**JAVASCRIPT**

* **General**
  + Always use **script tag** for JS. It can be **inside head or body**.
  + We can also add JS code in another file with extension of .js, then by source attribute in the script tag we can import it by using filename and path/link.
  + You can skip the window keyword(non-strict mode) as in JavaScript, the window object is the global scope object, that means that variables, properties, and methods by default belong to the window object. This also means that specifying the window keyword is optional.
  + Semicolons after every line is not mandatory but it is highly recommended.
  + When adding a number and a string, JavaScript will treat the number as a string.
  + JavaScript has **dynamic types or loosely type language/weakly typed language** . This means that the same variable can be used to hold different data types.

Ex

var x; // Now x is undefined

x = 5; // Now x is a Number

x = "John"; // Now x is a String

* + An empty value has nothing to do with undefined.
  + Undefined and null are equal in value but different in type.

typeof undefined // undefined

typeof null // object

null === undefined // false

null == undefined // true

* + Comparing two JavaScript objects will always return false.
  + In JavaScript there are 6 different data types that can contain values:
    - string
    - number
    - boolean
    - object
    - BigInt
    - symbol
  + And 2 data types that cannot contain values:
    - null
    - undefined
  + There are only six falsey values in JavaScript: undefined , null , NaN , 0 , "" (empty string), and false
  + A **polyfill** is a piece of code (usually JavaScript on the Web) used to provide modern functionality on older browsers that do not natively support it.
  + JavaScript uses the \ (backslash) as an escape character for:
    - \' single quote
    - \" double quote
    - \\ backslash
    - \n newline
    - \r carriage return
    - \t tab
    - \b backspace
    - \f form feed
    - \v vertical tab (Internet Explorer 9 and older treats '\v as 'v instead of a vertical tab ('\x0B). If cross-browser compatibility is a concern, use \x0B instead of \v.)
    - \0 null character (U+0000 NULL) (only if the next character is not a decimal digit; else it is an octal escape sequence)
    - \xFF character represented by the hexadecimal byte "FF"
    - Note that the \v and \0 escapes are not allowed in JSON strings.
* **JavaScript Output**
  + Writing into an HTML element, using **innerHTML**.
    - To access an HTML element, JavaScript can use the **document. getElementById(id**) method.
    - The id attribute defines the HTML element. The innerHTML property defines the HTML **content (text)**.
    - Changing the innerHTML property of an HTML element is a common way to display data in HTML.
    - EX.

document.getElementById("demo").innerHTML = "Hello Dolly."

* + Writing into the HTML output using **document.write ()**.
    - For testing purposes, it is convenient to use document.write().
    - It works same as innerHTML but for testing only.
    - Using document.write() after an HTML document is loaded, will delete all existing HTML.
  + Writing into an alert box, using **window.alert()**.
    - You can use an alert box to display data.
  + Writing into the browser console, using **console.log()**.
    - For debugging purposes, you can call the console.log() method in the browser to display data.
  + Printing a page, using **window.print()**.
    - JavaScript does not have any print object or print methods.
    - You cannot access output devices from JavaScript.
    - The only exception is that you can call the window.print() method in the browser to print the content of the current window.
* **Scope**
  + Scoping asks the question "Where do variables live?" or "Where can we access a certain variable, and where not?"
  + There are 3 types of scope in JavaScript: the global scope, scopes defined by functions, and scopes defined by blocks,
  + Only let and const variables are block scoped. Variables declared with var end up in the closest function scope;
  + In JavaScript, we have lexical scoping, so the rules of where we can access variables are based on exactly where in the code functions and blocks are written;
  + Every scope always has access to all the variables from all its outer scopes. This is the scope chain!
  + When a variable is not in the current scope, the engine looks up in the scope chain until it finds the variable it's looking for. This is called variable lookup.
  + The scope chain is a one-way street a scope will never ever have access to the variables of an inner scope:
  + The scope chain in a certain scope is equal to adding together all the variable environments of the all-parent scopes.
  + The scope chain has nothing to do with the order in which functions were called. It does not affect the scope chain at all!
  + Graphical user interface, website

    Description automatically generated
  + Graphical user interface, text, application, email

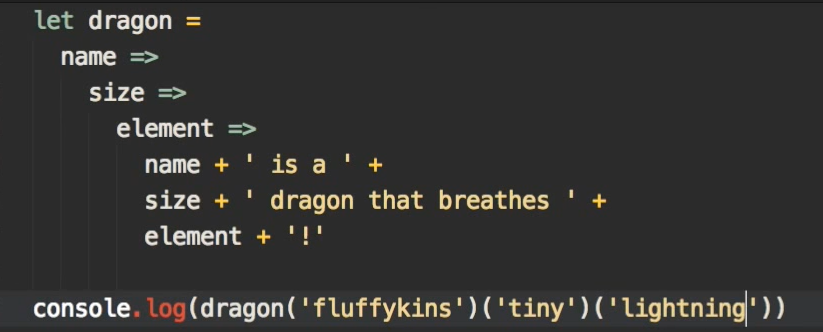
    Description automatically generated
  + **Block Scope**
    - **Block {}:** It is used to combine multiple statement and use them to the place where JS requires single statement. For ex if statement requires a single statement after condition, so we use block.
    - It is also known as Compound statement. It groups multiple statements.
    - What all variables and functions we can access inside a block is called block scope.
    - Let and const are blocked scope and each block has its own memory space, but var is not blocked scope and it gets attached to window object.
    - Graphical user interface, text, application, email

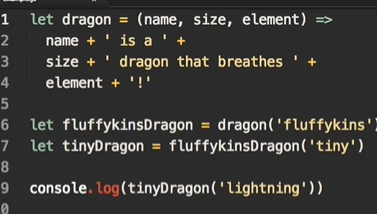
      Description automatically generated
  + **Shadowing**
    - It is a concept of overwriting a variables value inside a scope.
    - For let and const it works for all the scopes whereas for a var it works only for function scope.
    - Ex for var ->
      * A screenshot of a computer

        Description automatically generated with medium confidence
      * var value in line 10 will be 10 as it shared same global space as var is not blocked scope.
    - Ex for let/const ->
      * A screenshot of a computer

        Description automatically generated with medium confidence
      * Let value in line 10 will be 100 as it is in a different block
    - We cannot shadow a let variable with a var variable in a block as it will give syntax error but vice versa is possible because var is not block scoped but for the reverse let is blocked scope so it can shadow var.
* **Scope Chain**
  + Scope is where one can access a function or a variable in js.
  + Scope is directly dependent on a lexical environment which is local memory + memory of parent.
  + Meaning variable and functions are available in parent to child hierarchy.
  + In other words, a child block can access its parent functions and variables, but a parent cannot access its child function and variables.
  + Or whenever a new Execution environment is created it can access its local variable as well as its parent variables and continuing this for many execution environments makes a scope chain.
  + Parent(Lexical Environment) of Global Execution Environment is null.
  + The order in which function is called doesn’t affect the scope chain only where the function was declared does. For Eg in below image second() called third() still third will not get c and b value. This is known as lexical scoping.
  + Diagram

    Description automatically generated
* **Curring**





* Arrays (which method to use when)
  + Graphical user interface, application

    Description automatically generated
* **DOM**
  + Timeline

    Description automatically generated
* Graphical user interface, text

  Description automatically generated