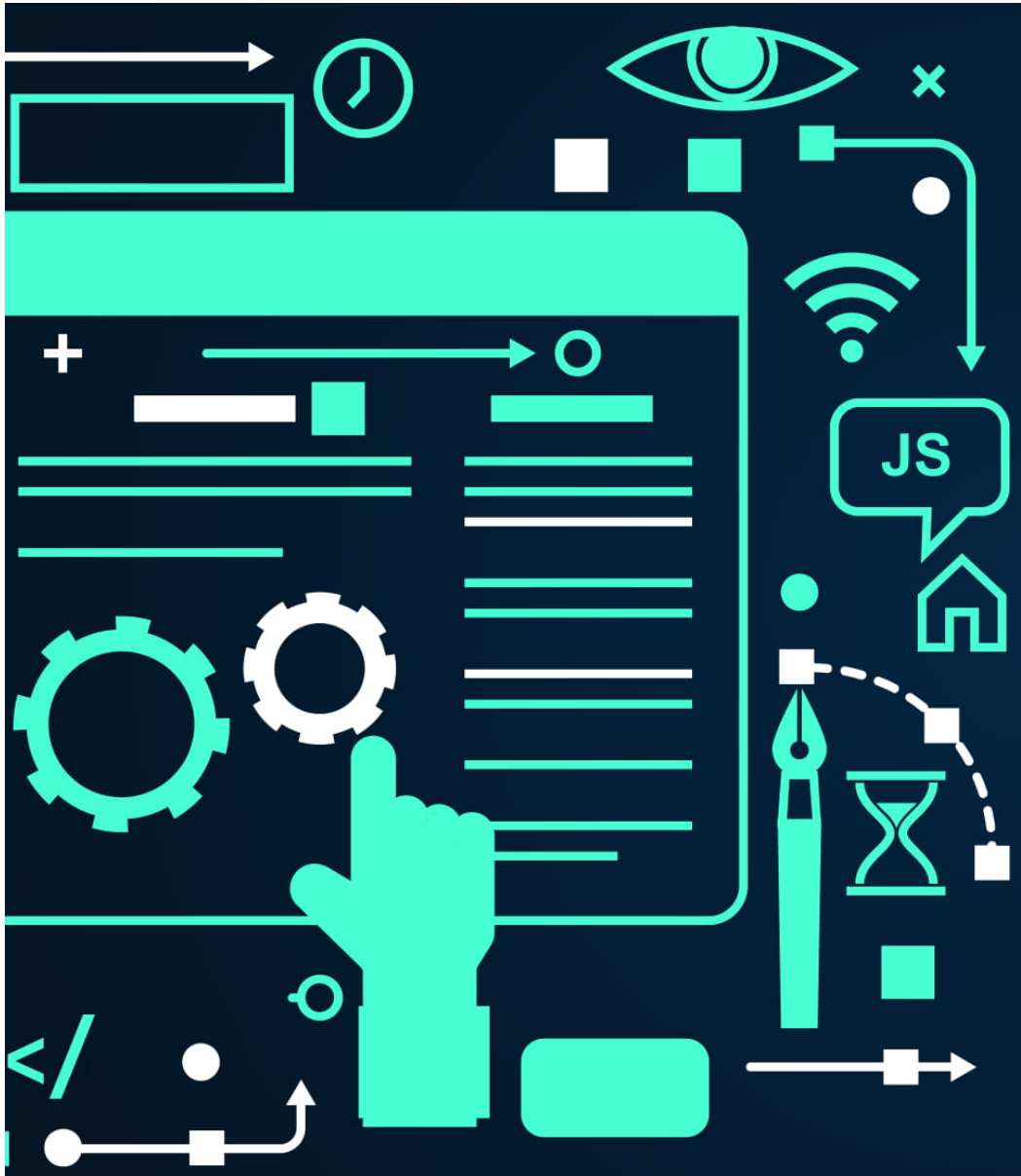


JAVASCRIPT

Programming language

Daniyal Manzoor

JavaScript

- JavaScript is the most popular and widely used client-side scripting language. Client-side scripting refers to scripts that run within your web browser. JavaScript is designed to add interactivity and dynamic effects to the web pages by manipulating the content returned from a web server.
- JavaScript was originally developed as LiveScript by Netscape in the mid 1990s. It was later renamed to JavaScript in 1995, and became an ECMA standard in 1997. Now JavaScript is the standard client-side scripting language for web-based applications, and it is supported by virtually all web browsers available today, such as Google Chrome, Mozilla Firefox, Apple Safari, etc.
- JavaScript is an object-oriented language, and it also has some similarities in syntax to Java programming language. But, JavaScript is not related to Java in any way.
- JavaScript is officially maintained by ECMA (European Computer Manufacturers Association) as ECMAScript. ECMAScript 6 (or ES6) is the latest major version of the ECMAScript standard.

What You Can Do with JavaScript

There are lot more things you can do with JavaScript.

- You can modify the content of a web page by adding or removing elements.
- You can change the style and position of the elements on a web page.
- You can monitor events like mouse click, hover, etc. and react to it.
- You can perform and control transitions and animations.
- You can create alert pop-ups to display info or warning messages to the user.
- You can perform operations based on user inputs and display the results.
- You can validate user inputs before submitting it to the server.

JAVASCRIPT
GETTING STARTED

Adding JavaScript to Your Web Pages

There are typically three ways to add JavaScript to a web page:

- Embedding the JavaScript code between a pair of <script> and </script> tag.
- Creating an external JavaScript file with the .js extension and then load it within the page through the src attribute of the <script> tag.
- Placing the JavaScript code directly inside an HTML tag using the special tag attributes such as onclick, onmouseover, onkeypress, onload, etc.

