JavaScript

Variable

1). Let: One time declare multiple time assign.

Let a = 5

Let a = 10 // not valid in second time

2). Var: Multiple time declare multipale time assign

Var x = 5

Var x = 10

3). Const: One time declare one time assign

Const a = 5 a=10 // not valid

Functions:

1). Basic Function:

```
Function funame(){
Return;
}
```

2). Arrow Function:

```
const calc=(a,b,c)=>{
  if(c=='+'){
   console.log(a+b)
  }
else if(c=='-'){
 console.log(a-b)
}
else if(c=='*'){
 console.log(a*b)
}
else if(c=='/'){
 console.log(a/b)
```

```
}
else{
 console.log("Operator is not valid")
}
}
calc(5,7,'*')
```

Loops:

1). For Loop: A for loop in JS is used to execute a block of code a specified number of time. It consists three optional expressions initialization , Condition and increment.

Example:

```
For(i=1;i<=5;i++){
Console.log(i)
}
```

Output: 1,2,3,4,5

2). For Of: Return element of element.

Example:

```
let arr=[5,7,9,2,4]
for(let val of arr){
   console.log(val)
}
```

Output: 5,7,9,2,4

3). For in: Return index of element.

Example:

```
let arr=[5,7]
for(let val in arr){
```

```
    console.log(val)
}
```

Output: 0,1

4). Map: The map function is a method in JavaScript used to create a new array by applying provided function to every element of the existing array. It transforms each element of the array based on the specified functions and returns a new array.

Example:

```
let arr=[1,2,3,4,5]
let ans=arr.map((val,idx)=>{
    return val+2
})
console.log(ans)
```

output: [3,4,5,6,7]

5). forEach: forEach method is used for traversing element in array. It allows to perform operation on each element without the need for a traditional for loop.

Example:

```
Let arr = [1,2,3,4,5]
Arr.forEach((val)=>{console.log(val)})
```

Output:

1

2

3

4

5

Array:

1). PUSH AND POP:

=> push: push the element in array

Example:

```
Let numbers = [1,2,3,4]
Number.push(5)
Console.log(number)
```

Output:[1,2,3,4,5]

=> pop: Remove the last element from array

Example:

```
Let numbers = [1,2,3,4,5]
number.pop()
console.log(number)
```

Output:[1,2,3,4]

2). Find: Return the first element which satisfies the condition.

Example:

```
let arr =['12','23','-52','85','-6','-52','15','30']
let ans = arr.find((val)=>{
        if(val > 0){
            return true
        }
})
console.log(ans)
```

output: 12

3). findIndex: Return the index of the first element which satisfies the condition.

Example:

```
let arr =['12','23','-52','85','-6','-52','15','30']
let ans = arr.findIndex((val)=>{
      if(val > 0){
         return true
      }
})
console.log(ans)
```

output: 0

4). Filter: Return all the element which satifies the condition in array form.

Example:

```
let arr =['12','23','-52','85','-6','-52','15','30']
let ans =arr.filter((val)=>{
   return val > 0
})
console.log(ans);
```

output: ['12', '23', '85', '15', '30']

5). Splice: It enable to add, remove, or replace element from any index. It modifies original array.

Example: [Delete]

```
let arr = [1,2,3,4,5]
arr.splice(3,1)
console.log(arr);
```

output:

[1, 2, 3, 5]

Example: [Add]

let arr1 = [1,2,3,4,5]

arr1.splice(3,2,7,8)

console.log(arr1)

output:

[1, 2, 3, 7, 8]

6). Slice: It is used to extract a section of an array or a portion of a string. It doesn't modify the original array or string but return a new one with the selected element.

Syntax: array.slice(StartIndex, EndIndex)

Example:

let arr = [1,5,6,2,8]

let ans = arr.slice(1,4)

console.log(ans);

output: [5, 6, 2]

7). Includes: It determines wheter an array include a particular value among its entries. It returns true it the value is found and otherwise it is given a false.

Example:

let fruits = ["apple", "banana", "mango"];

let hasBanana = fruits.includes("banana");

console.log(hasBanana);

Output: true

8). IndexOf: It return the first index at which a given element can be in the array. If the element is not present, it return -1.

Example:

Let arr = [2,4,5,6,7]

Arr.indexof.(5)

Output: 1

9). Sort: It is used to sort elements in array. It is used to sort the elements alphabetically or numerically in ascending or descending order.

