

Unit -2

Java script & jQuery

Syllabus :

Java script: Variables, Arrays, Objects, Loops, Conditionals, Switches, Functions, Events, Form validating, Ajax.

jQuery: Selectors & Mouse events, Form events, DOM Manipulation, Effects & Animation, Traversing & Filtering.

Introduction to JavaScript

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities. JavaScript was first known as Live Script, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name Live Script. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

Advantages of JavaScript:

The merits of using JavaScript are:

- Less server interaction: You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- Immediate feedback to the visitors: They don't have to wait for a page reload to see if they have forgotten to enter something.
- Increased interactivity: You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- Richer interfaces: You can use JavaScript to include such items as drag-and drop components and sliders to give a Rich Interface to your site visitors

variables :

Variables are used to store data in JavaScript. Variables are used to store reusable values.

The values of the variables are allocated using the assignment operator(“=”).

JavaScript Identifiers

JavaScript variables must have unique names. These names are called Identifiers.

- Basic rules to declare a variable in JavaScript:
- These are case-sensitive
- Can only begin with a letter, underscore(“_”) or “\$” symbol
- It can contain letters, numbers, underscore, or “\$” symbol
- A variable name cannot be a reserved keyword.

JavaScript is a dynamically typed language so the type of variables is decided at runtime.

Therefore there is no need to explicitly define the type of a variable.

We can declare variables in JavaScript in three ways:

1. Using the var keyword

The “var” variable has function scope, which means it is accessible within the function where it is defined or globally if defined outside of any function. In this example, the variables' name and age are defined using the var keyword. They are accessible within the sayHello() function and can also be accessed globally (outside the function) due to their global scope.

Example-1

```
<html>

<head>

<title>Various ways to define a variable in JavaScript</title>

<script>

var name = "John";

var age = 25;

function sayHello() {

var message = "Hello, " + name + "! You are " + age + " years old.";

console.log(message);
```

```
}  
sayHello();  
</script>  
</head>  
<body>  
<h1>Variable Example - var</h1>  
</body>  
</html>
```

2. Using the let Keyword

Introduced in ECMAScript 2015 (ES6) version update, the let keyword provides block scope for variables, i.e they are limited to the scope of a block only (i.e., within curly braces) where they are declared. In this example, the variables name and age are defined using let. They are accessible within the sayHello() function due to function scope, but they cannot be accessed outside the function because of block scope.

Example-2

```
<html>  
<head>  
<title>Various ways to define a variable in JavaScript</title>  
<script>  
let name = "John";  
let age = 25;  
function sayHello() {  
let message = "Hello, " + name + "! You are " + age + " years old.";  
console.log(message);  
}  
sayHello();
```

```
</script>
</head>
<body>
<h1>Variable Example - let</h1>
</body>
</html>
```

3. Using the const Keyword

The const keyword is used to define variables that are constants, meaning their value cannot be reassigned once initialized, but if the variable is an object or an array, its properties or elements can still be modified. Constants have block scope, like variables defined with let. In this example, the variables PI and maxAttempts are defined as constants using the const keyword. The area variable within the calculateArea () function is also defined as a constant, ensuring it does not change within the function.

Example-3:

```
<html>
<head>
<title>Various ways to define a variable in JavaScript</title>
<script>
const PI = 3.14159;
const maxAttempts = 3;
function calculateArea(radius) {
const area = PI * radius * radius;
console.log("The area is: " + area);
}
calculateArea(5);
```

```
</script>
</head>
<body>
<h1>Variable Example - const</h1>
</body>
</html>
```

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

Global Variables: A global variable has global scope which means it can be defined anywhere in your JavaScript code.

Local Variables: A local variable will be visible only within a function where it is defined.

Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable
function checkscope( ) {
    var myVar = "local"; // Declare a local variable
    document.write(myVar);
}
//-->
</script>
```

It will produce the following result:

Local

JavaScript Data Type

There are two kinds of data types as mentioned below

- ☐ Primitive Data Type
- ☐ Non Primitive Data Type

The following table describes Primitive Data Types available in JavaScript:

Sr.No.	Datatype Description
1.	String Can contain groups of character as single value. It is represented in double quotes. E.g. var x= "tutorial".
2.	Numbers Contains the numbers with or without decimal. E.g. var x=11, y=44.56;
3.	Booleans Contain only two values either true or false. E.g. var x=true, y=false.
4.	Undefined variable with no value is called Undefined. E.g. var x;
5.	Null If we assign null to a variable, it becomes empty. E.g. var x=null;

The following table describes Non-Primitive Data Types in java Script

Sr.No.	Datatype Description
1.	Array Can contain groups of values of same type. E.g. var x= {1,2,3,55};
2.	Objects Objects are stored in property and value pair. E.g. var rectangle = { length: 5, breadth: 3};

Arrays :

JavaScript Array is a data structure that allows you to store and organize multiple values within a single variable. It is a versatile and dynamic object. It can hold various data types, including numbers, strings, objects, and even other arrays. Arrays in JavaScript are zero-indexed i.e. the first element is accessed with an index 0, the second element with an index of 1, and so forth.

Declaration of an Array

There are basically two ways to declare an array i.e. Array Literal and Array Constructor.

1. Creating an Array using Array Literal

Creating an array using array literal involves using square brackets [] to define and initialize the array. This method is concise and widely preferred for its simplicity.

Syntax:

```
let arrayName = [value1, value2, ...];
```

2. Creating an Array using Array Constructor (JavaScript new Keyword)

The “Array Constructor” refers to a method of creating arrays by invoking the Array constructor function. This approach allows for dynamic initialization and can be used to create arrays with a specified length or elements.

Syntax:

```
let arrayName = new Array();
```

Basic Operations on JavaScript Arrays :

1. Accessing Elements of an Array

Any element in the array can be accessed using the index number. The index in the arrays starts with 0.

// Creating an Array and Initializing with Values

```
let courses = ["HTML", "CSS", "Javascript", "React"];
```

// Accessing Array Elements

```
console.log(courses[0]);
```

```
console.log(courses[1]);
```

```
console.log(courses[2]);
```

```
console.log(courses[3]);]
```

Output

HTML

CSS

Javascript

React

2. Accessing the First Element of an Array

The array indexing starts from 0, so we can access first element of array using the index number.

// Creating an Array and Initializing with Values

```
let courses = ["HTML", "CSS", "JavaScript", "React"];
```

// Accessing First Array Elements

```
let firstItem = courses[0];
```

```
console.log("First Item: ", firstItem);
```

output :

First Item: HTML

3. Accessing the Last Element of an Array

We can access the last array element using `[array.length - 1]` index number.

// Creating an Array and Initializing with Values

```
let courses = ["HTML", "CSS", "JavaScript", "React"];
```

// Accessing Last Array Elements

```
let lastItem = courses[courses.length - 1];
```

```
console.log("First Item: ", lastItem);
```

output :

First Item: React

4. Modifying the Array Elements

Elements in an array can be modified by assigning a new value to their corresponding index.

// Creating an Array and Initializing with Values

```
let courses = ["HTML", "CSS", "Javascript", "React"];
```

```
console.log(courses);
```

```
courses[1]= "Bootstrap";
```

```
console.log(courses);
```

output :

```
[ 'HTML', 'CSS', 'Javascript', 'React' ]  
[ 'HTML', 'Bootstrap', 'Javascript', 'React' ]
```

5. Adding Elements to the Array

Elements can be added to the array using methods like push() and unshift().

We can use push() method to add elements to the end of array

We can use unshift() method to add element in the first position. It is counter part of push()

method

// Creating an Array and Initializing with Values

```
let courses = ["HTML", "CSS", "Javascript", "React"];
```

// Add Element to the end of Array

```
courses.push("Node.js");
```

// Add Element to the beginning

```
courses.unshift("Web Development");
```

```
console.log(courses);
```

output :

```
[ 'Web Development', 'HTML', 'CSS', 'Javascript', 'React', 'Node.js' ]
```

6. Removing Elements from an Array

Remove elements using methods like pop(), shift(), or splice().

- We can use pop() method to remove and return last element of the array
- We can use shift() method to remove and return first element of the array. It is counter part to pop() Method

// Creating an Array and Initializing with Values

```
let courses = ["HTML", "CSS", "Javascript", "React", "Node.js"];
```

```
console.log("Original Array: " + courses);
```

// Removes and returns the last element

```
let lastElement = courses.pop();
```

```
console.log("After Removed the last elements: " + courses);
```

// Removes and returns the first element

```
let firstElement = courses.shift();
```

```
console.log("After Removed the First elements: " + courses);
```

// Removes 2 elements starting from index 1

```
courses.splice(1, 2);
```

```
console.log("After Removed 2 elements starting from index 1: " + courses);
```

output :

```
Original Array: HTML,CSS,Javascript,React,Node.js
After Removed the last elements: HTML,CSS,Javascript,React
After Removed the First elements: CSS,Javascript,React
After Removed 2 elements starting from index 1: CSS
```

7. Array Length

Get the length of an array using the length property.

// Creating an Array and Initializing with Values

```
let courses = ["HTML", "CSS", "Javascript", "React", "Node.js"];
```

```
let len = courses.length;
```

```
console.log("Array Length: " + len);    output : Array Length: 5
```

Difference Between JavaScript Arrays and Objects

JavaScript arrays use indexes as numbers.

objects use indexes as names..

When to use JavaScript Arrays and Objects?

Arrays are used when we want element names to be numeric.

Objects are used when we want element names to be strings.

Objects

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.

Here is an example of a JavaScript object.

