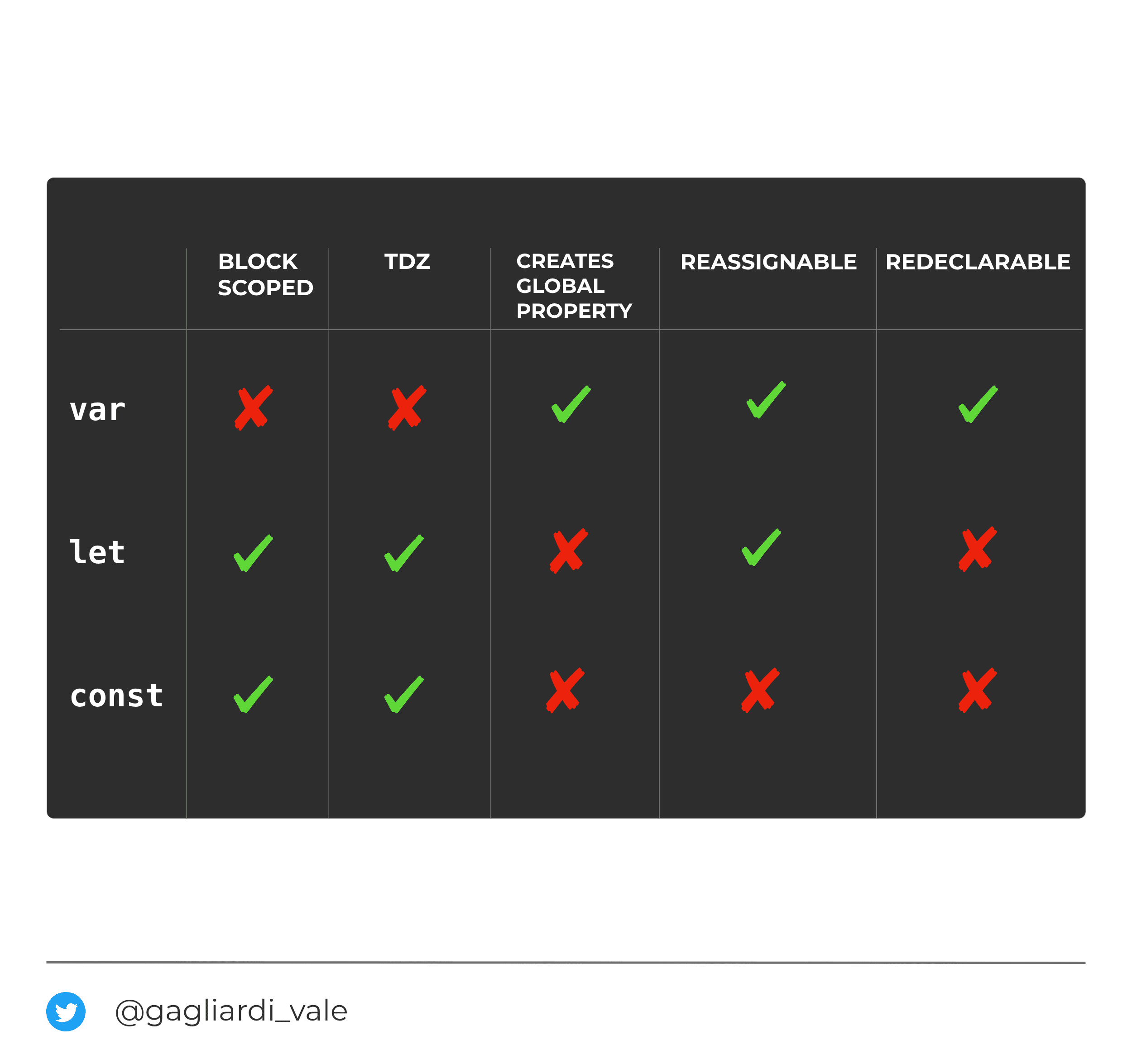
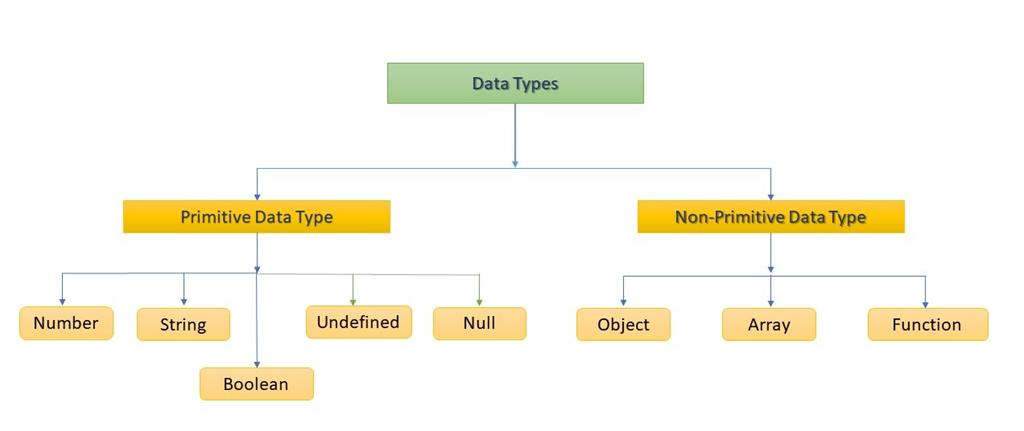
**JavaScript**

