

What is JavaScript

JavaScript is a programming language that executes on the browser. It turns static HTML web pages into interactive web pages by dynamically updating content, validating form data, controlling multimedia, animate images, and almost everything else on the web pages.

JavaScript is the third most important web technology after HTML and CSS. JavaScript can be used to create web and mobile applications, build web servers, create games, etc.

HTML script Tag

The HTML script tag `<script>` is used to embed data or executable client side scripting language in an HTML page. Mostly, JavaScript or JavaScript based API code inside a `<script></script>` tag.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
  <h1> JavaScript Tutorials</h1>

  <script>
    //write JavaScript code here..
    alert('Hello, how are you?')
  </script>
</body>
</html>
```

Multiple Script Tag:

```
<!DOCTYPE html>
<html>
<head>
  <script>
    alert('Executing JavaScript 1')
  </script>
</head>
<body>
  <h1> JavaScript Tutorials</h1>

  <script>
    alert('Executing JavaScript 2')
  </script>
```

```
<p>This page contains multiple script tags.</p>
```

```
<script>
    alert('Executing JavaScript 3')
</script>
</body>
</html>
```

JavaScript Syntax

JavaScript code can be written inside [HTML Script Tags](#) or in a separate file with .js extension.

```
<script>
    //Write javascript code here...

</script>
```

Case Sensitive

JavaScript is a case-sensitive scripting language. So, name of functions, variables and keywords are case sensitive. For example, myfunction and MyFunction are different, Name is not equal to nAme, etc.

Variables

In JavaScript, a variable is declared with or without the var keyword.

```
<script>
    var name = "Steve";
    id = 10;
</script>
```

Semicolon

JavaScript statements are separated by a semicolon. However, it is not mandatory to end a statement with a semicolon, but it is recommended.

```
<script>
    var one = 1; two = 2; three = 3; //three different statements

    var four = 4; //single statement
    var five = "Five" //single statement without ;
</script>
```

Whitespaces

JavaScript ignores multiple spaces and tabs. The following statements are the same.

```
<script>
  var one =1;
  var one  = 1;
  var one   =    1;
</script>
```

Code Comments

A comment is single or multiple lines, which give some information about the current program. Comments are not for execution.

Write comment after double slashes // or write multiple lines of comments between /* and */

```
<script>
  var one =1; // this is a single line comment

  /* this
  is multi line
  comment*/

  var two = 2;
  var three = 3;
</script>
```

String

A string is a text in JavaScript. The text content must be enclosed in double or single quotation marks.

```
<script>
  var msg = "Hello World" //JavaScript string in double quotes

  var msg = 'Hello World' //JavaScript string in single quotes
</script>
```

Number

JavaScript allows you to work with any type of number like integer, float, hexadecimal etc. Number must NOT be wrapped in quotation marks.

```
<script>
  var num = 100;

  var flot = 10.5;
</script>
```

Boolean

As in other languages, JavaScript also includes true and false as a boolean value.

```
<script>
  var yes = true;

  var no = false;
</script>
```

JavaScript Reserved Keywords

var	function	if
else	do	while
for	switch	break
continue	return	try
catch	finally	debugger
case	class	this
default	FALSE	TRUE
in	instanceOf	typeof
new	null	throw
void	width	delete

JavaScript Message Boxes: alert(), confirm(), prompt()

JavaScript provides built-in global functions to display popup message boxes for different purposes.

- alert(message): Display a popup box with the specified message with the OK button.
- confirm(message): Display a popup box with the specified message with OK and Cancel buttons.
- prompt(message, defaultValue): Display a popup box to take the user's input with the OK and Cancel buttons.

alert()

The `alert()` function displays a message to the user to display some information to users. This alert box will have the OK button to close the alert box.

The `alert()` function takes a parameter of any type e.g., string, number, boolean etc. So, no need to convert a non-string type to a string type.

```
alert("This is an alert message box."); // display string message
```

```
alert('This is a number: ' + 100); // display result of a concatenation
```

```
alert(100); // display number
```

```
alert(Date()); // display current date
```

confirm()

Use the `confirm()` function to take the user's confirmation before starting some task. For example, you want to take the user's confirmation before saving, updating or deleting data.

The `confirm()` function displays a popup message to the user with two buttons, OK and Cancel. The `confirm()` function returns true if a user has clicked on the OK button or returns false if clicked on the Cancel button. You can use the return value to process further.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
    <h1>Demo: confirm() </h1>
```

```
    <button onclick="save()">Save Data</button>
```

```
    <p id="msg"></p>
```

```
    <script>
```

```
        function save(){
            var userPreference;
```

```
            if (confirm("Do you want to save changes?") == true) {
                userPreference = "Data saved successfully!";
```

```
            } else {
                userPreference = "Save Canceled!";
            }
```

```
            document.getElementById("msg").innerHTML =
            userPreference;
        }
```

```
    </script>
```

```

</body>
</html>
prompt()
Use the prompt() function to take the user's input to do further actions.
For example, use the prompt() function in the scenario where you want
to calculate EMI based on the user's preferred loan tenure.
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: prompt()</h1>
    <button onclick="myinput()">Click to enter your name</button>
    <p id="msg"></p>

    <script>
        function myinput(){
            var name = prompt("Enter Your Name:");

            if (name == null || name == "") {
                document.getElementById("msg").innerHTML =
"You did not enter anything. Please enter your name again";
            }
            else
            {
                document.getElementById("msg").innerHTML =
"You entered: " + name;
            }
        }
    </script>
</body>
</html>

```

Declare a Variable

In JavaScript, a variable can be declared using var, let, const keywords.

- var keyword is used to declare variables since JavaScript was created. It is confusing and error-prone when using variables declared using var.
- let keyword removes the confusion and error of var. It is the new and recommended way of declaring variables in JavaScript.
- const keyword is used to declare a constant variable that cannot be changed once assigned a value.

```

<!DOCTYPE html>
<html>

```

```

<body>
  <h1>Demo: JavaScript Variables </h1>
  <p id="output"></p>

  <script>
    let msg;
    msg = "Hello JavaScript!"; // assigning a string value

    document.getElementById("output").innerHTML = msg;
  </script>
</body>
</html>

```

Another Example:

```

<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript Variables </h1>

  <p id="p1"></p>
  <p id="p2"></p>
  <p id="p3"></p>

  <script>
    let name = "Steve"; //assigned string value
    let num = 100; //assigned numeric value
    let isActive = true; //assigned boolean value

//let name = "Steve", num = 100, isActive = true;

    document.getElementById("p1").innerHTML = name;
    document.getElementById("p2").innerHTML = num;
    document.getElementById("p3").innerHTML = isActive;

/*
let num1 = 100;
    let num2 = num1;

    document.getElementById("p1").innerHTML = num1;
    document.getElementById("p2").innerHTML = num2;
*/

```

```

/*
let name = "Steve",
    num = 100,
    isActive
    =
    true;
*/
</script>
</body>
</html>

```

JavaScript Variable Naming Conventions

- Variable names are case-sensitive in JavaScript. So, the variable names msg, MSG, Msg, mSg are considered separate variables.
- Variable names can contain letters, digits, or the symbols \$ and _.
- A variable name cannot start with a digit 0-9.
- A variable name cannot be a reserved keyword in JavaScript, e.g. var, function, return cannot be variable names.

Constant Variables in JavaScript

Use const keyword to declare a constant variable in JavaScript.

- Constant variables must be declared and initialized at the same time.
- The value of the constant variables can't be changed after initialized them

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Variables </h1>

    <p id="p1"></p>

    <script>
        const num = 100; //constant variable
        document.getElementById("p1").innerHTML = num;
    </script>

```



```

        num = 200; //error

        const name; //error: must assign a value
        name = "Steve"; //error
    </script>
</body>
</html>

```

The value of a constant variable cannot be changed but the content of the value can be changed. For example, if an object is assigned to a const variable then the underlying value of an object can be changed.

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Variables </h1>

    <p id="p1"></p>

    <script>
        const person = { name: 'Steve'};
        person.name = "Bill";

        document.getElementById("p1").innerHTML =
person.name;
    </script>
</body>
</html>

```

Variable Scope

In JavaScript, a variable can be declared either in the global scope or the local scope.

Global Variables

Variables declared out of any function are called global variables. They can be accessed anywhere in the JavaScript code, even inside any function.

Local Variables

Variables declared inside the function are called local variables of that function. They can only be accessed in the function where they are declared but not outside.

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Global and Local Variables </h1>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>

    <script>
        let greet = "Hello " // global variable

        function myfunction(){
            let msg = "JavaScript!"; //local variable

            document.getElementById("p1").innerHTML =
greet + msg;
        }

        myfunction();

        document.getElementById("p2").innerHTML = greet;
        document.getElementById("p3").innerHTML =
msg; //error:can't access local variable
    </script>
</body>
</html>

```

Declare Variables without var and let Keywords

Variables can be declared and initialized without the var or let keywords. However, a value must be assigned to a variable declared without the var keyword.

The variables declared without the var keyword become global variables, irrespective of where they are declared.

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Variables without var</h1>
    <p id="p1"></p>

    <script>

```

```
function myfunction(){
    msg = "Hello JavaScript!";
}
myfunction();
document.getElementById("p1").innerHTML =
msg; // msg becomes global variable so can be accessed here
</script>
</body>
</html>
```

Javascript Operators

JavaScript includes operators same as other languages. An operator performs some operation on single or multiple operands (data value) and produces a result. For example, in `1 + 2`, the `+` sign is an operator and 1 is left side operand and 2 is right side operand. The `+` operator performs the addition of two numeric values and returns a result.

JavaScript includes following categories of operators.

1. [Arithmetic Operators](#)
2. [Comparison Operators](#)
3. [Logical Operators](#)
4. [Assignment Operators](#)
5. [Conditional Operators](#)
6. [Ternary Operator](#)

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations between numeric operands.

Operator	Description
+	Adds two numeric operands.
-	Subtract right operand from left operand
*	Multiply two numeric operands.

/	Divide left operand by right operand.
%	Modulus operator. Returns remainder of two operands.
++	Increment operator. Increase operand value by one.
--	Decrement operator. Decrease value by one.

```

<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript Arithmetic Operators</h1>
  <p>x = 5, y = 10, z;</p>
  <p id="p1">x+y=</p>
  <p id="p2">y-x=</p>
  <p id="p3">x*y=</p>
  <p id="p4">y/x=</p>
  <p id="p5">x%2=</p>

  <script>
    let x = 5, y = 10;
    let z = x + y
    document.getElementById("p1").innerHTML += z; //returns x+y=15

    z = y - x;
    document.getElementById("p2").innerHTML += z; //returns 5

    z = x * y;
    document.getElementById("p3").innerHTML += z; //returns 50

    z = y / x;
    document.getElementById("p4").innerHTML += z; //returns 2

    z = x % 2;
    document.getElementById("p5").innerHTML += z; //returns 1
  </script>
</body>
</html>

```

Example: Post and Pre Increment/Decrement

```

<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript ++ and -- Operators</h1>

```

```

<p>x = 5;</p>
<p id="p1">x++=</p>
<p id="p2">x=</p>
<p id="p3">++x=</p>
<p id="p4">x--=</p>
<p id="p5">x=</p>
<p id="p6">--x=</p>

<script>
    let x = 5;

    document.getElementById("p1").innerHTML += x++;
//post increment => x++ = 5
    document.getElementById("p2").innerHTML += x;
// value changes here => x=6

    document.getElementById("p3").innerHTML += ++x;
//pre increment & value changes here => ++x = 7

    document.getElementById("p4").innerHTML += x--;
//post decrement => x- = 7
    document.getElementById("p5").innerHTML += x;
//value changes here => x= 6

    document.getElementById("p6").innerHTML += --x;
//pre decrement and value changes here => --x = 5
</script>
</body>
</html>

```

String Concatenation

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript + Operator</h1>

    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <p id="p4"></p>
    <p id="p5"></p>

```