```
// object
```

```
const student = {  
    firstName: 'ram',  
    class: 10  
};
```

Here, student is an object that stores values such as strings and numbers.

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 { }.

JavaScript Object Properties

In JavaScript, "key: value" pairs are called properties. For example,

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

Here, name: 'John' and age: 20 are properties.

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.

```
objectName.key
```

For example,

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

```
// accessing property
```

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

Run Code

2. Using bracket Notation

Here is the syntax of the bracket notation.

```
objectName["propertyName"]
```

For example,

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
```

Here, a function is used as a value for the greet key. That's why we need to use person.greet() instead of person.greet to call the function inside the object.

A JavaScript method is a property containing a function declaration

```
const person = {  
  name: 'John',  
  age: 20,  
};  
  
// accessing property  
  
console.log(person["name"]); // John
```

Creating a JavaScript Object

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

- Create a single object, using an object literal.
- Create a single object, with the keyword **new**.
- Define an object constructor, and then create objects of the constructed type.
- Create an object using **Object.create()**.

Using an Object Literal

This is the easiest way to create a JavaScript Object.

Using an object literal, you both define and create an object in one statement.

An object literal is a list of name:value pairs (like age:50) inside curly braces {}.

The following example creates a new JavaScript object with four properties:

Example

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Try it Yourself »

Spaces and line breaks are not important. An object definition can span multiple lines:

Example

```
const person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};
```

Try it Yourself »

This example creates an empty JavaScript object, and then adds 4 properties:

Example

```
const person = {};  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";
```

Try it Yourself »

Using the JavaScript Keyword new

The following example create a new JavaScript object using `new Object()`, and then adds 4 properties:

Example

```
const person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";
```

Try it Yourself »

The examples above do exactly the same.

But there is no need to use `new Object()`.

For readability, simplicity and execution speed, use the object literal method.

Loops :

JavaScript Loops are powerful tools for performing repetitive tasks efficiently. Loops in JavaScript execute a block of code again and again while the condition is true.

JavaScript for Loop

The JS for loop provides a concise way of writing the loop structure. The for loop contains initialization, condition, and increment/decrement in one line thereby providing a shorter, easy-to-debug structure of looping.

Syntax

```
for (initialization; testing condition; increment/decrement) {  
    statement(s)  
}
```

Initialization condition: It initializes the variable and mark the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.

Test Condition: It is used for testing the exit condition of a for loop. It must return a boolean value. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.

Statement execution: Once the condition is evaluated to be true, the statements in the loop body are executed.

Increment/ Decrement: It is used for updating the variable for the next iteration.

Loop termination: When the condition becomes false, the loop terminates marking the end of its life cycle.

// JavaScript program to illustrate for loop

let x;

// for loop begins when x = 2

// and runs till x <= 4

```
for (x = 2; x <= 4; x++) {  
    console.log("Value of x: " + x);  
}
```

Output :

```
Value of x: 2  
Value of x: 3  
Value of x: 4
```

JavaScript while Loop

The JS while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax

```
while (boolean condition) {  
    loop statements...  
}
```

- While loop starts with checking the condition. If it is evaluated to be true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason, it is also called the Entry control loop
- Once the condition is evaluated to be true, the statements in the loop body are executed. Normally the statements contain an updated value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle

// JavaScript code to use while loop

```
let val = 1;

while (val < 6) {

    console.log(val);

    val += 1;

}
```

Output:

```
1
2
3
4
5
```

JavaScript do-while Loop

The JS do-while loop is similar to the while loop with the only difference is that it checks for the condition after executing the statements, and therefore is an example of an Exit Control Loop. It executes loop content at least once even the condition is false.

Syntax

```
do {

    Statements...

}
```

while (condition);

- The do-while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements and update of the variable value, the condition is checked for a true or false value. If it is evaluated to be true, the next iteration of the loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked and therefore is an example of the exit control loop.

```
let test = 1;

do {

    console.log(test);

    test++;

} while(test <= 5)
```

Output :

```
1
2
3
4
5
```

JavaScript for-in Loop

JS for-in loop is used to iterate over the properties of an object. The for-in loop iterates only over those keys of an object which have their enumerable property set to “true”.

Syntax

```
for(let variable_name in object_name) {

    // Statement

}

let myObj = { x: 1, y: 2, z: 3 };

for (let key in myObj) {

    console.log(key, myObj[key]);

}
```

Output :

```
x 1
y 2
z 3
```

JavaScript for-of Loop

JS for-of loop is used to iterate the iterable objects for example – array, object, set and map. It directly iterate the value of the given iterable object and has more concise syntax than for loop.

Syntax:

```
for(let variable_name of object_name) {  
    // Statement  
}
```

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

```
for (let value of arr) {  
    console.log(value);  
}
```

Output :

```
1  
2  
3  
4  
5
```

JavaScript Labeled Statement

JS label keyword does not include a goto keyword. Users can use the continue keyword with the label statement. Furthermore, users can use the break keyword to terminate the loop/block. You can also use the break keyword without defining the label but it terminates only the parent loop/block. To terminate the outer loop from the inner loop using the break keyword, users need to define the label.

Syntax

Label:

```
statement (loop or block of code)
```

Example

```
let sum = 0, a = 1;

// Label for outer loop
outerloop: while (true) {

  a = 1;

  // Label for inner loop
  innerloop: while (a < 3) {

    sum += a;

    if (sum > 12) {

      // Break outer loop from inner loop
      break outerloop;

    }

    console.log("sum = " + sum);

    a++;

  }

}
```

Output

```
sum = 1
sum = 3
sum = 4
sum = 6
sum = 7
sum = 9
sum = 10
sum = 12
```

JavaScript Break Statement

JS break statement is used to terminate the execution of the loop or switch statement when the condition is true.

Syntax

```
break;
```

Example

```
for (let i = 1; i < 6; i++) {  
    if (i == 4)  
        break;  
  
    console.log(i);  
}
```

Output

```
1  
2  
3
```

JavaScript Continue Statement

JS continue statement is used to break the iteration of the loop and follow with the next iteration. The break in iteration is possible only when the specified condition going to occur. The major difference between the continue and break statement is that the break statement breaks out of the loop completely while continue is used to break one statement and iterate to the next statement.

Syntax

```
continue;
```

Example

```
for (let i = 0; i < 11; i++) {
```

```
    if (i % 2 == 0)
        continue;

    console.log(i);
}
```

Output

```
1
3
5
7
9
```

JavaScript Infinite Loop (Loop Error)

One of the most common mistakes while implementing any sort of loop is that it may not ever exit, i.e. the loop runs for infinite times. This happens when the condition fails for some reason.

Example: This example shows an infinite loop.

```
// JavaScript program to illustrate infinite loop

// Infinite loop because condition is not false
// condition should have been i>0.
for (let i = 5; i != 0; i -= 2) {
    console.log(i);
}

let x = 5;
```

```
// Infinite loop because update statement
```

```
// is not provided
```

```
while (x == 5) {
```

```
    console.log("In the loop");  
}
```

Conditionals

JavaScript Conditional statements allow you to execute specific blocks of code based on conditions. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.

There are several methods that can be used to perform Conditional Statements in JavaScript.

if statement: Executes a block of code if a specified condition is true.

else statement: Executes a block of code if the same condition of the preceding if statement is false.

else if statement: Adds more conditions to the if statement, allowing for multiple alternative conditions to be tested.

switch statement: Evaluates an expression, then executes the case statement that matches the expression's value.

ternary operator (conditional operator): Provides a concise way to write if-else statements in a single line.

1. Using if Statement

The if statement is used to evaluate a particular condition. If the condition holds true, the associated code block is executed.

Syntax:

```
if ( condition ) {  
    // If the condition is met,  
    //code will get executed.  
}
```

Example: In this example, we are using the if statement to find given number is even or odd.

```
let num = 20;  
if (num % 2 === 0) {  
    console.log("Given number is even number.");  
}
```

```
if (num % 2 !== 0) {  
    console.log("Given number is odd number.");  
};
```

Output

Given number is even number.

Explanation: This JavaScript code determines if the variable `num` is even or odd using the modulo operator `%`. If `num` is divisible by 2 without a remainder, it logs “Given number is even number.” Otherwise, it logs “Given number is odd number.”

2. Using if-else Statement

The if-else statement will perform some action for a specific condition. Here we are using the else statement in which the else statement is written after the if statement and it has no condition in their code block.

Syntax:

```
if (condition1) {  
    // Executes when condition1 is true  
  
    if (condition2) {  
        // Executes when condition2 is true  
    }  
}
```

Example: In this example, we are using if-else conditional statement to check the driving licence eligibility date.

```
let age = 25;  
  
if (age >= 18) {  
    console.log("You are eligible of driving licence")  
} else {  
    console.log("You are not eligible for driving licence")  
};
```


Output

You are eligible of driving licence

Explanation: This JavaScript code checks if the variable `age` is greater than or equal to 18. If true, it logs “You are eligible for a driving license.” Otherwise, it logs “You are not eligible for a driving license.”