Embedding the JavaScript Code

```
<!DOCTYPE html>
 <html lang="en">
     <head>
          <meta charset="UTF-8">
          <title>Embedding JavaScript</title>
     </head>
     <body>
          <script>
          var greet = "Hello World!";
          document.write(greet); // Prints: Hello World!
          </script>
     </body>
 </html>
```

Calling an External JavaScript File

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

```
Console.log("Hello World!");
```

Embedding the JavaScript Code

```
<!DOCTYPE html>
 <html lang="en">
     <head>
          <meta charset="UTF-8">
          <title>Inlining JavaScript</title>
     </head>
     <body>
          <button onclick="alert('Hello World!')">Click Me</button>    </body>
 </html>
```

JAVASCRIPT

Generating Output

Writing Output to Browser Console

You can easily outputs a message or writes data to the browser console using the console.log() method. This is a simple, but very powerful method for generating detailed output. Here's an example:

```
console.log("Hello World!");
```

Displaying Output in Alert Dialog Boxes

You can also use alert dialog boxes to display the message or output data to the user. An alert dialog box is created using the alert() method. Here's is an example:

```
alert("Hello World!");
```

Writing Output to the Browser Window

You can use the document.write() method to write the content to the current document only while that document is being parsed. Here's an example:

```
document.write("Hello World!");
```

Inserting Output Inside an HTML Element

You can also write or insert output inside an HTML element using the element's innerHTML property. However, before writing the output first we need to select the element using a method such as getElementById(), as demonstrated in the following example:

```
<p id="greet"></p>
<p id="result"></p>

<script>
document.getElementById("greet").innerHTML = "Hello World!";
</script>
```

JAVASCRIPT
COMMENTS

Single Line Comments

A comment is simply a line of text that is completely ignored by the JavaScript interpreter. Comments are usually added with the purpose of providing extra information pertaining to source code. It will not only help you understand your code when you look after a period of time but also others who are working with you on the same project.

JavaScript support single-line as well as multi-line comments. Single-line comments begin with a double forward slash (//), followed by the comment text. Here's an example:

```
// This is my first JavaScript program
console.log("Hello World!");
```

Multi-line Comments

Whereas, a multi-line comment begins with a slash and an asterisk (/*) and ends with an asterisk and slash (*/). Here's an example of a multi-line comment.

```
/* This is my first program in JavaScript
Here's an example of a multi-line comment.
 */
console.log("Hello World!");
```

JAVASCRIPT
VARIABLES

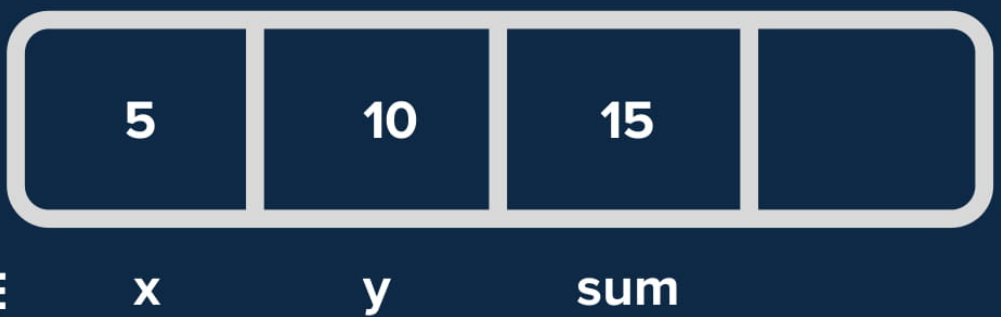
What is Variable?

Variables are fundamental to all programming languages. Variables are used to store data, like string of text, numbers, etc. The data or value stored in the variables can be set, updated, and retrieved whenever needed. In general, variables are symbolic names for values.

Program

```
var x = 5;
var y = 10;
var sum = x + y;
console.log(myVar);
```

Memory



Naming Conventions for JavaScript Variables

These are the following rules for naming a JavaScript variable:

- A variable name must start with a letter, underscore (_).
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters (A-z, 0-9) and underscores.
- A variable name cannot contain spaces.
- A variable name cannot be a JavaScript keyword or a JavaScript reserved word.

Declaring Multiple Variables at Once

In addition, you can also declare multiple variables and set their initial values in a single statement. Each variable are separated by commas, as demonstrated in the following example:

```
var name = "Peter Parker", age = 21, isMarried = false;

var name = "Peter Parker",
age = 21,
isMarried = false;
```

JAVASCRIPT

Data Types

The String Data Type

The string data type is used to represent textual data (i.e. sequences of characters). Strings are created using single or double quotes surrounding one or more characters, as shown below:

```
var a = 'Hi there!'; // using single quotes
var b = "Hi there!"; // using double quotes
console.log(typeof a);
console.log(typeof b);
```

The Number Data Type

- The number data type is used to represent positive or negative numbers with or without decimal place, or numbers written using exponential notation e.g. 1.5e-4 (equivalent to 1.5x10-4).
- The Number data type also includes some special values which are: Infinity, -Infinity and NaN.

```
var a = 25; // integer
var b = 80.5; // floating-point number
var c = 4.25e+6; // exponential notation, same as 4.25e6 or
4250000
var d = 4.25e-6; // exponential notation, same as 0.00000425
console.log(typeof a);
console.log(typeof b);
```

- The Number data type also includes some special values which are: Infinity, -Infinity and NaN. Infinity represents the mathematical Infinity ∞, which is greater than any number. Infinity is the result of dividing a nonzero number by 0, as demonstrated below:

```
alert(16 / 0); // Output: Infinity
alert(-16 / 0); // Output: -Infinity
alert(16 / -0); // Output: -Infinity
```

