JAVASCRIPT BASICS

* Javascript on client side:
* Js works along with HTML and CSS, HTML adds structure to web page, CSS styles it and Javascripte brings it to life by allowing users to interact with elements on the page, such as actions on clicking buttons, filling out forms and showing animations.
* Js on the client side is directly executed in the user’s browser.
* Applications of javascript:
* Js is used in a wide range of applications, from enhancing websites to building complex applications. Here are some example:
* Web Development: JS adds interactivity and dynamic behavior to static websites, with popular frameworks like **AngularJs** enhancing development.
* Web Applications: JS powers robust web applications, leveraging APIs,React and Electron to create dynamic user experiences like google maps.
* Server Applications**: NodeJs** brings Javascript to the server side, enabling powerful server applications and full-stack development.
* Game Development: LJS combined with HTML5 and libraries like **Ease Js**, enables the creation of interactive games for the web.
* Smartwatches: **Pebble JS** allows javascript to run on smartwatches supporting apps that require internet connectivity.

1. Variables and Datatypes in Javascript:

* Variables and data types are foundational concepts in programming, serving as the building blocks for storing and manipulating information within a program. In JavaScript, getting a good grasp of these concepts is important for writing code that works well and is easy to understand

1. Variables:

* A variable is like a container that holds data that can be reused or updated later in the program. In JavaScript, variables are declared using the keywords[var](https://www.geeksforgeeks.org/javascript-var/), [let](https://www.geeksforgeeks.org/javascript-let/), or [const](https://www.geeksforgeeks.org/javascript-const/).

1. Var keyword:- This is the older way to declare variables. Variables declared with var are function-scoped or globally scoped, meaning their accessibility can be broad, potentially leading to issues in larger codebases.
2. Let keyword:- Introduced in ES6, let provides block-scoped variables. This means a variable declared with let is only accessible within the block of code (e.g., inside an if statement or a for loop) where it's defined. let variables can be reassigned.
3. Const keyword:- Also introduced in ES6, const declares block-scoped constants. Once a value is assigned to a const variable, it cannot be reassigned. This is ideal for values that should not change during the program's execution.ell.
4. Datatypes:

* JavaScript supports various datatypes, which can be broadly categorized into primitive and non-primitive types.

1. Primitive Datatypes - Primitive datatypes represent single values and are immutable.
2. Number- For numerical values, including integers and floating-point numbers (e.g., 10, 3.14).
3. String- For textual data, enclosed in single or double quotes (e.g., "Hello World", 'JavaScript')
4. Boolean- Represents a logical entity with two possible values: true or false
5. Undefined- Represents a variable that has been declared but has not yet been assigned a value.
6. Null- Represents the intentional absence of any object value. It's a primitive value.
7. Symbol- Introduced in ES6, symbols are unique and immutable primitive values, often used for unique object property keys.
8. BigInt- For integer numbers of arbitrary length, exceeding the limit of the number type.
9. Non- Primitive Datatypes - Non-primitive types are objects and can store collections of data or more complex entities.
10. Object
11. Array
12. Function
13. Numbers:
    * JavaScript has a number type that can represent both integers and floating-point numbers.
14. String:
    * A **char** is a single character (For example: 1, 6, %, b, p, ., T, etc.)
    * The **string** type is a special type that consists of multiple **char**s.
    * To initialize a string value in a variable, enclose it within single or double quotation marks:
15. Boolean:

* A **bool** (Boolean) type has only 2 possible values: true or false.
* Booleans are building blocks for logic in the programs we write.

1. Naming conventions:

* In JavaScript, there are certain rules and conventions for naming variables:
* Variable names are case-sensitive, it means MyVariable & Myvariable are considered different.
* Variable names can only contain letters(a-z, A-Z), numbers(0-9), underscores(\_), and dollar signs($).
* Variable name must starts with a letters, underscore or dollar sign, they can’t begin with a number.
* Certain words are reserved by javascript and can’t be used as variable names such as **let, const, function, if, else etc**..
* It’s a common practice to use camelCase for variable names, where each word except the 1st starts with a capital letter(e.g., **m**y**V**ariable**N**ame).