This indicates eligibility for driving based on age.

3. else if Statement

The else if statement in JavaScript allows handling multiple possible conditions and outputs, evaluating more than two options based on whether the conditions are true or false.

Syntax:

```
if (1st condition) {  
    // Code for 1st condition  
} else if (2nd condition) {  
    // ode for 2nd condition  
} else if (3rd condition) {  
    // Code for 3rd condition  
} else {  
    // ode that will execute if all  
    // above conditions are false  
}
```

Example: In this example, we are using the above-explained approach.

```
const num = 0;  
  
if (num > 0) {  
    console.log("Given number is positive.");  
} else if (num < 0) {  
    console.log("Given number is negative.");  
} else {  
    console.log("Given number is zero.");  
}
```

```
};
```

Output

Given number is zero.

Explanation: This JavaScript code determines whether the constant `num` is positive, negative, or zero. If `num` is greater than 0, it logs “Given number is positive.” If `num` is less than 0, it logs “Given number is negative.” If neither condition is met (i.e., `num` is zero), it logs “Given number is zero.”

4. Using Switch Statement (JavaScript Switch Case)

As the number of conditions increases, you can use multiple else-if statements in JavaScript. but when we dealing with many conditions, the switch statement may be a more preferred option.

Syntax:

```
switch (expression) {  
    case value1:  
        statement1;  
        break;  
    case value2:  
        statement2;  
        break;  
    ...  
    case valueN:  
        statementN;  
        break;  
    default:  
        statementDefault;  
};
```

Example: In this example, we find a branch name Based on the student's marks, this switch statement assigns a specific engineering branch to the variable Branch. The output displays the student's branch name,

```
const marks = 85;

let Branch;

switch (true) {

  case marks >= 90:

    Branch = "Computer science engineering";

    break;

  case marks >= 80:

    Branch = "Mechanical engineering";

    break;

  case marks >= 70:

    Branch = "Chemical engineering";

    break;

  case marks >= 60:

    Branch = "Electronics and communication";

    break;

  case marks >= 50:

    Branch = "Civil engineering";

    break;

  default:

    Branch = "Bio technology";

    break;

}

console.log(`Student Branch name is : ${Branch}`);
```

Output

Student Branch name is : Mechanical engineering

Explanation:

This JavaScript code assigns a branch of engineering to a student based on their marks. It uses a switch statement with cases for different mark ranges. The student's branch is determined according to their marks and logged to the console.

5. Using Ternary Operator (?:)

The conditional operator, also referred to as the ternary operator (?:), is a shortcut for expressing conditional statements in JavaScript.

Syntax:

condition ? value if true : value if false

Example: In this example, we use the ternary operator to check if the user's age is 18 or older. It prints eligibility for voting based on the condition.

```
let age = 21;
```

```
const result =
```

```
    (age >= 18) ? "You are eligible to vote."
```

```
    : "You are not eligible to vote.";
```

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

Output

You are eligible to vote.

Explanation: This JavaScript code checks if the variable `age` is greater than or equal to 18. If true, it assigns the string "You are eligible to vote." to the variable `result`. Otherwise, it assigns "You are not eligible to vote." The value of `result` is then logged to the console.

Functions:

JavaScript function is a set of statements that take inputs, do some specific computation, and produce output.

or

A function is a block of code that performs a specific task.

A JavaScript function is executed when “something” invokes it (calls it).

Declaring a Function

The syntax to declare a function is:

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

- A function is declared using the function keyword.
- The basic rules of naming a function are similar to naming a variable. It is better to write a descriptive name for your function. For example, if a function is used to add two numbers, you could name the function add or addNumbers.
- The body of function is written within {}.

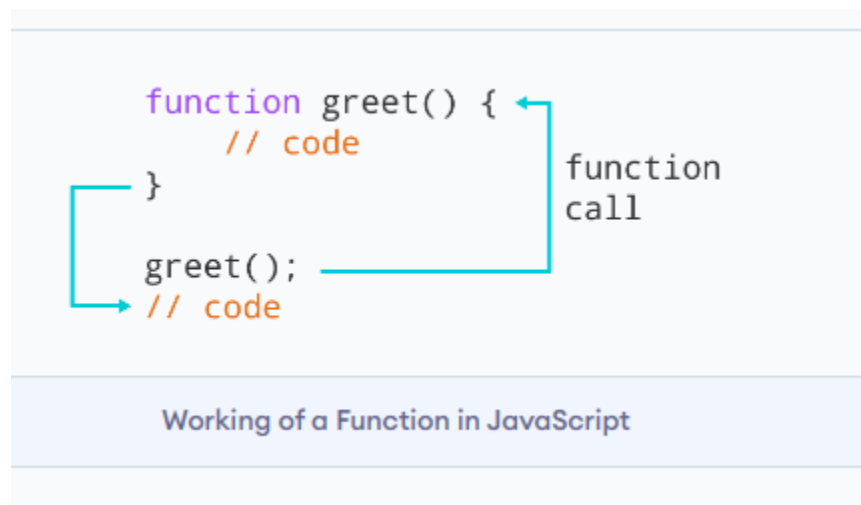
For example,

```
// declaring a function named greet()  
function greet() {  
    console.log("Hello there");  
}
```

Calling a Function

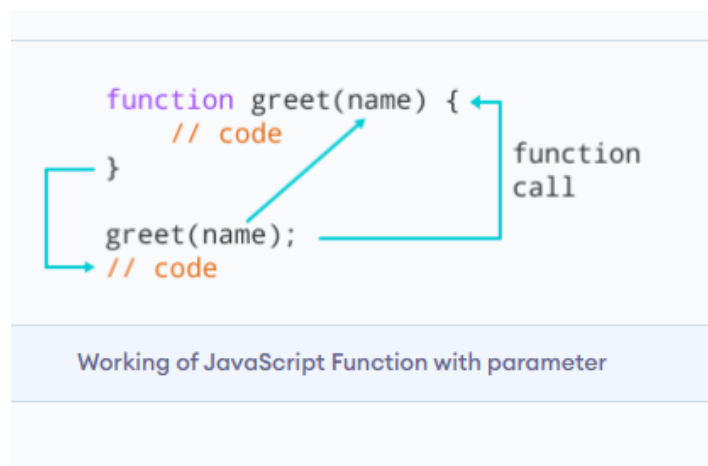
In the above program, we have declared a function named greet(). To use that function, we need to call it. Here's how you can call the above greet() function.

```
// function call  
greet();
```



Function Parameters

A function can also be declared with parameters. A parameter is a value that is passed when declaring a function.



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);
```

[Run Code](#)

Output

```
Enter a name: Simon  
Hello Simon :)
```

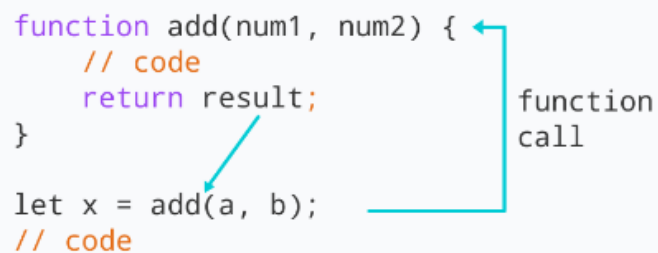
Note: When a value is passed when declaring a function, it is called **parameter**. And when the function is called, the value passed is called **argument**.

Function Return

The return statement can be used to return the value to a function call.

The return statement denotes that the function has ended. Any code after return is not executed.

If nothing is returned, the function returns an undefined value.



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

The diagram illustrates the flow of a function call. A blue arrow points from the function call `add(a, b)` in the line `let x = add(a, b);` to the opening curly brace of the `function add` definition. Another blue arrow points from the `return result;` statement back to the line `let x = add(a, b);`, indicating the return of a value.

Working of JavaScript Function with return statement

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);  
Run Code
```

Output

```
Enter first number: 3.4
Enter second number: 4
The sum is 7.4
```

Benefits of Using a Function

- Function makes the code reusable. You can declare it once and use it multiple times.
- Function makes the program easier as each small task is divided into a function.
- Function increases readability.

Anonymous Function

It is a function that does not have any name associated with it. Normally we use the function keyword before the function name to define a function in JavaScript, however, in anonymous functions in JavaScript, we use only the function keyword without the function name.

An anonymous function is not accessible after its initial creation, it can only be accessed by a variable it is stored in as a function as a value. An anonymous function can also have multiple arguments, but only one expression.

The below-enlightened syntax illustrates the declaration of an anonymous function using the normal declaration:

```
function() {  
    // Function Body  
}
```

We may also declare an anonymous function using the arrow function technique which is shown below:

```
(( ) => {  
    // Function Body...  
})());
```


The below examples demonstrate anonymous functions.

Example 1: In this example, we define an anonymous function that prints a message to the console. The function is then stored in the greet variable. We can call the function by invoking greet().

```
var greet = function () {  
    console.log("Welcome to GeeksforGeeks!");  
};  
  
greet();
```

Output:

Welcome to GeeksforGeeks!

