Module II : JavaScript & jQuery(14 hrs)

JavaScript (10)

Introduction to JavaScript: Variable, statements, Operators, Comments, constructs, Functions, expressions, JavaScript console, Scope, Events, String Methods, Numbers, Number Methods, Dates, Date Formats, Date, Methods, Arrays, Array Methods, Booleans, Comparisons

Control Structures: Conditions, Switch, Loop For, Loop While, Break

Functions: Function Definitions, Function Parameters, Function Invocation, Function Closures

Objects: Object Definitions, Object Properties, Object Methods, Object Prototypes

Object Oriented Programming: Method,Constructor,Inheritance,Encapsulation,Abstraction,Polymorphism,Javascript Validations, Document Object Document and Events (DOM Manipulation)

HTML DOM: DOM Intro, DOM Methods, DOM Document, DOM Elements, DOM HTML, DOM CSS,DOM Animations, DOM Events, DOM EventListener Navigation, DOM Nodes, DOM Node list, Debugging, Type Conversion, Regular expressions, Errors, Debugging

Forms: Forms Validation, Forms API,JS Browser BOM, Window, Screen, Location, History, Navigator, Popup Alert, Timing, Cookies, Javascript Wind code quality via JSLint tool, Security in Java Script

jQuery(4)

Basics of jQuery, jQuery selection and events, jQuery Effects, jquery traversal and manipulation, Data attributes and templates, jQuery Plugins, Jquery Web Toolkit

1. **Introduction to JavaScript:**
2. **Variable,**
3. **statements,**
4. **Operators,**
5. **Comments,**
6. **constructs,**
7. **Functions,**
8. **expressions,**
9. **JavaScript console,**
10. **Scope,**
11. **Events,**
12. **String Methods,**
13. **Numbers,**
14. **Number Methods,**
15. **Dates,**
16. **Date Formats,**
17. **Date Methods,**
18. **Arrays,**
19. **Array Methods,**
20. **Booleans,**
21. **Comparisons**

Introduction to JavaScript

JavaScript is the world's most popular programming language.

JavaScript is the programming language of the Web.

JavaScript is easy to learn.

Why Study JavaScript?

JavaScript is one of the **3 languages** all web developers **must** learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behavior of web pages

Javascript History

JavaScript was developed by Brendan Eich in 1995, which appeared in Netscape, a popular browser of that time.

The language was initially called LiveScript and was later renamed JavaScript. There are many programmers who think that JavaScript and Java are the same. In fact, **JavaScript and Java are very much unrelated. Java is a very complex programming language whereas JavaScript is only a scripting language**. The syntax of JavaScript is mostly influenced by the programming language C.

How to Run JavaScript?

Being a scripting language, **JavaScript cannot run on its own. In fact, the browser is responsible for running JavaScript code**. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it is up to the browser to execute it. The main advantage of JavaScript is that **all modern web browsers support** JavaScript. So, you do not have to worry about whether your site visitor uses Internet Explorer, Google Chrome, Firefox or any other browser. JavaScript will be supported. Also, JavaScript **runs on any operating system** including Windows, Linux or Mac. Thus, JavaScript overcomes the main disadvantages of VBScript (Now deprecated) which is limited to just IE(Internet Explorer) and Windows.

Tools You Need

To start with, you need a text editor to write your code and a browser to display the web pages you develop. You can use a text editor of your choice including Notepad++, Visual Studio Code, Sublime Text, Atom or any other text editor you are comfortable with. You can use any web browser including Google Chrome, Firefox, Microsoft Edge, Internet Explorer etc.

3 Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external javaScript)

JavaScript Where To insert

The <script> Tag

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

Example

```
<script type="text/javascript">
document.write("JavaScript is a simple language for javatpoint learners");
</script>
```

The **script** tag specifies that we are using JavaScript.

The **text/javascript** is the content type that provides information to the browser about the data.

The **document.write()** function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

NOTE: Old JavaScript examples may use a type attribute: <script type="text/javascript">.The type attribute is not required. JavaScript is the default scripting language in HTML.

1) JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

```
1. <script type="text/javascript">
2.  alert("Hello Javatpoint");
3. </script>
```

---

2) JavaScript Example : code between the head tag

Let's see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.

In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name as given below.

To call function, you need to work on event. Here we are using onclick event to call msg() function.

```
1. <html>
2. <head>
3. <script type="text/javascript">
4. function msg(){
5.  alert("Hello Javatpoint");
6. }
7. </script>
8. </head>
9. <body>
10. <p>Welcome to JavaScript</p>
11. <form>
12. <input type="button" value="click" onclick="msg()"/>
13. </form>
14. </body>
15. </html>
```

External JavaScript file

We can create external JavaScript file and embed it in many html page.

It provides **code re usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Let's create an external JavaScript file that prints Hello Javatpoint in a alert dialog box.

message.js

```
1. function msg(){
2.  alert("Hello Javatpoint");
3. }
```

Let's include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

```
1. <html>
2. <head>
3. <script type="text/javascript" src="message.js"></script>
4. </head>
5. <body>
6. <p>Welcome to JavaScript</p>
7. <form>
8. <input type="button" value="click" onclick="msg()"/>
9. </form>
10. </body>
11. </html>
```

Note: External Javascript file cannot contain `<script>` tags.

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags:

Example

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

External References

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
- With a file path (like /js/)
- Without any path

This example uses a **full URL** to link to myScript.js:

Example

```
<script src="https://www.w3schools.com/js/myScript.js"></script>
```

This example uses a **file path** to link to myScript.js:

Example

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

This example uses no path to link to myScript.js:

Example

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

JavaScript Variables

What are Variables?

Variables are containers for storing data (storing data values).

4 Ways to Declare a JavaScript Variable:

- Using `var`
- Using `let`
- Using `const`
- Using nothing

In this example, `x`, `y`, and `z`, are variables, declared with the `var` keyword:

Example

```
var x = 5;
var y = 6;
var z = x + y;
```

Try it Yourself »

In this example, `x`, `y`, and `z`, are variables, declared with the `let` keyword:

Example

```
let x = 5;
let y = 6;
let z = x + y;
```

Try it Yourself »

In this example, `x`, `y`, and `z`, are undeclared variables:

Example

```
x = 5;
y = 6;
z = x + y;
```

Try it Yourself »

From all the examples above, you can guess:

- x stores the value 5
- y stores the value 6
- z stores the value 11

When to Use JavaScript var?

Always declare JavaScript variables with `var`,`let`, or`const`.

