**Variables**

* Variables are the containers for storing data values
* var, const and let are the reserved keyword to declare a variable
* JavaScript identifiers are case-sensitive.
* You can assign a value to a variable using the equal to (=) operator when you declare it or before using it.

Keyword                      const              let                  var

Global scope              Yes                   Yes                 Yes

Function scope         Yes                  Yes                 Yes

Block scope               Yes                  Yes                 No

Can be reassigned    No                  Yes                 Yes

let variables can be updated but not re-declared;

const variables can neither be updated nor re-declared.

Var Keyword

<html>

<head>

<title>First HTML Page</title

</head>

<body>

<script src="./script.js"></script>

</body>

</html>

Create .js file and check it by console.log(“testing”)

You can check in browser by right click and inspect

In file🡪

\*\*\*var keyword\*\* \*\*\*\*\*\*/

// It can be updated and also can be re-declared

// var keyword two type of scope global and function

// not block level scope

var course = "zero to hero"

console.log(course)

var course="zero to hero in lwc"

console.log(course)

\*\*\*var keyword\*\* \*\*\*\*\*\*/

// It can be updated and also can be re-declared

// var keyword two type of scope global and function

// not block level scope

var course = "zero to hero" console.log(window.course)

function abc(){

var anotherCourse ="hero in lwc" console.log(anotherCourse)

}

abc ()

// console.log(anotherCourse) //block scope

if(2===2){

var x = 10

}

console.log(x)

Let keywords not support window.log

\*\*\*\*\*\*let keyword\*\*

// It can be updated and also cannot be re-declared

// let keyword support function and block level scope

let course = "zero to hero" course="zero to hero in lwc" console.log(course) console.log(window.course)

function abc(){

let x = "nikhil"

console.log(x)

}

abc ()

// console.log(x)

if(2==2){

let y = "salesforce"

console.log(y)

}

// console.log(y)

Const Keyword

\*\*\*const keyword\*\*\* \*/

// It can't be updated and also cannot be re-declared

// const keyword supports function and block level scope

const course = "zero to hero" console.log(course)

// const course="1wc"

function abc(){

const x = 20

console.log(x)

} abc ()

const x = 30

console.log(x)

if(1==1){

const y = 40

console.log(y)

}

console.log(y)

1. Data type in JS

#### Data types in JS

There are 8 basic data types in JavaScript.

1. number

5. undefined

2. string

6. null

3. boolean

7. object

8. Symbol

**Rest all types are Object in some form**

1. Array

2. Date

3. Math

4. String

5 etc.

4. bigint

1. Data Types Variables –
2. var x = 10.66 //±(253-1) console.log(typeof x)
3. // big int
4. var y= 1068768768n console.log(typeof y)
5. // string
6. var name='nikhil'
7. console.log(typeof name)
8. // boolean
9. var isMonday = true console.log(typeof isMonday)
10. // undefined
11. var z
12. console.log(typeof z)
14. //objects
15. var obj={}
16. console.log(typeof obj)
17. var arr = []
18. console.log(typeof arr)
19. //symbol
20. var sym= Symbol("id") console.log(typeof sym)
21. // null var x1 = 10
22. x1 = null
23. console.log(typeof x1)
24. // String() Boolean() console.log(typeof new String("hey"))
25. Null vs Undefined

|  |  |  |
| --- | --- | --- |
| SNO | Undefined | Null |
| 1 | If a variable is declared, but not initialized or assigned any value then JS automatically initialized it with undefined | It's a special data type which represent "nothing" or "empty" or "value unknown" |
| 2 | typeof undefined is undefined | Typeof null is object |
| 3 | undefined == null is true | Undefined === null is false |

Equality Comparison

== This operator does value comparison. Example 100 == "100"

=== This operator does value plus data type comparison Example 100 "100" ===

**3.1 Null vs Undefined**

// Equality comparison

// normal comparision

// strict comparison

//1. Normal comparision (==) || only compare value

console.log(3=="3")                         //true

console.log(3==3)                           //true

console.log("nikhil" == "nikhil")           //true

console.log("nikhil" == "nikhil")           //true

//2. strict comparision (===)

//compare value+data type

console.log(3===3)                          //true

/// Null vs Undefined

var x

console.log(x)                              //Undefined

var y = null

console.log(y)                              //null

console.log(x ===  y)                       //false

console.log(x ==  y)                        //true

1. Spread Operator

The operator's shape is three consecutive dots and is written as :

Usages of spread operator

1. Expanding String - Convert string into list of array

2. Combining Arrays - Combine array or add value to array

3. Combining Object - Combine Object or add value to Object

4. Creating New Shallow Copy of Arrays and objects

// Spread Operator

//1 Array

var arr = ["a", "b", "c"]

console.log(arr)                     //(3) ['a', 'b', 'c']