```

<script>
    let a = 5, b = "Hello ", c = "World!", d = 10;

    document.getElementById("p1").innerHTML = a + b;

    document.getElementById("p2").innerHTML = b + c;

    document.getElementById("p3").innerHTML = a + d;

    document.getElementById("p4").innerHTML = b + true;

    document.getElementById("p5").innerHTML = c - b;
</script>
</body>
</html>

```

Comparison Operators

JavaScript provides comparison operators that compare two operands and return a boolean value `true` or `false`.

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Comparison Operators</h1>
    <p>
        let a = 5, b = 10, c = "5", x = a;
    </p>

    <p id="p1">a == c returns </p>
    <p id="p2">a === c returns </p>
    <p id="p3">a == x returns </p>
    <p id="p4">a != b returns </p>
    <p id="p5">a > b returns </p>
    <p id="p6">a < b returns </p>
    <p id="p7">a >= b returns </p>
    <p id="p8">a <= b returns </p>

    <script>
        let a = 5, b = 10, c = "5", x = a;

        document.getElementById("p1").innerHTML += a == c;

```

```
document.getElementById("p2").innerHTML += a === c;

document.getElementById("p3").innerHTML += a == x;

document.getElementById("p4").innerHTML += a != b;

document.getElementById("p5").innerHTML += a > b;

document.getElementById("p6").innerHTML += a < b;

document.getElementById("p7").innerHTML += a >= b;

document.getElementById("p8").innerHTML += a <= b;
</script>
</body>
</html>
```

Logical Operators

In JavaScript, the logical operators are used to combine two or more conditions. JavaScript provides the following logical operators.

```
<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript Logical Operators</h1>

  <p id="p1"></p>
  <p id="p2"></p>
  <p id="p3"></p>
  <p id="p4"></p>
  <p id="p5"></p>

  <script>
    let a = 5, b = 10;
```

```
document.getElementById("p1").innerHTML = (a !=  
b) && (a < b);
```

```
document.getElementById("p2").innerHTML = (a >  
b) || (a == b);
```

```
document.getElementById("p3").innerHTML = (a <  
b) || (a == b);
```

```
document.getElementById("p4").innerHTML = !(a <  
b);
```

```
document.getElementById("p5").innerHTML = !(a >  
b);
```

```
</script>  
</body>  
</html>
```

Assignment Operators

JavaScript provides the assignment operators to assign values to variables with less key strokes.

```
<!DOCTYPE html>  
<html>  
<body>  
  <h1>Example: JavaScript Assignment Operators</h1>  
  
  <p id="p1"></p>  
  <p id="p2"></p>  
  <p id="p3"></p>  
  <p id="p4"></p>  
  <p id="p5"></p>  
  <p id="p6"></p>  
  
  <script>  
    let x = 5, y = 10;
```



```
x = y;
document.getElementById("p1").innerHTML = x;

x += 1;
document.getElementById("p2").innerHTML = x;

x -= 1;
document.getElementById("p3").innerHTML = x;

x *= 5;
document.getElementById("p4").innerHTML = x;

x /= 5;
document.getElementById("p5").innerHTML = x;

x %= 2;
document.getElementById("p6").innerHTML = x;
```

```
</script>
</body>
</html>
```

Ternary Operator

JavaScript provides a special operator called ternary operator `:?` that assigns a value to a variable based on some condition. This is the short form of the [if else condition](#).

```
<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript Ternary Operators</h1>

  <p id="p1"></p>
  <p id="p2"></p>

  <script>
```

```
let a = 10, b = 5;
```

```
let c = a > b? a : b; //10>5 c=a
```

```
let x = a>b?"Yes":"No"
```

```
let d = a > b? b : a;
```

```
document.getElementById("p1").innerHTML = c;
```

```
document.getElementById("p2").innerHTML = d;
```

```
</script>
```

```
</body>
```

```
</html>
```

JavaScript Data Types

In JavaScript, you can assign different types of values (data) to a variable e.g. string, number, boolean, etc.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>Demo: JavaScript Variables </h1>
```

```
<p id="p1"></p>
```

```
<p id="p2"></p>
```

```
<p id="p3"></p>
```

```
<script>
```

```
let myvariable = 1; // numeric value
```

```
document.getElementById("p1").textContent =  
myvariable;
```

```
myvariable = 'one'; // string value
```

```
        document.getElementById("p2").textContent =
myvariable;

        myvariable = true; // Boolean value
        document.getElementById("p3").textContent =
myvariable;
    </script>
</body>
</html>
```

JavaScript Strings

In JavaScript, a string is a primitive data type that is used for textual data. JavaScript string must be enclosed in single quotes, double quotes, or backticks.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript String</h1>

    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>

    <script>
        let str1 = "This is a double quoted string.";
        let str2 = 'This is a single quoted string.';
        let str3 = `This is a template string.`;

        document.getElementById("p1").innerHTML = str1;
        document.getElementById("p2").innerHTML = str2;
        document.getElementById("p3").innerHTML = str3;
    </script>
</body>
</html>
```

The template string (using backticks) is used when you want to include the value of a variable or expressions into a string. Use `${variable or expression}` inside backticks as shown below.

```
<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript String</h1>

  <p id="p1"></p>

  <script>
    let amount = 1000, rate = 0.05, duration = 3;
    //let check = amount * (1+rate*duration)
    let result = `Total Amount Payble: ${amount*(1 +
rate*duration)}` ;

    document.getElementById("p1").innerHTML = result;
  </script>
</body>
</html>
```

JavaScript strings can be accessed using a [for loop](#), as shown below.

```
<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript String</h1>

  <p id="p1"></p>
  <p id="p2"></p>
  <p id="p3"></p>

  <script>
```

```
let str = 'Hello World';
```

```
for(let i =0; i< str.length;i++)
```

```
document.getElementById("p2").innerHTML =  
document.getElementById("p2").innerHTML + str[i];
```

```
for(let ch of str)
```

```
document.getElementById("p3").innerHTML =  
document.getElementById("p3").innerHTML + ch;
```

```
</script>
```

```
</body>
```

```
</html>
```

If you want to include the same quotes in a string value as surrounding quotes then use a backward slash (\) before the quotation mark inside the string value.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>Demo: Quotes in String</h1>
```

```
<p id="p1"></p>
```

```
<p id="p2"></p>
```

```
<script>
```

```
let str1 = "This is \"simple\" string";
```

```
let str2 = 'This is \'simple\' string';
```

```
document.getElementById("p1").innerHTML = str1;
```

```
document.getElementById("p2").innerHTML = str2;
```

```
</script>
```

```
</body>
```

```
</html>
```

Example 1: if Statement

// check if the number is positive

```
const number = prompt("Enter a number: ");
```

// check if number is greater than 0

```
if (number > 0) {
```

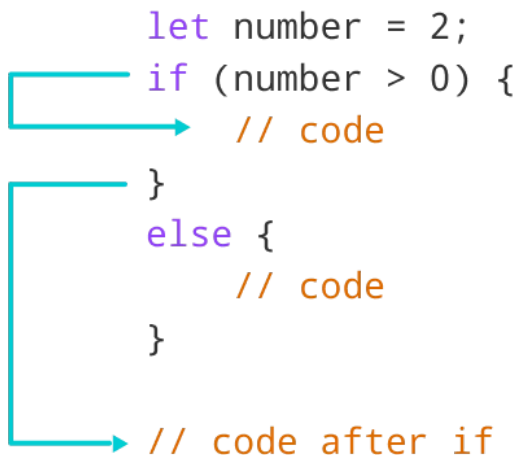
// the body of the if statement

```
  console.log("The number is positive");
```

```
}
```

```
console.log("The if statement is easy");
```

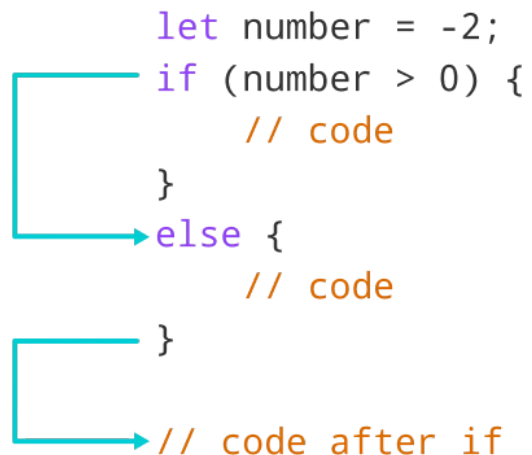
Condition is true



```
let number = 2;  
if (number > 0) {  
  // code  
}  
else {  
  // code  
}  
// code after if
```

The diagram illustrates the execution flow for the 'Condition is true' scenario. A teal arrow points from the 'if' statement to the code block inside the curly braces. Another teal arrow points from the 'else' statement to the code block inside its curly braces. A final teal arrow points from the end of the 'if...else' block to the code following it, indicating that the 'if' branch was executed.

Condition is false



```
let number = -2;  
if (number > 0) {  
  // code  
}  
else {  
  // code  
}  
// code after if
```

The diagram illustrates the execution flow for the 'Condition is false' scenario. A teal arrow points from the 'if' statement to the code block inside the curly braces. Another teal arrow points from the 'else' statement to the code block inside its curly braces. A final teal arrow points from the end of the 'if...else' block to the code following it, indicating that the 'else' branch was executed.

Example 2: if...else Statement

// check if the number is positive or negative/zero

```
const number = prompt("Enter a number: ");
```

```
// check if number is greater than 0
if (number > 0) {
  console.log("The number is positive");
}
// if number is not greater than 0
else {
  console.log("The number is either a negative number or 0");
}

console.log("The if...else statement is easy");
```

Example 3: if...else if Statement

```
// check if the number if positive, negative or zero
const number = prompt("Enter a number: ");

// check if number is greater than 0
if (number > 0) {
  console.log("The number is positive");
}
// check if number is 0
else if (number == 0) {
  console.log("The number is 0");
}
// if number is neither greater than 0, nor zero
else {
  console.log("The number is negative");
}

console.log("The if...else if...else statement is easy");
```

JavaScript for loop

The syntax of the for loop is:

```
for (initialExpression; condition; updateExpression) {
```

```
// for loop body
```

Example 1: Display a Text Five Times

```
// program to display text 5 times
const n = 5;

// looping from i = 1 to 5
for (let i = 1; i <= n; i++) {
    console.log(`I love JavaScript.`);
}
```

Output:

```
I love JavaScript.
I love JavaScript.
I love JavaScript.
I love JavaScript.
I love JavaScript.
```

Example 2: Display Numbers from 1 to 5

```
// program to display numbers from 1 to 5
const n = 5;

// looping from i = 1 to 5
// in each iteration, i is increased by 1
for (let i = 1; i <= n; i++) {
    console.log(i);    // printing the value of i
}
```

Example 3: Display Sum of n Natural Numbers

```
// program to display the sum of natural numbers
let sum = 0;
const n = 100

// looping from i = 1 to n
// in each iteration, i is increased by 1
```



```
for (let i = 1; i <= n; i++) {  
    sum += i; // sum = sum + i  
}
```

```
console.log('sum:', sum);
```

JavaScript while Loop

The syntax of the while loop is:

Example 1: Display Numbers from 1 to 5

```
// program to display numbers from 1 to 5  
// initialize the variable  
let i = 1, n = 5;  
  
// while loop from i = 1 to 5  
while (i <= n) {  
    console.log(i);  
    i += 1;  
}
```

Example 2: Sum of Positive Numbers Only

```
// program to find the sum of positive numbers  
// if the user enters a negative numbers, the loop ends  
// the negative number entered is not added to sum  
  
let sum = 0;  
  
// take input from the user  
let number = parseInt(prompt('Enter a number: '));  
  
while(number >= 0) {  
  
    // add all positive numbers  
    sum += number;
```

```
    // take input again if the number is positive
    number = parseInt(prompt('Enter a number: '));
}
```

```
// display the sum
console.log(`The sum is ${sum}.`);
```

JavaScript do...while Loop

The syntax of do...while loop is:

Example 3: Display Numbers from 1 to 5

```
// program to display numbers
let i = 1;
const n = 5;
```

```
// do...while loop from 1 to 5
do {
    console.log(i);
    i++;
} while(i <= n);
```

Example 4: Sum of Positive Numbers

```
// to find the sum of positive numbers
// if the user enters negative number, the loop terminates
// negative number is not added to sum
```

```
let sum = 0;
let number = 0;
```

```
do {
    sum += number;
    number = parseInt(prompt('Enter a number: '));
} while(number >= 0)
```

```
console.log(`The sum is ${sum}.`);
```

JavaScript break Statement

The `break` statement is used to terminate the loop immediately when it is encountered.

Example 1: break with for Loop

```
// program to print the value of i
for (let i = 1; i <= 5; i++) {
  // break condition
  if (i == 3) {
    break; //continue;
  }
  console.log(i);
}
```

Example 2: break with while Loop