The `var` keyword is used in all JavaScript code from 1995 to 2015.

The `let` and `const` keywords were added to JavaScript in 2015.

If you want your code to run in older browsers, you must use `var`.

When to Use JavaScript const?

If you want a general rule: always declare variables with `const`.

If you think the value of the variable can change, use `let`.

In this example, `price1`, `price2`, and `total`, are variables:

Example

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

Try it Yourself »

The two variables `price1` and `price2` are declared with the `const` keyword.

These are constant values and cannot be changed.

The variable `total` is declared with the `let` keyword.

This is a value that can be changed.

Just Like Algebra

Just like in algebra, variables hold values:

```
let x = 5;
let y = 6;
```

Just like in algebra, variables are used in expressions:

```
let z = x + y;
```

From the example above, you can guess that the total is calculated to be 11.

Note

Variables are containers for storing values.

JavaScript Identifiers

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter.
- Names can also begin with $ and _ (but we will not use it in this tutorial).
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) cannot be used as names.

Note

JavaScript identifiers are case-sensitive.

The Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

This is different from algebra. The following does not make sense in algebra:

```
x = x + 5
```

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x.

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

Note

The "equal to" operator is written like `==` in JavaScript.

JavaScript Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe".

In programming, text values are called text strings.

JavaScript can handle many types of data, but for now, just think of numbers and strings.

Strings are written inside double or single quotes. Numbers are written without quotes.

If you put a number in quotes, it will be treated as a text string.

Example

```
const pi = 3.14;
let person = "John Doe";
let answer = 'Yes I am!';
```

Try it Yourself »

Declaring a JavaScript Variable

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the `var` or the `let` keyword:

```
var carName;
```

or:

```
let carName;
```

After the declaration, the variable has no value (technically it is `undefined`).

To **assign** a value to the variable, use the equal sign:

```
carName = "Volvo";
```

You can also assign a value to the variable when you declare it:

```
let carName = "Volvo";
```

In the example below, we create a variable called `carName` and assign the value "Volvo" to it.

Then we "output" the value inside an HTML paragraph with id="demo":

Example

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

<script>
let carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

Try it Yourself »

Note

It's a good programming practice to declare all variables at the beginning of a script.

One Statement, Many Variables

You can declare many variables in one statement.

Start the statement with `let` and separate the variables by **comma**:

Example

```
let person = "John Doe", carName = "Volvo", price = 200;
```

Try it Yourself »

A declaration can span multiple lines:

Example

```
let person = "John Doe",
carName = "Volvo",
price = 200;
```

Try it Yourself »

Value = undefined

In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input.

A variable declared without a value will have the value `undefined`.

The variable carName will have the value `undefined` after the execution of this statement:

Example

```
let carName;
```

Try it Yourself »

Re-Declaring JavaScript Variables

If you re-declare a JavaScript variable declared with `var`, it will not lose its value.

The variable `carName` will still have the value "Volvo" after the execution of these statements:

Example

```
var carName = "Volvo";
var carName;
```

Try it Yourself »

Note

You cannot re-declare a variable declared with `let` or `const`.

This will not work:

```
let carName = "Volvo";
let carName;
```

JavaScript Arithmetic

As with algebra, you can do arithmetic with JavaScript variables, using operators like `=` and `+`:

Example

```
let x = 5 + 2 + 3;
```

Try it Yourself »

You can also add strings, but strings will be concatenated:

Example

```
let x = "John" + " " + "Doe";
```

Try it Yourself »

Also try this:

Example

```
let x = "5" + 2 + 3;
```

Try it Yourself »

Note

If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated.

Now try this:

Example

```
let x = 2 + 3 + "5";
```

Try it Yourself »

JavaScript Dollar Sign $

Since JavaScript treats a dollar sign as a letter, identifiers containing $ are valid variable names:

Example

```
let $ = "Hello World";
let $$$ = 2;
let $myMoney = 5;
```

Try it Yourself »

Using the dollar sign is not very common in JavaScript, but professional programmers often use it as an alias for the main function in a JavaScript library.

In the JavaScript library jQuery, for instance, the main function `$` is used to select HTML elements. In jQuery `$("p");` means "select all p elements".

JavaScript Underscore (_)

Since JavaScript treats underscore as a letter, identifiers containing _ are valid variable names:

Example

```
let _lastName = "Johnson";
let _x = 2;
let _100 = 5;
```

Try it Yourself »

Using the underscore is not very common in JavaScript, but a convention among professional programmers is to use it as an alias for "private (hidden)" variables.

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands. For example:

The **Assignment Operator** (=) assigns a value to a variable.

Assignment Examples

```
let x = 10;
```

Try it Yourself »

```
// Assign the value 5 to x
let x = 5;
// Assign the value 2 to y
let y = 2;
// Assign the value x + y to z:
let z = x + y;
```

Try it Yourself »

The **Addition Operator** (+) adds numbers:

Adding

```
let x = 5;
let y = 2;
let z = x + y;
```

Try it Yourself »

The **Multiplication Operator** (*) multiplies numbers.

Multiplying

```
let x = 5;
let y = 2;
let z = x * y;
```

Try it Yourself »

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

JavaScript Arithmetic Operators

Arithmetic Operators are used to perform arithmetic on numbers:

Arithmetic Operators Example

```
let a = 3;
let x = (100 + 50) * a;
```

Try it Yourself »

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)

/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Arithmetic operators are fully described in the **JS Arithmetic** chapter.

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

**=	x **= y	x = x ** y

JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

The **Addition Assignment Operator** (+=) adds a value to a variable.

Assignment

```
let x = 10;
x += 5;
```

Try it Yourself »

Assignment operators are fully described in the **JS Assignment** chapter.

Adding JavaScript Strings

The + operator can also be used to add (concatenate) strings.

Example

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

The result of text3 will be:

```
John Doe
```

Try it Yourself »

The += assignment operator can also be used to add (concatenate) strings:

Example

```
let text1 = "What a very ";
text1 += "nice day";
```

The result of text1 will be:

```
What a very nice day
```

Try it Yourself »

When used on strings, the + operator is called the concatenation operator.

Adding Strings and Numbers

Adding two numbers, will return the sum, but adding a number and a string will return a string:

Example

```
let x = 5 + 5;
let y = "5" + 5;
let z = "Hello" + 5;
```

The result of *x*, *y*, and *z* will be:

```
10
55
Hello5
```

Try it Yourself »

If you add a number and a string, the result will be a string!