                                    //0: "a"

                                    // 1: "b"

                                    // 2: "c"

                                    // 3: "3"

                                    // length: 4

// array index start from zero

console.log(arr[0])                 //a

console.log(arr[1])                 //b

arr.push("3")

console.log(arr)                    //['a', 'b', 'c', '3']

                                    // 0: "a"

                                    // 1: "b"

                                    // 2: "c"

                                    // 3: "3"

                                    // length: 4

//2 Objects

var obj={

"name" : "salesforce",

"age" : 23,

"full name" : "zero to hero"

}

console.log(obj. age)               //23

console.log(obj["full name"])       // zero to hero

obj.hobbies="cricket"

console.log(obj)                    //{name: 'salesforce', age: 23, full name: 'zero to hero', hobbies: 'cricket'}

                                    // age: 23

                                    // full name: "zero to hero"

                                    // hobbies: "cricket"

                                    // name: "salesforce"

//1. Expanding of string

let greeting = "Hello Everyone"

let charList = [...greeting]

console.log(charList)                   //['H', 'e', 'l', 'l', 'o', ' ', 'E', 'v', 'e', 'r', 'y', 'o', 'n', 'e']

// 2. Combining array

let arr1 = ["amazon", "google"]

let arr2 =["facebook", "instagram"]

let arr3 = [...arr1, ...arr2]

console.log(arr3)                       //['amazon', 'google', 'facebook', 'instagram']

//3. adding values to an array

let arr4 = ["a", "b", "c"]

let arr5 = ["nikhil",...arr4 ]

console.log(arr5)                       //(4) ['nikhil', 'a', 'b', 'c']

//4. Combining objects

let obj1 ={name:"salesforce", age:23, date:"23/10/2022"}

let obj2 = {name : "facebook", age : 55, "next" : "helo"}

let obj3 = {...obj1, ...obj2}

console.log(obj3)                                           //{name: 'facebook', age: 55, date: '23/10/2022', next: 'helo'}

                                                            //age: 55

                                                            // date: "23/10/2022"

                                                            // name: "facebook"

                                                            // next: "helo"

                                                            // [[Prototype]]: Object

//5. Shallow Copy

var arr10 = ["x", "y", "z"]

arr10.push (20)

console.log(arr10)                              //(4) ['x', 'y', 'z', 20]

var arr11 = arr10

arr11.push("nikhil")

console.log(arr10)                          //(5) ['x', 'y', 'z', 20, 'nikhil']

console.log(arr11)                          //(5) ['x', 'y', 'z', 20, 'nikhil']

//here arr10 push the value of its in arr11 that why its over it, to sol. this we can do following

//to avoid this we used shallow copy

//5. Shallow Copy

var arr10 = ["x", "y", "z"]

var arr11 = [...arr10]

arr11.push("nikhil")

console.log(arr10)                  //(3) ['x', 'y', 'z']

console.log(arr11)                  //(4) ['x', 'y', 'z', 'nikhil']

// but the problem in this it is useful in single obj or array. its not used in nested coding.

//6. Nested copy

var arrobj = [

    {name : "nikhil"},

    {name : "salesforce"}

    ]

    var arrobj1 = [...arrobj]

    console.log(arrobj1)                    //(2) [{…}, {…}]

                                            //0: {name: 'superman'}

                                            //1: {name: 'salesforce'}

                                            //length: 2[[Prototype]]: Array(0)

    arrobj1[0].name="superman"

    console.log(arrobj1)                    //(2) [{…}, {…}]

                                            //0: {name: 'superman'}

                                            //1: {name: 'salesforce'}

                                            //length: 2[[Prototype]]: Array(0)

    console.log(arrobj)                     //(2) [{…}, {…}]

                                            //0: {name: 'superman'}

                                            //1: {name: 'salesforce'}

                                            //length: 2[[Prototype]]: Array(0)

//for avoiding this problem we can do this

//6. Nested copy

var arrobj = [

    {name : "nikhil"},

    {name : "salesforce"}

    ]

    //var arrobj1 = [...arrobj]

    //console.log(arrobj1)

    //arrobj1[0].name="superman"

    //console.log(arrobj1)

    //console.log(arrobj)

    // Hack for nested copy

var arrobj1 = JSON.parse(JSON.stringify(arrobj))

arrobj1[0].name="superman"

console.log(arrobj1)                //(2) [{…}, {…}]

                                    // 0: {name: 'superman'}

                                    // 1: {name: 'salesforce'}

                                    // length: 2

console.log(arrobj)                 //(2) [{…}, {…}]

                                    // 0: {name: 'nikhil'}

                                    // 1: {name: 'salesforce'}

                                    // length: 2

1. Destructuring

The two most used data structures in JavaScript are Object and Array.