1. Empty Variables:

* In JavaScript, it's possible to declare variables without assigning them a specific value. This can be useful when you know you'll use a variable later in your code, but you don't yet know its initial value.
* To declare an empty variable in JavaScript, you can use the let keyword followed by the variable name, without assigning any value
* creates a variable named myVariable but doesn't assign any value to it. The variable is said to be **uninitialized.**

1. Constants:

* In JavaScript, sometimes you need variables that *never change*. This is where const comes in! It’s used to create constants—values that stay the same after being set.
* Use the const keyword, just like let, but once you set its value, you **cannot change it** later.

1. Concept of truthy & falsy value in Javascript and how it relates to javascript Booleans:

* Truthy values:
  + A value is considered "truthy" if it evaluates to true when coerced to a Boolean. Most values in JavaScript are truthy, including:
    - Non-zero numbers (e.g. `1`, `-1`, `3.14`).
    - Non-empty strings (e.g.`”hello”`, ””`).
    - Objects (e.g. `{ }`, `[ ]`).
    - Functions.
* Falsy values:
* A value is considered "falsy" if it evaluates to false when coerced to a Boolean. The following values are falsy in JavaScript:
  + `false`.
  + `0` (zero).
  + `” ”`(empty string).
  + `null`.
  + `undefined`.
  + `NaN` (Not-a-Number).

1. Operators: part-1
2. Arithmetic operators: ~it performs mathematical calculations.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Operation** | **Example** |
| **+** | Addition | 3 + 2 = 5 |
| **-** | Subtraction | 3 – 2 =1 |
| **\*** | Multiplication | 3 \* 2 = 6 |
| **/** | Division | 4 / 2 = 2 |

* Arithmetic shortcut Assignment operators:
* Javascript created a shortcut for self-arithmetic operations.
* This operation is valid for all arithmetic operations:

|  |  |
| --- | --- |
| **Operator** | **Shortcut** |
| **+** | **+=** |
| **-** | **-=** |
| **\*** | **\*=** |
| **/** | **/=** |
| **%** | **%=** |

* Modulo Operator:
* The Modulo operator % tells you what’s left over after dividing one number by another. Result = dividend % divisor.
* **Dividend**: The number being divided.
* **Divisor**: The number that divides the dividend.
* **Result**: The reminder of the division

1. Comparison operator:

* Comparison operators are used to compare two operands.
* Sometimes we need to check whether an operand is bigger/smaller/... than another operand. The following table shows possible operators for comparison:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| **==** | Equal | 1 == 2 returns **false** |
| **!=** | Not Equal | 1 != 2 returns **true** |
| **>** | Greater Than | 1 > 2 returns **false** |
| **<** | Lower Than | 1 < 2 returns **true** |
| **>=** | Greater or Equal | 1 >= 2 returns **false** |
| **<=** | Lower or Equal | 1 <= 2 returns **true** |

* Strict vs Loose Equality:
* In js there are two types of equality operators:

1. Loose Equality (==):

* Loose Equality (==) check if two values are equal after performing type coercion.
* This means if the operands have different types, Javascript will try to convert them to a common type before making the comparison.

1. Strict Equality (===):

* Strict Equality (===), on the other hand, checks if two values are equal without performing any type conversion.
* If operands have different types, they are considered not Equal.
* it's recommended to use strict equality (===) in most cases to avoid unexpected results due to type coercion.
* Similarly, there are also loose inequality (!=) and strict inequality (!==) operators that work in a similar way, but check
* for inequality instead of equality.
* Loose inequality (!=) checks if two values are not equal after performing type coercion, while strict inequality (!==) checks if two values are not equal without performing type conversion.

1. Operators: part-2
2. Logical operators part-1

* Logical operators are used to check combinations of comparisons that return true or false.
* For example the following statement contains two comparisons:
* Is 5 greater than 3 **and** less than 6?

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| **&&** | And- true if all operands are true | a && b |
| **||** | Or- true if any operands are true | a || b |
| **!** | Not- true if operands is false | !a |