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal

!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Comparison operators are fully described in the **JS Comparisons** chapter.

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

Logical operators are fully described in the **JS Comparisons** chapter.

JavaScript Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Type operators are fully described in the **JS Type Conversion** chapter.

JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4

<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.
Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.
~00000000000000000000000000000101 will return
11111111111111111111111111111010

JavaScript Statements

Example

```
let x, y, z;     // Statement 1
x = 5;           // Statement 2
y = 6;           // Statement 3
z = x + y;       // Statement 4
```

JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

A **JavaScript program** is a list of programming **statements**.

In HTML, JavaScript programs are executed by the web browser.

JavaScript Statements

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

This statement tells the browser to write "Hello Dolly." inside an HTML element with id="demo":

Example

```
document.getElementById("demo").innerHTML = "Hello Dolly.";
```

Try it Yourself »

Most JavaScript programs contain many JavaScript statements.

The statements are executed, one by one, in the same order as they are written.

JavaScript programs (and JavaScript statements) are often called JavaScript code.

Semicolons ;

Semicolons separate JavaScript statements.

Add a semicolon at the end of each executable statement:

Examples

```
let a, b, c;  // Declare 3 variables
a = 5;        // Assign the value 5 to a
b = 6;        // Assign the value 6 to b
c = a + b;    // Assign the sum of a and b to c
```

Try it Yourself »

When separated by semicolons, multiple statements on one line are allowed:

```
a = 5; b = 6; c = a + b;
```

Try it Yourself »

On the web, you might see examples without semicolons.
Ending statements with semicolon is not required, but highly recommended.

JavaScript White Space

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

```
let person = "Hege";
let person="Hege";
```

A good practice is to put spaces around operators (= + - * /):

```
let x = y + z;
```

JavaScript Line Length and Line Breaks

For best readability, programmers often like to avoid code lines longer than 80 characters.

If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

Example

```
document.getElementById("demo").innerHTML =
"Hello Dolly!";
```

Try it Yourself »

JavaScript Code Blocks

JavaScript statements can be grouped together in code blocks, inside curly brackets {...}.

The purpose of code blocks is to define statements to be executed together.

One place you will find statements grouped together in blocks, is in JavaScript functions:

Example

```
function myFunction() {
  document.getElementById("demo1").innerHTML = "Hello Dolly!";
  document.getElementById("demo2").innerHTML = "How are you?";
}
```

Try it Yourself »

In this tutorial we use 2 spaces of indentation for code blocks.
You will learn more about functions later in this tutorial.

JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Our Reserved Words Reference lists all JavaScript keywords.

Here is a list of some of the keywords you will learn about in this tutorial:

Keyword	Description
var	Declares a variable
let	Declares a block variable
const	Declares a block constant
if	Marks a block of statements to be executed on a condition
switch	Marks a block of statements to be executed in different cases

for	Marks a block of statements to be executed in a loop
function	Declares a function
return	Exits a function
try	Implements error handling to a block of statements

JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.

JavaScript Comments

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

JavaScript comments can also be used to prevent execution, when testing alternative code.

Single Line Comments

Single line comments start with //.

Any text between // and the end of the line will be ignored by JavaScript (will not be executed).

This example uses a single-line comment before each code line:

Example

```
// Change heading:
document.getElementById("myH").innerHTML = "My First Page";

// Change paragraph:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Try it Yourself »

This example uses a single line comment at the end of each line to explain the code:

Example

```
let x = 5;       // Declare x, give it the value of 5
let y = x + 2;   // Declare y, give it the value of x + 2
```

Try it Yourself »

Multi-line Comments

Multi-line comments start with `/*` and end with `*/`.

Any text between `/*` and `*/` will be ignored by JavaScript.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Try it Yourself »

It is most common to use single line comments.
Block comments are often used for formal documentation.

Using Comments to Prevent Execution

Using comments to prevent execution of code is suitable for code testing.

Adding // in front of a code line changes the code lines from an executable line to a comment.

This example uses // to prevent execution of one of the code lines:

Example

```
//document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

Try it Yourself »

This example uses a comment block to prevent execution of multiple lines:

Example

```
/*
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
*/
```

JavaScript Object Constructors

A constructor is a special function that creates and initializes an object instance of a class. In JavaScript, a constructor gets called when an object is created using the `new` keyword.

The purpose of a constructor is to create a new object and set values for any existing object properties.

In a constructor function `this` does not have a value. It is a substitute for the new object. The value of `this` will become the new object when a new object is created.

See Also:

The JavaScript **this** Tutorial

Object Types (Blueprints) (Classes)

The examples from the previous chapters are limited. They only create single objects.

Sometimes we need a "**blueprint**" for creating many objects of the same "type".

The way to create an "object type", is to use an **object constructor function**.

In the example above, `function Person()` is an object constructor function.

Objects of the same type are created by calling the constructor function with the `new` keyword:

```
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
```

Try it yourself »

What is **this**?

In JavaScript, the `this` keyword refers to an **object**.

Which object depends on how `this` is being invoked (used or called).

The `this` keyword refers to different objects depending on how it is used:

In an object method, `this` refers to the **object**.
Alone, `this` refers to the **global object**.

In a function, `this` refers to the **global object**.
In a function, in strict mode, `this` is `undefined`.
In an event, `this` refers to the **element** that received the event.
Methods like `call()`, `apply()`, and `bind()` can refer `this` to **any object**.

Note

`this` is not a variable. It is a keyword. You cannot change the value of `this`.

See Also:

The JavaScript **this** Tutorial

Adding a Property to an Object

Adding a new property to an existing object is easy:

Example

```
myFather.nationality = "English";
```

Try it Yourself »

The property will be added to myFather. Not to myMother. (Not to any other person objects).

Adding a Method to an Object

Adding a new method to an existing object is easy:

Example

```
myFather.name = function () {
  return this.firstName + " " + this.lastName;
};
```

Try it Yourself »

The method will be added to myFather. Not to myMother. (Not to any other person objects).

Adding a Property to a Constructor

You cannot add a new property to an object constructor the same way you add a new property to an existing object:

Example

```
Person.nationality = "English";
```

Try it Yourself »

To add a new property to a constructor, you must add it to the constructor function:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.nationality = "English";
}
```

Try it Yourself »

This way object properties can have default values.

Adding a Method to a Constructor

Your constructor function can also define methods:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.name = function() {
    return this.firstName + " " + this.lastName;
  };
}
```

Try it Yourself »

You cannot add a new method to an object constructor the same way you add a new method to an existing object.