Destructuring is a special syntax that allows us to "unpack" arrays or objects into a bunch of variables, as sometimes that's more convenient.

* Array destructuring
* Object destructuring

let arr = ["amazon", "google"]

// let company1 = arr[0]

// let company2 = arr[1]

let [company1, company2] = arr

console.log(company1)                   //amozon

console.log(company2)                   //gogle

// // object destructing

let options ={ title : "zero to hero", age : 23, type : "CRM" }

// var title = options.title

// var age = options.age

let {title, age, type} = options

console.log(title)                      //Zero to hero

console.log(age)                        //23

console.log(type)                       //CRM

1. String Interpolation

String interpolation allows you to embed expressions in the string

Template strings use back-ticks (``) rather than the single or double-quotes.

// String Interpolation

var name = "Sales"

var name1 = "force"

console.log(name     + name1)         //SalesForce

var name2 = "Nikhil"

var age = 23

var str = "my name is"+name2+ "and my age is"+ age

console.log(str)                    //my name isNikhiland my age is23

var a = 2

var b = 3

var str1 = "the sum of "+a+ "and"+b+ "is "+a+b

console.log(str1)                       //the sum of 2and3is 23

var name3 = "nikhil"

var age=23

console.log(`my name is ${name3}`)      //my name is nikhil

var a1 = 10

var b1 = 20

console.log(`the sum of ${a1} ${b1} is ${a1+b1}`)       //the sum of 10 20 is 30

var recordid = 966866876

console.log(`http://salesforce.com/${recordid}` )           //http://salesforce.com/966866876

var name4 = "Salesforce Troop";

console.log(`Hello, ${name4} !`);               //Hello, Salesforce Troop !

// String Interpolation with expression

var a2 = 10;

var b2 = 10;

console.log(`The sum of ${a2} and ${b2} is ${a2+b2}`)           //The sum of 10 and 10 is 20

1. String Methods

JavaScript provides Many methods to play with strings. Below are some of the most commonly used strings method in LWC

1. includes()

2.indexOf()

3. slice()

4. to LowerCase()

5.trim()

6.startsWith()

7.toUpperCase()

let str = "Hello guys doing i hope your are doing good"

// includes () -

//The includes () method determines whether a string contains

//the characters of a specified string.

let check = str.includes ("hope")

console.log(check)                          //true

// indexof

//The indexOf() method returns the position of the first

// occurrence of a specified value in a string.

let index = str.indexOf("doing")

console.log(index)                              //11 if word is not in line so it show -1

// startsWith()

// determines whether a string begins with the characters

// of a specified string.

console.log(str.startsWith("Hello"))            //true if I typed "heloo" it's show fale

//Slice ()

//method extracts parts of a string and returns the extracted parts in a new string.

let newStr = str.slice(0, 5)

console.log(newStr)                     //Hello

//toLowerCase()

// converts string to lowerCase letter

let x = "My name is Nikhil"

console.log(x.toLowerCase())                //my name is nikhil

// toUpperCase()

// converts string to uppercase letter

console.log(x.toUpperCase())                //MY NAME IS NIKHIL

//trim() - removes whitespaces from both sides of a string

let searchText = " salesforce lwc           "

console.log(searchText.trim())              //salesforce lwc

8. Object/JSON Operations

1. Object.keys()

3.JSON.stringify()

2. Object.values()

4.JSON.parse()

let obj={

    name : "Salesforce",

    age : 23,

    dob : '23/10/1990'

    }

    //Object.keys ()

    console.log(Object.keys (obj))          //['name', 'age', 'dob']

    // Object.values()

    console.log(Object.values (obj))        //['Salesforce', 23, '23/10/1990']

    //JSON.stringify

    console.log(JSON.stringify (obj))        //{"name":"Salesforce","age":23,"dob":"23/10/1990"}

    //JSON.stringify

    let str = JSON.stringify (obj)

    console.log(str)                        //{name: 'Salesforce', age: 23, dob: '23/10/1990'}

    // JSON.parse

    console.log(JSON.parse(str))        //{name: 'Salesforce', age: 23, dob: '23/10/1990'}

8. Array Methods

1. map() - loop over the array and return a new array based on the value return.

