Tokens:

Tokens are the small unit of programming language which contains entities like,

*Variables
*Identifiers
*Literals
*Operators
*Keywords, etc.,

Variable:

Variable are containers or variables are memory blocks to store the data.
Variable types are,
*Var
*Let
*Const

```
Ex: var username="john" (Less Recommended)
ES6 onwards,
        let user="john29"
        const name="John"
        Most recommended variables are let and const.
```

Var(1995-2015):

Var keyword is used to declare variable.
It's confusion and error prone when using variable declared using var.
Var can be declare separately and initialized separately

```
Ex: var a;
    a=10;
```

Var can be declared and initialized at once.

```
Ex: var a=10;
```

The variable declare with var can be reinitialized

```
Ex: var a=10;
            a="hi ji";
```

Var can be Redeclare.

```
Ex: var a=10;
     var a="hi ji";
```

Let:

Let keyword is used to declare variables.
Let keyword removes the confusion and error of var.
Let variables can be declare and initialized and reinitialize but can't be redeclare.

```
Ex: (I) let a;
        a=10;
   (II)let a=10;
        let a=10;
        a="HI"
  (III)let a=10:
        let a="hi"
```

Const:

Const keyword is used to declare a constant that can't be change once assign a value.
Const can't be declare and initialized separately. That can be declare and initialize in single line.

Ex:

Scope of Variables:

Scope determines variables can be accessibility from different path of the code.

It's having Global Scope, Local scope.
Var is global and local scope.
Let is Script scope, local scope and block scope .
Const is Script scope, local scope, block scope.

Identifiers:

identifiers is a name that is given to the entity like variable(memory block) Function, Object, etc.,

Rules for Naming Identifiers:

Identifiers name must start with either alphabet letter, $_,
Ex: let userName29$= "john"
We shouldn't use any special characteristics except $_
Identifier name can't be start with numbers but we can use in between.
Ex: let 12userName="john"-----------> It shows Error.
We shouldn't space in between the identifier names.
A reserved keywords of JavaScript can't be used as an identifiers.
Ex: let new="john"-----------> new is a reserved keyword
Length of an identifier can be any.
JavaScript identifiers are Case Sensitive.

Literals:

Literals are nothing but the values that is assigned to the variable.
Literals are nothing but data types.

Strict Mode in JS:

JavaScript Strict mode is a way to opt into a restricted variant in JavaScript.
Strict mode applied by using within the
0"use strict" eliminates some JavaScript silent errors by changing them to throw errors.
"use strict" applied to entire script or to individual functions. It doesn’t apply for block statements enclosed in {}
"use strict" should always be on the top of the JavaScript file for whole script.
and also should be on top of the functions.

```
Ex: // Whole strict
        "use strict"
        a=10; //throws error
        function test(){
                "use strict"
                b="Hi"
                }
                test();
```

Variable Hoisting:

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

```
Syntax: console.log(a) //Execution
          var a=10; //Declaration
          //var a;
         // console.log(a);
        //a=10;
```

Hoisting is not working for let and const. If we try to hoist by using let and const it throughs **Reference Error: Can't access variable 'a' before initialization**. This is called Temporal Dead Zone(TDZ).

Map Tag:

The <map> tag is to define the map-image on the user interface which is clickable. The <map> required the <img> tag that helps the relation between the image and map. The<map> element contains a number of <area> elements that define the clickable areas in the image map.

Syntax: <map name=""></map>

The <map> tag is used along with the <area> and <img> tags to define the image map.;
The use map attribute for the <img> tag is must be the same as the name attribute for the <map> tag creating a relationship between these two elements.

Syntax: <img src="https://cdn.pixabay.com/photo/2023/05/30/15/43/koala-8028992_1280.jpg" alt="Loris" usemap="#clickable">

```
<map name="clickable">
     <area shape="rect" coords="200,200,100,100" target="blank"
     href="https://pixabay.com/" alt="Pixaby">
     <area shape="circle" coords="300,400,300" target="blank"
     href="https://pixabay.com/" alt="pixaby">
     </map>
```

Datatypes:

Data Types specifies what kind of data can be stored and manipulated within a memory allocation.

In JavaScript, two types of data types are available. They are,

*Primitive Datatypes

*Non-Primitive Datatypes

Difference between primitive and non-primitive datatypes:

Primitive Datatypes	Non-Primitive Datatypes
• In primitive datatypes we can store only one values	• We can store multiple values or more than one values
• Primitive Datatypes are Immutable datatypes • Primitive datatypes are object de reference • Primitive data types are always contains some values	• Non-primitive datatypes are mutable datatypes • Whereas non-primitive datatypes are object reference • In non-primitive you can store the null value

Primitive Datatypes:

- String
- Number
- Boolean
- Undefined
- Null
- BigInt
- Symbol

String:

String is a sequence of characters used to represent the text.

Primitive way to store string value as

Ex: "" -----> Within double quotes

'' -------> Within a single quotes

`` ------> Within a Backticks(ES6 Onwards)

This is called template literals.

Template Literals:

Template literals or string literals are allowing embedded expression and variables.
You can use multiline strings and string interpolation features with them.

Syntax: `Strings`
`${Expressions}`
`${Variables}`

Ex: let a=10;
Let b=20;
Let c=a+b;
Console.log("the sum of a and b is"+" "+c)   // In Normal way
Console.log(`The sum of a and b is ${a*b}'); // backtick way

Ex: To Multiline String
Console.log{`John
John
John`}

How to concate in ES6:

Ex: let firstName="John";
Let last name="V";
Console.log("My name is"+" "+firstName+" "+lastName); // In Normal way
Console.log(`My name is ${firstName} ${lastName}`); // Backtick way

typeof() Operator:

Type of operator is used to check the datatype in JavaScript.
Ex:
Let userName="John"
Console.log(typeof(userName))

Note: String are immutable, string can't be changed. It's impossible to change the character in String.
Ex:
Var a="Hello World"
Console.log(a[0])
a[2]="H"
console.log(a) //It's not possible

Object reference way by using String constructors:

Ex:
```
let str=new String("Hi John");//Object creation
console.log(str); //In the form of object
console.log(typeof(str)); //to know the type of datatype we use
typeof operator.
```

valueOf Method:

It's used to convert object into primitive datatypes.
Ex:
```
let str=new String("Hi John");//Object creation
console.log(str); //In the form of object
console.log(typeof(str)); //to know the type of datatype we use
typeof operator.
let data=str.valueOf(); //To convert non primitive(object) into
primitive(string) we use valueOf operator
console.log(data);//Stored in the form of string
console.log(typeof(data));//type of datatype - String
```

Evaluation:

It evaluates JavaScript code or any operations within the String and executes into expressions.