Adding methods to an object constructor must be done inside the constructor function:

Example

```
function Person(firstName, lastName, age, eyeColor) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
  this.eyeColor = eyeColor;
  this.changeName = function (name) {
    this.lastName = name;
  };
}
```

The changeName() function assigns the value of name to the person's lastName property.

Now You Can Try:

```
myMother.changeName("Doe");
```

Try it Yourself »

JavaScript knows which person you are talking about by "substituting" **this** with **myMother**.

Built-in JavaScript Constructors

JavaScript has built-in constructors for native objects:

```
new String()    // A new String object
new Number()    // A new Number object
new Boolean()   // A new Boolean object
new Object()    // A new Object object
new Array()     // A new Array object
new RegExp()    // A new RegExp object
new Function()  // A new Function object
new Date()      // A new Date object
```

Try it Yourself »

The `Math()` object is not in the list. `Math` is a global object. The `new` keyword cannot be used on `Math`.

Did You Know?

As you can see above, JavaScript has object versions of the primitive data types `String`, `Number`, and `Boolean`. But there is no reason to create complex objects. Primitive values are much faster:

Use string literals `""` instead of `new String()`.

Use number literals `50` instead of `new Number()`.

Use boolean literals `true / false` instead of `new Boolean()`.

Use object literals `{}` instead of `new Object()`.

Use array literals `[]` instead of `new Array()`.

Use pattern literals `/()/` instead of `new RegExp()`.

Use function expressions `() {}` instead of `new Function()`.

Example

```
let x1 = "";              // new primitive string
let x2 = 0;               // new primitive number
let x3 = false;           // new primitive boolean

const x4 = {};            // new Object object
const x5 = [];            // new Array object
const x6 = /()/           // new RegExp object
const x7 = function(){}; // new function
```

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

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

Example

```
// Function to compute the product of p1 and p2
function myFunction(p1, p2) {
  return p1 * p2;
}
```

Try it Yourself »

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: ***(parameter1, parameter2, ...)***

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

Function Return

When JavaScript reaches a `return` statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example

Calculate the product of two numbers, and return the result:

```
let x = myFunction(4, 3);   // Function is called, return value will
end up in x

function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

The result in x will be:

```
12
```

Try it Yourself »

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Example

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

Try it Yourself »

The () Operator Invokes the Function

Using the example above, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result.

Accessing a function without () will return the function object instead of the function result.

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

Try it Yourself »

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Instead of using a variable to store the return value of a function:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

Try it Yourself »

You will learn a lot more about functions later in this tutorial.

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Try it Yourself »

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

JavaScript Expressions

Statements and expressions are two very important terms in JavaScript. Given how frequently these two terms are used to describe JavaScript code, it is important to understand what they mean and the distinction between the two.

Expressions

Any unit of code that can be evaluated to a value is an expression. Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation. As per the MDN documentation, JavaScript has the following expression categories.

Arithmetic Expressions:

Arithmetic expressions evaluate to a numeric value. Examples include the following

```
10;     // Here 10 is an expression that is evaluated to the
numeric value 10 by the JS interpreter10+13; // This is another
expression that is evaluated to produce the numeric value 23
```

String Expressions:

String expressions are expressions that evaluate to a string. Examples include the following

```
'hello';
'hello' + 'world'; // evaluates to the string 'hello world'
```

Logical Expressions:

Expressions that evaluate to the boolean value true or false are considered to be logical expressions. This set of expressions often

involve the usage of logical operators && (AND), ||(OR) and !(NOT). Examples include

```
10 > 9;   // evaluates to boolean value true
10 < 20;  // evaluates to boolean value false
true;     //evaluates to boolean value true
a===20 && b===30; // evaluates to true or false based on the
values of a and b
```

Primary Expressions:

Primary expressions refer to stand alone expressions such as literal values, certain keywords and variable values. Examples include the following

```
'hello world'; // A string literal
23;            // A numeric literal
true;          // Boolean value true
sum;           // Value of variable sum
this;          // A keyword that evaluates to the current object
```

Left-hand-side Expressions:

Also known as lvalues, left-hand-side expressions are those that can appear on the left side of an assignment expression. Examples of left-hand-side expressions include the following

```
// variables such as i and total
i = 10;
total = 0;// properties of objectsvar obj = {}; // an empty
object with no properties
obj.x = 10; // an assignment expression// elements of arrays
array[0] = 20;
array[1] = 'hello';// Invalid left-hand-side errors
++(a+1); // SyntaxError. Attempting to increment or decrement an
expression that is not an lvalue will lead to errors.
```

Now that we have covered the basics of expressions, let’s dive a bit deeper into expressions.

Assignment Expressions:

When expressions use the = operator to assign a value to a variable, it is called an assignment expression. Examples include

```
average = 55;var b = (a = 1); // here the assignment expression
(a = 1) evaluates to a value that is assigned to the variable b.
b = (a = 1) is another assignment expression. var is not part of
the expression.
```

The = operator expects an lvalue as its left-side operand. The value of an assignment expression is the value of the right-side operand such as 55 in the above example. As a side effect, the = operator assigns the value on the right side to the value on the left side.

Expressions with side effects:

As we just saw with assignment expressions, expressions with side effects are those that result in a change or a side effect such as setting or modifying the value of a variable through the assignment operator =, function call, incrementing or decrementing the value of a variable.