2. every() - return true or false if every element in the array satisfy the condition 3. filter() - return a new array with all the elements that satisfy the condition

4. some() return true if at least one element in the array satisfies

5. sort() - sort the elements of an array

6. reduce()- this method reduces the array to a single value (left to Right)

7. for Each this method calls for each array element

let arr = [2,3,5,7,9,10]

// map()

//syntax

// arr.methodName (function (currentItem, index, actualArray) {

// })

let newArray = arr.map(function (currentItem, index, array) {

console.log(`currentItem is ${currentItem} index ${index}`)

return currentItem\*2

})

console.log(arr)                        //(6) [2, 3, 5, 7, 9, 10]

console.log(newArray)                   //(6) [4, 6, 10, 14, 18, 20]

//filter()

let filteredValues = arr.filter (function (currentItem1, index, array){

return currentItem1 >5

})

let filteredValues1 = newArray.filter (function (currentItem1, index, array){

    return currentItem1 >5

    })

console.log(filteredValues)             //(3) [7, 9, 10]

console.log(filteredValues1)            // (5) [6, 10, 14, 18, 20]

//every()

let age = [32, 33, 18, 40]

let isAllAdult = age.every(function (currentItem) {

    return currentItem >=18

})

 console.log(isAllAdult)                //true

// some ( )

let agelist = [32, 33, 18, 40]

let isAdult = agelist.some (function (currentItem) {

return currentItem >32

})

console.log(isAdult)                    //true

//sort()

var names =["nikhil", "john", "smith"]

console.log(names.sort())                       //(3) ['john', 'nikhil', 'smith']

//sorting of number

var points = [10,39, 12, 80, 54]

let sortedValue = points.sort (function(a,b){

    return b-a

 })

console.log(sortedValue)                        //(5) [80, 54, 39, 12, 10]

//reduce methods array.reduce(function(){

//})

// reduce methods

// array.reduce(function(total, currentValue, index, array) {

// }, intialvalue)

let num= [12, 78, 30]

let sum = num.reduce(function (total, currentItem) {

return total+currentItem

},0)

console.log(sum)                        //120

//forEach()

num.forEach(function(currentItem)

{ console.log(currentItem) })               //12

                                            //78

                                            //30

1. Promise

The promise is an object that may produce a single value some time in the future Promises are used to handle asynchronous operations in JavaScript.

A Promise has three states

1. pending()

2. fulfilled()

3. rejected()

Use case from LWC point of view.

1. Fetching data from the server

2. Loading files from the system

function checkIsSuccess (data) {

    return new Promise(function(resolve, reject){

  if(data ==="success") {

  return resolve("successfully executed")

  } else {

   return resolve("unsuccessfully executed")

  }

}

  )}

console.log(checkIsSuccess ('success'))         // Promise {<fulfilled>: 'successfully executed'}

            //or

// to see in real time

checkIsSuccess('success').then(function(result) {

    console.log(result)                             //successfully executed

   })

        //or

   checkIsSuccess ('success') .then(function(result) {

   console.log(result)                              //successfully executed

}).catch(function(error) {

    console.error(error)

})

            //or

checkIsSuccess ('') .then(function(result) {

   console.log(result)

}).catch(function(error) {

    console.error(error)                            //unsuccessfully executed

})

//OR

function checkIsSuccess (data) {

    return new Promise(function(resolve, reject){

  if(data ==="success") {

  return resolve("successfully executed")

  } else {

   return reject("unsuccessfully executed")

  }

}

  )}

  checkIsSuccess ('') .then(function(result) {

    console.log(result)

 }).catch(function(error) {

     console.error(error)                           //unsuccessfully executed

 })

 fetch('https://api.github.com/users/karkranikhil').then(function(result){

console.log(result) })                              //Response {type: 'cors', url: 'https://api.github.com/users/karkranikhil',

                                                    //redirected: false, status: 200, ok: true, …}

fetch('https://api.github.com/users/karkranikhil').then(function(result){

return result.json()

 }).then (function(response){

console.log(response)                               //{login: 'karkranikhil', id: 11937732, node\_id:

'MDQ6VXNlcjExOTM3NzMy',

                          // avatar\_url:

'https://avatars.githubusercontent.com/u/11937732?v=4', gravatar\_id: '', …}

 })

1. Exports Import

Exporting - Use the export keyword to export many variables or many methods from a file export cont name = "Nikhil"

export function getName() {

return "nikhil"

}

Default export - Use the export default keyword to export only one variable or a method from a file

export default user =  "salesforce"

#### Imports

Importing - Use the import keyword to import a variable or method from a given file path or module.