- While NaN represents a special Not-a-Number value. It is a result of an invalid or an undefined mathematical operation, like taking the square root of -1 or dividing 0 by 0, etc.

```
alert("Some text" / 2); // Output: NaN
alert("Some text" / 2 + 10); // Output: NaN
alert(Math.sqrt(-1)); // Output: NaN
```

The Boolean Data Type

The Boolean data type can hold only two values: true or false. It is typically used to store values like yes (true) or no (false), on (true) or off (false), etc. as demonstrated below:

```
var isReading = true; // yes, I'm reading
var isSleeping = false; // no, I'm not sleeping
```

The Undefined Data Type

The undefined data type can only have one value-the special value undefined. If a variable has been declared, but has not been assigned a value, has the value undefined.

```
var a;
var b = "Hello World!“

alert(a) // Output: undefined
alert(b) // Output: Hello World!
```

The Null Data Type

This is another special data type that can have only one value-the null value. A null value means that there is no value. It is not equivalent to an empty string ("") or 0, it is simply nothing.

A variable can be explicitly emptied of its current contents by assigning it the null value.

```
var a = null;
alert(a); // Output: null
var b = "Hello World!“
alert(b); // Output: Hello World!
b = null;
alert(b) // Output: null
```

JAVASCRIPT
SYNTAX

Syntax

The syntax of JavaScript is the set of rules that define a correctly structured JavaScript program.

```
console.log("Hello World!");
document.write("Hello World!");
```

Case Sensitivity

JavaScript is case-sensitive. This means that variables, language keywords, function names, and other identifiers must always be typed with a consistent capitalization of letters.

For example, the variable myVar must be typed myVar not MyVar or myvar. Similarly, the method name getElementById() must be typed with the exact case not as getElementByID().

```
var myVar = "Hello World!";
console.log(myVar);
console.log(MyVar);
console.log(myvar);
```

JAVASCRIPT

Operators 01

Arithmetic Operators

The arithmetic operators are used to perform common arithmetical operations, such as addition, subtraction, multiplication etc. Here's a complete list of JavaScript's arithmetic operators:

Operator	Description	Example	Result
+	Addition	x + y	Sum of x and y
-	Subtraction	x - y	Difference of x and y.
*	Multiplication	x * y	Product of x and y.
/	Division	x / y	Quotient of x and y
%	Modulus	x % y	Remainder of x divided by y

Arithmetic Operators

```
var x = 10;
var y = 4;

alert(x + y); // 0utputs: 14
alert(x - y); // 0utputs: 6
alert(x * y); // 0utputs: 40
alert(x / y); // 0utputs: 2.5
alert(x % y); // 0utputs: 2
```

Assignment Operators

The assignment operators are used to assign values to variables.

Operator	Description	Example	Is The Same As
=	Assign	x = y	x = y
+=	Add and assign	x += y	x = x + y
-=	Subtract and assign	x -= y	x = x - y
*=	Multiply and assign	x *= y	x = x * y
/=	Divide and assign quotient	x /= y	x = x / y
%=	Divide and assign modulus	x %= y	x = x % y

Assignment Operators

```
var x; // Declaring Variable

x = 10;
alert(x); // Outputs: 10

x = 20;
x += 30;
alert(x); // Outputs: 50

x = 50;
x -= 20;
alert(x); // Outputs: 30
```

```
x = 5;
x *= 25;
alert(x); // Outputs: 125

x = 50;
x /= 10;
alert(x); // Outputs:

x = 100;
x %= 15;
alert(x); // Outputs: 10
```

String Operators

There are two operators which can also used be for strings.

Operator	Description	Example	Is The Same As
+	Concatenation	str1 + str2	Concatenation of str1 and str2
+=	Concatenation assignment	str1 += str2	Appends the str2 to the str1

String Operators

```
var str1 = "Hello";
var str2 = " World!";

alert(str1 + str2); // Outputs: Hello World

str1 += str2;
alert(str1); // Outputs: Hello World!
```

JAVASCRIPT

Events

Understanding Events and Event Handlers

An event is something that happens when user interact with the web page, such as when he clicked a link or button, entered text into an input box or textarea, made selection in a select box, pressed key on the keyboard, moved the mouse pointer, submits a form, etc.

When an event occur, you can use a JavaScript event handler (or an event listener) to detect them and perform specific task or set of tasks. By convention, the names for event handlers always begin with the word "on", so an event handler for the click event is called onclick, similarly an event handler for the load event is called onload, event handler for the blur event is called onblur

```
<button type="button" onclick="alert('Hello World!')">Click Me</button>
```

JAVASCRIPT

Mouse Events

The Click Event (onclick)

The click event occurs when a user clicks on an element on a web page. Often, these are form elements and links. You can handle a click event with an onclick event handler.

```
<button type="button" onclick="alert('You have clicked a
button!');">Click Me</button>

<a href="#" onclick="alert('You have clicked a
link!');">Click Me</a>
```

The Contextmenu Event (oncontextmenu)

The contextmenu event occurs when a user clicks the right mouse button on an element to open a context menu. You can handle a contextmenu event with an oncontextmenu event handler.

```
<button type="button" oncontextmenu="alert('You have
right-clicked a button!');">Right Click on Me</button>

<a href="#" oncontextmenu="alert('You have right-clicked a
link!');">Right Click on Me</a>
```

The Mouseover Event (onmouseover)

The mouseover event occurs when a user moves the mouse pointer over an element.

You can handle the mouseover event with the onmouseover event handler. The following example will show you an alert message when you place mouse over the elements.

```
<button type="button" onmouseover="alert('You have placed
mouse pointer over a button!');">Place Mouse Over Me</button>

<a href="#" onmouseover="alert('You have placed mouse pointer
over a link!');">Place Mouse Over Me</a>
```

The Mouseout Event (onmouseout)

The mouseout event occurs when a user moves the mouse pointer outside of an element.