**Day 1**

**var, let & const:**



**Data Types**



**Loops**

* [for statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration" \l "for_statement)
* [do...while statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration" \l "do...while_statement)
* [while statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration" \l "while_statement)
* [labeled statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration" \l "labeled_statement)
* [break statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration" \l "break_statement)
* [continue statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration" \l "continue_statement)
* [for...in statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration" \l "for...in_statement)
* [for...of statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration" \l "for...of_statement)

**Objects**

1. **What Is Javascript Objects?**

**Answer :**

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers −

* + Encapsulation − the capability to store related information, whether data or methods, together in an object.
  + Aggregation − the capability to store one object inside another object.
  + Inheritance − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
  + Polymorphism − the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

**String**

The **String** object is used to represent and manipulate a sequence of characters.

**Arrays and their basic functions:**

The Array object is used to store multiple values in a single variable:

const cars = ["Saab", "Volvo", "BMW"];

**Array Properties**

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.w3schools.com/jsref/jsref_constructor_array.asp) | Returns the function that created the Array object's prototype |
| [length](https://www.w3schools.com/jsref/jsref_length_array.asp) | Sets or returns the number of elements in an array |
| [prototype](https://www.w3schools.com/jsref/jsref_prototype_array.asp) | Allows you to add properties and methods to an Array object |

**Array Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [concat()](https://www.w3schools.com/jsref/jsref_concat_array.asp) | Joins two or more arrays, and returns a copy of the joined arrays |
| [copyWithin()](https://www.w3schools.com/jsref/jsref_copywithin.asp) | Copies array elements within the array, to and from specified positions |
| [entries()](https://www.w3schools.com/jsref/jsref_entries.asp) | Returns a key/value pair Array Iteration Object |
| [every()](https://www.w3schools.com/jsref/jsref_every.asp) | Checks if every element in an array pass a test |
| [fill()](https://www.w3schools.com/jsref/jsref_fill.asp) | Fill the elements in an array with a static value |
| [filter()](https://www.w3schools.com/jsref/jsref_filter.asp) | Creates a new array with every element in an array that pass a test |
| [find()](https://www.w3schools.com/jsref/jsref_find.asp) | Returns the value of the first element in an array that pass a test |
| [findIndex()](https://www.w3schools.com/jsref/jsref_findindex.asp) | Returns the index of the first element in an array that pass a test |
| [forEach()](https://www.w3schools.com/jsref/jsref_foreach.asp) | Calls a function for each array element |
| [from()](https://www.w3schools.com/jsref/jsref_from.asp) | Creates an array from an object |
| [includes()](https://www.w3schools.com/jsref/jsref_includes_array.asp) | Check if an array contains the specified element |
| [indexOf()](https://www.w3schools.com/jsref/jsref_indexof_array.asp) | Search the array for an element and returns its position |
| [isArray()](https://www.w3schools.com/jsref/jsref_isarray.asp) | Checks whether an object is an array |
| [join()](https://www.w3schools.com/jsref/jsref_join.asp) | Joins all elements of an array into a string |

**Hoisting:**

Hoisting is JavaScript's default behaviour of moving declarations to the top.