```
// program to find the sum of positive numbers
// if the user enters a negative numbers, break ends the loop
// the negative number entered is not added to sum
```

```
let sum = 0, number;
```

```
while(true) {
```

```
  // take input again if the number is positive
  number = parseInt(prompt('Enter a number: '));
```

```
  // break condition
  if(number < 0) {
    break;
  }
```

```
  // add all positive numbers
  sum += number;
```

```
}
```

```
// display the sum  
console.log(`The sum is ${sum}.`);
```

JavaScript continue Statement

The `continue` statement is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.

Example 1: Print the Value of i

```
// program to print the value of i  
for (let i = 1; i <= 5; i++) {
```

```
    // condition to continue  
    if (i == 3) {  
        continue;  
    }
```

```
    console.log(i);  
}
```

Example 2: Calculate Positive Number

```
// program to calculate positive numbers only  
// if the user enters a negative number, that number is skipped from  
calculation
```

```
// negative number -> loop terminate  
// non-numeric character -> skip iteration
```

```
let sum = 0;  
let number = 0;
```

```
while (number >= 0) {
```

```
// add all positive numbers
sum += number;
```

```
// take input from the user
number = parseInt(prompt('Enter a number: '));
```

```
// continue condition
if (isNaN(number)) {
    console.log('You entered a string.');
```

number = 0; // the value of number is made 0 again

```
    continue;
}
```

```
}
```

```
// display the sum
console.log(`The sum is ${sum}.`);
```

continue with Nested Loop

```
// nested for loops
```

```
// first loop
```

```
for (let i = 1; i <= 3; i++) {
```

```
    // second loop
```

```
    for (let j = 1; j <= 3; j++) {
```

```
        if (j == 2) {
```

```
            continue;
```

```
        }
```

```
        console.log(`i = ${i}, j = ${j}`);
```

```
    }
```

```
}
```

Example 1: Simple Program Using switch Statement

```
// program using switch statement
```

```
let a = 3;
```

```
switch (a) {
  case 1:
    a = 'one';
    break;
  case 2:
    a = 'two';
    break;
  default:
    a = 'not found';
}
console.log(`The value is ${a}`);
```

Example 2: Type Checking in switch Statement

```
// program using switch statement
let a = 1;
```

```
switch (a) {
  case "1":
    a = 1;
    break;
  case 1:
    a = 'one';
    break;
  case 2:
    a = 'two';
    break;

  default:
    a = 'not found';
    break;
}
console.log(`The value is ${a}`);
```

Example 3: Simple Calculator

```
// program for a simple calculator
```

```
let result;

// take the operator input
const operator = prompt('Enter operator ( either +, -, * or / ): ');

// take the operand input
const number1 = parseFloat(prompt('Enter first number: '));
const number2 = parseFloat(prompt('Enter second number: '));

switch(operator) {
  case '+':
    result = number1 + number2;
    console.log(`${number1} + ${number2} = ${result}`);
    break;
  case '-':
    result = number1 - number2;
    console.log(`${number1} - ${number2} = ${result}`);
    break;
  case '*':
    result = number1 * number2;
    console.log(`${number1} * ${number2} = ${result}`);
    break;
  case '/':
    result = number1 / number2;
    console.log(`${number1} / ${number2} = ${result}`);
    break;

  default:
    console.log('Invalid operator');
    break;
}
```

Example 4: switch With Multiple Case

```
// multiple case switch program
let fruit = 'apple';
switch(fruit) {
  case 'apple':
```

```
case 'mango':
case 'pineapple':
    console.log(`${fruit} is a fruit.`);
    break;
default:
    console.log(`${fruit} is not a fruit.`);
    break;
}
```

Declaring a Function

The syntax to declare a function is:

```
function nameOfFunction () {
    // function body
}
```

Example 1: Display a Text

```
// program to print a text
// declaring a function
function greet() {
    console.log("Hello there!");
}
```

```
// calling the function
greet();
```

Example 2: Function with Parameters

```
// program to print the text
// declaring a function
function greet(name) {
    console.log("Hello " + name + ":");
}
```

```
// variable name can be different
let name = prompt("Enter a name: ");
```



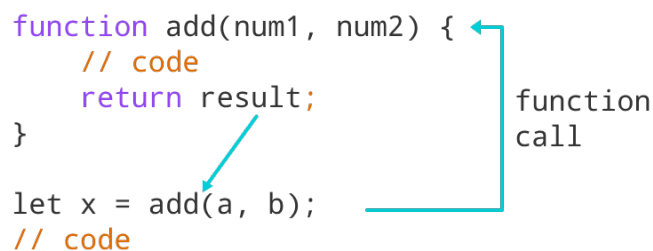
```
// calling function  
greet(name);
```

Example 3: Add Two Numbers

```
// program to add two numbers using a function  
// declaring a function  
function add(a, b) {  
    console.log(a + b);  
}
```

```
// calling functions  
add(3,4);  
add(2,9);
```

```
function add(num1, num2) {  
    // code  
    return result;  
}  
let x = add(a, b);  
// code
```



The diagram illustrates the execution of the function. A teal arrow points from the function definition's opening curly brace to the text "function call". Another teal arrow points from the "return result;" line to the "add(a, b)" part of the function call in the line below.

Example

4: Sum of Two Numbers

```
// program to add two numbers  
// declaring a function  
function add(a, b) {  
    return a + b;  
}
```

```
// take input from the user  
let number1 = parseFloat(prompt("Enter first number: "));  
let number2 = parseFloat(prompt("Enter second number: "));
```

```
// calling function  
let result = add(number1,number2);
```

```
// display the result
console.log("The sum is " + result);
```

Example 1: Local Scope Variable

```
// program showing local scope of a variable
let a = "hello";
```

```
function greet() {
  let b = "World"
  console.log(a + b);
}
```

```
greet();
console.log(a + b); // error
```

Example 2: block-scoped Variable

```
// program showing block-scoped concept
// global variable
let a = 'Hello';
```

```
function greet() {
```

```
  // local variable
  let b = 'World';
```

```
  console.log(a + ' ' + b);
```

```
  if (b == 'World') {
```

```
    // block-scoped variable
    let c = 'hello';
```

```
    console.log(a + ' ' + b + ' ' + c);
  }
```

```
// variable c cannot be accessed here
console.log(a + ' ' + b + ' ' + c);
}
```

```
greet();
```

String Concatenation

JavaScript string can be concatenated using the `+` operator or `string.concat()` method.

```
<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript String Concatenation</h1>

  <p id="p1"></p>
  <p id="p2"></p>

  <script>
    let str1 = 'Hello ';
    let str2 = "World ";

    let str3 = str1 + str2; //Hello World
    let str4 = str1.concat(str2); //Hello World

    document.getElementById("p1").innerHTML =
str3;
    document.getElementById("p2").innerHTML =
str4;
  </script>
</body>
</html>
```

Demo: Quotes in String

```
<!DOCTYPE html>
<html>
<body>
  <h1>Demo: Quotes in String</h1>

  <p id="p1"></p>
  <p id="p2"></p>

  <script>
    let str1 = "This is \"simple\" string";

    let str2 = 'This is \'simple\' string';

    document.getElementById("p1").innerHTML = str1;
    document.getElementById("p2").innerHTML = str2;
  </script>
</body>
</html>
```

Strings Comparison

Two strings can be compared using `<`, `>`, `==`, `===` operator, and `string.localeCompare(string)` method.

The mathematical operators `<` and `>` compare two strings and return a boolean (true or false) based on the order of the characters in the string.

The `==` operator compares the content of strings and `===` compares the reference equality of strings. The `localeCompare()` method compares two strings in the current locale. It returns `0` if strings are equal, else returns `1`.

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Strings Comparison</h1>

    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <p id="p4"></p>
    <p id="p5"></p>
    <p id="p6"></p>
    <p id="p7"></p>
    <p id="p8"></p>

    <script>
        document.getElementById("p1").innerHTML = "a" <
        "b"; //true
        document.getElementById("p2").innerHTML = "b" <
        "a"; //false
        document.getElementById("p3").innerHTML =
        "Apple" == "Apple"; //true
        document.getElementById("p4").innerHTML =
        "Apple" == "apple"; //false
        document.getElementById("p5").innerHTML =
        "Apple" === "Apple"; //true
        document.getElementById("p6").innerHTML =
        "Apple" === "apple"; //false
        document.getElementById("p7").innerHTML =
        "Apple".localeCompare("Apple"); //0
        document.getElementById("p8").innerHTML =
        "Apple".localeCompare("apple"); //1
    </script>
</body>
</html>

```

Note that the `===` operator compares the reference of strings objects and not the values.

```

<!DOCTYPE html>

```

```

<html>
<body>
    <h1>Demo: String & Object Comparison</h1>

    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <p id="p4"></p>

    <script>
        let str1 = "Hello";
        let str2 = 'Hello';
        let str3 = new String('Hello');

        document.getElementById("p1").innerHTML = str1
    == str2;
        document.getElementById("p2").innerHTML = str1
    === str2;
        document.getElementById("p3").innerHTML = str1
    == str3;
        document.getElementById("p4").innerHTML = str1
    === str3;
    </script>
</body>
</html>

```

String Properties

Property	Description
length	Returns the length of the string.

String Methods

Method	Description
charAt(position)	Returns the character at the specified position (in Number)

charAt(position)	Returns the character at the specified position (in Number).
charCodeAt(position)	Returns a number indicating the Unicode value of the character at the given position (in Number).
concat([string,,])	Joins specified string literal values (specify multiple strings separated by comma) and returns a new string.
indexOf(SearchString, Position)	Returns the index of first occurrence of specified String starting from specified number index. Returns -1 if not found.
lastIndexOf(SearchString, Position)	Returns the last occurrence index of specified SearchString, starting from specified position. Returns -1 if not found.
localeCompare(string position)	Compares two strings in the current locale.
match(RegExp)	Search a string for a match using specified regular expression. Returns a matching array.
replace(searchValue, replaceValue)	Search specified string value and replace with specified replace Value string and return new string. Regular expression can also be used as searchValue.
search(RegExp)	Search for a match based on specified regular expression.
slice(startNumber, endNumber)	Extracts a section of a string based on specified starting and ending index and returns a new string.
split(separatorString, limitNumber)	Splits a String into an array of strings by separating the string into substrings based on specified separator. Regular expression can also be used as separator.
substr(start, length)	Returns the characters in a string from specified starting position through the specified number of characters (length).
substring(start, end)	Returns the characters in a string between start and end indexes.
toLocaleLowerCase()	Converts a string to lower case according to current locale.
toLocaleUpperCase()	Converts a sting to upper case according to current locale.

split(separatorString, limitNumber)	Splits a String into an array of strings by separating the string into substrings based on specified separator. Regular expression can also be used as separator.
substr(start, length)	Returns the characters in a string from specified starting position through the specified number of characters (length).
substring(start, end)	Returns the characters in a string between start and end indexes.
toLocaleLowerCase()	Converts a string to lower case according to current locale.
toLocaleUpperCase()	Converts a string to upper case according to current locale.
toLowerCase()	Returns lower case string value.
toString()	Returns the value of String object.
toUpperCase()	Returns upper case string value.
valueOf()	Returns the primitive value of the specified string object.

The `length` property returns the number of characters in a string.

```
// defining a string
let sentence = "Program";
```

```
// returns number of characters in the sentence string
let len = sentence.length;
```

```
console.log(len);
```

```
// Output:
// 7
```


Example 2: length Property is Read Only

The `String.length` property is a read-only property. There will be no effect if we try to change it manually. For example:

```
let string2 = "Programming";

// assigning a value to string's length property
string2.length = 5;

// doesn't change the original string
console.log(string2); // Programming

// returns the length of 'Programming'
console.log(string2.length); // 11
```

The `charAt()` method returns the character at the specified index in a string.