Ex:

```
let x="10*20";//mathematical operations in string to perform
console.log(x)
console.log(typeof(x))//but not performing it returns as string
only
let y=eval(x);//to do a mathematical operations in string we use
evaluation method
console.log(y);//answer is 10*20=200 its working fine.
```

Number:

The Number datatype in JavaScript is primitive datatype.

Number datatype can hold integer, float, exponential value(2.563cpowerof

Ex:

```
let num=100;
console.log(num);
console.log(typeof num); //number datatype
let n1=10.00;
console.log(n1);//10
console.log(num===n1); //True. if we store 10.01 means false
```

Creating a number by using number constructor:(valueOf method(Instanceof object):

Ex:

```
let n2= new Number(100);
console.log(n2) //100
console.log(typeof n2) //Object
let n3=n2.valueOf();
console.log(n3) //100
console.log(typeof n3) //Number
```

Number() Method:

Which converts Number value within the string into a Number datatype

Ex:

```
let n4="100";
let n5=Number(n4);
console.log(n5); //100
console.log(typeof n5); //Number
```

Not a Number(NaN):

NaN is a special type of number that's not a legal number.

Ex:

```
let n6="10a";
let n7=Number(n6);
console.log(n7);//NaN
console.log(typeof n7);//Not a Number(NaN) is a special type of
number in JS. So it returns Number when you use typeOf operator.
```

Infinity:

Infinity is an property of global object.

The value of infinity that means a positive infinity is greater than any other number.

The negative infinity is lesser than any other number.

Ex:

```
console.log(15/0)//infinity(anything with 0 returns infinity)
console.log(-15/0)//-infinity
```

Prompt:

Prompt is a method used for collecting inputs from user.

The data is store in the form of string.

Ex:

```
let n8=prompt("Enter a value");
let n9=prompt("Enter a value");
console.log(n8);//In prompt it returns as String
console.log(n8+n9)//Concatination
```

Prompt Using Number method:

```
let n10=Number(prompt("Enter a value"));
let n11=Number(prompt("Enter a value"));
console.log(n10);//In Number prompt it returns as Number only
console.log(n10+n11) //Adding 2 Numbers
```

Undefined:

Undefined means a variable that has not be assigned a value is called Undefined.
A Variable is declared but not assigned a value that returns Undefined.
Undefined is global property which present in global scope.
Ex:

```
let j;
console.log(j); //Undefined
```

Null:

Null is one of the primitive datatype in JavaScript.
Null represents absence of object value and null is also an empty value.
The return type of **typeof(null)** is Object.
Ex:

```
let j1=null;
console.log(j1); //Null(Empty value)
console.log(typeof j1); //Object
```

Boolean:

Booleans is primitive datatype in JavaScript.
It can hold two values,
*True
*False
It's useful for controlling program flow like,
*if
*else if
*switch

By Using Boolean literals:

Ex:

```
let j2=true;
console.log(j2);//true
console.log(typeof j2);//Boolean
```

By using Comparisonal Operator:

It will return boolean value.
Ex:

```
console.log(10>100); //If condition false means it returns false
console.log(100>50); //If condition is true means it returns true.
```

By using Boolean Constructor:

For Condition False,
Ex:

```
let j4=new Boolean();// Not specified anything inside a
Boolean()method is false.
console.log(j4); /Boolean /False
console.log(typeof j4); //Object
```

For Condition True,
Ex:

```
let j3=new Boolean("Hey John!");
```

```
console.log(j3); //True
console.log(typeof j3); //Object
```

To Convert object into Boolean:

Ex:

```
let j5=new Boolean(1);//Object
let j6= j5.valueOf();
console.log(j6);//True
console.log(typeof j6);//Boolean
```

Symbol(ES6):

Symbol() is one of the primitive datatype in JavaScript.
It was introduced in ES6 Onwards.
Symbol() is used for internal purpose kept private.
Symbol() returns a Unique value.

Ex:

```
Var x=Symbol();
Var y=Symbol();
Console.log(x);//Returns the Symbol of Empty space like Symbol()
Console.log(typeof x);//Symbol
```

Ex2:

```
Let a=Symbol(100);
Let b=Symbol(100);
Console.log(a==b);//False
Console.log(a==a);//True
```

Symbol() Can't be created with instanceof Symbol(), if we try to use it will throw type Error.

Ex:

```
Let x=new Symbol();//TypeError
```

BigInt:

BigInt is one of the primitive type in JavaScript.
It was introduced in ES8 Onwards.
BigInt is used when the range of number, -2powerof 53-1 to 2powerof 53-1
Any number suffix with n is called BigInt.
BigInt we can't use Floating values. Because, BigInt will take only integers.

By Using Literals:

Ex:

```
let j9=543n;//BigInt By using literals - any integer suffix with n
is BigInt
console.log(j9);//543
console.log(typeof j9);//BigInt
let j11=10.2n// It will throw error immediately. Because BigInt not
accept floating values.
```

By Using Constructors:

Ex:

```
let j10=BigInt(543);//By Using Constructors - We dont need new
keyword, Symbol and BigInt itself a method. if we use new on Symbol
and BigInt it returns as TypeError
console.log(j10);//543
console.log(typeof j10);//BigInt
```

Operators:

Operator is a special symbol used to perform operations on operands.
Syntax: operand1 operator operand2.

Type of Operators:

*Arithmetic operators

*Assignment operators
*Comparison or relational operators
*Logical operators
*String operators(Concatenation)

Arithmetic Operators:

Arithmetic operators are used to perform arithmetic calculations.

Type of Arithmetic Operators:

*Addition(+) `console.log(10+5);`
*Subtraction(-) `console.log(5-1);`
*Multiplication(*) `console.log(2*5);`
*Division(/) `console.log(10/5);`
*Modulus(%) `console.log(10%2);`
*Exponential(**) `console.log(2**10);`
*Increment(++) `console.log(++a);`
*Decrement(--) `console.log(--a);`

Assignment Operators:

Assignment operators are used to assign values to variables.