Example 2: In this example, we pass arguments to the anonymous function.

```
var greet = function (platform) {  
    console.log("Welcome to ", platform);  
};
```

```
greet("GeeksforGeeks!");
```

Output:

Welcome to GeeksforGeeks!

As JavaScript supports Higher-Order Functions, we can also pass anonymous functions as parameters into another function.

Example 3: In this example, we pass an anonymous function as a callback function to the setTimeout() method. This executes this anonymous function 2000ms later.

```
setTimeout(function () {  
    console.log("Welcome to GeeksforGeeks!");  
}, 2000);
```

Output:

Welcome to GeeksforGeeks!

Another use case of anonymous functions is **to invoke the function immediately after initialization**, this is also known as Self Executing Function. This can be done by adding parenthesis so we can immediately execute the anonymous function.

Example 4: In this example, we have created a self-executing function.

```
(function () {  
    console.log("Welcome to GeeksforGeeks!");  
})();
```

Output:

Welcome to GeeksforGeeks!

Arrow functions

ES6 introduced a new and shorter way of declaring an anonymous function, which is known as Arrow Functions. In an Arrow function, everything remains the same, except here we don't need the function keyword also. Here, we define the function by a single parenthesis and then '=>' followed by the function body.

Example 5: In this example, we will see the use of arrow function.

```
var greet = () =>  
{  
    console.log("Welcome to GeeksforGeeks!");  
}  
  
greet();
```

Output:

Welcome to GeeksforGeeks!

If we have only a single statement in the function body, we can even remove the curly braces.

Example 6: In this example, we create a self-executing function.

```
let greet = () => console.log("Welcome to GeeksforGeeks!");  
  
greet();
```

Output:

Welcome to Geeksforgeeks!

Example-7: In this example, we will declare a self-executing anonymous function (without the name itself) and will see how we may declare it as well as how we may call it in order to print the resultant value.

```
((() => {  
  
    console.log("GeeksforGeeks");  
  
})());
```

Output:

GeeksforGeeks

Events :

Javascript has events to provide a dynamic interface to a webpage. These events are hooked to elements in the Document Object Model(DOM).

These events by default use bubbling propagation i.e, upwards in the DOM from children to parent. We can bind events either as inline or in an external script.

Syntax:

```
<HTML-element Event-Type = "Action to be performed">
```

Below are a few examples of common events that can happen on a website:

- The page finishes loading
- The user clicks a button
- The user hovers over a dropdown
- The user submits a form
- The user presses a key on their keyboard

Event Handlers and Event Listeners

When a user clicks a button or presses a key, an event is fired. These are called a click event or a keypress event, respectively.

An **event handler** is a JavaScript function that runs when an event fires.

An **event listener** attaches a responsive interface to an element, which allows that particular element to wait and “listen” for the given event to fire.

Event Listeners

The latest addition to JavaScript event handlers are event listeners. An **event listener** watches for an event on an element. Instead of assigning the event directly to a property on the element, we will use the `addEventListener()` method to listen for the event.

`addEventListener()` takes two mandatory parameters — the event it is to be listening for, and the listener callback function.

The HTML for our event listener

Events.html

```
<button>Click me</button>
```

```
<p>I will change.</p>
```

We will be using `changeText()` function here. We'll attach the `addEventListener()` method to the button

Js/events.js file

```
// Function to modify the text content of the paragraph
```

```
const changeText = () => {
```

```
    const p = document.querySelector('p');
```

```
    p.textContent = "I changed because of an event listener.";
```

```
}
```

```
// Listen for click event
```

```
const button = document.querySelector('button');
```

```
button.addEventListener('click', changeText);
```

Mouse Events

Mouse events are among the most frequently used events. They refer to events that involve clicking buttons on the mouse or hovering and moving the mouse pointer. These events also correspond to the equivalent action on a touch device.

Event	Description
click	Fires when the mouse is pressed and released on an element
dblclick	Fires when an element is clicked twice
mouseenter	Fires when a pointer enters an element
mouseleave	Fires when a pointer leaves an element
mousemove	Fires every time a pointer moves inside an element

A `click` is a compound event that is comprised of combined `mousedown` and `mouseup` events, which fire when the mouse button is pressed down or lifted, respectively.

Using `mouseenter` and `mouseleave` in tandem recreates a hover effect that lasts as long as a mouse pointer is on the element.

Form Events

Form events are actions that pertain to forms, such as `input` elements being selected or unselected, and forms being submitted.

Event	Description
<code>submit</code>	Fires when a form is submitted
<code>focus</code>	Fires when an element (such as an input) receives focus
<code>blur</code>	Fires when an element loses focus

Focus is achieved when an element is selected, for example, through a mouse click or navigating to it via the `TAB` key.

JavaScript is often used to submit forms and send the values through to a backend language. The advantage of using JavaScript to send forms is that it does not require a page reload to submit the form, and JavaScript can be used to validate required input fields.

Keyboard Events

Keyboard events are used for handling keyboard actions, such as pressing a key, lifting a key, and holding down a key.

Event	Description
<code>keydown</code>	Fires once when a key is pressed
<code>keyup</code>	Fires once when a key is released
<code>keypress</code>	Fires continuously while a key is pressed

Although they look similar, `keydown` and `keypress` events do not access all the exact same keys. While `keydown` will acknowledge every key that is pressed, `keypress` will omit keys that do not produce a character, such as `SHIFT`, `ALT`, or `DELETE`.

Keyboard events have specific properties for accessing individual keys.

If a parameter, known as an `event` object, is passed through to the event listener, we can access more information about the action that took place. Two properties that pertain to keyboard objects include `key` and `code`.

For example, if the user presses the letter `a` key on their keyboard, the following properties pertaining to that key will surface:

Property	Description	Example
<code>key</code>	Represents the character name	<code>a</code>
<code>code</code>	Represents the physical key being pressed	<code>KeyA</code>

To show how to gather that information via the JavaScript Console, we can write the following lines of code.

```
// Test the key and code properties
document.addEventListener('keydown', event => {
  console.log('key: ' + event.key);
  console.log('code: ' + event.code);
});
```

Copy

Once we press `ENTER` on the Console, we can now press a key on the keyboard, in this example, we'll press `a`.

```
Output
key: a
code: KeyA
```

The `key` property is the name of the character, which can change — for example, pressing `a` with `SHIFT` would result in a `key` of `A`. The `code` property represents the physical key on the keyboard.

Form validation :

Form validation refers to the process of ensuring that user input submitted through a form meets certain criteria or constraints before it is accepted and processed by a computer system.

The data entered into a form needs to be in the right format and certain fields need to be filled in to effectively use the submitted form. Username, password, and contact information are some details that are mandatory in forms and thus need to be provided by the user.

```
<body>
```

```
<script>
```

```
function validateform(){
```

```
var name=document.myform.name.value;
```

```
var password=document.myform.password.value;
```

```
if (name==null || name==""){
```

```
    alert("Name can't be blank");
```

```
    return false;
```

```
22
```

```
    }else if(password.length<6){
```

```
        alert("Password must be at least 6 characters
```

```
long.");
```

```
        return false;
```

```
    }
```

```
}
```

```
</script>
```

```
<body>
```

```
<form name="myform" method="post"
```

```
action="http://www.javatpoint.com/javascriptpag
```

```
es/valid.jsp" onsubmit="return validateform()"
```


>

Name: <input type="text" name="name">

Password: <input type="password"

name="password">

<input type="submit" value="register">

</form>

</body>

Ajax:

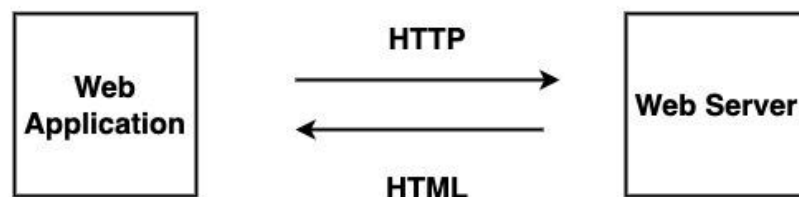
AJAX stands for asynchronous Javascript and XML. AJAX is not a programming language or technology, but it is a combination of multiple web-related technologies like HTML, XHTML, CSS, JavaScript, DOM, XML, XSLT and XMLHttpRequest object. The AJAX model allows web developers to create web applications that are able to dynamically interact with the user. It will also be able to quickly make a background call to web servers to retrieve the required application data. Then update the small portion of the web page without refreshing the whole web page.

AJAX applications are much more faster and responsive as compared to traditional web applications. It creates a great balance between the client and the server by allowing them to communicate in the background while the user is working in the foreground.