```
sum = 20; // here sum is assigned the value of 20sum++; //
increments the value of sum by 1function modify(){
  a *= 10;
}var a = 10;
modify(); // modifies the value of a to 100.
```

Console in JavaScript

The **console object** provides access to the browser's debugging console.

The **console object** is a property of the **window object**.

The **console object** is accessed with:

`window.console` or just `console`

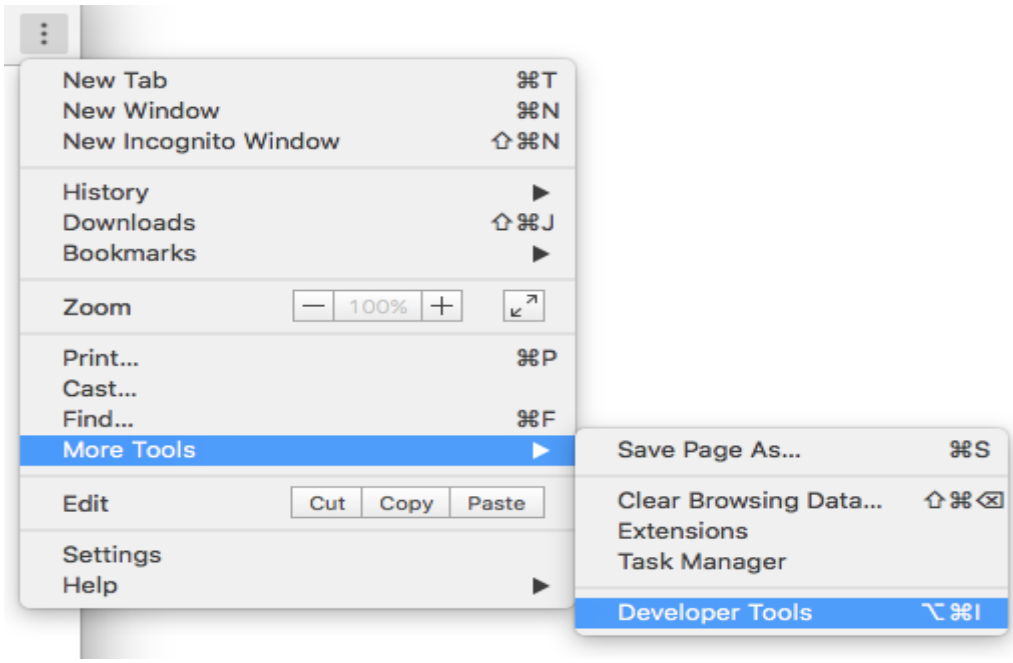
Browser Support

`console` is supported in all browsers:

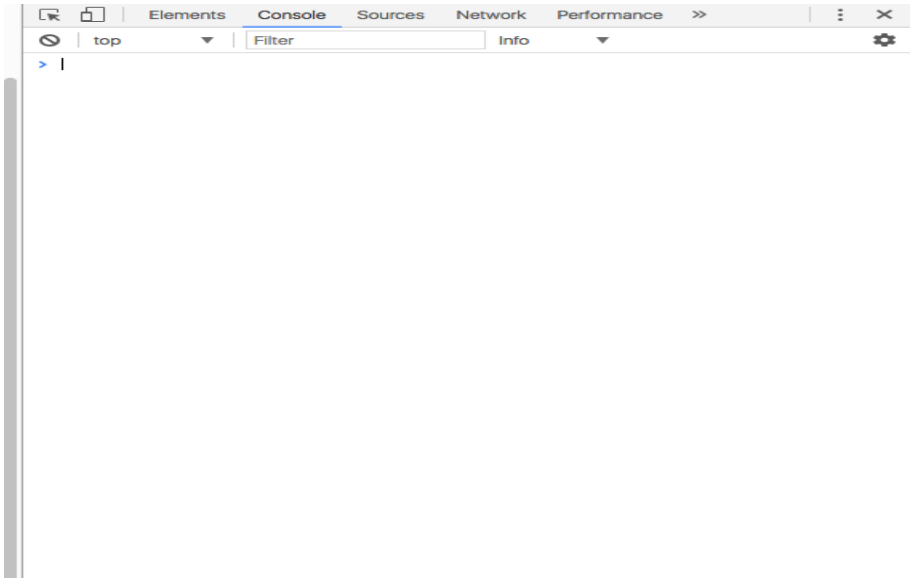
Chrome	IE	Edge	Firefox	Safari	Opera
Yes	8-11	Yes	Yes	Yes	Yes

Chrome

To open the JavaScript Console in Chrome, you can navigate to the menu at the top-right of your browser window signified by three vertical dots. From there, you can select **More Tools** then **Developer Tools**.



This will open a panel where you can click on **Console** along the top menu bar to bring up the JavaScript Console if it is not highlighted already:



You can also enter into the JavaScript Console by using the keyboard shortcut `CTRL` + `SHIFT` + `J` on Linux or Windows, or `COMMAND` + `OPTION` + `J` on macOS, which will bring focus immediately to the Console.

Now that you have accessed the Console, you can begin working within it in JavaScript.

Working in the Console

Within the Console, you can type and execute JavaScript code.

Start with an alert that prints out the string `Hello, World!`:

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

```

Copy

Once you press the `ENTER` key following your line of JavaScript, an alert popup will appear in your browser:

Note that the Console will also print the result of evaluating an expression, which will read as `undefined` when the expression does not explicitly return something.

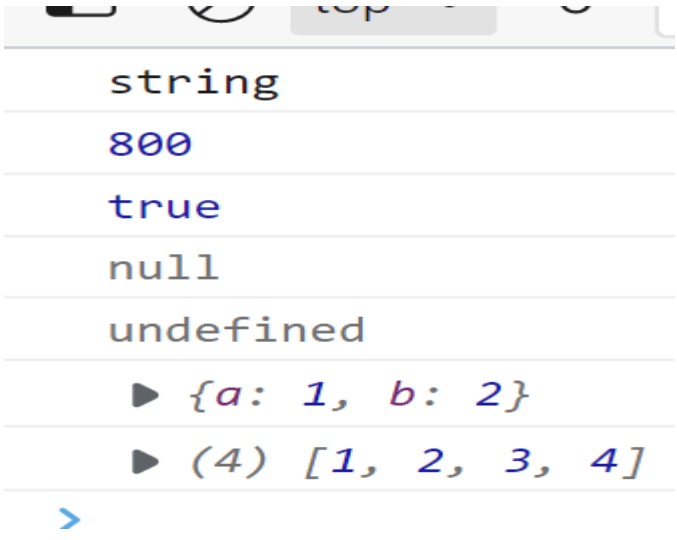
Types of Console

- console.log()
- console.error()
- console.warn()
- console.clear()
- console.time()
- console.table()
- console.count()
- console.group()

1. console.log()

Mainly used to log/print the output to the console. We can put any type inside the log(), be it strong, array, object, boolean etc.