Type of Assignment Operators:

*Assignment operator(=) `a=10;`
*Addition assignment operator(+=) `console.log(a+=5);`
*Subtraction assignment operator(-=) `console.log(a-=5);`
Multiplication assignment operator(=) `console.log(a*=5);`
*Division assignment operator(/=) `console.log(a/=5);`
*Modulus assignment operator(%) `console.log(a%=2);`
*Exponential assignment operator(**=) `console.log(a**=2)`

Comparison operator:

Comparison operators are comparing two values and return boolean values either true or false.

Types of Comparison Operators:

*Equal to(==) `console.log("10"==10);//Check the values only`
*Strictly equal to(===) `console.log("100"==="100");//It check the values and datatypes if both are same it returns true only.`
*Not equal to(!=) `console.log(10!=5)`
*Strictly equal to(!==) `console.log("10!"==10);`
*Greater than(>) `console.log(10>5);`
*Less than(<) `console.log(10<5);`
*Greater than or equal to(>=) `console.log(10>=12);`
*Less than or equal to(<=) `console.log(10<=10);`

Logical operators:

Logical operators are perform logical operations and returns the bo0lean values either true or false.

Types of Logical Operators:

*Logical AND(&&) `console.log((10>5)&&(10<10));`
*Logical OR(||) `console.log((10>5)||(10<10));`
*Logical Not(!) `console.log(!10);`

String Operators(Concatenation):

In JavaScript, we can use addition operator to concatenate(Joins) two or more strings.
Ex:

```
console.log("hello"+" "+"world");
```

Functions:

Functions are one of the fundamental built-in blocks in JavaScript.

A set of statements or instructions that performs a task or calculations of values.
Write once, Execute many times.
Functions are non-primitive datatype and object reference.
Syntax:

```
function function_name(parameters){   ---------->Function Declaration
  //Set of instructions/Statements
}

function_name(arguments) ----> Function Calling
```

Function Declaration:

Function declaration or statement consists of,
"Function" keyword followed by Name of the function, optional parameters within the parenthesis() and enclosed with curly bracers{} where we can write statements.
Ex:

```
Function john(){ //Function Declaration
    var course="Java Full stack developer"
    console.log(course);
}
John(); //Function Calling
```

Type of Functions:

*Named Function
*Anonymous Function
*Function Expression
*Immediate Invoke Function Expression(ITFE)
*Arrow Function

Named Function:

A function which is having a name and calling a function by that name.
Ex:

```
Function john(){ //Function Declaration
    var course="Java Full stack developer"
    console.log(course);
}
John(); //Function Calling
```

Anonymous Function:

A function without having a name.
Ex:

```
function(){
     console.log("Function anonymous not working")
 };
 ()
```

Anonymous function itself will not execute it throws syntax error for invoke(calling) anonymous function we need function expression or variable declaration.
Ex:

```
let data=function(){
    console.log("Function anonymous working with the function is
fine")
};
data();
```

Function Expression:

A function is declaring with the help of variable explicitly.
Ex:

```
let data=function(){
    console.log("Function anonymous working with the function is
fine")
};
data();
```

Immediately invoke function expression:

A immediately invoke function expression is one of the function to execute immediately as soon as they created. And they can only once being called.
If we try to call this function more than once it will throw the type error for the second calling function.
Ex:

```
(function(){
    console.log("Immediate invoke function is working")
})();
```

First Class or First Citizen Function:

A function which is passed as a value to a variable is called first class or first citizen functions.
Ex:

```
function(){
    console.log("John")
};
```

Default Parameters (ES6 Onwards):

The default parameters is a way to set default values for function parameters when an argument or value is not passed for function.
Ex1:

```
function add(a=10, b=5){ //Default parameters
console.log(a+b);
console.log(a);
console.log(b);
}
add()
```

Ex2:

```
let userName="John"
let institute="QSP"
let stuid="QSP03"
function studProfile(user=userName, ins=institute,
id=stuid){//Naming convention
    console.log(user, ins, id);
}
studProfile()//If we pass no arguments then default values are
printing.
studProfile("Pradeep","QSP","QSP07") //If we pass arguments or both
the default values and arguments in a single function means
arguments are executed.
```

Pure Function:

A Pure function is a function the return value only determine by its arguments without any side effects.
Pure functions are functions that doesn't have side effects when we call with the same input parameters will always return same output.
Ex:

```
function pure(a,b){
console.log(a+b);//If Expected output comes then it will be called
pure functions.
}
pure(20,20)
```

Impure Function:

A Impure function is a type of function the given same input parameters and the same number of arguments pass but it may produce different output from external factors or external sources.

Ex:

```
function impure(a,b){
    console.log(a+b+c)//1040. If we not get the expected output or
our output affected by external sources means impure functions
}
impure(20,20)
```

Higher Order Function:

A higher order function is a function that takes one or more functions as an arguments which is called Higher order function(HOF) or a higher order a function that accepts an another function as an argument and returns a functions as its result.

CallBack function:

A callback function is a function that's passed as an value or arguments into another function is called callback function.

Ex:

```
function HOF(callback){  //Higher Order Function(HOF)
    return callback;
}
HOF(callback()); //CallBack function
HOF(callback1());
function callback(){
    console.log("I'm Callback function");
}
function callback1(){
    console.log("I'm Callback function1");
}
```

Closure:

A closure is the combination of a function bundled together with references to its surrounding states(the lexical environment(nested function)) .

In other words, a closure gives an access to an outer most functions scope and global scope from an inner most function.

A closure will be created everytime when a nested function is created.

Ex:

```
function x(){
    let a=100;
    let b=null;
    let c="john";
    console.log(a);
    console.log(b);
    console.log(c);
    function y(){
    console.log(a);
    console.log(b);
    console.log(c);
    }
    y()
}
x()
```

Arrow Function(ES6):

ES6's Most used feature.
Arrow function is the alternative for traditional function expression.
Arrow function doesn't support any function declaration.
Arrow function doesn't have argument object.
Arrow function is not suitable for call apply bind methods.
Arrow function Can't be used as constructors.
Arrow function doesn't use new keyword.
Syntax:

```
()=>{};
```

Ex1:
```
let t=()=>{
    console.log("John what are you doing right now");
    console.log("I'm just called you but no response")
}
t();
```

Ex2:
```
let r=username=>username
console.log("John")
```

Ex3:
```
let s=_=>console.log("Hey john i like you da");
s()
```

Implicit return:

A function returns the values without using return keyword is called implicit return

Ex:
```
Console.log(array());
let m=()=>console.log("Hello") //return implicitly, but always in
the first line only it will return
m()
```

Note:

Only arrow function have implicit return, whereas normal function doesn't support implicit return.

