**JavaScript Notes**

**Data Types**

* **Primitives:** When you access a primitive type you work directly on its value are string, number, boolean, null, undefined, symbol, bigint.
* **Complex**: When you access a complex type you work on a reference to its value are object, array, function.

**Rules for choosing variables names**

* Letter, digits, underscores, & $ sign allowed.
* Must begin with a $, \_ or a letter.
* JavaScript reserved words cannot be used as a variable name.

**var vs let in JavaScript**

* var is globally scoped while let & const are block scoped.
* var can be updated & re-declared within its scope.
* let can be updated but not re-declared.
* const can neither be updated nor be re-declared.
* const must be initialized during declaration unlike let and var.

**Template Literals**

* It is a literals delimited with backtick(`) characters, allowing for multi-line strings, string interpolation with embedded expressions, and special constructs called tagged templates.

**DOM Manipulating**

* DOM (Document Object Model) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.
* **addEventListener():** This method allows you to set up functions to be called when a specified event happens, such as when a user clicks a button.

**Creating and placing new nodes**

* **querySelector():** The Document method querySelector() returns the first Element within the document that matches the specified selector, or group of selectors. If no matches are found, null is returned.
* **textContents**: It is all text contained by an element and all its children that are for formatting purposes only.
* **Document.createElement**: The document.createElement() method creates the HTML element specified by tagName.
* **Node.appendChild():** The appendChild() method of the Node interface adds a node to the end of the list of children of a specified parent node. If the given child is a reference to an existing node in the document, appendChild() moves it from its current position to the new position.
* **Document.createTextNode():** create a new text node. This method can be used to escape HTML characters.

**Moving and removing elements**

* **Ṇode.removeChild():** The removeChild() method of the Node interface removes a child node from the DOM and returns the removed node.
* **Element.remove():** This method removes the element from the DOM.

**Manipulating styles**

* **Document.styleSheets:** The styleSheets read-only property of the Document interface returns a StyleSheetList of CSSStyleSheet objects, for stylesheets explicitly linked into or embedded in a document.
* **Element.setAttribute():** this takes two arguments, the attribute you want to set on the element, and the value you want to set it to. Example: Element.setAttribute(‘class’, ‘className’)

**Asynchronous Actions**

* Asynchronous actions are the actions that we initiate now and they finish later. Eg. setTimeOut

**Callback funcitons**

* It is a function passed into another function as an argument which is then invoked inside the outer function to complete an action.

**Pyramid of Doom**

* When we have callback inside callback , the code gets difficult to manage.

**Promises**

* The Solution of a callback hell is promises.
* A promises is a ‘promise of code execution’. The code either executes or fails, in both the cases the users will be notified.

**Syntax:**

let promise = new Promise(function(resolve, reject) { //execute})

* **then function** – Its runs when the promise is resolved or reject and receives the value.
* **catch function** – The call .catch(f) is a complete analog of .then(null, f), it’s just a shorthand.
* **Finally function** – The call .finally(f) is similar to .then(f, f) in the sense that f runs always, when the promise is settled: be it resolve or reject.

**Async/await**

* **Async functions -** There’s a special syntax to work with promises in a more comfortable fashion, called “async/await”. It’s surprisingly easy to understand and use.
* The keyword **await** makes JavaScript wait until that promise settles and returns its result.

**JavaScript Cookies**

* Cookies are small strings of data stored directly in the browser.

**Local Storage**

* Local Storage is a data storage of web storage.
* This allows the javaScript sites and apps to store and access the data without any expiration date.

**Session Storage**

* Session Storage exists only within the current browser tabs. Another tab with some page will have a different storage.
* The data survives page refresh, but not closing/opening tabs.

**TypeScript**

* Typescript is the superset of JavaScript. It is a typed superset of JavaScript that compiles to plain JavaScript.
* Goal: writing Typed JavaScript.
* 2 Main Pros:
  + Type checking errors => catch before they occur in production.
  + Providing better documentation => makes it easier for you and others to understand the code.
* Features:
  + Strong/static typing (var msg:string = “Hello World”)
  + Object Oriented

**Type Assertion**

* It is a mechanism which tells the compiler about the type of a variable. It has two forms:
  + Angle-brackets syntax: **(<DataType>VariableName)**
  + As syntax: Variable **as** DataType

**Interface in TypeScript**

* Capable of describing the wide range of shapes that JavaScript objects can take. **Syntax:**

interface detailsInfo {

    firstName:string,

    lastName:string,

}

let namets = (fullDetails:detailsInfo) => {

    console.log(fullDetails.firstName + " " + fullDetails.lastName)

}

namets({

    firstName: "Anuj",

    lastName: "Kesarwani",

})