You can handle the mouseout event with the onmouseout event handler. The following example will show you an alert message when the mouseout event occurs.

```
<button type="button" onmouseout="alert('You have moved out of the button!');">Place Mouse Inside Me and Move Out</button>

<a href="#" onmouseout="alert('You have moved out of the link!');">Place Mouse Inside Me and Move Out</a>
```

JAVASCRIPT

Keyboard Events

The Keydown Event (onkeydown)

The keydown event occurs when the user presses down a key on the keyboard.

You can handle the keydown event with the onkeydown event handler. The following example will show you an alert message when the keydown event occurs.

```
<input type="text" onkeydown="alert('You have pressed a key inside text input!')">

<textarea onkeydown="alert('You have pressed a key inside textarea!')"></textarea>
```

The Keyup Event (onkeyup)

The keyup event occurs when the user releases a key on the keyboard.

You can handle the keyup event with the onkeyup event handler. The following example will show you an alert message when the keyup event occurs.

```
<input type="text" onkeyup="alert('You have released a key
inside text input!')">

<textarea onkeyup="alert('You have released a key inside
textarea!')"></textarea>
```

The Keypress Event (onkeypress)

The keypress event occurs when a user presses down a key on the keyboard that has a character value associated with it. For example, keys like Ctrl, Shift, Alt, Esc, Arrow keys, etc. will not generate a keypress event, but will generate a keydown and keyup event.

You can handle the keypress event with the onkeypress event handler. The following example will show you an alert message when the keypress event occurs

```
<input type="text" onkeypress="alert('You have pressed a key inside text input!')">

<textarea onkeypress="alert('You have pressed a key inside textarea!')"></textarea>
```

JAVASCRIPT
FUNCTIONS

What is Function?

A function is a group of statements that perform specific tasks and can be kept and maintained separately form main program. Functions provide a way to create reusable code packages which are more portable and easier to debug. Here are some advantages of using functions:

Functions reduces the repetition of code within a program — Function allows you to extract commonly used block of code into a single component. Now you can perform the same task by calling this function wherever you want within your script without having to copy and paste the same block of code again and again.

Functions makes the code much easier to maintain — Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.

Functions makes it easier to eliminate the errors — When the program is subdivided into functions, if any error occur you know exactly what function causing the error and where to find it. Therefore, fixing errors becomes much easier.

SYNTAX

The declaration of a function start with the function keyword, followed by the name of the function you want to create, followed by parentheses i.e. () and finally place your function's code between curly brackets {}. Here's the basic syntax for declaring a function:

```
function functionName() {
    // Code to be executed
}
```

Defining and Calling a Function

```
// Defining function
function sayHello(){
   alert("Hello, welcome to this website!");
}

// Calling function
sayHello(); // 0utputs: Hello, welcome to this website!
```

A function name must start with a letter or underscore character not with a number, optionally followed by the more letters, numbers, or underscore characters. Function names are case sensitive, just like variable names.

NOTE

Adding Parameters to Functions

You can specify parameters when you define your function to accept input values at run time. The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.

```
function functionName(parameter1, parameter2, parameter3) {
    // Code to be executed
}
```

```
// Defining function
function displaySum(num1, num2) {
   var total = num1 + num2;
   alert(total);
}

// Calling function
displaySum(6, 20); // 0utputs: 26
displaySum(-5, 17); // 0utputs: 12
```

Returning Values from a Function

A function can return a value back to the script that called the function as a result using the return statement. The value may be of any type, including arrays and objects.

```
// Defining function
function getSum(num1, num2) {
var total = num1 + num2;
return total;
}
// Displaying returned value
alert(getSum(6, 20)); // 0utputs: 26
alert(getSum(-5, 17)); // 0utputs: 12
```

Working with Function Expressions

The syntax that we've used before to create functions is called function declaration. There is another syntax for creating a function that is called a function expression.

Once function expression has been stored in a variable, the variable can be used as a function:

```
// Function Expression
var getSum = function(num1, num2) {
   var total = num1 + num2;
   return total;
};
alert(getSum(5, 10));
```

There is no need to put a semicolon after the closing curly bracket in a function declaration. But function expressions, on the other hand, should always end with a semicolon.

NOTE

```
declaration(); // Outputs: Hi, I'm a function declaration!
function declaration() {
   alert("Hi, I'm a function declaration!");
}

expression(); // Uncaught TypeError: undefined is not a function
var expression = function() {
alert("Hi, I'm a function expression!");
};
```

JAVASCRIPT

FUNCTIONS

ES6

Adding Parameters to Functions

you can specify default values to the function parameters. This means that if no arguments are provided to function when it is called these default parameters values will be used

```
function sayHello(name = 'Guest'){
alert('Hello, ' + name);
}

sayHello(); // 0utputs: Hello, Guest
sayHello('John'); // 0utputs: Hello, John
```

Arrow Functions

Arrow Functions are another interesting feature in ES6. It provides a more concise syntax for writing function expressions by opting out the function and return keywords.

Arrow functions are defined using a new syntax, the fat arrow (=>) notation. Let's see how it looks:

```
// Arrow function
var sum = (a, b) => a + b;
console.log(sum(2, 3)); // 5
```

Arrow Functions

As you can see there is no function and return keyword in arrow function declaration.

You can also skip the parentheses i.e. () in case when there is exactly one parameter, but you will always need to use it when you have zero or more than one parameter.

Additionally, if there's more than one expression in the function body, you need to wrap it braces ({}). In this case you also need to use the return statement to return a value.