Example

```
// string declaration
const str1 = "Hello World!";

// finding character at index 1
let index1 = str1.charAt(1);

console.log("Character at index 1 is " + index1);

// Output:
// Character at index 1 is e
```

Example 2: A Non-integer Index Value in `charAt()`

```
const string = "Hello World";

// finding character at index 6.3
let result1 = string.charAt(6.3);
```

```
console.log("Character at index 6.3 is " + result1);
```

```
// finding character at index 6.9
```

```
let result2 = string.charAt(6.9);
```

```
console.log("Character at index 6.9 is " + result2);
```

```
// finding character at index 6
```

```
let result3 = string.charAt(6);
```

```
console.log("Character at index 6 is " + result3);
```

Example 3: Without passing parameter in charAt()

```
let sentence = "Happy Birthday to you!";
```

```
// passing empty parameter in charAt()
```

```
let index4 = sentence.charAt();
```

```
console.log("Character at index 0 is " + index4);
```

Example 1: Using charCodeAt() Method

```
const greeting = "Good morning!";
```

```
// UTF-16 code unit of character at index 5
```

```
let result1 = greeting .charCodeAt(5);
```

```
console.log(result1);
```

```
// UTF-16 code unit of character at index 5.2
```

```
let result2 = greeting .charCodeAt(5.2);
```

```
console.log(result2);
```

```
// UTF-16 code unit of character at index 5.9
```

```
let result3 = greeting.charCodeAt(5.9);
```

```
console.log(result3);
```

Example 2: charCodeAt() Method for Index Out of Range

```
const greeting = "Good morning!";
```

```
// passing index greater than length of string
```

```
let result3 = greeting.charCodeAt(18);
```

```
console.log(result3);
```

```
// passing non-negative index value
```

```
let result4 = greeting.charCodeAt(-9);
```

```
console.log(result4);
```

Output:

NaN

NaN

Example 1: Using codePointAt() method

```
let fruit = "Apple";
```

```
// unicode code point of character A
```

```
let codePoint = fruit.codePointAt(0);
```

```
console.log("Unicode Code Point of 'A' is " + codePoint);
```

Output:

Unicode Code Point of 'A' is 65

Example 2: codePointAt() with Default Parameter

```
let message = "Happy Birthday";
```

```
// without passing parameter in codePointAt()
```

```
let codePoint = message.codePointAt();
```

```
console.log(codePoint);
```

```
// passing 0 as parameter
```

```
let codePoint0 = message.codePointAt(0);
```

```
console.log(codePoint0);
```

Output:

```
72
```

```
72
```

Example: Using concat() method

```
console.log("").concat({}); // [object Object]
```

```
console.log("").concat(null); // null
```

```
console.log("").concat(true); // true
```

```
console.log("").concat(4, 5); // 45
```

```
let str1 = "Hello";
```

```
let str2 = "World";
```

```
// concatenating two strings
```

```
let newStr = str1.concat(" ", str2, "!");
```

```
console.log(newStr); // Hello, World!
```

Output:

```
[object Object]
```

```
null
```

```
true
```

```
45
```

```
Hello, World!
```

Example 1: Using endsWith() Method

```
// string definition
```

```
let sentence = "JavaScript is fun";
```

```
// checking if the given string ends with "fun"
```

```
let check = sentence.endsWith("fun");
```

```
console.log(check);
```

```
// checking if the given string ends with "is"  
let check1 = sentence.endsWith("is");
```

```
console.log(check1);
```

Example 2: endsWith() for Case Sensitive Strings

The `endsWith()` method is case sensitive. For example,

```
// string definition  
let sentence = "JavaScript is fun";
```

```
// checking if the given string ends with "fun"  
let check = sentence.endsWith("fun");
```

```
console.log(check);
```

```
// checking if the given string ends with "Fun"  
let check1 = sentence.endsWith("Fun");
```

```
console.log(check1);
```

Javascript String match()

The `match()` method returns the result of matching a string against a regular expression.

```
const message = "JavaScript is a fun programming language.";
```

```
// regular expression that checks if message contains 'programming'  
const exp = /programming/;
```

```
// check if exp is present in message  
let result = message.match(exp);  
console.log(result);
```

```
/*  
Output: [  
  'programming',  
  index: 20,  
  input: 'JavaScript is a fun programming language.',  
  groups: undefined  
]  
*/
```

JavaScript String repeat()

The `repeat()` method creates a new string by repeating the given string a specified number of times and returns it.

```
const holiday = "Happy holiday!";
```

```
// repeating the given string 3 times  
const result = holiday.repeat(3);
```

```
console.log(result);
```

```
// Output:
```

```
// Happy holiday!Happy holiday!Happy holiday!
```

Example 1: Using repeat() Method

```
// string declaration  
const holiday = "Happy holiday!";
```

```
// repeating the given string 2 times  
const result = holiday.repeat(2);
```

```
console.log(result);
```

```
// using 0 as a count value  
// returns an empty string  
let result2 = holiday.repeat(0);
```

```
console.log(result2);
```

Example 2: Using Non-integer as a Count Value in repeat()

```
let sentence = "Happy Birthday to you!";
```

```
// using non-integer count value  
let result1 = sentence.repeat(3.2);
```

```
console.log(result1);
```

```
// using non-integer count value  
let result2 = sentence.repeat(3.7);
```

```
console.log(result2);
```

Example 3: Using Negative Number as a Count Value

The count value in the `repeat()` method must be a non-negative number. Otherwise, it throws an error. For example:

```
let sentence = "Happy Birthday to you!";
```

```
// using negative number as count value  
let result3 = sentence.repeat(-1);
```

```
console.log(result3);
```

Output:

```
RangeError: Invalid count value
```

JavaScript String replace()

The `replace()` method returns a new string with the specified string/regex replaced.

```
const message = "ball bat";
```

```
// replace the first b with c
```

```
let result = message.replace('b', 'c');  
console.log(result);
```

```
// Output: call bat
```

Example 1: Replace the first occurrence

```
const text = "Java is awesome. Java is fun."
```

```
// passing a string as the first parameter
```

```
let pattern = "Java";  
let new_text = text.replace(pattern, "JavaScript");  
console.log(new_text);
```

```
// passing a regex as the first parameter
```

```
pattern = /Java/;  
new_text = text.replace(pattern, "JavaScript");  
console.log(new_text);
```

JavaScript Booleans

The boolean (not Boolean) is a primitive data type in JavaScript. It can have only two values: true or false.

```
<!DOCTYPE html>  
<html>  
<body>  
  <h1>Demo: JavaScript Boolean</h1>  
  <script>  
    var YES = true;  
    var NO = false;  
  
    alert(YES);
```



```
        alert(NO);
    </script>
</body>
</html>
```

The following example demonstrates how a boolean value controls the program flow using the [if condition](#).

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Boolean</h1>
    <script>
        var YES = true;
        var NO = false;

        if(YES)
        {
            alert("This code block will be executed");
        }

        if(NO)
        {
            alert("This code block will not be executed");
        }
    </script>
</body>
</html>
```

The comparison expressions return boolean values to indicate whether the comparison is true or false. For example, the following expressions return boolean values.

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1>Demo: JavaScript Boolean</h1>
<p id="p1">1 > 2 = </p>
<p id="p2">a < b = </p>
<p id="p3">a > b = </p>
<p id="p4">a + 20 > b + 5 = </p>
<script>
    var a = 10, b = 20;

    var result = 1 > 2; // false
    document.getElementById("p1").textContent
+= result;

    result = a < b; // true
    document.getElementById("p2").textContent
+= result;

    result = a > b; // false
    document.getElementById("p3").textContent
+= result;

    result = a + 20 > b + 5; // true
    document.getElementById("p4").textContent
+= result;
</script>
</body>
</html>
```

JavaScript Objects: Create Objects, Access Properties & Methods

An object is a non-primitive, structured data type in JavaScript. Objects are same as variables in JavaScript, the only difference is that an object holds multiple values in terms of properties and methods.

In JavaScript, an object can be created in two ways: 1) using Object Literal/Initializer Syntax 2) using the `Object()` Constructor function with the [new keyword](#). Objects created using any of these methods are the same.

```
var p1 = { name:"Steve" }; // object literal syntax
```

```
var p2 = new Object(); // Object() constructor function  
p2.name = "Steve"; // property
```

Create Object using Object Literal Syntax

The object literal is a short form of creating an object. Define an object in the `{ }` brackets with key:value pairs separated by a comma. The key would be the name of the property and the value will be a literal value or a function.

```
var emptyObject = {}; // object with no properties or methods
```

```
var person = { firstName: "John" }; // object with single property
```

```
// object with single method
```

```
var message = {  
    showMessage: function (val) {  
        alert(val);  
    }  
};
```

```
// object with properties & method
var person = {
    firstName: "James",
    lastName: "Bond",
    age: 15,
    getFullName: function () {
        return this.firstName + '
' + this.lastName
    }
};
```

Create Objects using Objects() Constructor

Another way of creating objects is using the `Object()` constructor function using the [new](#) keyword. Properties and methods can be declared using the dot notation `.property-name` or using the square brackets `["property-name"]`, as shown below.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Object</h1>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <p id="p4"></p>
    <p id="p5"></p>

    <script>
        var person = new Object();
```

```

    // Attach properties and methods to person
object
    person.firstName = "James";
    person["lastName"] = "Bond";
    person.age = 25;
    person.getFullName = function () {
        return this.firstName + ' ' +
this.lastName;
    };

    // access properties & methods
    document.getElementById("p1").innerHTML =
person.firstName;
    document.getElementById("p2").innerHTML =
person.lastName;

    document.getElementById("p3").innerHTML =
person["firstName"];
    document.getElementById("p4").innerHTML =
person["lastName"];

    document.getElementById("p5").innerHTML =
person.getFullName();

</script>
</body>
</html>

```

Example: Variables as Object Properties

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Nested Object</h1>
    <p id="p1"></p>

```

```
<p id="p2"></p>

<script>
    var firstName = "James";
    var lastName = "Bond";

    var person = { firstName, lastName }

    document.getElementById("p1").textContent =
person.firstName;
    document.getElementById("p2").textContent =
person.lastName;

</script>
</body>
</html>
```

Access JavaScript Object Properties & Methods

An object's properties can be accessed using the dot notation `obj.property-name` or the square brackets `obj["property-name"]`. However, method can be invoked only using the dot notation with the parenthesis, `obj.method-name()`, as shown below.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Object</h1>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <p id="p4"></p>
    <p id="p5"></p>
```

```

<script>
    var person = {
        firstName: "James",
        lastName: "Bond",
        age: 25,
        getFullName: function () {
            return this.firstName + ' ' +
this.lastName
        }
    };

    document.getElementById("p1").innerHTML =
person.firstName;
    document.getElementById("p2").innerHTML =
person.lastName;

    document.getElementById("p3").innerHTML =
person["firstName"];
    document.getElementById("p4").innerHTML =
person["lastName"];

    document.getElementById("p5").innerHTML =
person.getFullName();

</script>
</body>
</html>

```

The properties and methods will be available only to an object where they are declared.

```

<!DOCTYPE html>
<html>

```

```
<body>
  <h1>Demo: JavaScript Nested Object</h1>
  <p id="p1"></p>
  <p id="p2"></p>
  <p id="p3"></p>
  <p id="p4"></p>
  <p id="p5"></p>
  <p id="p6"></p>

  <script>
    var p1 = new Object();
    p1.firstName = "James";
    p1.lastName  = "Bond";

    var p2 = new Object();

    document.getElementById("p1").textContent =
p2.firstName;
    document.getElementById("p2").textContent =
p2.lastName;

    p3 = p1; // assigns object
    p3.firstName; // James
    p3.lastName; // Bond

    document.getElementById("p3").textContent =
p3.firstName;
    document.getElementById("p4").textContent =
p3.lastName;

    p3.firstName = "Sachin"; // assigns new value
    p3.lastName  = "Tendulkar"; // assigns new
value
```



```
        document.getElementById("p5").textContent =  
p3.firstName;  
        document.getElementById("p6").textContent =  
p3.lastName;
```

```
    </script>  
</body>  
</html>
```

Enumerate Object's Properties

Use the `for in` loop to enumerate an object, as shown below

```
<!DOCTYPE html>  
<html>  
<body>  
    <h1>Demo: JavaScript Object using for-in Loop</h1>  
  
    <script>  
        var person = new Object();  
        person.firstName = "James";  
        person.lastName = "Bond";  
  
        for(var prop in person){  
            alert(prop); // access property name  
            alert(person[prop]); // access property  
value  
            };  
  
    </script>  
</body>
```

</html>

Pass by Reference

Object in JavaScript passes by reference from one function to another.