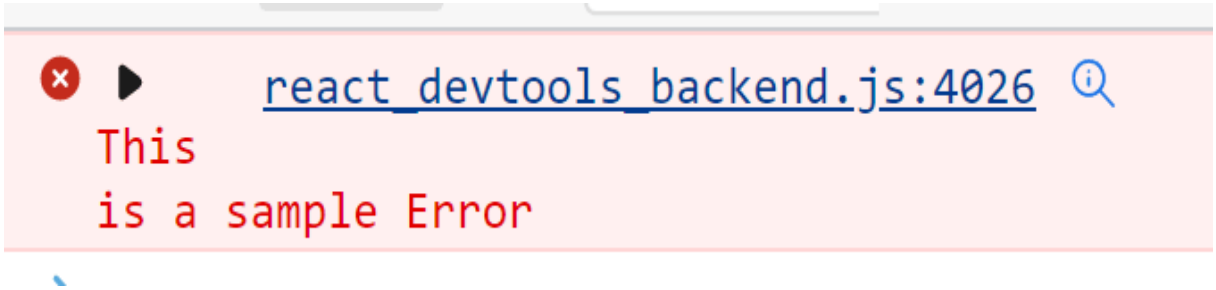
```
//console.log() method
console.log('string')
console.log(800)
console.log(true)
console.log(null)
console.log(undefined)
console.log({a:1, b:2}) // object inside log()
console.log([1,2,3,4]) // array inside log()
```



2. console.error()

This method is used to log error messages to the console. IUseful in testing of code. By default, the error message will be highlighted in red color.

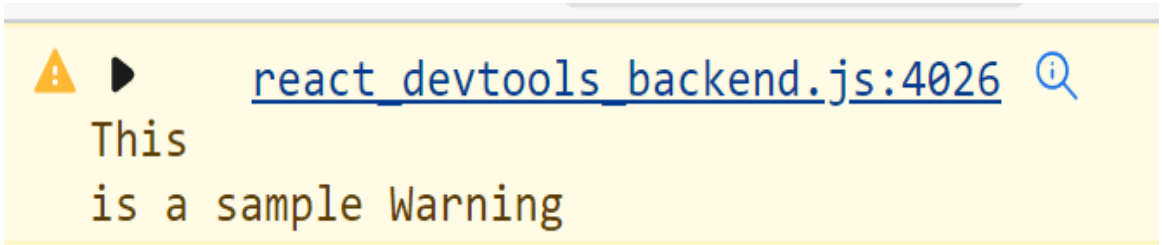
```
// console.error() method
console.error('This is a sample Error')
```



3. console.warn()

Used to log warning messages to the console. By default, the warning message will be highlighted with yellow color.

```
// console.warn() method
console.warn('This is a sample Warning')
```



4. console.clear()

Used to clear the console. This console will be cleared, incase Chromium based browsers, a simple overlayed text will be printed like one shown in the screenshot below stating '*Console was cleared*', while in Firefox, no message is returned.

```
// console.clear() method
console.clear()
```

```
Console was cleared
>
```

5. console.time() and console.timeEnd()

Whenever we want to know the amount of time spent by a block of code, or a function, we can make use ot the time() & timeEnd() methods provided by the JavaScript console object. They take a label which mst be same, and the code inside can be anything(a function, object, simple console.log())

```
// console.time() and console.timeEnd() method
// console.time() method
console.time("Let's see time console")
let time1 = function(){
   console.log('time1 function is running')
}
let time2 = function(){
   console.log('time2 function is running')
}
time1()
time2()
// console.timeEnd() method
console.timeEnd('Time Taken')
```

```
time1 function is running
time2 function is running
Time Taken: 0.14794921875
ms
```

6. console.table()

This method allows us to generate a table inside a console. The input must be an array or an object which will be shown as a table.

```
// console.table() method
console.table({a:1, b:2, c:3, d:4, e:5})
```

(index)	Value
a	1
b	2
c	3
d	4
e	5

▶ Object

7. console.count()

This method allows us to generate a table inside a console. The input **MUST** be an array or object which will be shows as a table.

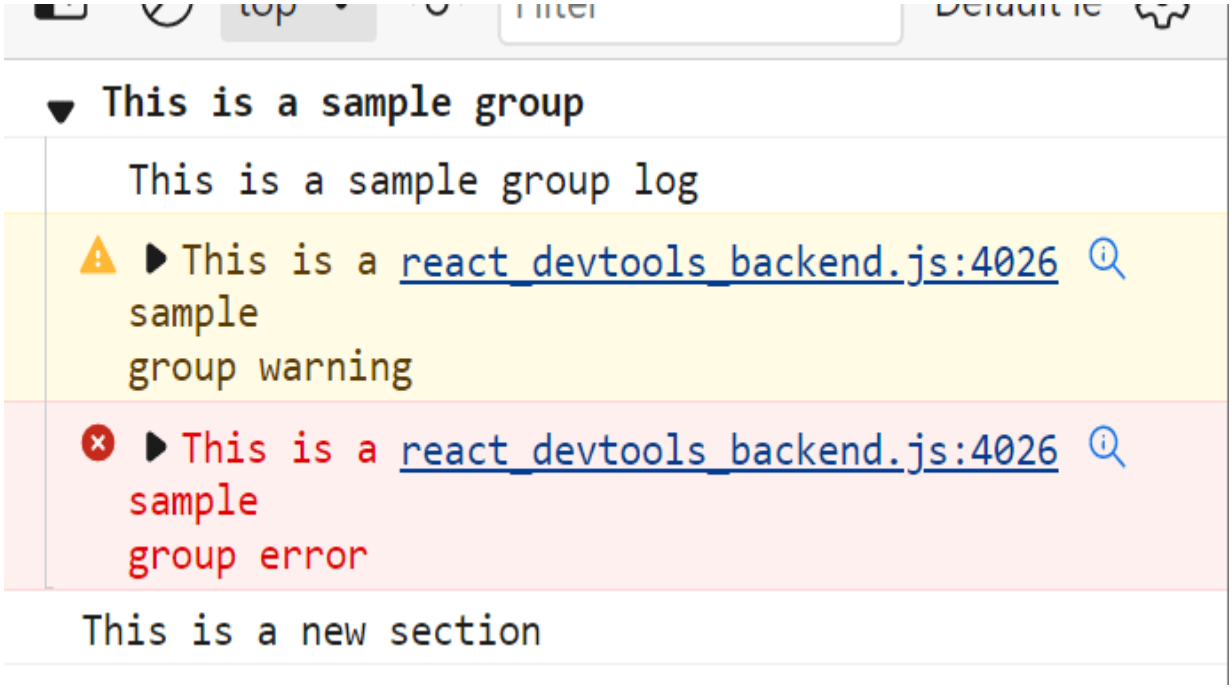
```
// console.count() method
console.log('This is a sample count')
for(let i=0; i<10; i++){
   console.count('This is iteration number', i)
}
```

```
This is a sample count
This is iteration number: 1
This is iteration number: 2
This is iteration number: 3
This is iteration number: 4
This is iteration number: 5
This is iteration number: 6
This is iteration number: 7
This is iteration number: 8
This is iteration number: 9
This is iteration number: 10
```

8. console.group() and console.groupEnd()

group() and groupEnd() methods of the console object allows us to group contents in a separate block of code, which will be indented. Just like the time() & timeEnd(), they also accept label, again of the same value.

```
// console.group() and console.groupEnd() method
// console.group() method
console.group('This is a sample group')
console.log('This is a sample group log')
console.warn('This is a sample group warning')
console.error('This is a sample group error')
// console.groupEnd() method
console.groupEnd('This is a sample group')
console.log('This is a new section')
```



JavaScript Scope

❮ Previous Next ❯

Scope determines the accessibility (visibility) of variables.