Explicit return:

A function returns the value with the help of return keyword it's called explicit return.
Normal function supports only explicit return.
Arrow function also supports explicit return.

Ex: Explicit return for Normal function:
```
function array(){
    // console.log("Hello");
    return "Hello"    //It will return value only not printing and
we mandatorily use return keyword because it's Explicit return
}
```

Ex: Explicit return for Arrow function:
```
let q=(a,b)=>{
return a+b;
}
console.log(q(10,5))
```

Argument Object:

Argument is an array like object which is holding indefinite no.of argument values.
Argument object is available only in normal function.
Argument object will not work in arrow function. If we try to written argument in arrow function it will through(Uncaught ReferenceError: arguments is not defined).

Argument object for Normal function Ex:
```
function argument(){
    return arguments;
}
let u=argument(5,2,5,3,8);
```

```
console.log(u) //It will return in the form of object(impure array)
```

Argument object for Arrow Function Ex:

```
//WE can't use argument object in array function
// let j=()=>{
//     return arguments;
// }
// console.log(j(1,23,4,5)) //Uncaught type referenceError:
arguments is not defined. we can't define argument in arrow
function.
```

Construction Function:

A constructor is a special type of function that creates and initialize an object instance of an class.
Constructor is called when an object is created using the new keyword(Memory allocated in heap area).
A function constructor is a function that is called each time an object is created.
Ex:

Write a Details about conditional Statement:

Conditional statements included in the JavaScript code assist with decision making based on certain conditions.
The condition is specified in the conditional statement can either be true or false.
We have 4 types of conditional statements in JavaScript. They are,

- If
- If else
- If else if(ladder)
- Switch

If:

If statement will execute only if the given condition is true.
If condition will results in false, nothing will be printed.
Syntax: if(condition){
    Statements….}

If-else:

If else statements executes a block of code specified condition is true. If the condition is false another block of code can be executed with else block.
Syntax:

Else-if Ladder:

Else-if condition is used when these are more than one condition to be checked.

Switch:

Switch statement is used to performs different actions based on the different conditions.
It select one of many code blocks to be executed.
Syntax:

Switch Expression is evaluated only once.
The value of expression is compared with the value of each case.
If there is a match, the associated block of code will be executed.

If there's no match; default block will be executed.

Break Keyword:

When JavaScript searches a break keyword; it breaks out from the switch block.

Default Keyword:

It specifies the code to run if there is not matching case value.

Ternary Operator: (Operator 1?(Operator2:Operator3);

Ternary operator is a shortend way or writing if-else statement.

It allows you to write concise, cleaner and easier to read lines of code.

Syntax:

Looping Statements:

JavaScript loops are used to iterate the codes 'n' number of times or specified number of times.

There are four types of loops,

- For loop
- While loop
- Do-while loop
- For-in loop

For Loop:

The for loop iterated the element for the fixed number of times; it shoud be used if number of iteration is known.

Syntax:

While loop:

The for loop iterates the element infinite no.of times. It should be used to no.of iteration is not known.

Syntax:

Do-While loop:

The do-while loop iterated the element for the infinite no.of times like while loop. But, the code is executed atleast once, whether the condition is true or false.

Difference between Property and Method:

Length is not a method it's a property.

Method is nothing but a function.

String Methods:

The String object is used to represent and manipulate a sequence of characters.

String methods are,

- length()
- charAt()
- indexOf()
- lastIndexOf()
- concate()
- startswith()
- endsWith()
- includes()
- repeat()
- replace()
- split()
- match()

- search()
- slice()
- substring()
- substr()
- trim()
- toLowerCase()
- toUpperCase()
- toLocaleLowercase()
- toLocaleUppercase()
- charCodeAt()
- codePointAt()
- toString()
- valueOf()

length():

The length property is used to findout the size of the string or no.of characters.

Syntax:

```
console.log(str.length); //Length property to know the length of an
string
```

charAt():

This method provides the character values present at the specified index.

Syntax:

```
console.log(str.charAt(4)); //If we pass the index value it will
return which character is on that index value
```

indexOf():

It provides index position of character present in the given string.

Syntax:

```
console.log(str.indexOf("a")); //if we pass the character it will
return index value
```

lastIndexOf():

It provides the index position of a character value present in the given string by searching character from the last position.

Syntax:

```
console.log(str.lastIndexOf("a")); //same character exist many times
we use 2 default method only. it will return last index
```

concate():

This method is used for combination of two or more strings.

Syntax for using concate method:

```
console.log(str.concat(" "+str1+" "+str2));//concatinate two string
using string methods
```

Syntax for using additional operators:

```
console.log(str1+str1+str1);//By using Additional operators
```

startWith():

This method determines whether the given string begin with the character or specified values given in the method.

Syntax:

```
console.log(str.startsWith("Hello"));//Start with the character or
word it will return boolean values. If it is present it will return
true. If it is not present it will return false.
```

endsWith():

This method determines whether the string ends with the specified character or value.

Syntax:

```
console.log(str.endsWith("script"));//End with the character or word
it will return boolean values
```

includes():

This method determines whether the given string contains specified character or value within the string.

Syntax:

```
console.log(str.includes("q"));//If the specified value(char or
word) exist on original string it will return true or false
```

repeat():

This method is used to repeat the given String for no.of times.

Syntax:

```
console.log(str.repeat(3)); //repeat method takes count.
```

replace():

It replaces the given string with the specified string or values.

Syntax:

```
console.log(str.replace("Hello Javascript","Hi React
Js"));//replacing an entire string or single character
```

split():

This method splits a string into an array and then it returns the newly created array value.

Syntax:

```
console.log(str.split(""));//Each and every character will be like
an array element. it splits as an single character into the array
element
console.log(str.split(" "));//When we give single space hello will
be one word and javascript will be one word. it splits as word.
```

match():

match() method search a specified character or regular expression in a given string and returns that specified value or regular expression if a match occurs.

If the match not found it returns null.