**JavaScript Declarations are Hoisted**

In JavaScript, a variable can be declared after it has been used.

In other words; a variable can be used before it has been declared.

**Closures:**

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives you access to an outer function’s scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

**arrow funcition vs .bind(this):**

An **arrow function expression** is a compact alternative to a traditional [function expression](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function), but is limited and can't be used in all situations.

**Differences & Limitations:**

* Does not have its own bindings to [this](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this) or [super](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super), and should not be used as [methods](https://developer.mozilla.org/en-US/docs/Glossary/Method).
* Does not have [new.target](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new.target) keyword.
* Not suitable for [call](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call), [apply](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/apply) and [bind](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind) methods, which generally rely on establishing a [scope](https://developer.mozilla.org/en-US/docs/Glossary/Scope).
* Can not be used as [constructors](https://developer.mozilla.org/en-US/docs/Glossary/Constructor).
* Can not use [yield](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield), within its body.

The **bind()** method creates a new function that, when called, has its this keyword set to the provided value, with a given sequence of arguments preceding any provided when the new function is called.

**Day 2**

**Promises + callbacks + async await:**

I **promise** to do this **whenever** that is true. If it isn't true, then I won't.

This is a simple illustration of JavaScript Promises. Sounds like an IF statement? We’ll soon see a huge difference.

A promise is used to handle the asynchronous result of an operation. JavaScript is designed to not wait for an asynchrnous block of code to completely execute before other synchronous parts of the code can run. For instance, when making API requests to servers, we have no idea if these servers are offline or online, or how long it takes to process the server request.

With Promises, we can defer execution of a code block until an async request is completed. This way, other operations can keep running without interruption.

Promises have three states:

* **Pending**: This is the initial state of the Promise before an operation begins
* **Fulfilled**: This means the specified operation was completed
* **Rejected**: The operation did not complete; an error value is usually thrown

**Async:**

An async function is a modification to the syntax used in writing promises. You can call it syntactic sugar over promises. It only makes writing promises easier.

An async function returns a promise -- if the function returns a value, the promise will be resolved with the value, but if the async function throws an error, the promise is rejected with that value.

**Await:**

Await is only used with an async function. The await keyword is used in an async function to ensure that all promises returned in the async function are synchronized, ie. they wait for each other. Await eliminates the use of callbacks in .then() and .catch(). In using async and await, async is prepended when returning a promise, await is prepended when calling a promise. try and catch are also used to get the rejection value of an async function.

**callback hell:**

Callback hell is a **phenomenon that afflicts a JavaScript developer when he tries to execute multiple asynchronous operations one after the other**. **By nesting callbacks** in such a way, we easily end up with error-prone, hard to read, and hard to maintain code.

**Immutability:**

an immutable value is something that cannot be changed. Especially when we’re developing our applications, we might end up in some situations where we want to create a new object in our code, containing a new property or value while also maintaining the original value. The concept of immutability can help us to create new objects, making sure that we’re not changing the original value.

In JavaScript, we have primitive types and reference types. Primitive types include numbers, strings, boolean, null, undefined. And reference types include objects, arrays and functions.

The difference between those types is that the primitive types are immutable (or unchangeable), and the reference types are mutable (changeable). For example, the string type is immutable:

**Shallow vs deep copy JS:**

**Spread operator:**

The **spread operator** is a new addition to the set of operators in JavaScript ES6. It takes in an iterable (e.g an array) and expands it into individual elements.

The spread operator is commonly used to make shallow copies of JS objects. Using this operator makes the code concise and enhances its readability.