Example:

```
// //ascending
let arr = [5,8,9,2,3,5,6]
let ans = arr.sort((a,b)=>{
   return a-b
})
console.log(ans)
```

output: [2, 3, 5, 5, 6, 8, 9]

```
 //descending
let arr1 = [5,8,9,2,3,5,6]
let ans2 = arr.sort((a,b)=>{
   return b-a
})
console.log(ans2)
```

output: [9, 8, 6, 5,5, 3, 2]

10). Reverse: It is used to reverse the order of elements in an arrays. It modify the original array and return a reference to the same array.

Example:

```
let arr = [1,2,3,4,5,6,7]

arr.reverse()

console.log(arr)
```

Output: [7, 6, 5, 4, 3, 2, 1]

Object:

1). Object.key(): Return all the keys of the object in array.

Example:

```
let obj = {
   name: "John",
   age: 25,
   occu:"value"
}
console.log(Object.keys(obj));
```

output:

```
[ 'name', 'age', 'occu' ]
```

2). Object.Value(): Return all the value of object in array;

Example:

```
let obj = {
   name: "John",
   age: 25,
   occu:"value"
}
console.log(Object.values(obj));
```

Output:

['John', 25, 'value']

3). JSON.stringify(): It converts a JavaScript object or value to a JSON string.

Example:

```
let obj = {
   name: "John",
   age: 25,
   occu:"value"
}
console.log(JSON.stringify(obj));
```

Output:

{"name":"John","age":25,"occu":"value"}

4). JSON.parse(): It parses a JSON string and return a JavaScript object.

Example:

```
let obj1 = '{"name":"John","age":25,"occu":"value"}'
console.log(JSON.parse(obj1))
```

Output:

{name: 'John', age: 25, occu: 'value'}

Array of Object:

An array of objects is a collection in which each element is an object, allowing you to store and manage multiple objects within a single array. This structure is particularly useful when you want to organize and manipulate groups of related data items.

Example:

```
let pro = [
    {id:1,product:'AC',cost:20000},
    {id:2,product:'Coolar',cost:1000},
    {id:3,product:'pen',cost:20},
    {id:4,product:'pencil',cost:10},
]
let sum=0
for(let val of pro){
   console.log(val.cost)
}
```

Output:

```
20000
1000
20
10
```

String:

1). charAt(index): Return the character at the specified index in a string.

Example:

```
let str = 'Hello world'
console.log(str.charAt(1));
```

output:

```
e
```

2). Concat(Str1,str2): Coobines teo or more string and retrin a new string

Example:

```
let str = 'Hello world'

let str1 = 'Karan'

console.log(str.concat(' ',str1));
```

Output:

Hello world Karan

3). Includes: Checks if a string contains the specified substring and retrun a boolean.

Example:

```
let str = 'Hello world'

console.log(str.includes("Hacker"));
```

Output:

False

4). indexOf: Returns the index of the first occurrence of a substring in a string. Return -1 if the substring is not found.

Exmaple:

```
let str = 'Hello world'

console.log(str.indexOf("Vishal"));
```

Output:

-1

5). Replace (Search Value, Replace Value): Replace a substring with a new value in a string.

Example:

```
let str = 'Hello Vishal’

console.log(str.replace("Vishal","Hacker"));
```

Output:

Hello Hacker

6). Substring (Start Index, EndIndex): Extracts a portion of a string between the specified indices. The EndIndex is optional and if not provided. It goes till the end of the String.

Example:

et str = 'Hello world'

console.log(str.substring(6,10));

Output: wor

7). Split: The split() method splits a string into an array of substring. The split() method returns the new array. The split() method does not changes the original string.

Example:

let str = 'Hello world'

console.log(str.split(' '));

Output: ['Hello', 'world']

8). Length; The length property in JavaSacript return the number of charcters in a string. It is a read-only and provides the length of the strring in UTF-16 code units.

Example:

let str = 'Hello world'

console.log(str.length);

Output: 11

9). toUpperCase: Convert the string to uppercase letters:

Example:

let str = 'Hello world'

console.log(str.toUpperCase());

Output: HELLO WORLD

10). toLowerCase: convert the string to lowercase lletter.

Example:

```
let str = 'Hello world'

console.log(str.toLowerCase());
```

Output:

hello world

Spread Operator:

The spread operator (...) in JavaScript is used to expend element of an alterable (like an array of a string) into individual element. It is often used to copy arrays concatenate array, and pass element of an array as function argument.