There are several variations of how you can write arrow functions. Here are the most commonly used:

```
// Single parameter, single statement
var greet = name => alert("Hi " + name +
"!");

greet("Peter");// Hi Peter!


// Multiple arguments, single statement
var multiply = (x, y) => x * y;

alert(multiply(2, 3)); // 6


// Single parameter, multiple statements
var test = age => {
  if(age > 18) {
  alert("Adult"); }
  else { alert("Teenager"); }
}
test(21); // Adult


// Multiple parameters, multiple
statements
var divide = (x, y) => {
 if(y != 0) {
 return x / y; }
}

alert(divide(10, 2)); // 5


// No parameter, single statement
var hello = () => alert('Hello
World!');

hello(); // Hello World!
```

JAVASCRIPT

Document/Window Events

The Load Event (onload)

The load event occurs when a web page has finished loading in the web browser.

You can handle the load event with the onload event handler. The following example will show you an alert message as soon as the page finishes loading.

```
<body onload="window.alert('Page is loaded successfully!');">
   <h1>This is a heading</h1>
   <p>This is paragraph of text.</p>
</body>
```

The Unload Event (onunload)

The unload event occurs when a user leaves the current web page.

You can handle the unload event with the onunload event handler. The following example will show you an alert message when you try to leave the page.

```
<body onunload="alert('Are you sure you want to leave this
page?');">
   <h1>This is a heading</h1>
   <p>This is paragraph of text.</p>
</body>
```

The Resize Event (onresize)

The resize event occurs when a user resizes the browser window. The resize event also occurs in situations when the browser window is minimized or maximized.

You can handle the resize event with the onresize event handler. The following example will show you an alert message when you resize the browser window to a new width and height.

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

<script>
   function displayWindowSize() {
       var w = window.outerWidth;
       var h = window.outerHeight;
       var txt = "Window size: width=" + w + ", height=" + h;
       document.getElementById("result").innerHTML = txt;
}
   window.onresize = displayWindowSize;
</script>
```

JAVASCRIPT

Form Events

The Focus Event (onfocus)

The focus event occurs when the user gives focus to an element on a web page.

You can handle the focus event with the onfocus event handler. The following example will highlight the background of text input in yellow color when it receives the focus.

Note: The value of this keyword inside an event handler refers to the element which has the handler on it (i.e. where the event is currently being delivered).

```
<input type="text" onfocus="highlightInput(this)">
<buttontype="button">Button</button>

<script>
   function highlightInput(elm){
   elm.style.background = "yellow";
   }
</script>
```

The Blur Event (onblur)

The blur event occurs when the user takes the focus away from a form element or a window.

You can handle the blur event with the onblur event handler. The following example will show you an alert message when the text input element loses focus.

```
<input type="text" onblur="alert('Text input loses focus!')">

<button type="button">Submit</button>
```

The Change Event (onchange)

The change event occurs when a user changes the value of a form element.

You can handle the change event with the onchange event handler. The following example will show you an alert message when you change the option in the select box.

```
<select onchange="alert('You have changed the selection!');">
  <option>Select</option>
  <option>Male</option>
  <option>Female</option>
</select>
```

The Submit Event (onsubmit)

The submit event only occurs when the user submits a form on a web page.

You can handle the submit event with the onsubmit event handler. The following example will show you an alert message while submitting the form to the server.

```
<form onsubmit="alert('Form data will be submitted to the
server!');">
  <label>First Name:</label>
  <input type="text" name="first-name" required>
  <input type="submit" value="Submit">
</form>
```

JAVASCRIPT

Operators 02

Comparison Operators

Operator	Description	Example	Is The Same As
==	Equal	x == y	True if x is equal to y
===	Identical	x === y	True if x is equal to y, and they are of the same type
!=	Not equal	x != y	True if x is not equal to y
!==	Not identical	x !== y	True if x is not equal to y, or they are not of the same type
<	Less than	x < y	True if x is less than y
>	Greater than	x > y	True if x is greater than y
>=	Greater than or equal to	x >= y	True if x is greater than or equal to y
<=	Less than or equal to	x <= y	True if x is less than or equal to y

Comparison Operators

```
var x = 25;
var y = 35;
var z = "25";

alert(x == z); // Outputs: true

alert(x === z); // Outputs: false

alert(x != y); // Outputs: true

alert(x !== z); // Outputs: true
```

```
alert(x < y); // Outputs: true

alert(x > y); // Outputs: false

alert(x <= y); // Outputs: true

alert(x >= y); // Outputs: false
```

JAVASCRIPT

Conditions - Control Flow

Conditions - Control Flow

Conditional statements are used to perform different action based on different condition

Conditional statements allow the developer to make correct decisions and perform right actions as per condition

It helps to perform different actions for different decisions

We can use the following conditional statements in JavaScript to make decisions:

- If Statement
- If...else Statement
- If...else if...else Statement
- Switch...Case Statement

The if statement

If the conditional statement is the simplest and basic control statement make decisions and execute statements conditionally

The if statement is used to execute a block of code only if the specified condition evaluates to true

It evaluates the content only if an expression is true

```
if(condition) {
    // Code to be executed
}
```

```
var now = new Date();
var dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

if(dayOfWeek == 5) {
   alert("Have a nice weekend!");
}
```

The if else statement

If The JavaScript if...else statement is used to execute the code weather condition is true or false

The developer can enhance the decision-making capabilities by providing an alternative choice through adding an else statement to the if statement

The condition can be any expression that evaluates to true or false

If the condition evaluates to true, statements_1 are executed; otherwise, statements_2 are executed

```
if(condition) {
    // Code to be executed if condition is true
} else {
    // Code to be executed if condition is false
}
```

```
var now = new Date();
var dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

if(dayOfWeek == 5) {
alert("Have a nice weekend!");
}
else {
alert("Have a nice day!");
}
```

The if...else if...else Statement

The if...else if...else a special statement that is used to combine multiple if...else statements

It is an advanced form of if...else that allows us to make a correct decision out of several conditions