**Rest parameters:**

A function definition's last parameter can be prefixed with "..." (three U+002E FULL STOP characters), which will cause all remaining (user supplied) parameters to be placed within a ["standard" JavaScript array.](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array). Only the last parameter in a function definition can be a rest parameter.

**Day 3**

**Lexical scope**

A lexical scope in JavaScript means that **a variable defined outside a function can be accessible inside another function defined after the variable declaration**. But the opposite is not true; the variables defined inside a function will not be accessible outside that function.

he lexical scope allows a function scope to access statically the variables from the outer scopes. Finally, a closure is a **function that captures variables from its lexical scope**.

**Web APIs (setTimeout, setInterval):**

We may decide to execute a function not right now, but at a certain time later. That’s called “scheduling a call”.

There are two methods for it:

* setTimeout allows us to run a function once after the interval of time.
* setInterval allows us to run a function repeatedly, starting after the interval of time, then repeating continuously at that interval.

These methods are not a part of JavaScript specification. But most environments have the internal scheduler and provide these methods. In particular, they are supported in all browsers and Node.js.

**Eventloop:**

JavaScript has a concurrency model based on an **event loop**, which is responsible for executing the code, collecting and processing events, and executing queued sub-tasks. This model is quite different from models in other languages like C and Java.

**JS is a single threaded async concurrent language:**

Javascript is a single threaded language. This means it has one call stack and one memory heap. As expected, it executes code in order and must finish executing a piece code before moving onto the next. It's synchronous, but at times that can be harmful. For example, if a function takes awhile to execute or has to wait on something, it freezes everything up in the meanwhile.

**Diff b/w ES5 -> ES6:**

|  |  |  |
| --- | --- | --- |
| **Based on** | **ES5** | **ES6** |
| **Definition** | ES5 is the fifth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International) | ES6 is the sixth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International). |
| **Release** | It was introduced in 2009. | It was introduced in 2015. |
| **Data-types** | ES5 supports primitive data types that are **string, number, boolean, null,** and **undefined**. | In ES6, there are some additions to JavaScript data types. It introduced a new primitive data type **'symbol'** for supporting unique values. |
| **Defining Variables** | In ES5, we could only define the variables by using the **var** keyword. | In ES6, there are two new ways to define variables that are **let** and **const**. |
| **Performance** | As ES5 is prior to ES6, there is a non-presence of some features, so it has a lower performance than ES6. | Because of new features and the shorthand storage implementation ES6 has a higher performance than ES5. |
| **Support** | A wide range of communities supports it. | It also has a lot of community support, but it is lesser than ES5. |
| **Object Manipulation** | ES5 is time-consuming than ES6. | Due to destructuring and speed operators, object manipulation can be processed more smoothly in ES6. |
| **Arrow Functions** | In ES5, both **function** and **return** keywords are used to define a function. | An arrow function is a new feature introduced in ES6 by which we don't require the **function** keyword to define the function. |
| **Loops** | In ES5, there is a use of **for** loop to iterate over elements. | ES6 introduced the concept of **for...of** loop to perform an iteration over the values of the iterable objects. |

**React**

**Day 4**

**What is React and how it works:**

React is a JavaScript library (not a framework) that **creates user interfaces (UIs)** in a predictable and efficient way using declarative code. You can use it to help build single page applications and mobile apps, or to build complex apps if you utilise it with other libraries.

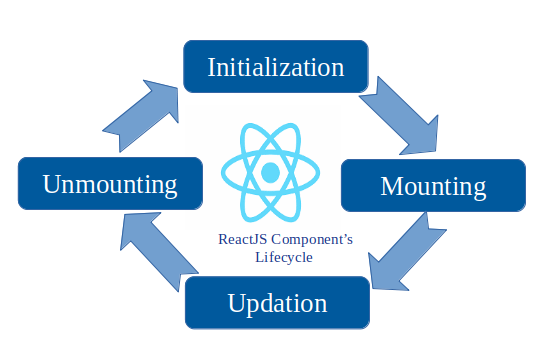
**Virtual DOM vs Real DOM:**

A virtual DOM is a lightweight JavaScript representation of the DOM used in declarative web frameworks such as React, Vue.js, and Elm. Updating the virtual DOM is comparatively faster than updating the actual DOM.