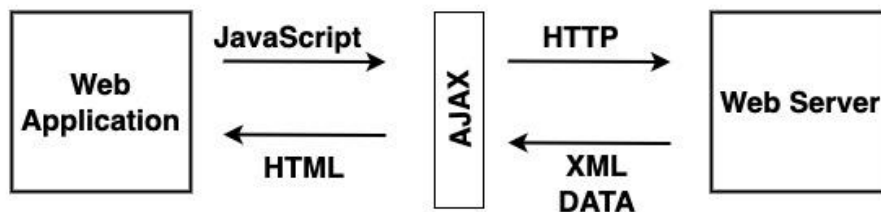
In the AJAX applications, the exchange of data between a web browser and the server is asynchronous means AJAX applications submit requests to the web server without pausing the execution of the application and can also process the requested data whenever it is returned. For example, Facebook uses the AJAX model so whenever we like any post the count of the like button increase instead of refreshing the whole page.

Working of AJAX

Traditional web applications are created by adding loosely web pages through links in a predefined order. Where the user can move from one page to another page to interact with the different portions of the applications. Also, HTTP requests are used to submit the web server in response to the user action. After receiving the request the web server fulfills the request by returning a new webpage which, then displays on the web browser. This process includes lots of pages refreshing and waiting.



AJAX change this whole working model by sharing the minimum amount of data between the web browser and server asynchronously. It speedup up the working of the web applications. It provides a desktop-like feel by passing the data on the web pages or by allowing the data to be displayed inside the existing web application. It will replace loosely integrated web pages with tightly integrated web pages. AJAX application uses the resources very well. It creates an additional layer known as AJAX engine in between the web application and web server due to which we can make background server calls using JavaScript and retrieve the required data, can update the requested portion of a web page without causing full reload of the page. It reduces the page refresh timing and provides a fast and responsive experience to the user. Asynchronous processes reduce the workload of the web server by dividing the work with the client computer. Due to the reduced workload web servers become more responsive and fast.



AJAX Technologies

The technologies that are used by AJAX are already implemented in all the Modern browsers. So the client does not require any extra module to run the AJAX application. The technologies used by AJAX are

Javascript – It is an important part of AJAX. It allows you to create client-side functionality. Or we can say that it is used to create AJAX applications.

XML – It is used to exchange data between web server and client.

The XMLHttpRequest – It is used to perform asynchronous data exchange between a web browser and a web server.

HTML and CSS – It is used to provide markup and style to the webpage text.

DOM – It is used to interact with and alter the webpage layout and content dynamically.

Advantages of AJAX

The following are the advantages of AJAX –

- It creates responsive and interactive web applications.
- It supports the development of patterns and frameworks that decrease the development time.
- It makes the best use of existing technology and feature instead of using some new technology.
- It makes an asynchronous call to the web server which means the client doesn't have to wait for the data to arrive before starting rendering.

What are AJAX use cases?

You can use AJAX to create various features in web applications.

Autocomplete

Search engines provide autocomplete options in real time when users search for a specific keyword in the search bar. AJAX allows the webpage to relay each character input to the web server and return a list of relevant recommendations on the existing page.

Form verification

AJAX allows web applications to validate specific information in forms before users submit them. For example, when a new user creates an account, the webpage can automatically verify if a username is available before the user moves to the next section.

Chat functionality

Text messengers and chatbots use AJAX to display real-time conversations on browsers. AJAX sends the text written by a user to the server and publishes it simultaneously in other users' chat interfaces.

Social media

Social media platforms use AJAX to update users' feeds with the latest content without loading a new page on the browser. For example, Twitter refreshes your feed immediately whenever someone you follow tweets an update.

Voting and rating systems

Some forums and social bookmarking sites use AJAX to display the rating or votes of specific posts in real time. For example, you can upvote or downvote a post on Reddit without refreshing the entire page.

Using the XMLHttpRequest object

In this approach, we will use the XMLHttpRequest object to make an Ajax call. The XMLHttpRequest() method creates an XMLHttpRequest object which is used to make a request with the server.

Syntax:

```
let xhttp = new XMLHttpRequest();
```

Above syntax is used to create an XMLHttpRequest object. This object has many different methods used to interact with the server to send, receive or interrupt responses from the server. In the response, we get a string from the server that we print.

Example: In this example, we will use the [XMLHttpRequest object](#) to make an Ajax call.

- Javascript

```
function run() {  
  
    // Creating Our XMLHttpRequest object  
  
    let xhr = new XMLHttpRequest();  
  
    // Making our connection  
  
    let url = 'https://jsonplaceholder.typicode.com/todos/1';  
  
    xhr.open("GET", url, true);  
  
    // function execute after request is successful  
  
    xhr.onreadystatechange = function () {  
  
        if (this.readyState == 4 && this.status == 200) {  
  
            console.log(this.responseText);  
  
        }  
  
    }  
  
    // Sending our request  
  
    xhr.send();  
  
}  
  
run();
```

Output:

```
"{"  
  "userId": 1,  
  "id": 1,  
  "title": "delectus aut autem",  
  "completed": false  
}"
```

jQuery:

Selectors

What is jQuery?

jQuery is a lightweight, "write less, do more", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

jQuery Syntax

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: `$(selector).action()`

- A \$ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

Examples:

- `$(this).hide()` - hides the current element.
- `$("p").hide()` - hides all <p> elements.

jQuery Selectors

jQuery selectors allow you to select and manipulate HTML element(s).

jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing [CSS Selectors](#), and in addition, it has some own custom selectors.

All selectors in jQuery start with the dollar sign and parentheses: `$()`.

The element Selector

The jQuery element selector selects elements based on the element name.

You can select all `<p>` elements on a page like this:

```
$("p")
```

Example

When a user clicks on a button, all `<p>` elements will be hidden:

Example

```
$(document).ready(function(){
  $("button").click(function(){
    $("p").hide();
  });
})
```

The #id Selector

The jQuery `#id` selector uses the id attribute of an HTML tag to find the specific element.

An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

To find an element with a specific id, write a hash character, followed by the id of the HTML element:

```
$("#test")
```

Example

When a user clicks on a button, the element with `id="test"` will be hidden:

Example

```
$(document).ready(function(){
    $("button").click(function(){
        $("#test").hide();
    });
});
```

The .class Selector

The jQuery *.class* selector finds elements with a specific class.

To find elements with a specific class, write a period character, followed by the name of the class:

```
$(".test")
```

Example

When a user clicks on a button, the elements with class="test" will be hidden:

Example

```
$(document).ready(function(){
    $("button").click(function(){
        $(".test").hide();
    });
});
```

Mouse events :

Mouse Events in jQuery

The Mouse Event represents events that occur due to the user interacting with a pointing device (such as a mouse). Common events using this interface include click, double click, mouse, mouse enter, mouse out, etc.

If you want to attach any handler with these mouse events, then jQuery provides a bunch of mouse event handling methods. In this article, we will learn about these mouse events.

List of jQuery Mouse Events

Here is the list of mouse events used in jQuery.

click(): The event occurs when the user clicks on an element

contextmenu(): The event occurs when the user right-clicks on an element to open a context menu

dblclick(): The event occurs when the user double-clicks on an element

hover(): attach one or two handlers to the matched elements, to be executed when the mouse pointer enters and leaves the elements.

mouseenter(): The event occurs when the pointer is moved onto an element

mouseleave(): The event occurs when the pointer is moved out of an element

mouseover(): The event occurs when the pointer is moved onto an element, or onto one of its children

mouseout(): The event occurs when a user moves the mouse pointer out of an element, or out of one of its children

Syntax:

All the events take a handler function as parameter for example,

`$(selector).mouseenter(function(){ })` or

`$(selector).mouseleave(function(){ })`

But `hover()` method can take two handlers as parameters. The first one is for mouse enter and the second one is for mouse leave

`$(selector).hover(handlerIn, handlerOut)`

Actually, this is the shorthand for `$(selector).mouseenter(handlerIn).mouseleave(handlerOut);`

Example: jQuery Mouse Click Event

Now by some examples, we will see that how to attach event handlers to these events. Let's see the click event first. In the below example, we have `<div>` element. When we will click on the div notice at the change i.e. it will append the text You clicked the div as well as also change the background color of the div element to yellow.

```
<html>
<head>
<title>jQuery Mouse Events</title>
</head>
<body>
<h2>jQuery Mouse Events</h2>
<section>
<div>
```



```

<h3>This is a Title</h3>
<p>
Lorem ipsum dolor, sit amet consectetur adipisicing elit. Nisi,
laudantium.
</p>
</div>
</section>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
<script>
$(document).ready(function () {
$("div").click(function () {
$(this).css("background-color", "yellow");
$("div p").append("<br>" + "You clicked the div");
});
});
</script>
</body>
</html>

```

Now, run the above code and you will see the following page in the browser. Now, click anywhere between the div element which shown in between the red color as shown in the below image.