```
<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript Object</h1>
  <p id="p1"></p>
  <p id="p2"></p>

  <script>
    function changeFirstName(per)
    {
      per.firstName = "Steve";
    }

    var person = { firstName : "Bill" };

    document.getElementById("p1").textContent =
person.firstName;

    changeFirstName(person)

    document.getElementById("p2").textContent =
person.firstName;

  </script>
</body>
</html>
```



Points to Remember :

JavaScript object is a standalone entity that holds multiple values in terms of properties and methods.

Object property stores a literal value and method represents function.

An object can be created using object literal or object constructor syntax.

Object literal:

```
var person = {  
    firstName: "James",  
    lastName: "Bond",  
    age: 25,  
    getFullName: function () {  
        return this.firstName + ' ' +  
this.lastName  
    }  
};
```

Object constructor:

```
var person = new  
Object();  
  
person.firstName = "James";  
person["lastName"] = "Bond";  
person.age = 25;  
person.getFullName = function () {  
    return this.firstName + ' ' +  
this.lastName;  
};
```

Object properties and methods can be accessed using dot notation or [] bracket.

An object is passed by reference from one function to another.

An object can include another object as a property.

JavaScript Date: Create, Convert, Compare Dates in JavaScript

JavaScript provides Date object to work with date & time, including days, months, years, hours, minutes, seconds, and milliseconds.

Use the `Date()` function to get the string representation of the current date and time in JavaScript. Use the `new` keyword in JavaScript to get the Date object.

```
<!DOCTYPE html>
<html>
<body>
  <h1>Demo: Current Date</h1>
  <p id="p1"></p>
  <p id="p2"></p>

  <script>

    document.getElementById("p1").innerHTML = Date();

    var currentDate = new Date();

    document.getElementById("p2").innerHTML = currentDate;

  </script>
</body>
</html>
```

Date() Syntax

```
new Date()
new Date(value)
new Date(dateString)
new Date(year, monthIndex)
new Date(year, monthIndex, day)
```

```
new Date(year, monthIndex, day, hours)
new Date(year, monthIndex, day, hours, minutes)
new Date(year, monthIndex, day, hours, minutes,
seconds)
new Date(year, monthIndex, day, hours, minutes,
seconds, milliseconds)
```

Parameters:

- **No Parameters:** A date object will be set to the current date & time if no parameter is specified in the constructor.
- **value:** An integer value representing the number of milliseconds since January 1, 1970, 00:00:00 UTC.
- **dateString:** A string value that will be parsed using `Date.parse()` method.
- **year:** An integer value to represent a year of a date. Numbers from 0 to 99 map to the years 1900 to 1999. All others are actual years.
- **monthIndex:** An integer value to represent a month of a date. It starts with 0 for January till 11 for December
- **day:** An integer value to represent day of the month.
- **hours:** An integer value to represent the hour of a day between 0 to 23.
- **minutes:** An integer value to represent the minute of a time segment.
- **seconds:** An integer value to represent the second of a time segment.
- **milliseconds:** An integer value to represent the millisecond of a time segment. Specify numeric milliseconds in the constructor to get the date and time elapsed from 1/1/1970.

The following example shows various formats of a date string that can be specified in a `Date()` constructor.

```
<!DOCTYPE html>
<html>
<body>
  <h1>Demo: JavaScript Date Object</h1>
  <p id="p1"></p>
  <p id="p2"></p>
  <p id="p3"></p>
  <p id="p4"></p>
  <p id="p5"></p>
  <p id="p6"></p>
  <p id="p7"></p>
  <p id="p8"></p>
  <p id="p9"></p>

  <script>
    var date1 = new Date("3 march 2015");

    var date2 = new Date("3 February, 2015");

    var date3 = new Date("3rd February,
2015"); // invalid date

    var date4 = new Date("2015 3 February");

    var date5 = new Date("3 2015 February ");

    var date6 = new Date("February 3 2015");

    var date7 = new Date("February 2015 3");
```

```

        var date8 = new Date("3 2 2015");

        var date9 = new Date("3 march 2015
20:21:44");

        document.getElementById("p1").innerHTML =
date1;
        document.getElementById("p2").innerHTML =
date2;
        document.getElementById("p3").innerHTML =
date3;
        document.getElementById("p4").innerHTML =
date4;
        document.getElementById("p5").innerHTML =
date5;
        document.getElementById("p6").innerHTML =
date6;
        document.getElementById("p7").innerHTML =
date7;
        document.getElementById("p8").innerHTML =
date8;
        document.getElementById("p9").innerHTML =
date9;
    </script>
</body>
</html>

```

You can use any valid separator in the date string to differentiate date segments.

```

<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Date Object</h1>
    <p id="p1"></p>

```

```
<p id="p2"></p>
<p id="p3"></p>
<p id="p4"></p>
<p id="p5"></p>
<p id="p6"></p>
<p id="p7"></p>
<p id="p8"></p>
<p id="p9"></p>
```

```
<script>
```

```
    var date1 = new Date("February 2015-3");
```

```
    var date2 = new Date("February-2015-3");
```

```
    var date3 = new Date("February-2015-3");
```

```
    var date4 = new Date("February,2015-3");
```

```
    var date5 = new Date("February,2015,3");
```

```
    var date6 = new Date("February*2015,3");
```

```
    var date7 = new Date("February$2015$3");
```

```
    var date8 = new Date("3-2-2015"); // MM-dd-
```

```
YYYY
```

```
    var date9 = new Date("3/2/2015"); // MM-dd-
```

```
YYYY
```

```
    document.getElementById("p1").innerHTML =
date1;
```

```
    document.getElementById("p2").innerHTML =
date2;
```



```
        document.getElementById("p3").innerHTML =
date3;
        document.getElementById("p4").innerHTML =
date4;
        document.getElementById("p5").innerHTML =
date5;
        document.getElementById("p6").innerHTML =
date6;
        document.getElementById("p7").innerHTML =
date7;
        document.getElementById("p8").innerHTML =
date8;
        document.getElementById("p9").innerHTML =
date9;
    </script>
</body>
</html>
```

Compare Dates in JavaScript

Use comparison operators to compare two date objects.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Date comparison</h1>
    <p id="p1"></p>
    <p id="p2"></p>

    <script>
        var date1 = new Date('4-1-2015');
        var date2 = new Date('4-2-2015');

        if (date1 > date2)
```

```
document.getElementById("p1").innerHTML =  
date1.toDateString() + ' is greater than ' +  
date2.toDateString();  
    else (date1 < date2 )
```

```
document.getElementById("p2").innerHTML    =  
date1.toDateString() + ' is less than ' +  
date2.toDateString();
```

```
    </script>  
</body>  
</html>
```

JavaScript Arrays: Create, Access, Add & Remove Elements

We have learned that a variable can hold only one value. We cannot assign multiple values to a single variable. JavaScript array is a special type of variable, which can store multiple values using a special syntax.

The following declares an array with five numeric values.

```
let numArr = [10, 20, 30, 40, 50];
```

In the above array, `numArr` is the name of an array variable. Multiple values are assigned to it by separating them using a comma inside square brackets as `[10, 20, 30, 40, 50]`. Thus, the `numArr` variable stores five numeric values. The `numArr` array is created using the literal syntax and it is the preferred way of creating arrays.

Another way of creating arrays is using the `Array()` constructor, as shown below.

```
let numArr = new Array(10, 20, 30, 40, 50);
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
  <h1>Demo: JavaScript Arrays</h1>
```

```
  <p id="p1"></p>
```

```
  <p id="p2"></p>
```

```
  <p id="p3"></p>
```

```
  <p id="p4"></p>
```

```
  <p id="p5"></p>
```

```
  <script>
```

```
    let stringArray = ["one", "two", "three"];
```

```
    let numericArray = [1, 2, 3, 4];
```

```
    let decimalArray = [1.1, 1.2, 1.3];
```

```
    let booleanArray = [true, false, false,  
true];
```

```
    let data = [1, "Steve", "DC", true,  
255000, 5.5];
```

```
    document.getElementById("p1").innerHTML =  
stringArray;
```

```
    document.getElementById("p2").innerHTML =  
numericArray;
```

```
    document.getElementById("p3").innerHTML =  
decimalArray;
```

```
        document.getElementById("p4").innerHTML =
booleanArray;
        document.getElementById("p5").innerHTML =
data;
    </script>
</body>
</html>
```

It is not required to store the same type of values in an array. It can store values of different types as well.

```
let data = [1, "Steve", "DC", true, 255000, 5.5];
```

Get Size of an Array

Use the `length` property to get the total number of elements in an array. It changes as and when you add or remove elements from the array.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Getting Array Size</h1>
    <p id="p1"></p>
    <p id="p2"></p>

    <script>
        let cities = ["Mumbai", "New York", "Paris",
"Sydney"];
        document.getElementById("p1").innerHTML =
cities.length;

        cities[4] = "Delhi";
```

```
        document.getElementById("p2").innerHTML =
cities.length;
    </script>
</body>
</html>
```

Accessing Array Elements

Array elements (values) can be accessed using an index. Specify an index in square brackets with the array name to access the element at a particular index like `arrayName[index]`. Note that the index of an array starts from zero.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Accessing Array Elements in
JavaScript</h1>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <p id="p4"></p>
    <p id="p5"></p>
    <p id="p6"></p>
    <p id="p7"></p>
    <p id="p8"></p>
    <p id="p9"></p>
    <p id="p10"></p>

    <script>
        let numArr = [10, 20, 30, 40, 50];
        document.getElementById("p1").innerHTML =
numArr[0];
```

```
        document.getElementById("p2").innerHTML =
numArr[1];
        document.getElementById("p3").innerHTML =
numArr[2];
        document.getElementById("p4").innerHTML =
numArr[3];
        document.getElementById("p5").innerHTML =
numArr[4];

        let cities = ["Mumbai", "New York", "Paris",
"Sydney"];

        document.getElementById("p6").innerHTML =
cities[0];
        document.getElementById("p7").innerHTML =
cities[1];
        document.getElementById("p8").innerHTML =
cities[2];
        document.getElementById("p9").innerHTML =
cities[3];

        //accessing element from nonexistence index
        document.getElementById("p10").innerHTML =
cities[4]; // undefined
    </script>
</body>
</html>
```

Example: Accessing Array using at()

```
let numArr = [10, 20, 30, 40, 50];
console.log(numArr.at(0)); // 10
console.log(numArr.at(1)); // 20
console.log(numArr.at(2)); // 30
```

```
console.log(numArr.at(3)); // 40
console.log(numArr.at(4)); // 50
console.log(numArr.at(5)); // undefined

//passing negative index
console.log(numArr.at(-1)); // 50
console.log(numArr.at(-2)); // 40
console.log(numArr.at(-3)); // 30
console.log(numArr.at(-4)); // 20
console.log(numArr.at(-5)); // 10
console.log(numArr.at(-6)); // undefined
```

Update Array Elements

You can update the elements of an array at a particular index using `arrayName[index] = new_value` syntax.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Update Array Elements in
JavaScript</h1>
    <p id="p1"></p>
    <p id="p2"></p>

    <script>
        let cities = ["Mumbai", "New York", "Paris",
"Sydney"];
        document.getElementById("p1").innerHTML =
cities;

        cities[0] = "Delhi";
        cities[1] = "Los angeles";
```

```
        document.getElementById("p2").innerHTML =
cities;
    </script>
</body>
</html>
```

Adding New Elements

You can add new elements using `arrayName[index] = new_value` syntax. Just make sure that the index is greater than the last index. If you specify an existing index then it will update the value.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Add Array Elements in JavaScript</
h1>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <p id="p4"></p>

    <script>
        let cities = ["Mumbai", "New York", "Paris",
"Sydney"];
        document.getElementById("p1").innerHTML =
cities;

        cities[4] = "Delhi"; //add new element at last
        document.getElementById("p2").innerHTML =
cities;
```



```
        cities[cities.length] = "London";//use length
property to specify last index
        document.getElementById("p3").innerHTML =
cities;

        cities[9] = "Pune";
        document.getElementById("p4").innerHTML =
cities;
    </script>
</body>
</html>
```

Example: Add Element At Last using push()

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Add Array Elements in JavaScript</h1>
    <p id="p1"></p>
    <p id="p2"></p>

    <script>
        let cities = ["Mumbai", "New York", "Paris",
"Sydney"];
        document.getElementById("p1").innerHTML =
cities;

        cities.push("Delhi");
        document.getElementById("p2").innerHTML =
cities;
    </script>
</body>
</html>
```

Use the `unshift()` method to add an element to the beginning of an array.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Add Array Elements in JavaScript</h1>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>

    <script>
        let cities = ["Mumbai", "New York", "Paris", "Sydney"];
        document.getElementById("p1").innerHTML = cities;

        cities.unshift("Delhi"); //adds new element at the beginning
        document.getElementById("p2").innerHTML = cities;

        cities.unshift("London", "Pune"); //adds new element at the beginning
        document.getElementById("p3").innerHTML = cities;
    </script>
</body>
</html>
```

Remove Array Elements

The `pop()` method returns the last element and removes it from the array.

