JAVASCRIPT

* Introduction:
* JavaScript is a high-level, Object-oriented, Multi-paradigm programming language.
* JavaScript allows us to add dynamic effects in pages.
* And it is also use for building entire web applications in the browser.
* Inline JavaScript:
* Inline JavaScript means the JavaScript written in HTML file.
* External JavaScript:
* There is separate JavaScript file with all logic having .js extension and attach in html file with script tag.
* This file make code modular and clear.
* Syntax: <script src=” filename.js”></script>
* **Values and Variable:**
* Values are an information or data.
* Variable are block whose store this value or information or data.
* Variable name should be camelCase
* Variable only contains letters, numbers, underscore ’\_’ , $ sign.
* Variable should not start with numbers. Eg: 3name = ‘hii’; ❌
* Variable may start with Underscore and dollar $
* For constant variable write variable in UPPERCASE. Eg: PI = 3.14 ✅
* **Data Types:**
* JavaScript has a dynamic typing -> We do not have manually define the data type of the value stored in a variable. Instead, data types are determined automatically.
* To check data type of variable we use **typeof** operator. Eg: **typeof** name

1. **Primitive Data Types:**
   1. **Numbers:** Floating point numbers -> Used for decimals and integers. Eg: let age = 23;
   2. **String:** Sequence of characters -> Used for text. Eg: let name = “js”;
   3. **Boolean:** Logical type that can only be true or false -> Used for decision taking. Eg: let isLogin = true;
   4. **Undefined:** Value taken by a variable that is not yet defined (‘empty value’) Eg: let children;
   * If we only declare variable the value and type both are undefined.
   1. **Null:** Also means ‘empty value’.
      * **NOTE:**  Null in JavaScript has a typeof **object,** but it is a biggest bug in JavaScript language.
   2. **Symbol:** Value that is unique and cannot be changed.
   3. **BigInt:** Larger integer than the Number type can told.
2. **Non – Primitive Data Types:**

* **Dynamic typing:**

*let* lastName = "Jangale";

console.log(typeof lastName);  *//String*

lastName = 100;

console.log(typeof lastName); *//number*

* String can be converted to number dynamically.
* **Let, const and var:  
  1. Var:** 
  + Var is oldest keyword in JavaScript for declaring a variable.
  + It has a global scoped or function scope that means variable defined outside a function can be accessed globally and variable defined inside a particular function can be accessed within the function.
  + Eg:

A computer screen with colorful text

Description automatically generated

* + We can re-declare a variable with same name in the same scope using var keyword, which give no error in case of var keyword

Eg:

A screenshot of a computer

Description automatically generated

1. **Let:** 
   * The let keyword is an improved version of the var keyword.
   * It is introduced in ES6 or ECMAScript 2015.
   * These variables have the block scope. It can’t be accessible outside the particular code block.

Eg:

**A computer screen with text and numbers

Description automatically generated**

* + Redeclaration in same scope of let variables is not allowed in JavaScript and it is the biggest advantages of let variables over var variables.

**A black background with colorful text

Description automatically generated**

* + But redeclaration in different scope is allowed in JavaScript.

Eg.

**A computer screen shot of a code

Description automatically generated**

* + We can modify the let value after declaring the value.

1. **Const keyword:** 
   * Const has all the properties that are the same as the let keyword, except the user cannot update it and must assign it with a value at the time of declaration.
   * These variables also have the block scope.
   * It is mainly used to create constant variables whose values cannot be changed once they are initialized with a value like value of PI.

Eg:

**A computer screen with colorful text

Description automatically generated**

**It gives TypeError: Assignment to constant variables.**

* **Difference between let, var and const variables:**

|  |  |  |
| --- | --- | --- |
| **var** | **let** | **const** |
| Scope is functional or global | Scope is block scope only. | Scope is block scope only. |
| It can be updated and re-declared in the same scopes. (Mutable) | It can be updated but cannot be re-declared in the same scope. (Mutable) | It cannot be updated or re-declared in any scope.  (Immutable) |
| It can be declared without initialization. | It can be declared without initialization. | It cannot be declared without initialization. |
| It can be accessed without initialization as its default value is “undefined”. | It cannot be accessed without initialization otherwise it will give ‘referenceError’ | It cannot be accessed without initialization, as it cannot be declared without initialization. |
| These variables are hoisted. | These variables are hoisted but stay in the temporal dead zone until the initialization. | These variables are hoisted but stays in the temporal dead zone until the initialization. |

* **NOTE:** Don’t use var keyword for declaring variable because var is outdated variable and it is not safe because it has a global scope. And there is no error occurs when we redeclare var variables with same name.
* **Global Variable:**- If we declare variable without using var, let and const this variable consider as a global scope variable. We can access this variable from out off the scope and also inside the other scope.