Syntax:

```
console.log(str.match(/a/));//if we match using regular expression
it will select first occuring only with the return type of array.
console.log(str.match(/a/g)); //thats why we use g. g stands for
globally search. in this regular expression we can achieve
everything.
console.log(str.match(/a/gi));//Case insensitive whether its a or A
present in the og string it will take both in this regular
expression.
console.log(str.match(/[a-f]/gi));//It will take all the character
from a-f case insentively and globally search
```

search():

This method search a specified value or a regular expression in a given string and it will return index position if the match occurs.

If the match not found it returns -1.

Syntax:

```
console.log(str.search("Script")); // If the match not occurs it
will returns-1 returns
console.log(str.search(/Script/i));//using case insensitive so it
returns starting index postion
```

slice():

This method is used to fetch or slice out(Removing) the part of the given string.

It allows us to assign positive as well as negative index values.

It will allow two parameters which is starting index and ending index except the end index value.

Syntax:

```
console.log(str.slice(6,10));//Not taking end index value and in
between characters are output
console.log(str.slice(4));//If we pass one parameter it will remove
first 4 index values and print other. it takes 4 as an starting
index
console.log(str.slice(-6));//script
console.log(str.slice(-6,-7)); //If its more than the first value
then it will return just an empty space.
```

substring():

It is used to fetch the part of the given string based on the specified index.

It can't be assigned with negative values.

Syntax:

```
console.log(str.substring(6,10));//substring and slice are both same
performance. the difference between both is substring will not
accept negative values
console.log(str.substring(-6));//If we pass negative values it will
throw the original string as an output
```

substr():(Depricated):

This method is used to fetch the part of the string on the basis of starting position and length of characters to be removed.

Syntax:

```
console.log(str.substr(6,4));//Starting as a first parameter and 2nd
parameter as a no.of elements to be removed.
```

trim():

This method is used to remove the whitespace from the left and right side of the string.

Syntax:

```
let str3="                    Hello
javascript                      ";
console.log(str3); //It wont trim. it will print how it is actually
console.log(str3.trim());//It will remove the whitespaces and gives
the output as an normal string.
```

toLowerCase():

It converts a given string into lowercase letters.

Syntax:

```
console.log(str.toLowerCase());//it will print as an lowercase
letters.
```

toUpperCase():

It converts a given string into uppercase letters.

Syntax:

```
console.log(str.toUpperCase());//it will print as an uppercase
letters
```

toLocaleLowerCase():

It converts the string into lowercase letters on the basis of current local.

Syntax:

```
console.log(str6.toLocaleLowerCase());
```

toLocaleUpperCase():

It converts a given string into uppercase letters on the basis of current local.

Syntax:

```
console.log(str6.toLocaleUpperCase());
```

charCodeAt():

It provides unicode value of an character present at the specified index.

Syntax:

```
console.log(str.charCodeAt(1));//ASCII value of e is 101 is output
```

codePointAt():

Both charCodeAt() and codePointAt() both works as same.

Syntax:

```
console.log(str.charCodeAt(1));//ASCII value of e is 101 is output
```

toString:

It provides a string representing to the particular object.

Syntax:

```
let str5="1000";
console.log(str5.toString());//1000 as a string
```

valueOf():

This method provides primitive value of string object.

Syntax:

```
console.log(str5.valueOf());//1000 as a string
```

Array:

Array is also one of the JavaScript object reference datatype, Non primitive datatype.
Arrays are used to store multiple values in a single variable(Array literals).
An array which can hold more than one value at a time.
Array is used to store both homogeneous and heterogeneous datatypes.

Syntax of an array:

By using Array literals:

Variablename Arrayname=[];
Ex: let arr=[]
let arr=["John","Pradeep","Manoj","Tharun","Sathish","Sanjeev"]//Homogeneous data
let arr1=["john",1,69.0,"B.sc",true,{college:"msu"},[1,2,3,4,5]]//Heterogeneous data

By using Array Constructor:

Ex: let arr=new Array();

```
let arr3=new Array(1,"hi",2);//Creating array using contstructors
```

```
Ex of Accessing array element by using index:
    console.log(arr[3]);
We Can add elements in the array using index values:
    let arr4=[];
    Arr4[0]="Hi";
```

Array Methods():

- Push()
- Pop()
- Unshift()
- Shift()
- Splice()
- Slice()
- Joins()
- Reverse()
- Concat()
- indexOf()
- lastIndexOf()
- Find()
- FindIndex()
- Fill()
- Flat()

- Some()
- Every()
- Includes()
- Sort()
- Keys()
- Values()
- Entries()

Push():

Array.push() appends or adding(insert) an array element at the last index.

Syntax:

```
arr.push("Hey guys GM");
```

Pop():

Pop() remove last element in an array.

Syntax:

```
console.log(arr.pop());
```

Note: pop() method is one of the method to empty an array.

Unshift():

Unshift() is used to add an element to the first index of an array.

Syntax:

```
arr.unshift("Hi guys");
```

Note: unshift() will alter the index position of an element.

It will shift an elements index position from lower index to higher index.

Shift():

Shift() remove an first element in an array.

Syntax:

```
arr.shift();
```

Note: shift() elements index position from higher index to lower index.

Splice():

Splice() change values or deleting values in an array with particular position.

Syntax:

```
//Splice - It will take 3 parameters. 1st parameter start index, 2nd
is count how many you want to remove. the third one is optional
which is used to add the values on the removed elements.
let y5=arr.splice(5,2,"Radha","Mahalingam","Kodi")//WE can add
multiple values in the 3rd parameters
console.log(arr);//It will alter the original array
console.log(y5);
```

Slice():

Slice() returns extract value from an array or slice() is used to remove an element from an array.

Syntax:

```
//Slice - we can use negative value to remove from right to left
let y4=arr.slice(2,4);//Slice out the elements from the array. First
parameter is starting index 2nd parameter is end index
console.log(arr);//It will not alter the original array
console.log(y4.slice(1));
console.log(arr.slice(3));
console.log(y4);//Removing elements only printing
```

Note: slice() will not modify the original value.

The return type of slice() is extracted value of an array.

Join():

Join() return new string from an array.

```
console.log(arr.join());//It will return as single string
console.log(arr.join(""));//It will give space in between the
multiple words
console.log(arr.join("/"));//it will print / between words
console.log(arr.join("/OR/"));
```

Reverse():

Reverse() reverse an array. First element becomes last element and last element becomes first.