Multiple imports

import {name, getName} from './filepath'

Imports all exported members

import \* as Utils from './filepath'

Imports a module with a default member

import user from './filepath'

Export File

// Normal Export

// export const PI = 3.14

// export function add(a,b){

//         console.log(a+b)                    //5

// }

//OR

// Normal Export

const PI = 3.14

function add(a,b){

//console.log(a+b)

}

export {PI, add}

//OR

//want send data with alias name

// Normal Export , export with a Alias

// const PI\_Data = 3.14

// function add(a,b){

// console.log(a+b)

// }

// export {PI\_Data as PI, add}

//OR

// export data default means you can send data without curly braces

// export default function minus(a,b){

//     console.log(a-b)

//     }

export  function minus(a,b){

    console.log(a-b)

    }

Import File

// import {PI, add} from './Export.js'

// console.log(PI)                     //3.14

// console.log(add (2,3))              //undefined

//OR

// import minus, {PI, add} from './Export.js'

// console.log(PI)

// console.log(add (2,3))

// console.log(minus(10,7))

// import minus, {PI, add} from './utils.js'

//import as \*

import \* as UTILS from './Export.js'

console.log(UTILS.PI)

console.log(UTILS.add(2,3))

console.log(UTILS.minus (8,5))

//remove defualt in export class

1. querySelector

The querySelector() method returns the first element that matches a specified CSS selector(s) in the document.

document.querySelector (selector);

# querySelector All

The querySelectorAll() method returns all elements in the document that matches a specified CSS selector(s), as a static NodeList object.

document.querySelector All (selector);

//HTML file

<html>

<head>

<title>First HTML Page</title

</head>

<body>

    <div>Hi everyone</div>

    <div class="abc">Hi everyone</div>

    <div>Hi everyone LWC Developer</div>

    <div>Hi Wazid</div>

<script src="./QuerySelector.js"></script>

</body>

<!-- Hi everyone

Hi everyone LWC Developer

Hi Wazid -->

//JS File

let element = document.querySelector('div')

console.log(element)                        //<div> Hi everuone</div>

element.style.color='Orange'                //only first div in irange color

let elementAll = document.querySelectorAll('div')

console.log(elementAll)                 //NodeList(3) [div, div, div]

Array.from(elementAll).forEach(function(item){

    item.style.color='Pink'                 //All in ink Color

})

//if you used map funcition

let elementtwo = document.querySelector('div')

console.log(elementtwo.innerHTML)                        //Hi everyone

console.log(elementtwo.innerText)                       //Hi everyone

elementtwo.style.color='Orange'                //only first div in irange color

let elementAlltwo = document.querySelectorAll('div')

console.log(elementAlltwo)                 //NodeList(3) [div, div, div]

Array.from(elementAlltwo).map(function(item){

    return item.style.color='Pink'                 //All in ink Color

})

//if we used class element in HTML used that in Js class

let elementthree = document.querySelector('.abc')

console.log(elementthree.innerText)                       //Hi everyone

document.querySelector('.abc')

this.template.querySelector('.abc')

1. Event

An event is an action that occurs in the web browser, which the web browser feedbacks to you so that you can respond to it.

For example, when users click a button on a webpage, you may want to respond to this click event by displaying an alert box.

Event handler - It is a block of code that will execute when the event occurs. It is also known as an event listener.

Two Common ways to add events

1. HTML Event Handler attribute - When we add an event through HTML, the Event always begins with on keyword like onclick, onchange, onkeyup

2. Event Listener - Event Handlers provide two main methods for dealing with the registering/deregistering event listeners :

• addEventListener() - register an event handler

● removeEventListener() - remove an event handler

//Event html file

<html>

<head>

<title>First HTML Page</title

</head>

<body>

    <button onclick="firstfunction()">1.Click</button> <br><br>

    <button onclick="secondfunction">2.Click</button><br><br>

    <button onclick="removeHandler()">Remove</button>

    <!-- <button>click</button> -->

    <div class="demo"></div>

<script src="./events.js"></script>

</body>

</html>

//Event.js file

// function firstfunction(){

//     console.log('Hurrah Wazid')     //when you click on Click buttoin which created by html file it will show "Hurrah Wazid"

// }

function secondfunction(){

    console.log('dusri bar click kiya')

}

let btn = document.querySelector("button")

btn.addEventListener("click", secondfunction)

document.addEventListener("mousemove", handler)

function handler(){

    document.querySelector(".demo").innerHTML = Math.random()       //by this mouse pointer position will show

}

function removeHandler(){

    document.removeEventListener("mousemove", handler)      //by this mouse pointer postion will stop to show

}

Event Propagation

Event Propagation explains the order in which events are received on the page from the element where the event occurs and propagated through the DOM tree. There are two main event models

1. Event bubbling

2. Event Capturing