Once you click anywhere between the div element, then the text “You clicked the div” is appended as well as the background color of the div element is changed to yellow as shown in the below image

jQuery Mouse Events

This is a Title

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Nisi, laudantium.
You clicked the div

Example DB click Mouse Event

To bind a dblclick event with <div> where an alert box is displayed whenever you double click on any of the divs.

```
<!doctype html>

<html>

<head>

<title>The jQuery Example</title>

<script src="https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>

<script>

$(document).ready(function() {

$("div").dblclick(function(){

alert('Hi there!');

});

});

</script>

<style>

div{ margin:10px;padding:12px; border:2px solid #666;

width:60px;cursor:pointer}

</style>
```

```
</head>

<body>

<p>Double click on any of the squares to see the result:</p>

<div>One</div>

<div>Two</div>

<div>Three</div>

</body>

</html>
```

3. Example Mouse enter Event example

Example:

To bind a mouse enter event with <div> where an alert box is displayed whenever you bring cursor over any of the divs.

```
<!doctype html>

<html>

<head>

<title>The jQuery Example</title>

<script src="https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>

<script>

32

$(document).ready(function() {

$("div").mouseenter(function(){

alert('Cursor is in!');

});
```

```
});  
  
</script>  
  
<style>  
  
div{ margin:10px;padding:12px; border:2px solid #666;  
  
width:60px;cursor:pointer}  
  
</style>  
  
</head>  
  
<body>  
  
<p>Bring cursor over any of the squares to see the  
result:</p>  
  
<div>One</div>  
  
<div>Two</div>  
  
<div>Three</div>  
  
</body>  
  
</html>
```

4. Mouse leave Event

To bind a mouse leave event with <div> where an alert box is displayed whenever you take cursor out of the div.

Example:

```
<!doctype html>  
  
<html>  
  
<head>  
  
<title>The jQuery Example</title>  
  
<script src="https://www.tutorialspoint.com/jquery/jquery-
```

```
3.6.0.js"></script>
```

```
<script>
```

```
$(document).ready(function() {
```

```
$("#div").mouseleave(function(){
```

```
alert('Curosr is out!');
```

```
});
```

```
});
```

```
</script>
```

```
<style>
```

```
div{
```

```
margin:10px;
```

```
padding:12px;
```

```
border:2px solid #666;
```

```
width:60px;
```

```
cursor: pointer;
```

```
}
```

```
</style>
```

```
33
```

```
</head>
```

```
<body>
```

```
<p>Take cursor out any of the squares to see the result:</p>
```

```
<div>One</div>
```

```
<div>Two</div>
```

```
<div>Three</div>
```

```
</body>
```

```
</html>
```

5. Mouse Down Event

To bind a mouse down event with `<div>` where an alert box is displayed whenever the left, middle or right mouse button is pressed down over any of the divs.

Example:

```
<!doctype html>
```

```
<html>
```

```
<head>
```

```
<title>The jQuery Example</title>
```

```
<script src="https://www.tutorialspoint.com/jquery/jquery-3.6.0.js"></script>
```

```
<script>
```

```
$(document).ready(function() {
```

```
  $("div").mousedown(function(){
```

```
    alert('Mouse button is down!');
```

```
  });
```

```
});
```

```
</script>
```

```
<style>
```

```
div{ margin:10px;
```

```
padding:12px;
```

```
border:2px solid #666;
```

```
width:60px;
```

```
cursor:pointer}

</style>

</head>

<body>

<p>Press mouse button down over any of the squares to see the
result:</p>

<div>One</div>

<div>Two</div>

<div>Three</div>

</body>

</html>
```

34

6. Mouse up Event

To bind a mouse up event with <div> where an alert box is displayed whenever the left, middle or right mouse button is released over any of the divs.

Example:

```
<!doctype html>

<html>

<head>

<title>The jQuery Example</title>

<script src="https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>

<script>

$(document).ready(function() {
```

```
$("#div").mouseup(function(){
    alert('Mouse button is released!');
});

});

</script>

<style>

div{ margin:10px;padding:12px; border:2px solid #666;
width:60px;cursor:pointer}

</style>

</head>

<body>

<p>Release mouse button over any of the squares to see the
result:</p>

<div>One</div>

<div>Two</div>

<div>Three</div>

</body>

</html>
```

Form Events:

A The HTML Form Element interface symbolizes a form element within the DOM, offering access and modification options for various form aspects and components. Form events are standard JavaScript events, augmented by jQuery for added support. They're triggered when form controls gain or lose focus, or when users modify control values, such as typing in a text input or selecting an option in a dropdown

Examples:

1. Submit
2. Select
3. Focus
4. Blur

1. submit () event

The submit event is a commonly used form event, second in popularity to the change event. By using the submit () method, you can attach an event handler that executes a function when the submit event is triggered. This event takes place whenever a form is submitted, enabling you to manage actions like form validation or data processing before the form is actually submitted to a server.

Example:

```
<html>

<head>

<title>jQuery Submit</title>

<script

src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.m

in.js"></script>

<script type="text/javascript">

$(document).ready(function(){

$("#myForm").submit(function () {

alert("Form submitted");

});
```

```
});  
  
</script>  
  
</head>  
  
<body>  
  
<form id="myForm">  
  
<input name="submit" id="submit" type="submit" />  
  
</form>  
  
</body>  
  
</html>
```

2. Select Event

In jQuery, there is no `select()` method that corresponds to a "select" event. Instead, the `select()` method in jQuery is used to select elements in the DOM based on a given selector. The "select" event in jQuery is used to capture the selection of text within an input or text area element. This event is triggered when a user highlights or selects text within the input or textarea using their mouse or keyboard. Here's an example of how you might use the "select" event in jQuery:

Example:

```
<html>  
  
<head>  
  
<title>jQuery</title>  
  
<script  
  
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.m  
  
in.js"></script>  
  
<script type="text/javascript">  
  
$(document).ready(function(){
```

```
$("#select").change(function(){  
  
    var selectedColor = $(this).find(":selected").val();  
  
    alert("You have selected - " + selectedColor);  
  
});  
  
});  
  
</script>  
  
</head>  
  
<body>  
  
<select>  
  
<option>Red</option>  
  
<option>Blue</option>  
  
<option>Green</option>  
  
</select>  
  
</body>  
  
</html>
```

3. Focus Event

The focus() method in jQuery serves two purposes:

Triggering the Focus Event: If you call focus() on an element, it triggers the focus event for that element. This event occurs when an element gains focus, usually due to user interaction, like clicking on an input field.

Attaching a Function to the Focus Event: You can also use focus() to attach a function that runs when the focus event is triggered for the specified element. This function executes when the element gains focus, allowing you to perform actions or validations at that point.

Example:

```
<html>

<head>

<title>jQuery</title>

<script

src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.m

in.js"></script>

<script type="text/javascript">

$(document).ready(function(){

$("input").focus(function(){

$(this).css('background-color', 'pink');

});

});

</script>

</head>

<body>

Enter: <input type="text" size="30"/>

</body>

</html>
```

Dom Manipulation :

The Document Object Model (DOM) is a programming interface for HTML and XML (Extensible Markup Language) documents. It specifies the logical structure of documents as well as how they are accessed and manipulated. DOM is a method of representing a webpage in a structured and systematic order, making it easier for programmers and users to navigate the document. Using the Document object's commands or methods, we can easily access and manipulate HTML tags, IDs,

classes, Attributes, or Elements. DOM provides JavaScript with access to the HTML and CSS of the web page, as well as the ability to add actions to HTML elements. So, in simple terms, the Document Object Model is an API that represents and interacts with HTML or XML documents.

Properties of DOM

Let's take a look at the document object's properties that can be accessed and modified.

Window Object: A window object is a browser object that is always at the top of the hierarchy. It is similar to an API in that it is used to set and access all of the browser's properties and methods. The browser generates it automatically.

Document object: A document object is created when an HTML document is loaded into a window. The 'document' object has several properties that refer to other objects that allow access to and modification of the web page's content. If we need to access any element in an HTML page, we always begin with the 'document' object. The document object is a window object property.

Form Object: It is defined by form tags.

Link Object: Link tags are used to describe it.

Anchor Object: A href tag is used to represent it.

Form Control Elements: A form may contain a number of control elements, including text fields, buttons, radio buttons, checkboxes, and more.

The following table lists some important methods to add/remove new DOM elements.

Method	Description
append()	Inserts content to the end of element(s) which is specified by a selector.
before()	Inserts content (new or existing DOM elements) before an element(s) which is specified by a selector.
after()	Inserts content (new or existing DOM elements) after an element(s) which is specified by a selector.
prepend()	Insert content at the beginning of an element(s) specified by a selector.
remove()	Removes element(s) from DOM which is specified by selector.
replaceAll()	Replace target element(s) with specified element.
wrap()	Wrap an HTML structure around each element which is specified by selector.

1. after() Method:

The jQuery after() method inserts content (new or existing DOM elements) after target element(s) which is specified by a selector. First of all, specify a selector to get the reference of target element(s) after which you want to add the content and then call after() method. Pass the content string as a parameter. Content string can be any valid HTML element.

Example:

```
<!DOCTYPE html>

<html>

<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">

</script>

<script>

$(document).ready(function () {

$('#div1').after('<div style="background-color:yellow"> New div </div>');

});
```

```
</script>

<style>

div{

border: 1px solid;

background-color:red;

margin: 2px 0 2px 0;

}

</style>

</head>

<body>

<h1>Demo: jQuery after() method </h1>

<div id="div1">div 1

</div>

<div id="div2">div 2

</div>

</body>

</html>
```

2. Before Method()

The jQuery before() method inserts content (new or existing DOM elements) before target element(s) which is specified by a selector. Specify a selector to get the reference of target element(s) before which you want to add the content and then call before() method. Pass the content string that can be any valid HTML element as parameter.