Syntax:

```
console.log(arr.reverse());//inbuilt method reversing the array
elements
```

```
//To Reverse an string and print using array
let str="Peculiar John";
console.log(str);
let m=str.split("");
console.log(m);
let c=m.reverse();
console.log(c);
console.log(c.join(""));
//Method changing method to get output in a single line
let n=str.split("").reverse().join("");
console.log(n);
```

IndexOf():

indexOf() returns the first index that the given element in an array.

If index not found it returns -1.

indexOf() returns index of the first occurs of the value in an array

IndexOf() check an element from left to right

Syntax:

```
console.log(arr.indexOf("John"));//This method gives the index
position of the given value.
```

LastIndexOf():

lastIndexOf() returns the index of the last occurs of the specified value in an array.

It will return -1 if the index not found.

lastIndexOf() check the element from right to left.

Syntax:

```
console.log(arr.lastIndexOf("Pradeep"));//Its check the element from
right to left. From right first occur index will be the output.
```

Concat():

Concat() used to add two or more array into an single array.

Syntax:

```
console.log(arr.concat(arr1));//We can concat multiple array in the
single array.
```

find():

Find() returns the value of the first element in the provided array that satisfies the given testing condition.

If no value satisfies the testing condition it returns **undefined**.

Syntax:

```
let x=arr.find((str)=>str=="John");
```

findIndex():

findIndex() returns the index of the first element in the array that satisfies the provided testing condition.
If the condition isn't satisfies It will return -1.
Syntax:
```
console.log(arr.findIndex((val)=>val="John"));
```
Fill():
Fill() method change the all element in an array to a static value(same value), from start index to an end index.
It returns modified array.
Syntax:
```
console.log(arr.fill("Hey Pappa ne konjam nillu"));
```
Flat():
Flat() creates a new array with all sub elements array.
Flat() converts a multi-dimensional array into single-dimensional array.
Syntax:

Some():
Some() method test whether atleast one element in the array passes the test condition.
It returns true if the condition is satisfies or else it return false,
Syntax:
```
console.log(arr.some((man)=>man>"Tharun"));
```
Every():
Every() tests whether all element in the array pass the test condition.
It returns true if all the element passed test condition else returns false.
Syntax:
```
console.log(arr.every((check)=> check="John"));
```
Include():
Include() determines whether the array includes a certain values among its elements.
It returns true if the element present or else return false.
Syntax:
```
console.log(arr.includes("Manoj"));
```
Sort():
Sort() the element of an array.
The default sort order is ascending. We can descending by using callback function
Syntax:
```
let y3=arr.sort((a,b)=>a-b);//b-a for descending. a-b for ascending.
by default ascending
```
Keys():
Keys() returns new array iterator object that contains the keys for each index in the array.
Syntax:
```
let y1=arr.keys();//Keys will return index position
//By using next method
console.log(y1.next());
console.log(y1.next());

for(let keys of y1){
    console.log(keys);
}
```
Values():
Value() returns a new array iterator object that contains the values for each index in the array.
Syntax:

```
let y2=arr.values();//values will return the elements only
console.log(y2.next());
console.log(y2.next());
for(let va of y2){
    console.log(va);
}
```

Entries():

Entries() returns a new array iterator object that contains the key: value pairs for each index in the array.

Syntax:

```
let y=(arr.entries());//Entries will return the index position and
elements
console.log(y.next().value);//We use next method to iterate the
values and print it.
console.log(y.next().value);
//By using for of loop we can easily iterate the values and print
for(let val of y){
    console.log(val);
}
```

Map():

Map() method calls a define callback function on each element of an array and returns an array(copied array) that contains the result.

Map() returns new array instance.

map() supports method changing.

Syntax:

```
let arr6=[5,4,6,8,2,0,9,1];
let j=arr6.map((value,index)=>{
    console.log(value);
    console.log(index);
    return value +5;// map method only add and reduce the values. if
we pass condition on map it will return boolean values.
});
console.log(j);//it will return the modified value
console.log(arr6);//It will not affect the original value of an
array
```

forEach():

forEach() is one of the array instance method in JavaScript for iterate or hydrating an array element.

forEach() calls the callback function one time for each element in the array.

When we use an return keyword the output value will be undefined.

forEach() will not support method changing.

Syntax:

```
let h=arr6.forEach((val,i)=>{
    console.log(val);//Iterating the values
    console.log(i);
    return val;
 });
console.log(h);//It will return undefined
```

Filter():

Filter() creates a new array with all elements that pass the test condition given in the method.

Filter() supports method changing.

Syntax:

```
let i=arr6.filter((val)=>{ //Index is not mandatory for filter.
        console.log(val);
        return val >4;//It checks the condition
    });
    console.log(i);
    console.log(arr6);
```

Reduce():

Reduce() iterate each array element and returns single output value.

Syntax:

```
let k=arr6.reduce((accumlator,val)=>{
    return accumlator*val;
    });
console.log(k);
```

Array.isArray():

This method determines whether the array is pure or impure array.

Syntax:

```
let hi=[1,2,3,4,5];
console.log(Array.isArray(hi));
```

Array.from():

This method will create a new shallow copies array instance from array like object or iterable object.

This method is also used for converting impure array into pure array.

Syntax:

```
let str3="How was the day john?";
let d=str3.split("");
console.log(d);
let e=Array.from(str3);
console.log(e);
function demo(){
return arguments;
}
let data=demo(1,2,3,4,5,6,7);
console.log(data);
console.log(Array.isArray(data));
let val=Array.from(data)
console.log(val);
console.log(Array.isArray(val));
```

Array.of():

Array.of() creates new array instance from the variable no.of arguments.

Syntax:

```
let z=Array.of(10,20,30,40,50);
console.log(z);//Given argument as an array
console.log(Array.of(10));
console.log(Array.of("HTML","CSS","JS"));
```

Rest Parameter():

Rest parameter that has the prefix of (…)

Rest parameter allows you to represent an indefinite no.of arguments as an array.

The rest parameter should always be written as the last parameter in the list.

Syntax:

```
function test(a,b,...c){
    console.log(a);
    console.log(b);
    console.log(c);
}
```

```
let ab=test(1,2,3,4,5,6,7)//We pass n number of arguments but a and
b took 1 and 2 remaining arguments are taken and store by the rest
parameter variable
console.log(Array.isArray(ab));//In rest parameter the return type
of array should be impure
```

Spread operator(…):

Spread operator is also denoted with(…) allows us to copy all or part of the existing array or object into another array or object.