**Eg:**

**A screen shot of a computer program

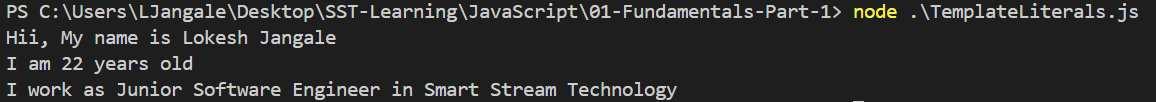
Description automatically generated**

* But you should not create variable without var, let and const keyword until and unless any requirement available.
* **Template Literals:**
* If we want to insert variable in string, we use backticks (``) and pass variable in ${}.
* Template literals allowing us for multi-line string or string interpolation with embedded expressions.

Eg:

A screen shot of a computer program

Description automatically generated

Output:  
 

* **Type Conversion:**
* JavaScript provide a way to convert data type manually.

1. **String to Number:**
   * Syntax: Number(value)
   * If we contain a string having number and we want to convert this string to number we use these syntax.

Eg:

let value = “23”;  
console.log (Number(value)) //output: 23🡪number

* + If variable contain string as a word the conversion output is NaN (Not a Number)

Eg:

let value = “Lokesh”;

console.log (Number(value)) //output: NaN -> number

1. **Number to String:**
   * Syntax: String(value);
   * I we want to convert number to string then we use above syntax.

* **Type coercion:  
  -** Type coercion refers to the process of automatic or implicit conversion of value from one data type to another data type.
* This includes conversion from Number to String, String to Number, Boolean to number etc.
* For String if we use ‘+’ then numberString is concatenated but if we use ‘-‘ then string is converted to number.

Eg:

|  |  |
| --- | --- |
| A screen shot of a computer screen  Description automatically generated | A black screen with yellow numbers  Description automatically generated |

* **Falsy value:**
  + In JavaScript there are 5 falsy values are available [ 0, undefined, ‘ ’ , NaN, null].
  + These 5 falsy values are used for false conditions.
* **Equality Operator == vs ===:**

1. **=== or Strict Equality operator:**
   * It is checking value and also data type of value are equal or not.
   * It does not perform type coercion.

Eg:

console.log(18 === 18)  //true

console.log ('18' === 18) //false

1. **== or Loose Equality Operator:**
   * It is checking value but not check data type of value.
   * It performs type coercion.
   * That’s why sometime bugs found

Eg:

console.log (18 == 18) //true

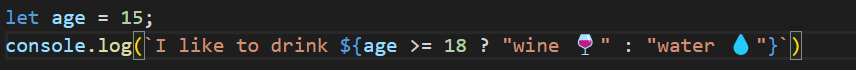
console.log ('18' == 18) //true

* **Note:** Always use strict equality operator (===) to check equality for value until and unless if any particular requirement are not present for ==.
* **Ternary Operator (?):**
* It is a conditional operator which is used to check conditions true or false.
* It works as a single line if – else statement.
* Syntax:

**Condition ? true statement : false statement;**

* We call it a ternary operator because it has a tree main type condition, true statement and false statement.
* We can use ternary operators in template literals but we cannot use if-else statement in template literals.

Eg:

****

* **Strict Mode:**
* Being a scripting language, sometimes the JavaScript code displays the code displays the correct result even it has some errors. Hence this is cause a bug In a program. To overcome this problem we can use the JavaScript strict mode.
* JavaScript provides “**use strict**”; expression to enable the strict mode. If there is any silent error or mistake in the code, it throws an error.
* We can write more safe code using strict mode.
* By using strict mode we can perform strict type checking in code. And it avoid global variable declaration
* The purpose of "use strict" is to indicate that the code should be executed in "strict mode".
* With strict mode, you cannot use undeclared variables or we cannot declare variable without let, const and var.

Eg:

**A computer screen with text

Description automatically generated**

* **Function:**
* Function is a piece of code which we can use again and again by just calling that function.
* We can also pass arguments to function and function can also return some value.
* This is also called as function declaration.
* Functions allow us to write more maintainable code.
  + Syntax:  
    **function** funcName(**param**){} //Function Declaration

funcName(val); //Function Calling

Eg:

A screen shot of a computer program

Description automatically generated

* **Types of Function:**

1. **Anonymous Function:**
   * Function without name is called Anonymous Function.
   * This function is declaring as variable declaration
   * This is also called as function expression.
   * Syntax:

**const** funcName = **function (param){}**

**Eg:**

const calAge = function (param) {
    return 2024 - param;
}
const age = calAge(2002);
console.log(age);

* **Note:** The main difference between function declaration and function expression is in function declaration we can call function before declaration but in function expression we cannot call function before declaration.