```
if(condition1) {
    // Code to be executed if condition1 is true
} else if(condition2) {
    // Code to be executed if the condition1 is false and condition2 is true
} else {
    // Code to be executed if both condition1 and condition2 are false
}
```

```
var now = new Date();
var dayOfWeek = now.getDay(); // Sunday - Saturday : 0 - 6

if(dayOfWeek == 5) {
alert("Have a nice weekend!");
}
else if(dayOfWeek == 0) {
alert("Have a nice Sunday!");
}
else {
alert("Have a nice day!");
}
```

The Ternary Operator

The ternary operator provides a shorthand way of writing the if...else statements. The ternary operator is represented by the question mark (?) symbol and it takes three operands: a condition to check, a result for true, and a result for false. Its basic syntax is:

```
var result = (condition) ? value1 : value2
```

```
var age = 21;

var userType = age < 18 ? 'Child' : 'Adult';

alert(userType); // Displays Adult
```

The switch...case statement

The switch...case statement is alternative to an if...else if...else statement, both do almost the same thing

This matches the case and the value of condition and if the case matches, the subsequent block is executed and if none of the case matches default block is executed

The JavaScript switch statement is used to execute one code from multiple blocks of expressions

```
switch(x){
    case value1:
        // Code to be executed if x === value1
        break;
    case value2:
        // Code to be executed if x === value2
        break;
    ...
    default:
        // Code to be executed if x is different from all values
}
```

```
var d = new Date();
switch(d.getDay()) {
    case 0:
          alert("Today is Sunday.");
          break;
    case 1:
          alert("Today is Monday.");
          break;
    case 2:
          alert("Today is Tuesday.");
          break;
    case 3:
          alert("Today is Wednesday.");
          break;
    case 4:
          alert("Today is Thursday.");
          break;
    case 5:
          alert("Today is Friday.");
          break;
    case 6:
          alert("Today is Saturday.");
          break;
    default:
          alert("No information available for that day.");
          break;
    }
```

JAVASCRIPT

Operators 03

Logical Operators

The logical operators are typically used to combine conditional statements.

Operator	Name	Example	Result
&&	And	x && y	True if both x and y are true
||	Or	x || y	True if either x or y is true
!	Not	!x	True if x is not true

Logical Operators

```
// && (Logical AND) - returns true if both operands are true

console.log('true && true: ', true && true);
console.log('true && false: ', true && false);
console.log('false && true: ', false && true);
```

Logical Operators

```
// || (Logical OR) - returns true if one of the operand is true

console.log('true || true: ', true || true);
console.log('true || false: ', true || false);
console.log('false || true: ', false || true);
```

Logical Operators

```
// ! (Logical NOT) True if operand is not true (means I will be true if
other is false)
let isSeniorCitizen = true;

let isYoungGeneration = !isSeniorCitizen;
console.log('isYoungGeneration: ', isYoungGeneration);
```

Logical Operators

```
var year = 2018; // Leap years are divisible by 400 or by 4 but not 100

if((year % 400 == 0) || ((year % 100 != 0) && (year % 4 == 0))){

alert(year + " is a leap year.");
}
else{
alert(year + " is not a leap year.");
}
```

JAVASCRIPT

OBJECT

OBJECT

A JavaScript object is just a collection of named values. These named values are usually referred to as properties of the object.

```
var person = {
   name: "Peter",
   age: 28,
   gender: "Male",
   displayName() {
   alert(this.name);}
};
```

Accessing Object's Properties

```
var book = {
"name": "Harry Potter and the Goblet of Fire",
"author": "J. K. Rowling",
"year": 2000
}; // Dot notation

document.write(book.author); // Prints: J. K. Rowling // Bracket
notation
document.write(book["year"]); // Prints: 2000
```

JAVASCRIPT
ARRAY

Array

```
var myArray = [element0, element1, ..., elementN];
```

```
var colors = ["Red", "Green", "Blue"];
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
var cities = ["London", "Paris", "New York"];
var person = ["John", "Wick", 32];
```

Accessing the Elements of an Array

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
document.write(fruits[0]); // Prints: Apple
document.write(fruits[1]); // Prints: Banana
document.write(fruits[2]); // Prints: Mango
document.write(fruits[fruits.length - 1]); // Prints: Papaya
```

Getting the Length of an Array

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];

document.write(fruits.length); // Prints: 5
```

Adding Last New Elements to an Array

```
var colors = ["Red", "Green", "Blue"];
colors.push("Yellow");

document.write(colors); // Prints: Red,Green,Blue,Yellow
document.write(colors.length); // Prints: 4
```

```
var colors = ["Red", "Green", "Blue"];
colors.unshift("Yellow");

document.write(colors); // Prints: Yellow,Red,Green,Blue
document.write(colors.length); // Prints: 4
```

Removing Elements from an Array

```
var colors = ["Red", "Green", "Blue"];
var last = colors.pop();

document.write(last); // Prints: Blue
document.write(colors.length); // Prints: 2
```

```
var colors = ["Red", "Green", "Blue"];
var first = colors.shift();

document.write(first); // Prints: Red
document.write(colors.length); // Prints: 2
```

Adding or Removing Elements at Any Position

The splice() method is a very versatile array method that allows you to add or remove elements from any index, using the syntax

This method takes three parameters: the first parameter is the index at which to start splicing the array, it is required; the second parameter is the number of elements to remove (use 0 if you don't want to remove any elements), it is optional; and the third parameter is a set of replacement elements, it is also optional. The following example shows how it works:

```
arr.splice(startIndex, deleteCount, elem1, ..., elemN)
```

```
var colors = ["Red", "Green", "Blue"];
var removed = colors.splice(0,1); // Remove the first element
document.write(colors); // Prints: Green,Blue
document.write(removed); // Prints: Red (one item array)

removed = colors.splice(1, 0, "Pink", "Yellow"); // Insert two items at position
one
document.write(colors); // Prints: Green,Pink,Yellow,Blue
document.write(removed); // Empty array
document.write(removed.length); // Prints: 0

removed = colors.splice(1, 1, "Purple", "Voilet"); // Insert two values, remove
one
document.write(colors); //Prints: Green,Purple,Voilet,Yellow,Blue
document.write(removed); // Prints: Pink (one item array)
document.write(removed.length); // Prints: 1
```

Creating a String from an Array

here may be situations where you simply want to create a string by joining the elements of an array. To do this you can use the join() method.