Syntax Ex1:

```
let str4="Try to be nice";
let cd=[...str4]
console.log(cd);
console.log(Array.isArray(cd));//In spread operator the return type
of array is pure
```

Ex2:

```
//by using spread operator - it's create a new array so it will not
affect the original array
let value=[...ac];
value.splice(1,3);
console.log(value);
console.log(ad);
```

Ex3 Using spread operator in function:

```
function student(student1,student2, student3, student4){
console.log(student1, student2, student3, student4);
};
student(...arr)
```

Object:

Object is a type of variable that stores the data in the form of key: value pairs,
Object is an entity which stores the data in the form of key: value pairs.
Syntax:

```
let obj={
    name:"John",
    age:23,
    course:"Core java"
}
console.log(obj);
```

An object is a collection of properties
A propety value can be a string, function, null, undefined, symbol,etc.,
In JavaScript almost everything is an object.
Boolean can be object
Number, string can be object.
Data, math, arrray, functions and objects are always an js objects.
Retrieving the Data:

- By using . notation
- By using [] notation

Syntax for By using .notaton:

```
// !Retrieving the data
console.log(obj.name);//Dot notation
console.log(obj.age);
console.log(obj.course);
```

Syntax for [] notation:

```
console.log(obj['name']);//Bracket notation
console.log(obj['age']);
console.log(obj['course']);
```

To add the data:

```
//!Declaring an empty object and adding values through dot and bracket
notation
obj2.designation="Soft dev.";//Adding data
obj2['city']="vandalur";
console.log(obj2);
```

Reinitializing the Existing value or object mutation:

Syntax:

```
obj={
    name:"Peculiar John",
    age:23,
    Course:"Web Technology"
};
console.log(obj);
```

Deleting the key from an object using DELETE Property:

Syntax:

```
delete obj.age;
console.log(obj);
```

Multiple object creation and Accessing the object values:

Syntax:

Object Methods:

- Object.entries()
- Object.values()
- Object.keys()
- Object.seal()
- Object.isSealed()
- Object.freeze()
- Objet.isFrozen()
- Object.assign()
- Object.create()

Object.entries():

This method returns the array of a given object own enumerable key and value pairs.

Syntax:

```
let entry=Object.entries(obj3)
console.log(entry);
```

This method returns the array of both key and value of an object.

Object.values():

This method returns array of a given object on enumerable property of an values.

Syntax:

```
let val=Object.values(obj3)
console.log(val);
```

Object.keys():

Object.keys() returns an array of a given object enumerable property names(keys).

Syntax:

```
let key=Object.keys(obj3)
console.log(key);
```

Object.seal():

Object.seal() prevents new properties been added and it prevents other codes from deleting properties of an object.
In seal() we can't add a new property but we can modify the existing property values.
Syntax:
```
Object.seal(obj3);
obj3.salary="100000";//If we used seal metod we can't add the entries
console.log(obj3);
obj3.name="John";//We can reinitialize the value. values can be mutable in seal
console.log(obj3);
```
Object.isSealed():

This method determines if an object is sealed or not and return boolean values.
Syntax:
```
console.log(Object.isSealed((obj3)));// checking whether the object is sealed or not. if it's it returns true or else false.
```
Object.freeze():

This method freezes an object so that, other codes can't delete or change its properties.
In freeze() we can't add a new property as well as we can't modify the existing property values also.
Syntax:
```
Object.freeze(obj3);
obj3.location="Chennai";//We can't add the values when freeze method on
console.log(obj3);
```
Object.isFrozen():

This method determines if an object is freezed or not and returns boolean value
Syntax:
```
console.log(Object.isFrozen(obj3));//checking whether the object is frozen or not. if it's it returns true or else false.
```
Object.assign():

This method copies all enumerable own properties from source object to a target object. And it returns a modified target object.
Syntax:
```
let x=Object.assign({},obj3);// Accept 2 values. they are, Target and source object
console.log(x);
x.location="Chennai";
console.log(obj3);
console.log(x);//If we modified the value on the copied object it will not affect the original object.
Ex2:
//!if we want add the values on the existing object we can also use assign method:
let y=Object.assign(obj3,{location:"Puthukudi",salary:1000000});
//We can assign the values for the existing object also
console.log(y);
```
Ex3:
```
//!Ovverride a key name of an source object into the target object
let tar={
    name:"john",
    company:"Mindtree",
    role:"Full-stackdeveloper"
}
let src={
    company:"Wipro",
    city:"Chennai"
}
let z=Object.assign(tar,src)
console.log(z);//If target and source attribut has a same key
property it will override the source property.
```

Object.create():

This method create an new object using an existing object as an prototype of the newly created object.

The object can also be create by passing null in the create().

Syntax Ex1:

```
//!if we want add the values on the existing object we can also
use assign method
let y=Object.assign(obj3,{location:"Puthukudi",salary:1000000});
//We can assign the values for the existing object also
console.log(y);
```

Ex2:

```
//!Create object method
let x1=Object.create(obj3)
console.log(obj3);
console.log(x1);//Obj3 informations are store it in the
prototype based object
console.log(x1.name);//We can access the prototype object
properties.
```

Ex3:

```
//!If we want to create an empty object we use null in create
method
let y1=Object.create(null);
console.log(y1);//Empty object is created
y1.name="Pradeep";
y1.company="TCS";
y1.role="Front-End developer";
console.log(y1);//Properties are added to the object.
```

Apply():

Apply() method calls the function directly this to the first argument passed to the apply() and if any other arguments provided as an array or passed as an arguments to the function.

Syntax:

Bind():

The bind () creates a new function or bound function and when that function is called First it sets this keyword to the first argument which is passed to the bind(), if any other sequences of arguments or passed to the bind method then they are passed as an argument to the new function.

Syntax:

```
let x5=demo.apply(values,[30,40])
console.log(x5);
```

Ex1:

```
function demo (c,d){
        return this.a+this.b+c+d;
    }
    let x7=demo.bind(values,5,5);
    console.log(x7());
```

Math objects:

Math objects are used to perform mathematical performance on numbers.