Example:

```
let arr = [1,2,3,4,5]

let arr1 = [...arr,6,7,8,9,10]

console.log(arr1)
```

Output:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Example:

```
let obj= {
   name: 'vishal',  // name is key vishal is value
}
let obj1={
   age: 25,
  }
let obj2 = {
   ...obj,
   ...obj1,
   ocuu:'developer',
```

```
}
console.log(obj2)
```

Output:

```
{ name: 'vishal', age: 25, ocuu: 'developer' }
```

Rest Operator:

The rest operator (…) is used in function parameters to collect all remaining arguments into a single array. It allows function to accept a variable number of arguments making them more flexible and dynamic.

Example:

```
function sum(a,b,c,d,e,f,g){
   return a+b+c+d+e+f+g


}
let arrr = [1,2,3,4,5,6,7]
console.log(sum(...arrr))
```

Output:

28

Example:

```
function sum(...num){
   sum = 0
   for(let a of num){
      sum += a
   }
   console.log(sum)


}
console.log(sum(7,8,9,4,5,6,1,3,5))
```

output: 48

DE structuring:

Object destructuring allows you to extract values from an object and assign them to variables based on their property names.

Example:

```
let obj = {
   name:'vishal',
   course:'React',
   fees:'17000'
}
const{name,course,fees}=obj
console.log(course,fees)
```

Output:

React 17000\

Call Back:

Function inside function is called callbacks. A function, IF passed as an argument them it is called callback.

Example:

```
function add(a,b,c){
   setTimeout(()=>{
      console.log(a+b)
      c()
   },2000)
}
function addcompleted(){
   console.log('completed')
}
add(5,9,addcompleted)
```

Output:

14

Completed

setTimeout:

It is used to excute a specified function or code snippet once after a specified delay (In Milliseconds)

Example:

```
setTimeout (()=>{
console.log('setTimeout');},2000);
```

Output: setTimeoout

setInterval:

It is used repeatedly execute a specified function or code snippet at defined intervals.

Example:

```
let ans = setInterval(()=>{
   console.log('Vishal')
},2000)
setTimeout(()=>{
   clearInterval(ans)
},10000)
```

Output:

Vishal

Vishal

Vishal

Vishal

Vishal

Async:

The async keyword is used to define an asynchronous function. When you prefix a function with async, it always returns a promise. If the function returns a value, JavaScript automatically wraps it in a resolved promise.

Example:

```
async function myFunction() {
return "Hello";
}
myFunction().then(value => console.log(value));
```

Output: Hello

=>. In the example above, myFunction is an async function that returns a promise resolved with the value "Hello".

Await Keyword

The await keyword is used to pause the execution of an async function until a promise is resolved. It can only be used inside an async function.

Example:

```
async function getData() {
let promise = new Promise((resolve, reject) => {
setTimeout(() => resolve("Done!"), 1000);
});
let result = await promise; // Waits until the promise resolves
console.log(result); // Output: Done!
}
getData();
```

=>. In this example, the await keyword pauses the execution of getData until the promise resolves.

Error Handling in Async/Await

JavaScript provides predefined arguments for handling promises: resolve and reject.

resolve: Used when an asynchronous task is completed successfully.

reject: Used when an asynchronous task fails, providing the reason for failure.

Example:

```
const me=()=>{
    return new Promise((resolve, reject) => {
        setTimeout(()=>{
            let success = true
            if(success){
                resolve('Task1 is completed')
            }
            else{
                reject('Error')
            }
        },3000)
    })
}
const task1=()=>{
    return new Promise((resolve,reject)=>{
        setTimeout(()=>{
           let success1 = true
           if(success1){
            resolve('Task2 is complited')
           }
           else{
            reject('Error')
```

```
        }
    },2000)
  })
}
async function getdata(){
  try{
    console.log('Before')
    let vi = await me()
    console.log(vi)
    let ans1 = await task1()
    console.log(ans1)
    console.log('After')
  }
  catch{
    console.log('Error')
  }
}
getdata()
```

Output:

Before

Task1 is completed

Task2 is complited

After

Promises:

Promises are object used for handling asynchronous operations. There are 3 stages for promises pending, Fulfilled and Rejected.

Example:

```
const fetchdata=()=>{
 return new Promise((resolve, reject) => {
       setTimeout(()=>{
          let success = true;
          if(success){
             resolve({id:1,name:'vishal',occu:'Developer'})
          }
          else{
             reject('Data Not Define')
          }
       },2000)
       })
}
fetchdata()
.then((res)=>{
   console.log(res)
   return res
})
.then((ans)=>{
   console.log('Hello',ans.name)
   return ans
})
.catch((err)=>{
```

```
    console.log(err)
})
```

Output:

```
{ id: 1, name: 'vishal', occu: 'Developer' }
Hello vishal
```