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Remove Last Element from an Array in
JavaScript</h1>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>

    <script>
        let cities = ["Mumbai", "New York", "Paris",
"Sydney"];
        document.getElementById("p1").innerHTML =
cities;

        let removedCity = cities.pop();
        document.getElementById("p2").innerHTML =
cities;
        document.getElementById("p3").innerHTML =
removedCity;
    </script>
</body>
</html>
```

The `shift()` method returns the first element and removes it from the array.

```
<!DOCTYPE html>
<html>
<body>
```

```
<h1>Demo: Remove First Element from Array in
JavaScript</h1>
<p id="p1"></p>
<p id="p2"></p>
<p id="p3"></p>

<script>
    let cities = ["Mumbai", "New York", "Paris",
"Sydney"];
    document.getElementById("p1").innerHTML =
cities;

    let removedCity = cities.shift();
    document.getElementById("p2").innerHTML =
cities;
    document.getElementById("p3").innerHTML =
removedCity;
</script>
</body>
</html>
```

Array Methods Reference

The following table lists all the Array methods.

Method	Description
concat()	Returns new array by combining values of an array that is specified as parameter with existing array values.
every()	Returns true or false if every element in the specified array satisfies a condition specified in the callback function. Returns false even if single element does not satisfy the condition.

filter()	Returns a new array with all the elements that satisfy a condition specified in the callback function.
forEach())	Executes a callback function for each elements of an array.
indexOf())	Returns the index of the first occurrence of the specified element in the array, or -1 if it is not found.
join()	Returns string of all the elements separated by the specified separator
lastIndex Of()	Returns the index of the last occurrence of the specified element in the array, or -1 if it is not found.
map()	Creates a new array with the results of calling a provided function on every element in this array.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements at the end of an array and returns the new length of the array.
reduce()	Pass two elements simultaneously in the callback function (till it reaches the last element) and returns a single value.
reduceRight() ght()	Pass two elements simultaneously in the callback function from right-to-left (till it reaches the last element) and returns a single value.
reverse())	Reverses the elements of an array. Element at last index will be first and element at 0 index will be last.
shift()	Removes the first element from an array and returns that element.
slice()	Returns a new array with specified start to end elements.

<code>some()</code>	Returns true if at least one element in this array satisfies the condition in the callback function.
<code>sort()</code>	Sorts the elements of an array.
<code>splice()</code>	Adds and/or removes elements from an array.
<code>toString() ()</code>	Returns a string representing the array and its elements.
<code>unshift()</code>	Adds one or more elements to the front of an array and returns the new length of the array.

JavaScript Array concat()

```
let primeNumbers = [2, 3, 5, 7]
let evenNumbers = [2, 4, 6, 8]
```

```
// join two arrays
let joinedArrays = primeNumbers.concat(evenNumbers);
console.log(joinedArrays);
```

```
/* Output:
```

```
[
  2, 3, 5, 7,
  2, 4, 6, 8
]
*/
```

Another Example

```
var languages1 = ["JavaScript", "Python", "Java"];
var languages2 = ["C", "C++"];
```

```
// concatenating two arrays
var new_arr = languages1.concat(languages2);
console.log(new_arr); // [ 'JavaScript', 'Python', 'Java', 'C', 'C++' ]
```

```
// concatenating a value and array
var new_arr1 = languages2.concat("Lua", languages1);
console.log(new_arr1); // [ 'C', 'C++', 'Lua', 'JavaScript', 'Python', 'Java' ]
```

Javascript Array entries()

The `entries()` method returns a new Array Iterator object containing key/value pairs for each array index

```
// defining an array named alphabets
const alphabets = ["A", "B", "C"];
```

```
// array iterator object that contains
// key-value pairs for each index in the array
let iterator = alphabets.entries();
```

```
// iterating through key-value pairs in the array
for (let entry of iterator) {
  console.log(entry);
}
```

```
// Output:
// [ 0, 'A' ]
// [ 1, 'B' ]
// [ 2, 'C' ]
```

The `fill()` method returns an array by filling all elements with a specified value.

```
// defining an array
var fruits = ['Apple', 'Banana', 'Grape'];
```

```
// filling every element of the array with 'Cherry'
```

```
fruits.fill("Cherry");
```

```
console.log(fruits);
```

```
// Output:
```

```
// [ 'Cherry', 'Cherry', 'Cherry' ]
```

Example 1: Using fill() Method

```
var prices = [651, 41, 4, 3, 6];
```

```
// filling every element of the array with '5'
```

```
new_prices = prices.fill(5);
```

```
console.log(prices);
```

```
console.log(new_prices);
```

```
Output:
```

```
[ 5, 5, 5, 5, 5 ]  
[ 5, 5, 5, 5, 5 ]
```

Example 2: fill() Method with Three Arguments

```
// array definition
```

```
var language = ["JavaScript", "Python", "C", "C++"];
```

```
// replacing element of array from index 1 to 3 by 'JavaScript'
```

```
language.fill("JavaScript", 1, 3);
```

```
// printing the original array
```

```
console.log(language);
```

```
Output:
```

```
[ 'JavaScript', 'JavaScript', 'JavaScript', 'C++' ]
```

Example 3: fill() Method with Invalid Indexes


```
var rank = [8, 9, 3, 7];

// on passing negative index, counting starts from back
rank.fill(15, -2);

// prints the modified 'rank' array
console.log(rank); // [ 8, 9, 15, 15 ]

// passing invalid index result in no change
rank.fill(15, 7, 8);

console.log(rank); // [ 8, 9, 15, 15 ]

// passing invalid indexes
rank.fill(15, NaN, NaN);

console.log(rank); // [ 8, 9, 15, 15 ]
```

Output:

```
[ 8, 9, 15, 15 ]
[ 8, 9, 15, 15 ]
[ 8, 9, 15, 15 ]
```

The `filter()` method returns a new array with all elements that pass the test defined by the given function.

```
let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

```
// function to check even numbers
function checkEven(number) {
  if (number % 2 == 0)
    return true;
  else
    return false;
}
```

```
// create a new array by filter even numbers from the
numbers array
let evenNumbers = numbers.filter(checkEven);
console.log(evenNumbers);
```

```
// Output: [ 2, 4, 6, 8, 10 ]
```

The `filter()` method returns a new array with all elements that pass the test defined by the given function.

```
let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

```
// function to check even numbers
function checkEven(number) {
  if (number % 2 == 0)
    return true;
  else
    return false;
}
```

```
// create a new array by filter even numbers from the
numbers array
let evenNumbers = numbers.filter(checkEven);
console.log(evenNumbers);
```

```
// Output: [ 2, 4, 6, 8, 10 ]
```

filter() Parameters

The `filter()` method takes in:

- `callback` - The test function to execute on each array element; returns `true` if element passes the test, else `false`. It takes in:

- element - The current element being passed from the array.
- thisArg (optional) - The value to use as this when executing callback. By default, it is undefined.

filter() Return Value

- Returns a new array with only the elements that passed the test.

Example 1: Filtering out values from Array

```
const prices = [1800, 2000, null, 3000, 5000, "Thousand", 500, 8000]
```

```
function checkPrice(element) {  
  return element > 2000 && !Number.isNaN(element);  
}
```

```
let filteredPrices = prices.filter(checkPrice);  
console.log(filteredPrices); // [ 3000, 5000, 8000 ]
```

```
// using arrow function  
let newPrices = prices.filter((price) => (price > 2000 && !  
Number.isNaN(price)));  
console.log(newPrices); // [ 3000, 5000, 8000 ]
```

Output

```
[ 3000, 5000, 8000 ]  
[ 3000, 5000, 8000 ]
```

Here, all the numbers **less than or equal to 2000**, and all the **non-numeric** values are filtered out.

Example 2: Searching in Array

```
const languages = ["JavaScript", "Python", "Ruby", "C", "C++",  
"Swift", "PHP", "Java"];
```

```
function searchFor(arr, query) {  
  function condition(element) {  
    return element.toLowerCase().indexOf(query.toLowerCase()) !  
    == -1;  
  }  
  return arr.filter(condition);  
}
```

```
let newArr = searchFor(languages, "ja");  
console.log(newArr); // [ 'JavaScript', 'Java' ]
```

```
// using arrow function  
const searchArr = (arr, query) => arr.filter(element =>  
element.toLowerCase().indexOf(query.toLowerCase()) !== -1);
```

```
let newLanguages = searchArr(languages, "p");  
console.log(newLanguages); // [ 'JavaScript', 'Python', 'PHP' ]
```

Output

```
[ 'JavaScript', 'Java' ]  
[ 'JavaScript', 'Python', 'PHP' ]
```

The `find()` method returns the value of the first array element that satisfies the provided test function.

```
let numbers = [1, 3, 4, 9, 8];
```

```
// function to check even number  
function isEven(element) {  
  return element % 2 == 0;  
}
```

```
// get the first even number  
let evenNumber = numbers.find(isEven);  
console.log(evenNumber);
```

```
// Output: 4
```

Example 1: Using find() method

```
function isEven(element) {  
  return element % 2 == 0;  
}
```

```
let randomArray = [1, 45, 8, 98, 7];
```

```
let firstEven = randomArray.find(isEven);  
console.log(firstEven); // 8
```

```
// using arrow operator
```

```
let firstOdd = randomArray.find((element) => element % 2 == 1);  
console.log(firstOdd); // 1
```

Output:

8

1

Example 2: find() with Object elements

```
const team = [  
  { name: "Bill", age: 10 },  
  { name: "Linus", age: 15 },  
  { name: "Alan", age: 20 },  
  { name: "Steve", age: 34 },  
];
```

```
function isAdult(member) {  
  return member.age >= 18;  
}
```

```
console.log(team.find(isAdult)); // { name: 'Alan', age: 20 }
```

```
// using arrow function and deconstructing
```

```
let adultMember = team.find(({ age }) => age >= 18);
```

```
console.log(adultMember); // { name: 'Alan', age: 20 }
```

Output:

```
{ name: 'Alan', age: 20 }  
{ name: 'Alan', age: 20 }
```

The `findIndex()` method returns the index of the first array element that satisfies the provided test function or else returns -1.