Math objects are,

- Math.PI()
- Math.random()
- Math.floor()
- Math.ceil()
- Math.round()
- Math.trunc()
- Math.abs()
- Math.min()
- Math.max()
- Math.pow()
- Math.sqrt()
- Math.cbrt()

Math.PI():

It returns the PI value of 3.14.

Syntax:

```
console.log(Math.PI);//Pi value - 3.14
```

Math.random():

This method generate random numbers between 0-1 by default.

If we want random numbers between any number - we have to provide the number with random method.

Syntax:

```
console.log(Math.random()*10);//By default 0-1. It will randomly
pick numbers from 0-10. and not repeating the values.
```

Math.floor():

This method push value to the nearest lower value.

Syntax:

```
console.log(Math.floor(10.9));//It gives lowest value upto 10.99
```

Math.ceil():

This method push to the next highest integer or value

Syntax:

```
console.log(Math.ceil(10.9));//It gives highest value from 10.01
to 10.99 and gives 11.
```

Math.round():

This method round of the number to the nearest integer.

If the number from 10.01-10.49 it push the value to 10.

If the values from 10.5-10.99 it pushes to the next highest integer 11.

Syntax:

```
console.log(Math.round(10.49));//It's round of to the nearest
value upto 10.49 its give 10 as an output. from 10.5 it gives
11.
```

Math.trunc():

This method eliminates the decimal value of an integer.

Syntax:

```
console.log(Math.trunc(10.1));//It will removes the decimal
values and gives output as an integer of the lowest value of it
```

Math.abs():

This method returns the absolute or the positive value of an individuals if the input is negative value and it returns the positive value for negative integer.

Syntax:

```
console.log(Math.abs(-10.5));//It will remove the negative
values and print the exact value in positive
```

Math.min():

It returns minimum values among the values passed as an argument.

Syntax:

```
console.log(Math.min(2,5,30,439,7993));//Gives the minimum value
```

Math.max():

It returns maximum values among the values passed as an argument.

Syntax:

```
console.log(Math.max(2,5,30,439,7993));//Gives the maximum value
```

Math.pow():

This method valuates the power of the numbers.

Syntax:

```
console.log(Math.pow(2,5));//First value base value, 2nd value
is power value, 2power 5 is 32
```

Math.sqrt():

It returns the square of a number.

Syntax:

```
console.log(Math.sqrt(4));//Square root of 4 is 2
```

Math.cbrt():

It returns the cube of a number.

Syntax:

```
console.log(Math.cbrt(27));//Cube root of 27 is 3
```

JSON Object:

JSON stands for JavaScript Object Notation.

JSON is a light-weight format for storing an transporting data from client to server and vice-versa.

Rule for Syntax:

Data is in name and value pairs.
Data is separated by commas.
Both key and values except number should be closed with double quotes.

Syntax:

```
{
"userName"="John",
"age"=23
}
```

What is the difference between JavaScript object and JSON object:

JavaScript Object	JSON Object

Let X={
userName="John",
Age:20;
	{
	"userName"="John",
	"age"=23

	}
• We use variable to declare an object	• We don't use variables
• Keys shouldn't in double quotes	• Keys and values except number should be inside a double quotes
• JS supports undefined, symbol and bigInt.	• JSON not supports symbol, undefined and bigInt

JSON Methods:

- JSON.stringfy
- JSON.parse

JSON.stringfy:
When sending data to a webserver(from client to server) the data has to be a string.
Convert a JavaScript into an string with JSON.stringfy method.
Syntax:

JSON.parse:
JSON.parse method parse a string and returns a JavaScript object.
Syntax:

Get attribute:
The get attribute method of the element interface returns the value of a specif9ied attribute of an element.
It a given attribute doesn't exists, it return either null or empty string.
Syntax:
<div class="demo">hello</div>
Let div=document.querySelector("demo");
Console.log(div.getAttribute("class"));

Remove attribute:
Thie method removes the attribute with the specified name from the element.
Syntax:
Console.log(div

ClassList():
Class is used to add class to an element.
By using class list if we are adding a class name this will not override the existing class value.

DOM token list method:

- Classlist.Add()
- Classlist.Remove()
- Classlist.Toggle()

classList.add():
This method is used to add given token to the list.
Syntax:
classList.add("tokenName");

classList.remove():

Thismethod is used to remove the token from the list.
classList.remove("tokenName");

classList.toggle():

This method of the DOM token list interface removes a given token from the list and returns false.
If token doesn't exist in the element the function return true.
Syntax:
classList.toggle("token");
Ex: HTML
```
<button>click</button>
<div class="test">hello DOM</div>
```
Ex: JavaScript
```
Let btn=document.QuerySelector("button");
Btn.addEventListener("click",e=>{
Div.classList.toggle("hide")
})
```

Events:

Events are actions or occurences that happen in the system your programming which the system tells you about. So you can response to them in same way.
Events available are,

- onClick()
- mouseOver()
- mouseLeave()
- onDblClick()
- keyUp()
- keyDown()
- onChange()
- onSumbit()
- onLoad()..etc.,

Events Can be added in 3 ways,

First Way of Adding events:
```
HTML: <button onclick="data()">Click</button>
JavaScript: function data(){
Console.log("Clicked")
}
```
Second way of Adding Events:
```
HTML: <div>click</div>
JavaScript: let div=document.querySelector("div")
Div.ondblClick=function(){
Document.body.style.bodyground="red"
}
```
Third Way of Adding Events:
```
HTML: <button>click</button>
JavaScript: let btn=document.querySelector("button");
Btn.addEventListener("mouseenter",()=>{
Document.body.style.background="crimson"
})
Btn.addEventListener("mouseleave",()=>{
Document.body.style.background="teal"
})
```

Event deligation:

Capturing and bubbling allow us to implement one of the most powerful event handling patterns called event deligations.

Event deligations to prevent bubbling phase as a methods such as,

- Stop propogations
- Prevent default..Etc.,

Syntax:

This event deligation is an pattern or idea where we can use when we have a lot of elements handled in a similar way. Then, instead of assigning a handler to each of them we put a single handler on their common anchestor.

Syntax:

Web Storage:

Web storage consists of 2 storages..

- Local Storage-10MB(Permanent Storage)
- Session Storage-5MB(Session period of time)

Local Storage:

Local storage is a form of web storage that stores data for a long time.

Local storage allows developer to store and retrieve data in the browser.

The data stored in local storage will not expired.