JavaScript has 3 types of scope:

- Block scope

- Function scope
- Global scope

Block Scope

Before ES6 (2015), JavaScript had only **Global Scope** and **Function Scope**.

ES6 introduced two important new JavaScript keywords: `let` and `const`.

These two keywords provide **Block Scope** in JavaScript.

Variables declared inside a { } block cannot be accessed from outside the block:

Example

```
{
  let x = 2;
}
// x can NOT be used here
```

Variables declared with the `var` keyword can NOT have block scope.

Variables declared inside a { } block can be accessed from outside the block.

Example

```
{
  var x = 2;
}
// x CAN be used here
```

Local Scope

Variables declared within a JavaScript function, become **LOCAL** to the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
```

```
}

// code here can NOT use carName
```

Try it Yourself »

Local variables have **Function Scope**:

They can only be accessed from within the function.

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

Function Scope

JavaScript has function scope: Each function creates a new scope.

Variables defined inside a function are not accessible (visible) from outside the function.

Variables declared with `var`, `let` and `const` are quite similar when declared inside a function.

They all have **Function Scope**:

```
function myFunction() {
  var carName = "Volvo";   // Function Scope
}

function myFunction() {
  let carName = "Volvo";   // Function Scope
}

function myFunction() {
  const carName = "Volvo";   // Function Scope
}
```

Global JavaScript Variables

A variable declared outside a function, becomes **GLOBAL**.

Example

```
let carName = "Volvo";
// code here can use carName

function myFunction() {
// code here can also use carName
}
```

Try it Yourself »

A global variable has **Global Scope**:

All scripts and functions on a web page can access it.

Global Scope

Variables declared **Globally** (outside any function) have **Global Scope**.

Global variables can be accessed from anywhere in a JavaScript program.

Variables declared with `var`, `let` and `const` are quite similar when declared outside a block.

They all have **Global Scope**:

```
var x = 2;       // Global scope
let x = 2;       // Global scope
const x = 2;       // Global scope
```

JavaScript Variables

In JavaScript, objects and functions are also variables.

Scope determines the accessibility of variables, objects, and functions from different parts of the code.

Automatically Global

If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.

This code example will declare a global variable `carName`, even if the value is assigned inside a function.

Example

```
myFunction();

// code here can use carName

function myFunction() {
  carName = "Volvo";
}
```

Try it Yourself »

Strict Mode

All modern browsers support running JavaScript in "Strict Mode".

You will learn more about how to use strict mode in a later chapter of this tutorial.

In "Strict Mode", undeclared variables are not automatically global.

Global Variables in HTML

With JavaScript, the global scope is the JavaScript environment.

In HTML, the global scope is the window object.

Global variables defined with the `var` keyword belong to the window object:

Example

```
var carName = "Volvo";
// code here can use window.carName
```

Try it Yourself »

Global variables defined with the `let` keyword do not belong to the window object:

Example

```
let carName = "Volvo";
// code here can not use window.carName
```

Try it Yourself »

Warning

Do NOT create global variables unless you intend to.

Your global variables (or functions) can overwrite window variables (or functions).
Any function, including the window object, can overwrite your global variables and functions.

The Lifetime of JavaScript Variables

The lifetime of a JavaScript variable starts when it is declared.

Function (local) variables are deleted when the function is completed.

In a web browser, global variables are deleted when you close the browser window (or tab).

Function Arguments

Function arguments (parameters) work as local variables inside functions.

JavaScript Events

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

HTML Events

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

With single quotes:

```
<element event='some JavaScript'>
```

With double quotes:

```
<element event="some JavaScript">
```

In the following example, an `onclick` attribute (with code), is added to a `<button>` element:

Example

```
<button onclick="document.getElementById('demo').innerHTML =
Date()">The time is?</button>
```

Try it Yourself »

In the example above, the JavaScript code changes the content of the element with id="demo".

In the next example, the code changes the content of its own element (using `this.innerHTML`):

Example

```
<button onclick="this.innerHTML = Date()">The time is?</button>
```

Try it Yourself »

JavaScript code is often several lines long. It is more common to see event attributes calling functions:

Example

```
<button onclick="displayDate()">The time is?</button>
```

Try it Yourself »

Common HTML Events

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element

onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

The list is much longer: [W3Schools JavaScript Reference HTML DOM Events](#).

JavaScript Event Handlers

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

JavaScript Strings

- JavaScript strings are for storing and manipulating text.
- A JavaScript string is zero or more characters written inside quotes.
- **Example**
- `let text = "John Doe";`
- Try it Yourself »
- You can use single or double quotes:

- ## Example

- ```
  let carName1 = "Volvo XC60";  // Double quotes
  let carName2 = 'Volvo XC60';  // Single quotes
  ```
- Try it Yourself »

- You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

- ## Example

- ```
  let answer1 = "It's alright";
  let answer2 = "He is called 'Johnny'";
  let answer3 = 'He is called "Johnny"';
  ```

- Try it Yourself »
-

- # String Length

- To find the length of a string, use the built-in `length` property:

- ## Example

- ```
  let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  let length = text.length;
  ```
- Try it Yourself »
-
-

- # Escape Character

- Because strings must be written within quotes, JavaScript will misunderstand this string:

- ```
  let text = "We are the so-called "Vikings" from the north.";
  ```

- The string will be chopped to "We are the so-called ".

- The solution to avoid this problem, is to use the **backslash escape character**.
- The backslash (\) escape character turns special characters into string characters:

Code	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

- The sequence \" inserts a double quote in a string:

Example

```
let text = "We are the so-called \"Vikings\" from the north.";
```

Try it Yourself »

- The sequence \' inserts a single quote in a string:

Example