1. **Function Declaration:**A screen shot of a computer code

   Description automatically generated

Output:  
A screen shot of a computer

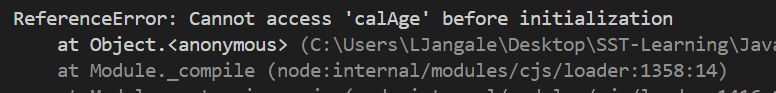
Description automatically generated

* This function can execute before declaration
* We cannot use function declaration for callback function.

1. **Function Expression**:  
   A computer screen shot of text

   Description automatically generated

Output:



* This function is not execute before declaration
* We can use function expression for callback function.

1. **Arrow Function:**
   * It is concise way of writing JavaScript functions in shorter way.
   * They make our code is more structured and readable.
   * Arrow function is anonymous function that is function without a name but they are often assigned to any variable.
   * They are also called as **Lambda function.**
   * Syntax:  
     **const** funcName = () => {}
   * The return statement and function brackets are optional for single-line functions.

Eg:

A computer screen shot of text

Description automatically generated

* **All Function type in one image:** 👇

A screenshot of a computer program

Description automatically generated

* **Array:**
* An array is a group of similar elements or a data item of the same type collected at contiguous memory locations.
* Syntax:  
  **const** array\_name = [item1, item2, ...];

Or

**const** array\_name = **new Array(**item1, item2, …**);**

**Eg:**

const friends = ["Lokesh", "Yeshwant", "Vishal"];

console.log(friends);

//Another way to declare array by using new keyword

const year = new Array(2001, 2002, 2003, 2004);

console.log(year);

//Find the length of array

console.log(friends.length);

//Find the last element of array

console.log(friends[friends.length - 1])

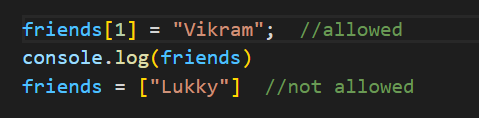
//Change the last element of array

friends[friends.length - 1] = "Karan";

console.log(friends);

* **Note:** If array is declared as a **const** we can modify the element of array but we cannot assign new array to that variable

**Eg:**



**Output:**A screen shot of a computer screen

Description automatically generated

* **Object:**
* Object is a most important data type in JavaScript.
* Objects are store value as like array but in different way.
* Array can store value in the form of index but object can store value is in the form of key-value pair.
* We can access element of array using index but we can access element of object by there keys.
* Object is like container in JavaScript which can hold multiple value in it. These value are store as a properties of object with it own key
* Syntax:  
  **const** object\_name = {

**Key : value,**

**….**

}

* We can access key by using dot (**.**) or square brackets (**[‘Key’]**)

Eg:

console.log(lokesh.firstName);

console.log(lokesh['firstName'])

* The main difference in between dot and square brackets we can pass the expression in brackets, but we can’t pass expression in dot.

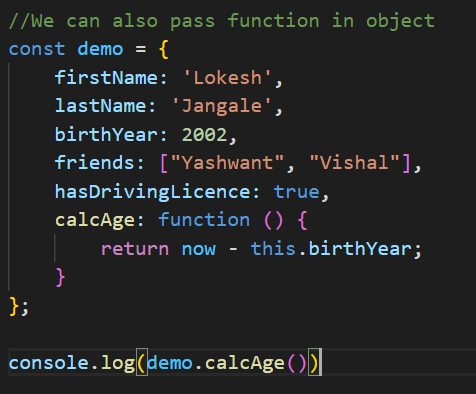
Eg:

A black background with white text and red and blue letters

Description automatically generated

* If we want to find value through any expression then only use bracket notation otherwise use dot notation it give more cleaner and simple code
* We can also pass function as a property in object

Eg:



* **Note:** **this.** Represent current object.
* **Type of console. :**

1. **console.log();** -> Generate log value
2. **console.warn();** -> use to generate warning in console.
3. **console.error();** -> use to generate error in console.
4. **console.table();** -> log value display in table format.

**Eg:**

A screen shot of a computer

Description automatically generated

* **Software Bug:**
* Defect or problem in a computer program.
* Basically, any unexpected or unintended behaviour of a computer program is a software bug.