**State vs props:**

* Components receive data from outside with props, whereas they can create and manage their own data with state
* Props are used to pass data, whereas state is for managing data
* Data from props is read-only, and cannot be modified by a component that is receiving it from outside
* State data can be modified by its own component, but is private (cannot be accessed from outside)
* Props can only be passed from parent component to child (unidirectional flow)
* Modifying state should happen with the setState ( ) method

**React Component LifeCycle;**



**Diff b/w class functional components:**

|  |  |
| --- | --- |
| **Functional Components** | **Class Components** |
| A functional component is just a plain JavaScript function that accepts props as an argument and returns a React element. | A class component requires you to extend from React. Component and create a render function which returns a React element. |
| There is no render method used in functional components. | It must have the render() method returning HTML |
| Also known as Stateless components as they simply accept data and display them in some form, that they are mainly responsible for rendering UI. | Also known as Stateful components because they implement logic and state. |
| React lifecycle methods (for example, componentDidMount) cannot be used in functional components. | React lifecycle methods can be used inside class components (for example, componentDidMount). |

**Day 5**

**Initialize a React App using create-react-app (Zainab see react-native init instead)**

**1. Hello World**

**2. Introducing JSX**

**3. Rendering Elements**

Rendering is a **process that is triggered by a change of state in some component of your application**, when a state change occurs React: It will collect from the root of your App all the components that requested a re-render because their state or their props changed.

**4. Components and Props**

**5. State and Lifecycle**

**6. Handling Events**

**How do we style React Components**

**Day 6**

**7. Conditional Rendering**

**8. Lists and Keys**

**9. Forms**

**10. Lifting State Up**

**11. Composition vs Inheritance**

**12. Thinking In React**

**Day 7**

**React Router (Zainab see React Navigation v5 instead)**

**Implement React Router and create atleast 3 routes (Zainab add Screens use StackNavigators instead)**

**What is a single Page Application**

**Network request (Fetch & axios)**

**Use an Open API and consume it in your React Application using axios**

**Basic hooks**

**useState**

**useEffect**

**useRef**

**Implement these hooks in your react application**

**Learn how to implement mount, update and unmount callbacks using useEffect**

**Day 8**

**React Advanced Concepts**

**Code-Splitting (Zainab see bundles for different architectures instead, see how do we make builds for Android & iOS)**

**Context**

**Error Boundaries**

**Forwarding Refs**

**Fragments**

**Higher-Order Components**

**Integrating with Other Libraries**

**JSX In Depth**

**Day 9**

**Reconciliation**

**Refs and the DOM**

**Render Props**

**Static Type Checking**

**Typechecking With PropTypes**

**Uncontrolled Components**

**Day 10**

**Create a signup Form in React**

**Fields =>**

**username**

**email**

**password**

**confirmPassword**

**Implement Formik and Yup for form validation**

**Day 11**

**Flux**

**Learn Redux**

**Implement Redux Thunk in your React Application**

**Redux persist (whitelist & blacklist)**

**General**

**Day 12**

**npm vs yarn**

**what is npx**

**Babel**

**Webpack**

**Webpack Plugins and try to install some new and update some existing ones in CRA**

**What is Gulp and Grunt(Just Overview)**

**Https vs Http**

**NEXTJS**

**Day 13**

**SSR vs CSR**

**Create a NextJS Hello World App**

**Learn Routing in NextJS**

**Serve static Assets using public directory**

**Create a listing Page in Next JS with a search bar and results**

**(Will add the exact project requirements and APIs)**

**Day 14**

**Project continued**

**Day 15**

**Project continued**

**API Routes**

**Browser**

**Day 16**

**PWA + Service Workers**

**Caching**

**Network Request life cycle**

**SEO (high level) + google page indexing**

**How Nextjs can be helpful for SEO**

**Local storage and cookie**

**Redirection**

**CDN**

**Request Statuses**

**CORS**