```
// function that returns odd number  
function isOdd(element) {  
  return element % 2 !== 0;  
}
```

```
// defining an array of integers  
let numbers = [2, 8, 1, 3, 4];
```

```
// returns the index of the first odd number in the array  
let firstOdd = numbers.findIndex(isOdd);
```

```
console.log(firstOdd);
```

```
// Output: 2
```

Example 1: Using `findIndex()` method

```
// function that returns even number  
function isEven(element) {  
  return element % 2 == 0;  
}
```

```
// defining an array of integers  
let numbers = [1, 45, 8, 98, 7];
```

```
// returns the index of the first even number in the array
```

```
let firstEven = numbers.findIndex(isEven);
```

```
console.log(firstEven); // 2
```

Example 2: `findIndex()` with Arrow Function

```
// defining an array
```

```
let days = ["Sunday", "Wednesday", "Tuesday", "Friday"];
```

```
// returns the first index of 'Wednesday' in the array
```

```
let index = days.findIndex((day) => day === "Wednesday");
```

```
console.log(index); // 1
```

Example 3: `findIndex()` with Object Elements

```
// defining an object
```

```
const team = [  
  { name: "Bill", age: 10 },  
  { name: "Linus", age: 15 },  
  { name: "Alan", age: 20 },  
  { name: "Steve", age: 34 },  
];
```

```
// function that returns age greater than or equal to 18
```

```
function isAdult(member) {  
  return member.age >= 18;  
}
```

```
// returns the index of the first element which is
```

```
// greater than or equal to 18
```

```
console.log(team.findIndex(isAdult)); // 2
```

The `forEach()` method executes a provided function for each array element.

Example

```
let numbers = [1, 3, 4, 9, 8];
```

```
// function to compute square of each number
function computeSquare(element) {
  console.log(element * element);
}
```

```
// compute square root of each element
numbers.forEach(computeSquare);
```

```
/* Output:
```

```
1
```

```
9
```

```
16
```

```
81
```

```
64
```

```
*/
```

Example 1: Printing Contents of Array

```
function printElements(element, index) {
  console.log('Array Element ' + index + ': ' + element);
}
```

```
const prices = [1800, 2000, 3000, , 5000, 500, 8000];
```

```
// forEach does not execute for elements without values
// in this case, it skips the third element as it is empty
prices.forEach(printElements);
```

The `includes()` method checks if an array contains a specified element or not.

Example

```
// defining an array
```

```
let languages = ["JavaScript", "Java", "C"];
```

```
// checking whether the array contains 'Java'
```



```
let check = languages.includes("Java");
```

```
console.log(check);
```

```
// Output: true
```

Example 1: Using includes() method

```
let languages = ["JavaScript", "Java", "C", "C++"];
```

```
// checking whether the array contains 'C'
```

```
let check1 = languages.includes("C");
```

```
console.log(check1); // true
```

```
// checking whether the array contains 'Ruby'
```

```
let check2 = languages.includes("Ruby");
```

```
console.log(check2); // false
```

Example 2: includes() for Case-Sensitive Search

The `includes()` method is case sensitive. For example:

```
let languages = ["JavaScript", "Java", "C", "Python"];
```

```
// checking whether the array contains 'Python'
```

```
let check1 = languages.includes("Python");
```

```
console.log(check1); // true
```

```
// checking whether the array contains 'python'
```

```
let check2 = languages.includes("python");
```

```
console.log(check2); // false
```

Example 3: includes() with two Parameters

```
let languages = ["JavaScript", "Java", "C", "Python"];
```

```
// second argument specifies position to start the search
let check1 = languages.includes("Java", 2);
```

```
console.log(check1); // false
```

```
// the search starts from third last element
let check2 = languages.includes("Java", -3);
```

```
console.log(check2); // true
```

Example 1: Using indexOf() method

```
var priceList = [10, 8, 2, 31, 10, 1, 65];
```

```
// indexOf() returns the first occurrence
var index1 = priceList.indexOf(31);
console.log(index1); // 3
```

```
var index2 = priceList.indexOf(10);
console.log(index2); // 0
```

```
// second argument specifies the search's start index
var index3 = priceList.indexOf(10, 1);
console.log(index3); // 4
```

```
// indexOf returns -1 if not found
var index4 = priceList.indexOf(69.5);
console.log(index4); // -1
```

Example 2: Finding All the Occurrences of an Element

```
function findAllIndex(array, element) {
  indices = [];
  var currentIndex = array.indexOf(element);
  while (currentIndex !== -1) {
    indices.push(currentIndex);
    currentIndex = array.indexOf(element, currentIndex + 1);
  }
}
```

```
    return indices;
}
```

```
var priceList = [10, 8, 2, 31, 10, 1, 65, 10];
```

```
var occurrence1 = findAllIndex(priceList, 10);
console.log(occurrence1); // [ 0, 4, 7 ]
```

```
var occurrence2 = findAllIndex(priceList, 8);
console.log(occurrence2); // [ 1 ]
```

```
var occurrence3 = findAllIndex(priceList, 9);
console.log(occurrence3); // []
```

Example 3: Finding If Element exists else Adding the Element

```
function checkOrAdd(array, element) {
  if (array.indexOf(element) === -1) {
    array.push(element);
    console.log("Element not Found! Updated the array.");
  } else {
    console.log(element + " is already in the array.");
  }
}
```

```
var parts = ["Monitor", "Keyboard", "Mouse", "Speaker"];
```

```
checkOrAdd(parts, "CPU"); // Element not Found! Updated the
array.
console.log(parts); // [ 'Monitor', 'Keyboard', 'Mouse', 'Speaker',
'CPU' ]
```

```
checkOrAdd(parts, "Mouse"); // Mouse is already in the array.
```

Example: Using join() method

```
var info = ["Terence", 28, "Kathmandu"];
```

```
var info_str = info.join(" | ");
```

```
// join() does not change the original array
console.log(info); // [ 'Terence', 28, 'Kathmandu' ]
```

```
// join() returns the string by joining with separator
console.log(info_str); // Terence | 28 | Kathmandu
```

```
// empty argument = no separator
var collection = [3, ".", 1, 4, 1, 5, 9, 2];
console.log(collection.join("")); // 3.141592
```

```
var random = [44, "abc", undefined];
console.log(random.join(" and ")); // 44 and abc and
```

Example: Using split()

```
console.log("ABCDEF".split("")); // [ 'A', 'B', 'C', 'D', 'E', 'F' ]
```

```
const text = "Java is awesome. Java is fun.";
```

```
let pattern = ".";
let newText = text.split(pattern);
console.log(newText); // [ 'Java is awesome', ' Java is fun', " ]
```

```
let pattern1 = ".";
// only split string to maximum to parts
let newText1 = text.split(pattern1, 2);
console.log(newText1); // [ 'Java is awesome', ' Java is fun' ]
```

```
const text2 = "JavaScript ; Python ;C;C++";
let pattern2 = ";";
let newText2 = text2.split(pattern2);
console.log(newText2); // [ 'JavaScript ', ' Python ', 'C', 'C++' ]
```

```
// using RegEx
let pattern3 = /\s*(?:;|$)\s*/;
let newText3 = text2.split(pattern3);
console.log(newText3); // [ 'JavaScript', 'Python', 'C', 'C++' ]
```

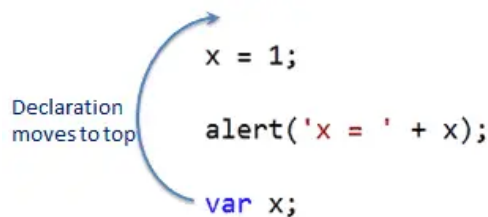
JavaScript Hoisting

Hoisting in JavaScript is a behavior in which a function or a variable can be used before declaration. For example,

```
// using test before declaring  
console.log(test); // undefined  
var test;
```

```
// using test before declaring  
var test;  
console.log(test); // undefined
```

Since the variable `test` is only declared and has no value, `undefined` value is assigned to it.



Variable Hoisting

In terms of variables and constants, keyword `var` is hoisted and `let` and `const` does not allow hoisting.

For example,

```
// program to display value  
a = 5;  
console.log(a);  
var a; // 5
```

However in JavaScript, initializations are not hoisted. For example,

```
// program to display value
```

```
console.log(a);
```

```
var a = 5;
```

Output:

undefined

Also, when the variable is used inside the function, the variable is hoisted only to the top of the function. For example,

```
// program to display value
```

```
var a = 4;
```

```
function greet() {  
    b = 'hello';  
    console.log(b); // hello  
    var b;  
}
```

```
greet(); // hello
```

```
console.log(b);
```

Output:

hello

Uncaught ReferenceError: b is not defined

In the above example, variable `b` is hoisted to the top of the function `greet` and becomes a local variable. Hence `b` is only accessible inside the function. `b` does not become a global variable.

If a variable is used with the `let` keyword, that variable is not hoisted. For example,

```
// program to display value
```

```
a = 5;  
console.log(a);  
let a; // error
```

Output

```
Uncaught ReferenceError: Cannot access 'a' before initialization
```

Function Hoisting

A function can be called before declaring it. For example,

```
// program to print the text
```

```
greet();  
  
function greet() {  
    console.log('Hi, there.');
```

```
}
```

Output:

```
Hi, there
```

However, when a function is used as an **expression**, an error occurs because only declarations are hoisted. For example;

```
// program to print the text  
greet();
```

```
let greet = function() {
```

```
    console.log('Hi, there.');
```

Output:

Uncaught ReferenceError: greet is not defined

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: Hoisting</h1>

    <script>
        alert(UseMe);

        var UseMe;

        function UseMe()
        {
            alert("UseMe function called");
        }

    </script>
</body>
</html>
```

JavaScript Recursion

Recursion is a process of calling itself. A function that calls itself is called a recursive function.

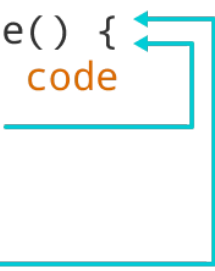
The syntax for recursive function is:

```
function recurse() {
    // function code
    recurse();
    // function code
}
```

```
recurse();
```



```
function recurse() {  
    // function code  
    recurse();  
}  
  
recurse();
```



function
call

Example 1:

Print

Numbers

```
// program to count down numbers to 1  
function countDown(number) {
```

```
    // display the number  
    console.log(number);
```

```
    // decrease the number value  
    const newNumber = number - 1;
```

```
    // base case  
    if (newNumber > 0) {  
        countDown(newNumber);  
    }  
}
```

```
countDown(4);
```

Example 2: Find Factorial

```
// program to find the factorial of a number  
function factorial(x) {
```

```
    // if number is 0  
    if (x === 0) {  
        return 1;  
    }
```

```
    // if number is positive  
    else {
```

```
    return x * factorial(x - 1);  
  }  
}
```

```
const num = 3;
```

```
// calling factorial() if num is non-negative  
if (num > 0) {  
  let result = factorial(num);  
  console.log(`The factorial of ${num} is ${result}`);  
}
```

JavaScript Object Declaration

The syntax to declare an object is:

```
const object_name = {  
  key1: value1,  
  key2: value2  
}
```

Here, an object `object_name` is defined. Each member of an object is a **key: value** pair separated by commas and enclosed in curly braces `{}`.

```
// object creation  
const person = {  
  name: 'John',  
  age: 20  
};  
console.log(typeof person); // object
```

You can also define an object in a single line.

```
const person = { name: 'John', age: 20 };
```

Accessing Object Properties

You can access the **value** of a property by using its **key**.

1. Using dot Notation

Here's the syntax of the dot notation.

```
const person = {  
  name: 'John',  
  age: 20,  
};
```

```
// accessing property  
console.log(person.name); // John
```

2. Using bracket Notation

Here is the syntax of the bracket notation.

```
const person = {  
  name: 'John',  
  age: 20,  
};
```

```
// accessing property  
console.log(person["name"]); // John
```

JavaScript Nested Objects

An object can also contain another object. For example,

```
// nested object  
const student = {  
  name: 'John',  
  age: 20,  
  marks: {
```

```
    science: 70,  
    math: 75  
  }  
}
```

```
// accessing property of student object  
console.log(student.marks); // {science: 70, math: 75}
```

```
// accessing property of marks object  
console.log(student.marks.science); // 70
```

JavaScript Object Methods

In JavaScript, an object can also contain a function. For example,

```
const person = {  
  name: 'Sam',  
  age: 30,  
  // using function as a value  
  greet: function() { console.log('hello') }  
}
```

```
person.greet(); // hello
```

JavaScript Methods and this Keyword

In JavaScript, objects can also contain functions. For example,

```
// object containing method  
const person = {
```

```
  name: 'John',  
  greet: function() { console.log('hello'); }  
};
```

Adding a Method to a JavaScript Object

You can also add a method in an object. For example,

```
// creating an object  
let student = { };
```

```
// adding a property  
student.name = 'John';
```

```
// adding a method  
student.greet = function() {  
  console.log('hello');  
}
```

```
// accessing a method  
student.greet(); // hello
```

JavaScript this Keyword

To access a property of an object from within a method of the same object, you need to use the `this` keyword. Let's consider an example.

```
const person = {  
  name: 'John',  
  age: 30,
```

```
  // accessing name property by using this.name  
  greet: function() { console.log('The name is' + ' ' + this.name); }
```

```
};
```

```
person.greet();
```

Another Example:

```
const person = {  
  name: 'John',  
  age: 30,  
  greet: function() {  
    let surname = 'Doe';  
    console.log('The name is' + ' ' + this.name + ' ' + surname); }  
};
```

```
person.greet();
```

JavaScript Constructor Function

In JavaScript, a constructor function is used to create objects. For example,

```
// constructor function  
function Person () {  
  this.name = 'John',  
  this.age = 23  
}
```

```
// create an object  
const person = new Person();
```

Create Multiple Objects with Constructor Function

In JavaScript, you can create multiple objects from a constructor function. For example,

```
// constructor function
function Person () {
  this.name = 'John',
  this.age = 23,

  this.greet = function () {
    console.log('hello');
  }
}

// create objects
const person1 = new Person();
const person2 = new Person();

// access properties
console.log(person1.name); // John
console.log(person2.name); // John
```