```
var colors = ["Red", "Green", "Blue"];
document.write(colors.join()); // Prints: Red,Green,Blue
document.write(colors.join("")); // Prints: RedGreenBlue
document.write(colors.join("-")); // Prints: Red-Green-Blue
document.write(colors.join(", ")); // Prints: Red, Green, Blue
```

Merging Two or More Arrays

The concat() method can be used to merge or combine two or more arrays. This method does not change the existing arrays, instead it returns a new array. For example:

```
var pets = ["Cat", "Dog", "Parrot"];
var wilds = ["Tiger", "Wolf", "Zebra"]; // Creating new array by
combining pets and wilds arrays
var animals = pets.concat(wilds);
document.write(animals); // Prints: Cat,Dog,Parrot,Tiger,Wolf,Zebra
```

Searching Through an Array

If you want to search an array for a specific value, you can simply use the indexOf() and lastIndexOf(). If the value is found, both methods return an index representing the array element. If the value is not found, -1

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
document.write(fruits.indexOf("Apple")); // Prints: 0
document.write(fruits.indexOf("Banana")); // Prints: 1
document.write(fruits.indexOf("Pineapple")); // Prints: -1
```

Searching Through an Array

You can also use includes() method to find out whether an array includes a certain element or not.

```
var arr = [1, 0, 3, 1, false, 5, 1, 4, 7];
document.write(arr.includes(1)); // Prints: true
document.write(arr.includes(6)); // Prints: false
document.write(arr.includes(1, 2)); // Prints: true
document.write(arr.includes(3, 4)); // Prints: false
```

JAVASCRIPT

ARRAY FUNCTION

ES6

8 most Important Method

JavaScript Array filter() Method

The filter() method creates an array filled with all array elements that pass a test (provided as a function)

```
array.filter(function(currentValue, index, arr), thisValue)
```

```
var ages = [32, 33, 16, 40];
function checkAdult(age) {
  return age >= 18;
}
function myFunction() {
  document.getElementById("demo").innerHTML = ages.filter(checkAdult);
}
```

JavaScript Array map() Method

The map() method creates a new array with the results of calling a function for every array element.

```
array.map(function(currentValue, index, arr), thisValue)
```

```
var numbers = [65, 44, 12, 4];
var newarray = numbers.map(myFunction)
function myFunction(num) {
  return num * 10;
}

document.getElementById("demo").innerHTML = newarray;
```

JavaScript Array find() Method

The find() method returns the value of the first element in an array that pass a test.

```
array.find(function(currentValue, index, arr),thisValue)
```

```
var ages = [3, 10, 18, 20];
function checkAdult(age) {
  return age >= 18;
}
function myFunction() {
  document.getElementById("demo").innerHTML = ages.find(checkAdult);
}
```

JavaScript Array forEach() Method

The forEach() method calls a function once for each element in an array, in order.

```
array.forEach(function(currentValue, index, arr), thisValue)
```

```
var numbers = [65, 44, 12, 4];
numbers.forEach(myFunction)

function myFunction(item, index, arr) {
  arr[index] = item * 10;
}
```

JavaScript Array some() Method

The map() method creates a new array with the results of calling a function for every array element.

```
array.some(function(currentValue, index, arr), thisValue)
```

```
var ages = [3, 10, 18, 20];
function checkAdult(age) {
  return age >= 18;
}
function myFunction() {
  document.getElementById("demo").innerHTML = ages.some(checkAdult);
}
```

JavaScript Array every() Method

If it finds an array element where the function returns a false value, every() returns false (and does not check the remaining values)

If no false occur, every() returns true

```
array.every(function(currentValue, index, arr), thisValue)
```

```
var ages = [32, 33, 16, 40];
function checkAdult(age) {
  return age >= 18;
}
function myFunction() {
  document.getElementById("demo").innerHTML = ages.every(checkAdult);
}
```

JavaScript Array reduce() Method

The reduce() method reduces the array to a single value.

The reduce() method executes a provided function for each value of the array (from left-to-right).

The return value of the function is stored in an accumulator (result/total).

```
array.reduce(function(total, currentValue, currentIndex, arr), initialValue)
```

```
var numbers = [175, 50, 25];

document.getElementById("demo").innerHTML = numbers.reduce(myFunc);
function myFunc(total, num) {
  return total - num;
}
```

JavaScript Array includes() Method

You can also use includes() method to find out whether an array includes a certain element or not.

```
var arr = [1, 0, 3, 1, false, 5, 1, 4, 7];
document.write(arr.includes(1)); // Prints: true
document.write(arr.includes(6)); // Prints: false
document.write(arr.includes(1, 2)); // Prints: true
document.write(arr.includes(3, 4)); // Prints: false
```

JAVASCRIPT

Operators 04

Incrementing and Decrementing Operators

The increment/decrement operators are used to increment/decrement a variable's value.