Example:

```
<!DOCTYPE html>

<html>

<head>
```

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
</script>
<script>
$(document).ready(function(){
$('#div1').before('<div style="background-color:yellow"> New div </div>');
});
</script>
<style>
div{
border: 1px solid;
background-color:red;
margin: 2px 0 2px 0;
}
</style>
</head>
<body>
<h1>Demo: jQuery before() method </h1>
<div id="div1">div 1 </div>
<div id="div2">div 2</div>
</body>
</html>
```

3. append() Method

The jQuery append() method inserts content to the end of target element(s) which is specified by a selector. First specify a selector expression to get the reference of an element(s) to which you want to append content, then call append() method and pass content string as a parameter.

Example:

```
<!DOCTYPE html>

<html>

<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">

</script>

<script>

$(document).ready(function () {

$('p').append('World!');

});

</script>

</head>

<body>

<h1>Demo: jQuery append() method </h1>

<p>Hello </p>

</body>

</html>
```

4. prepend() Method

The jQuery prepend() method inserts content at the beginning of an element(s) specified by a selector. First specify a selector expression to get the reference of an element(s) to which you want to prepend the content, then call prepend() method and pass content string as a parameter.

Example:

```
<!DOCTYPE html>

<html>

<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"> </script>
```

```
<script>

$(document).ready(function () {

$('p').prepend('World');

});

</script>

</head>

<body>

<h1>Demo: jQuery prepend() method </h1>

<p>Hello</p>

</body>

</html>
```

41

5. remove() Method

The jQuery remove() method removes element(s) as specified by a selector. First specify a selector expression to get the reference of an element(s) which you want to remove from the document and then call

remove() method.

Syntax:

```
<!DOCTYPE html>

<html>

<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">

</script>

<script>

$(document).ready(function () {

$('label').remove ();

});
```

```
</script>
</head>
<body>
<h1>Demo: jQuery remove() method </h1>
<div>This is div.
<label>This is label.</label>
</div>
</body>
</html>
```

6. replace All() Method

The jQuery replaceAll() method replaces all target elements with specified element(s). Here, syntax is different. First specify a content string as replacement element(s) and then call replaceAll() method with selector expression to specify a target element(s).

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
</script>
<script>
$(document).ready(function () {
$(<span>This is span</span>').replaceAll('p');
});
</script>
</head>
<body>
```

```
<h1>Demo: jQuery replaceAll() method </h1>
```

```
<div>
```

```
<p>This is paragraph.</p>
```

```
</div>
```

```
<p>This is another paragraph.</p>
```

```
</body>
```

```
</html>
```

Effects & Animation :

jQuery effects add an X factor to your website interactivity. jQuery provides a trivially simple interface for doing various kind of amazing effects like show, hide, fade-in, fade-out, slide-up, slide-down, toggle etc. jQuery methods allow us to quickly apply commonly used effects with a minimum configuration. This tutorial covers all the important jQuery methods to create visual effects.

jQuery Effect - Hiding Elements

jQuery gives simple syntax to hide an element with the help of hide() method:

```
$(selector).hide( [speed, callback] );
```

You can apply any jQuery selector to select any DOM element and then apply jQuery hide() method to hide it. Here is the description of all the parameters which gives you a solid control over the hiding effect

speed – This optional parameter represents one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

callback – This optional parameter represents a function to be executed whenever the animation completes; executes once for each element animated against.

Example:

```
<!doctype html>
```

```
<html>
```

```
<head>
```

```
<title>The jQuery Example</title>
```

```
<script src="https://www.tutorialspoint.com/jquery/jquery-3.6.0.js"></script>
```

```
<script>
```

```

$(document).ready(function() {

    $("div").click(function(){

        $(this).hide(1000);

    });

});

</script>

<style>

    div{ margin:10px;padding:12px; border:2px solid #666; width:60px; cursor:pointer}

</style>

</head>

<body>

    <p>Click on any of the squares to see the result:</p>

    <div>Hide Me</div>

    <div>Hide Me</div>

    <div>Hide Me</div>

</body>

</html>

```

jQuery Effect - Show Elements

jQuery gives simple syntax to show a hidden element with the help of show() method:

```
$(selector).show( [speed, callback] );
```

```

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>show demo</title>

```

```
<style>

div {

background: #def3ca;

margin: 3px;

width: 80px;

display: none;

float: left;

text-align: center;

}

</style>

<script src="https://code.jquery.com/jquery-3.7.0.js"></script>

</head>

<body>

<button id="showr">Show</button>

<button id="hidr">Hide</button>

<div>Hello 3,</div>

<div>how</div>

<div>are</div>

<div>you?</div>

<script>

$( "#showr" ).on( "click", function() {

$( "div" ).first().show( "fast", function showNext() {

$( this ).next( "div" ).show( "fast", showNext );

});

});

});
```

jQuery Effect - Toggle Elements

jQuery provides toggle() methods to toggle the display state of elements between revealed or hidden. If the element is initially displayed, it will be hidden; if hidden, it will be shown.

```
$(selector).toggle( [speed, callback] );
```

Example:

```
<!doctype html>

<html>

<head>

<title>The jQuery Example</title>

<script src="https://www.tutorialspoint.com/jquery/jquery-3.6.0.js"></script>

<script>

    $(document).ready(function() {

        $("#button").click(function(){

            $("#box").toggle(1000);

        });

    });

</script>

<style>

    button{ margin:3px;width:125px;cursor:pointer;}

    #box{ margin:3px;padding:12px; height:100px; width:100px;background-color:#9c9cff;}

</style>

</head>

<body>

    <p>Click on the Toggle Box button to see the box toggling:</p>

    <div id="box">This is Box</div>

    <button id="button">Toggle Box</button>

</body>
```

```
</html>
```

jQuery Effect - Fading Elements

jQuery gives us two methods - `fadeIn()` and `fadeOut()` to fade the DOM elements in and out of visibility.

```
$(selector).fadeIn( [speed, callback] );
```

```
$(selector).fadeOut( [speed, callback] );
```

The jQuery `fadeIn()` method is used to fade in a hidden element where as `fadeOut()` method is used to fade out a visible element

```
<html>
```

```
<head>
```

```
<title>The jQuery Example</title>
```

```
<script src="https://www.tutorialspoint.com/jquery/jquery-3.6.0.js"></script>
```

```
<script>
```

```
$(document).ready(function() {
```

```
    $("#show").click(function(){
```

```
        $("#box").fadeIn(1000);
```

```
    });
```

```
    $("#hide").click(function(){
```

```
        $("#box").fadeOut(1000);
```

```
    });
```

```
});
```

```
</script>
```

```
<style>
```

```
    button{ cursor:pointer; }
```

```
    #box{ margin-bottom:5px;padding:12px;height:100px; width:150px; background-color:#9c9cff; }
```

```
</style>
```

```
</head>
```

```
<body>
```


<p>Click on fadeOut and fadeIn buttons to see the result:</p>

<div id="box">This is Box</div>

<button id="hide">fadeOut Box</button>

<button id="show">fadeIn Box</button>

</body>

</html>

jQuery Effect - Sliding Elements

jQuery gives us two methods - **slideUp()** and **slideDown()** to slide up and slide down the DOM elements respectively. Following is the simple syntax for these two methods:

```
$(selector).slideUp( [speed, callback] );  
$(selector).slideDown( speed, [callback] );
```

The jQuery **slideUp()** method is used to slide up an element where as **slideDown()** method is used to slide down

<!doctype html>

<html>

<head>

<title>The jQuery Example</title>

<script src="https://www.tutorialspoint.com/jquery/jquery-3.6.0.js"></script>

<script>

```
$(document).ready(function() {  
    $("#show").click(function(){  
        $("#box").slideDown(1000);  
    });  
    $("#hide").click(function(){  
        $("#box").slideUp(1000);
```

```
    });  
  
});  
  
</script>  
  
<style>  
  
    button{cursor:pointer;}  
  
    #box{margin-bottom:5px;padding:12px;height:100px; width:120px; background-color:#9c9cff;}  
  
</style>  
  
</head>  
  
<body>  
  
    <p>Click on slideUp and slideDown buttons to see the result:</p>  
  
  
    <div id="box">This is Box</div>  
  
    <button id="hide">slideUp </button>  
  
    <button id="show">slideDown </button>  
  
</body>  
  
</html>
```

jQuery Animations - The animate() Method

The jQuery animate() method is used to create custom animations.

Syntax:

```
$(selector).animate({params},speed,callback);
```

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.

The following example demonstrates a simple use of the `animate()` method; it moves a `<div>` element to the right, until it has reached a left property of 250px:

Example

```
$("#button").click(function(){  
  
    $("#div").animate({left: '250px'});  
  
});
```

Traversing & Filtering

DOM traversing using jQuery is used to find (or select) HTML elements based on their relationship to other elements. Begin with one option and work your way through it until you reach the desired elements.

An HTML page is depicted as a tree in the image below (DOM tree). You can easily move up(ancestors), down(descendants), and sideways(siblings) in the tree using jQuery traversing, starting from the selected (current) element. This is known as traversing - or moving through - the DOM tree.