JavaScript Constructor Function Parameters

You can also create a constructor function with parameters. For example,

```
// constructor function
function Person (person_name, person_age,
person_gender) {

  // assigning parameter values to the calling object
  this.name = person_name,
```

```
this.age = person_age,  
this.gender = person_gender,
```

```
    this.greet = function () {  
        return ('Hi' + ' ' + this.name);  
    }  
}
```

```
// creating objects
```

```
const person1 = new Person('John', 23, 'male');  
const person2 = new Person('Sam', 25, 'female');
```

```
// accessing properties
```

```
console.log(person1.name); // "John"  
console.log(person2.name); // "Sam"
```

However, if an object is created with an object literal, and if a variable is defined with that object value, any changes in variable value will change the original object. For example,

```
// using object lateral
```

```
let person = {  
    name: 'Sam'  
}
```

```
console.log(person.name); // Sam
```

```
let student = person;
```

```
// changes the property of an object  
student.name = 'John';
```



```
// changes the origins object property  
console.log(person.name); // John
```

Adding Properties And Methods in an Object

You can add properties or methods in an object like this:

```
// constructor function  
function Person () {  
    this.name = 'John',  
    this.age = 23  
}
```

```
// creating objects  
let person1 = new Person();  
let person2 = new Person();
```

```
// adding property to person1 object  
person1.gender = 'male';
```

```
// adding method to person1 object  
person1.greet = function () {  
    console.log('hello');  
}
```

```
person1.greet(); // hello
```

```
// Error code  
// person2 doesn't have greet() method  
person2.greet();
```

Accessor Property

In JavaScript, accessor properties are methods that get or set the value of an object. For that, we use these two keywords:

- `get` - to define a getter method to get the property value
- `set` - to define a setter method to set the property value

JavaScript Getter

In JavaScript, getter methods are used to access the properties of an object. For example,

```
const student = {
```

```
  // data property  
  firstName: 'Monica',
```

```
  // accessor property(getter)  
  get getName() {  
    return this.firstName;  
  }  
};
```

```
// accessing data property  
console.log(student.firstName); // Monica
```

```
// accessing getter methods  
console.log(student.getName); // Monica
```

```
// trying to access as a method  
console.log(student.getName()); // error
```

JavaScript Setter

In JavaScript, setter methods are used to change the values of an object. For example,

```
const student = {  
  firstName: 'Monica',
```

```
  //accessor property(setter)  
  set changeName(newName) {  
    this.firstName = newName;  
  }  
};
```

```
console.log(student.firstName); // Monica
```

```
// change(set) object property using a setter  
student.changeName = 'Sarah';
```

```
console.log(student.firstName); // Sarah
```

Types of Errors

In programming, there can be two types of errors in the code:

Syntax Error: Error in the syntax. For example, if you write `consol.log('your result');`, the above program throws a syntax error. The spelling of `console` is a mistake in the above code.

Runtime Error: This type of error occurs during the execution of the program. For example, calling an invalid function or a variable.

These errors that occur during runtime are called **exceptions**. Now, let's see how you can handle these exceptions.

JavaScript try...catch Statement

The try...catch statement is used to handle the exceptions. Its syntax is:

Example 1: Display Undeclared Variable

```
// program to show try...catch in a program
```

```
const numerator= 100, denominator = 'a';
```

```
try {  
    console.log(numerator/denominator);
```

```
    // forgot to define variable a  
    console.log(a);
```

```
}  
catch(error) {  
    console.log('An error caught');  
    console.log('Error message: ' + error);  
}
```

Output:

```
NaN  
An error caught  
Error message: ReferenceError: a is not defined
```

Example 2: try...catch...finally Example

```
const numerator= 100, denominator = 'a';
```

```
try {
```

```

    console.log(numerator/denominator);
    console.log(a);
}
catch(error) {
    console.log('An error caught');
    console.log('Error message: ' + error);
}
finally {
    console.log('Finally will execute every time');
}

```

Output:

NaN

An error caught

Error message: ReferenceError: a is not defined

Finally will execute every time

Example 1: try...catch...throw Example

```

const number = 40;
try {
    if(number > 50) {
        console.log('Success');
    }
    else {

        // user-defined throw statement
        throw new Error('The number is low');
    }

    // if throw executes, the below code does not execute
    console.log('hello');
}
catch(error) {
    console.log('An error caught');
    console.log('Error message: ' + error);
}

```

Output:

An error caught

Error message: Error: The number is low

Rethrow an Exception

You can also use `throw` statement inside the `catch` block to rethrow an exception. For example,

```
const number = 5;
try {
  // user-defined throw statement
  throw new Error('This is the throw');
}
catch(error) {
  console.log('An error caught');
  if( number + 8 > 10) {

    // statements to handle exceptions
    console.log('Error message: ' + error);
    console.log('Error resolved');
  }
  else {
    // cannot handle the exception
    // rethrow the exception
    throw new Error('The value is low');
  }
}
```

JavaScript Modules

As our program grows bigger, it may contain many lines of code. Instead of putting everything in a single file, you can use modules to separate codes in separate files as per their functionality. This makes our code organized and easier to maintain.

Module is a file that contains code to perform a specific task. A module may contain variables, functions, classes etc. Let's see an example,

Suppose, a file named **greet.js** contains the following code:

```
// exporting a function
export function greetPerson(name) {
  return `Hello ${name}`;
}
```

Now, to use the code of **greet.js** in another file, you can use the following code:

```
// importing greetPerson from greet.js file
import { greetPerson } from './greet.js';
```

```
// using greetPerson() defined in greet.js
let displayName = greetPerson('Jack');
```

```
console.log(displayName); // Hello Jack
```

Export Multiple Objects

It is also possible to export multiple objects from a module. For example,

In the file **module.js**

```
// exporting the variable
export const name = 'JavaScript Program';
```

```
// exporting the function
```

```
export function sum(x, y) {  
  return x + y;  
}
```

In main file,

```
import { name, sum } from './module.js';
```

```
console.log(name);  
let add = sum(4, 9);  
console.log(add); // 13
```

Javascript setTimeout()

The setTimeout() method executes a block of code after the specified time. The method executes the code only once.

The commonly used syntax of JavaScript setTimeout is:

```
setTimeout(function, milliseconds);
```

Its parameters are:

- **function** - a function containing a block of code
- **milliseconds** - the time after which the function is executed

The setTimeout() method returns an **intervalID**, which is a positive integer.

Example 1: Display a Text Once After 3 Second

// program to display a text using setTimeout method

```
function greet() {  
  console.log('Hello world');  
}
```

```
setTimeout(greet, 3000);  
console.log('This message is shown first');
```

Output:

```
This message is shown first  
Hello world
```

In the above program, the setTimeout() method calls the greet() function after **3000** milliseconds (**3** second).

Hence, the program displays the text Hello world only once after **3** seconds.

The setTimeout() method returns the interval id. For example,
// program to display a text using setTimeout method
function greet() {
 console.log('Hello world');
}

```
let intervalId = setTimeout(greet, 3000);  
console.log('Id: ' + intervalId);
```

Output:

```
Id: 3  
Hello world
```

Example 2: Display Time Every 3 Second

```
// program to display time every 3 seconds  
function showTime() {
```

```
  // return new date and time  
  let dateTime = new Date();
```

```
  // returns the current local time  
  let time = dateTime.toLocaleTimeString();
```

```
  console.log(time)
```

```
  // display the time after 3 seconds  
  setTimeout(showTime, 3000);  
}
```

```
// calling the function  
showTime();
```

Output:

```
5:45:39 PM  
5:45:43 PM  
5:45:47 PM  
5:45:50 PM  
.....
```

JavaScript clearTimeout()

As you have seen in the above example, the program executes a block of code after the specified time interval. If you want to stop this function call, you can use the `clearTimeout()` method.

The syntax of `clearTimeout()` method is:

Example 3: Use clearTimeout() Method

```
// program to stop the setTimeout() method
```

```
let count = 0;
```

```
// function creation
```

```
function increaseCount(){
```

```
    // increasing the count by 1
```

```
    count += 1;
```

```
    console.log(count)
```

```
}
```

```
let id = setTimeout(increaseCount, 3000);
```

```
// clearTimeout
```

```
clearTimeout(id);
```

```
console.log('setTimeout is stopped.');
```

Output:

```
setTimeout is stopped.
```

In the above program, the `setTimeout()` method is used to increase the value of `count` after 3 seconds. However, the `clearTimeout()` method stops the function call of the `setTimeout()` method. Hence, the `count` value is not increased.

JavaScript ES6

JavaScript **ES6** (also known as **ECMAScript 2015** or **ECMAScript 6**) is the newer version of JavaScript that was introduced in 2015.

[ECMAScript](#) is the standard that JavaScript programming language uses. ECMAScript provides the specification on how JavaScript programming language should work.

JavaScript let Vs var

Here's the overview of the differences between `let` and `var`.

let	var
let is block-scoped.	var is function scoped.
let does not allow to redeclare variables.	var allows to redeclare variables.
Hoisting does not occur in let.	Hoisting occurs in var.

JavaScript Arrow Function

Arrow function is one of the features introduced in the ES6 version of JavaScript. It allows you to create functions in a cleaner way compared to regular functions. For example,

```
// function expression
let x = function(x, y) {
  return x * y;
}
```

can be written as

```
// using arrow functions  
let x = (x, y) => x * y;
```

Arrow Function Syntax

The syntax of the arrow function is:

```
let myFunction = (arg1, arg2, ...argN) => {  
  statement(s)  
}
```

Here,

- myFunction is the name of the function
- arg1, arg2, ...argN are the function arguments
- statement(s) is the function body

Example 1: Arrow Function with No Argument

If a function doesn't take any argument, then you should use empty parentheses. For example,

```
let greet = () => console.log('Hello');  
greet(); // Hello
```

Example 2: Arrow Function with One Argument

If a function has only one argument, you can omit the parentheses. For example,

```
let greet = x => console.log(x);  
greet('Hello'); // Hello
```

Example 3: Arrow Function as an Expression

You can also dynamically create a function and use it as an expression. For example,

```
let age = 5;
```

```
let welcome = (age < 18) ?  
  () => console.log('Baby') :
```

```
() => console.log('Adult');
```

```
welcome(); // Baby
```

Example 4: Multiline Arrow Functions

If a function body has multiple statements, you need to put them inside curly brackets `{}`. For example,

```
let sum = (a, b) => {  
  let result = a + b;  
  return result;  
}
```

```
let result1 = sum(5,7);  
console.log(result1); // 12
```

this with Arrow Function

Inside a regular function, [this keyword](#) refers to the function where it is called.

However, this is not associated with arrow functions. Arrow function does not have its own this. So whenever you call this, it refers to its parent scope. For example,

Inside a regular function

```
function Person() {  
  this.name = 'Jack',  
  this.age = 25,  
  this.sayName = function () {
```

```
    // this is accessible  
    console.log(this.age);
```

```
  }  
  function innerFunc() {
```

```
    // this refers to the global object  
    console.log(this.age);  
    console.log(this);  
  }
```

```
        innerFunc();
    }
}
```

```
let x = new Person();
x.sayName();
```

Output:

```
25
undefined
Window {}
```

Here, `this.age` inside `this.sayName()` is accessible because `this.sayName()` is the method of an object.

However, `innerFunc()` is a normal function and `this.age` is not accessible because `this` refers to the global object (Window object in the browser). Hence, `this.age` inside the `innerFunc()` function gives `undefined`.

Inside an arrow function

```
function Person() {
    this.name = 'Jack',
    this.age = 25,
    this.sayName = function () {
```

```
        console.log(this.age);
        let innerFunc = () => {
            console.log(this.age);
        }
    }
}
```

```
        innerFunc();
    }
}
```

```
const x = new Person();
x.sayName();
```

Output:

25

25

Arguments Binding

Regular functions have arguments binding. That's why when you pass arguments to a regular function, you can access them using the `arguments` keyword. For example,

```
let x = function () {  
  console.log(arguments);  
}  
x(4,6,7); // Arguments [4, 6, 7]
```

Arrow functions do not have arguments binding.

When you try to access an argument using the arrow function, it will give an error. For example,

```
let x = () => {  
  console.log(arguments);  
}
```

```
x(4,6,7);  
// ReferenceError: Can't find variable: arguments
```