Operator	Name	Effect
++x	Pre-increment	Increments x by one, then returns x
x++	Post-increment	Returns x, then increments x by one
--x	Pre-decrement	Decrements x by one, then returns x
x--	Post-decrement	Returns x, then decrements x by one

Incrementing and Decrementing Operators

```
var x; // Declaring Variable

x = 10;
alert(++x); // Outputs: 11
alert(x); // Outputs: 11

x = 10;
alert(x++); // Outputs: 10
alert(x); // Outputs: 11

x = 10;
alert(--x); // Outputs: 9
alert(x); // Outputs: 9

x = 10;
alert(x--); // Outputs: 10
alert(x); // Outputs: 9
```

JAVASCRIPT
LOOP

LOOP

- while — loops through a block of code as long as the condition specified evaluates to true.
- do…while — loops through a block of code once; then the condition is evaluated. If the condition is true, the statement is repeated as long as the specified condition is true.
- for — loops through a block of code until the counter reaches a specified number.
- for…in — loops through the properties of an object.
- for…of — loops over iterable objects such as arrays, strings, etc.

While Loop

```
while(condition) {
    // Code to be executed
}
```

```
var i = 1;
while(i <= 5) {
    document.write("<p>The number is " + i + "</p>");
    i++;
}
```

Note: Make sure that the condition specified in your loop eventually goes false. Otherwise, the loop will never stop iterating which is known as infinite loop. A common mistake is to forget to increment the counter variable (variable i in our case)

NOTE

do...while Loop

```
do {
    // Code to be executed
}
while(condition);
```

```
var i = 1;
do {
    document.write("<p>The number is " + i + "</p>");
    i++;
}while(i <= 5);
```

While Loop

```
while(condition) {
    // Code to be executed
}
```

```
var i = 1;
while(i <= 5) {
    document.write("<p>The number is " + i + "</p>");
    i++;
}
```

Difference Between while and do...while Loop

- The while loop differs from the do-while loop in one important way — with a while loop, the condition to be evaluated is tested at the beginning of each loop iteration, so if the conditional expression evaluates to false, the loop will never be executed.

- With a do-while loop, on the other hand, the loop will always be executed once even if the conditional expression evaluates to false, because unlike the while loop, the condition is evaluated at the end of the loop iteration rather than the beginning.

for Loop

```
for(initialization; condition; increment) {
    // Code to be executed
}
```

```
for(let i=1; i<=5; i++){
    document.write("<p>The number is " + i + "</p>");
}
```

For Loop Statement

- initialization — it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.
- condition — it is evaluated at the beginning of each iteration. If it evaluates to true, the loop statements execute. If it evaluates to false, the execution of the loop ends.
- increment — it updates the loop counter with a new value each time the loop runs.

for...in Loop

The for-in loop is a special type of a loop that iterates over the properties of an object, or the elements of an array. The generic syntax of the for-in loop is:

```
for(variable in object) {
    // Code to be executed
}
```

```
let person = {"name": "Clark", "surname": "Kent", "age": "36"};

for(var prop in person) {
    document.write("<p>" + prop + " = " + person[prop] + "</p>");
}
```

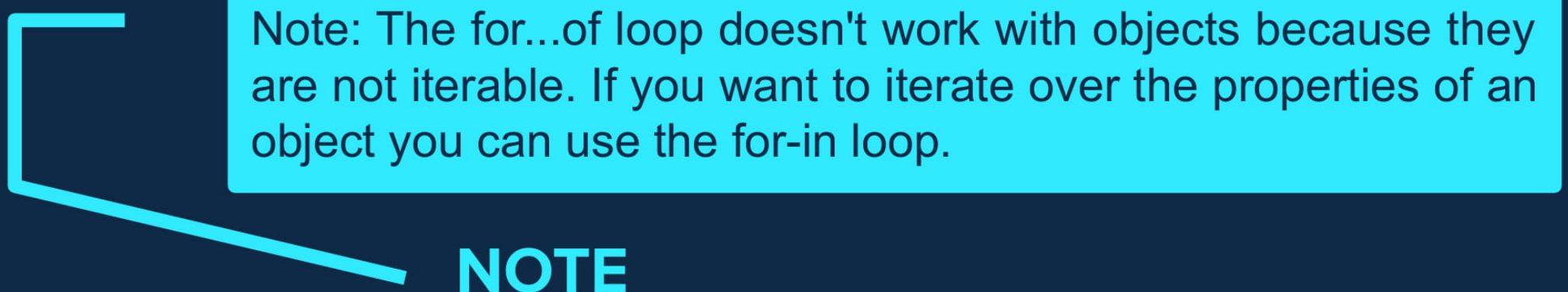
JAVASCRIPT

LOOP

ES6

For Loop Statement

- ES6 introduces a new for-of loop which allows us to iterate over arrays or other iterable objects (e.g. strings) very easily. Also, the code inside the loop is executed for each element of the iterable object.



for Loop

```
for(initialization; condition; increment) {
    // Code to be executed
}
```

```
let letters = ["a", "b", "c", "d", "e", "f"];
for(let letter of letters) {
    console.log(letter); // a,b,c,d,e,f
}
```

JAVASCRIPT

VARIABLES

ES6

The let and const Keywords

ES6 introduces two new keywords let and const for declaring variables.

The const keyword works exactly the same as let, except that variables declared using const keyword cannot be reassigned later in the code. Here's an example:

Unlike var, which declare function-scoped variables, both let and const keywords declare variables, scoped at block-level ({}). Block scoping means that a new scope is created between a pair of curly brackets {}.

var vs let vs const

	var	let	const
Stored in Global Scope	✓	✗	✗
Function Scope	✓	✓	✓
Block Scope	✗	✓	✓
Can be Reassigned?	✓	✓	✗
Can be Redeclared?	✓	✗	✗
Can Be Hoisted?	✓	✗	✗