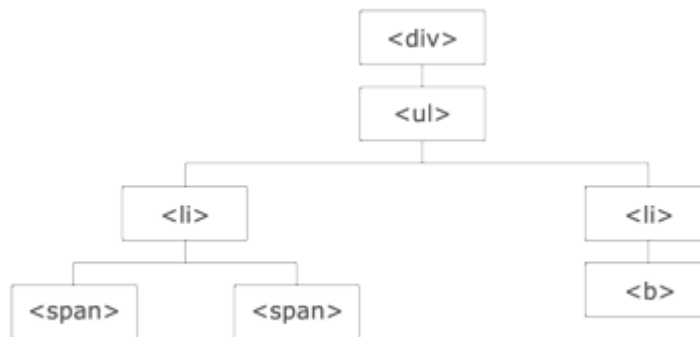


Illustration explained:

The `<div>` element is the parent of ``, and an ancestor of everything inside of it

The `` element is the parent of both `` elements, and a child of `<div>`

The left `` element is the parent of ``, child of `` and a descendant of `<div>`

The `` element is a child of the left `` and a descendant of `` and `<div>`

The two `` elements are siblings (they share the same parent)

The right `` element is the parent of ``, child of `` and a descendant of `<div>`

The `` element is a child of the right `` and a descendant of `` and `<div>`

Note :

An ancestor is a parent, grandparent, great-grandparent, and so on.

A descendant is a child, grandchild, great-grandchild, and so on.

Siblings share the same parent.

Traversing the DOM Tree

jQuery provides a number of methods for traversing the DOM.

Tree traversal is the most common type of traversal method. An ancestor is a parent, grandparent, great-grandparent, and so on in logical relationships.

jQuery provides useful methods such as `parent()`, `parents()`, and `parentsUntil()` that can be used to traverse the DOM tree on single or multiple levels to quickly get the parent or other ancestors of an element in the hierarchy.

The majority of DOM Traversal Methods do not modify the jQuery DOM object and are used to filter out elements from a document based on specified conditions.

jQuery provides methods for traversing in three directions:

Traversing Upwards - This path leads to the ancestors (Parent, Grandparent, Great-grandparent, etc.).

Traversing Downwards entails traversing the descendants in this direction (Child, Grandchild, Great-grandchild, etc.).

Sideways - Traveling in this direction means passing through the ancestors and siblings (For example, brother and sisters are at the same level).

jQuery DOM Traversing Methods

Ancestors:

`parent()`

it gives parent element of specified selector

Syntax:

- `$(selector).parent();`

- **parents()**
it gives all ancestor elements of the specified selector.

Syntax:

```
$(selector).parents();
```

- **parentsUntil()**
it gives all ancestor elements between specified selector and arguments.
Syntax:

- `$(selector).parentsUntil(selector, filter element)`
- `$(selector).parentsUntil(element, filter element)`

- **offsetParent()**

it gives the first positioned parent element of specified selector.

Syntax:

```
$(selector).offsetParent();
```

- **closest()**

it gives the first ancestor of the specified selector.

Syntax:

```
$(selector).closest(selector);
$(selector).closest(selector, context);
$(selector).closest(selection);
$(selector).closest(element);
```

Descendants:

- **children()**

it gives the children of each selected elements, optionally filtered by a selector.

Syntax:

- `$(selector).children();`

- **find()**

it gives descendant elements of specified elements, filtered by a selector, jQuery object, or element.

Syntax:

```
$(selector).find('selector to find');
```

Siblings:

- **siblings()**

it gives all siblings of the specified selector.

Syntax:

```
$(selector).siblings();
```

- **next()**

it gives the next sibling element of the specified selector.

Syntax:

- `$(selector).next();`

- **nextAll()**

it gives all next sibling elements of the specified selector.

Syntax:

- `$(selector).nextAll();`

- **nextUntil()**

it gives all next sibling elements between specified selector and arguments.

Syntax:

- `$(selector).nextUntil();`

- **prev()**

it gives the previous sibling element of the specified selector.

Syntax:

- `$(selector).prev(selector);`

- `$(selector).prev()`
- **prevAll()**
it gives all previous sibling elements of the specified selector.
Syntax:
 - `$(selector).prevAll(selector, filter element)`
 - `$(selector).prevAll(element, filter element)`
- **prevUntil()**
it gives all previous sibling elements between specified selector and arguments.
Syntax:
 - `$(selector).prevUntil(selector, filter element)`
 - `$(selector).prevUntil(element, filter element)`

Query DOM Filtering Methods

To narrow the search for elements in a DOM tree, jQuery provides several methods such as `filter()`, `first()`, `last()`, `eq()`, `slice()`, `has()`, `not()`, and so on

Ancestors:

- `parent()`

it gives parent element of specified selector

Syntax:

`$(selector).parent();`

- `parents()`

it gives all ancestor elements of the specified selector.

Syntax:

`$(selector).parents();`

- `parentsUntil()`

it gives all ancestor elements between specified selector and arguments.

Syntax:

`$(selector).parentsUntil(selector, filter element)`

`$(selector).parentsUntil(element, filter element)`

- `offsetParent()`

it gives the first positioned parent element of specified selector.

Syntax:

```
$(selector).offsetParent();
```

- `closest()`

it gives the first ancestor of the specified selector.

Syntax:

```
$(selector).closest(selector);
```

```
$(selector).closest(selector, context);
```

```
$(selector).closest(selection);
```

```
$(selector).closest(element);
```

Descendants:

- `children()`

it gives the children of each selected elements, optionally filtered by a selector.

Syntax:

```
$(selector).children();
```

- `find()`

it gives descendant elements of specified elements, filtered by a selector, jQuery object, or element.

Syntax:

```
$(selector).find('selector to find');
```

Siblings:

- `siblings()`

it gives all siblings of the specified selector.

Syntax:

```
$(selector).siblings();
```

- `next()`

it gives the next sibling element of the specified selector.

Syntax:

```
$(selector).next();
```

- `nextAll()`

it gives all next sibling elements of the specified selector.

Syntax:

```
$(selector).nextAll();
```

- `nextUntil()`

it gives all next sibling elements between specified selector and arguments.

Syntax:

```
$(selector).nextUntil();
```

- `prev()`

it gives the previous sibling element of the specified selector.

Syntax:

```
$(selector).prev(selector);
```

```
$(selector).prev()
```

- `prevAll()`

it gives all previous sibling elements of the specified selector.

Syntax:

```
$(selector).prevAll(selector, filter element)
```

```
$(selector).prevAll(element, filter element)
```


- `prevUntil()`

it gives all previous sibling elements between specified selector and arguments.

Syntax:

```
$(selector).prevUntil(selector, filter element)
```

```
$(selector).prevUntil(element, filter element)
```

jQuery DOM Filtering Methods

To narrow the search for elements in a DOM tree, jQuery provides several methods such as `filter()`, `first()`, `last()`, `eq()`, `slice()`, `has()`, `not()`, and so on.

- `first()`

it gives the first element of the specified selector.

Syntax:

```
$(selector).first();
```

- `last()`

it gives the last element of the specified selector.

Syntax:

```
$(selector).last();
```

- `eq()`

it gives an element with a specific index number of the specified selector.

Syntax:

```
$(selector).eq(index);
```

```
$(selector).eq( indexFromEnd );
```

- `filter()`

it remove/detect an elements that are matched with specified selector.

Syntax:

```
$(selector).filter(selector)
```

```
$(selector).filter(function)
```

`$(selector).filter(selection)`

`$(selector).filter(elements)`

- `has()`

it gives all elements that have one or more elements within, that are matched with specified selector.

Syntax:

`$(selector).has(selector);`

- `is()`

it checks if one of the specified selector is matched with arguments.

Syntax:

`.is(selector)`

`.is(function)`

`.is(selection)`

`.is(elements)`

- `map()`

Pass each element in the current matched set through a function, producing a new jQuery object containing the return values

Syntax:

`.map(callback)`

- `slice()`

it selects a subset of specified selector based on its argument index or by start and stop value.

Syntax:

`$(selector).slice(start, end);`

`$(selector).slice(start);`

Example :

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
.siblings * {  
    display: block;  
    border: 2px solid lightgrey;  
    color: lightgrey;  
    padding: 5px;  
    margin: 15px;  
}
```

```
</style>
```

```
<script src=
```

```
"https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js">
```

```
</script>
```

```
<script>
```

```
$(document).ready(function() {  
    $("h2").siblings().css({  
        "color": "red",  
        "border": "2px solid red"  
    });  
    $("h2").parent().css({  
        "color": "green",  
        "border": "2px solid blue"
```

```
    });

    $("p").first().css(
        "background-color", "yellow");

    $("p").has("span").css(
        "background-color", "indigo");

    });
</script>
</head>

<body class="siblings">

    <div>GeeksforGeeks (parent)

        <p>GeeksforGeeks</p>

        <p><span>GeeksforGeeks</span></p>

        <h2>GeeksforGeeks</h2>

        <h3>GeeksforGeeks</h3>

        <p>GeeksforGeeks</p>

    </div>

</body>

</html>
```

GeeksforGeeks (parent)

GeeksforGeeks

GeeksforGeeks

GeeksforGeeks

GeeksforGeeks

GeeksforGeeks