```
let text= 'It\'s alright.';
```

Try it Yourself »

- The sequence \\ inserts a backslash in a string:

Example

```
let text = "The character \\ is called backslash.";
```

- Try it Yourself »

- Six other escape sequences are valid in JavaScript:

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

- The 6 escape characters above were originally designed to control typewriters, teletypes, and fax machines. They do not make any sense in HTML.

-

Breaking Long Code Lines

- For best readability, programmers often like to avoid code lines longer than 80 characters.

- If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

- ## Example

- `document.getElementById("demo").innerHTML = "Hello Dolly!";`
- Try it Yourself »

- You can also break up a code line **within a text string** with a single backslash:

- ## Example

- `document.getElementById("demo").innerHTML = "Hello \ Dolly!";`
- Try it Yourself »

- The `\` method is not the preferred method. It might not have universal support.
  Some browsers do not allow spaces behind the `\` character.

- A safer way to break up a string, is to use string addition:

- ## Example

- `document.getElementById("demo").innerHTML = "Hello " + "Dolly!";`
- Try it Yourself »

- You cannot break up a code line with a backslash:

- ## Example

- `document.getElementById("demo").innerHTML = \ "Hello Dolly!";`
- Try it Yourself »
-

JavaScript Strings as Objects

- Normally, JavaScript strings are primitive values, created from literals:

- `let x = "John";`

- But strings can also be defined as objects with the keyword `new`:

- `let y = new String("John");`

Example

- ```
  let x = "John";
  let y = new String("John");
  ```

- Try it Yourself »

- Do not create Strings objects.
- The `new` keyword complicates the code and slows down execution speed.
- String objects can produce unexpected results:
- When using the `==` operator, x and y are **equal**:

- ```
  let x = "John";
  let y = new String("John");
  ```
- Try it Yourself »

- When using the `===` operator, x and y are **not equal**:

- ```
  let x = "John";
  let y = new String("John");
  ```
- Try it Yourself »

- Note the difference between `(x==y)` and `(x===y)`.
- `(x == y)` true or false?

- ```
  let x = new String("John");
  let y = new String("John");
  ```
- Try it Yourself »

- `(x === y)` true or false?

- `let x = new String("John");`
  `let y = new String("John");`
- Try it Yourself »

- Comparing two JavaScript objects **always** returns **false**.

JavaScript String Methods

❮ PreviousNext ❯

String length	String trim()
String slice()	String trimStart()
String substring()	String trimEnd()
String substr()	String padStart()
String replace()	String padEnd()
String replaceAll()	String charAt()
String toUpperCase()	String charCodeAt()
String toLowerCase()	String split()
String concat()	

String search methods are covered in the next chapter.

JavaScript String Length

The `length` property returns the length of a string:

Example

```
let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let length = text.length;
```

Try it Yourself »

Extracting String Parts

There are 3 methods for extracting a part of a string:

- `slice(start, end)`
- `substring(start, end)`

- `substr(start, length)`

JavaScript String slice()

`slice()` extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: start position, and end position (end not included).

Example

Slice out a portion of a string from position 7 to position 13:

```
let text = "Apple, Banana, Kiwi";
let part = text.slice(7, 13);
```

Try it Yourself »

Note

JavaScript counts positions from zero.

First position is 0.

Second position is 1.

Examples

If you omit the second parameter, the method will slice out the rest of the string:

```
let text = "Apple, Banana, Kiwi";
let part = text.slice(7);
```

Try it Yourself »

If a parameter is negative, the position is counted from the end of the string:

```
let text = "Apple, Banana, Kiwi";
let part = text.slice(-12);
```

Try it Yourself »

This example slices out a portion of a string from position -12 to position -6:

```
let text = "Apple, Banana, Kiwi";
let part = text.slice(-12, -6);
```

Try it Yourself »

JavaScript String substring()

`substring()` is similar to `slice()`.

The difference is that start and end values less than 0 are treated as 0 in `substring()`.

Example

```
let str = "Apple, Banana, Kiwi";
let part = str.substring(7, 13);
```

Try it Yourself »

If you omit the second parameter, `substring()` will slice out the rest of the string.

JavaScript String substr()

`substr()` is similar to `slice()`.

The difference is that the second parameter specifies the **length** of the extracted part.

Example

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(7, 6);
```

Try it Yourself »

If you omit the second parameter, `substr()` will slice out the rest of the string.

Example

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(7);
```

Try it Yourself »

If the first parameter is negative, the position counts from the end of the string.

Example

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(-4);
```

Try it Yourself »

Replacing String Content

The `replace()` method replaces a specified value with another value in a string:

Example

```
let text = "Please visit Microsoft!";
let newText = text.replace("Microsoft", "W3Schools");
```

Try it Yourself »

Note

The `replace()` method does not change the string it is called on.

The `replace()` method returns a new string.

The `replace()` method replaces **only the first** match

If you want to replace all matches, use a regular expression with the /g flag set. See examples below.

By default, the `replace()` method replaces **only the first** match:

Example

```
let text = "Please visit Microsoft and Microsoft!";
let newText = text.replace("Microsoft", "W3Schools");
```

Try it Yourself »

By default, the `replace()` method is case sensitive. Writing MICROSOFT (with upper-case) will not work:

Example

```
let text = "Please visit Microsoft!";
let newText = text.replace("MICROSOFT", "W3Schools");
```

Try it Yourself »

To replace case insensitive, use a **regular expression** with an `/i` flag (insensitive):

Example

```
let text = "Please visit Microsoft!";
let newText = text.replace(/MICROSOFT/i, "W3Schools");
```

Try it Yourself »

Note

Regular expressions are written without quotes.

To replace all matches, use a **regular expression** with a `/g` flag (global match):

Example

```
let text = "Please visit Microsoft and Microsoft!";
let newText = text.replace(/Microsoft/g, "W3Schools");
```

Try it Yourself »

Note

You will learn a lot more about regular expressions in the chapter JavaScript Regular Expressions.

JavaScript String ReplaceAll()

In 2021, JavaScript introduced the string method `replaceAll()`:

Example

```
text = text.replaceAll("Cats","Dogs");
text = text.replaceAll("cats","dogs");
```

Try it Yourself »

The `replaceAll()` method allows you to specify a regular expression instead of a string to be replaced.

If the parameter is a regular expression, the global flag (g) must be set set, otherwise a TypeError is thrown.

Example

```
text = text.replaceAll(/Cats/g,"Dogs");
text = text.replaceAll(/cats/g,"dogs");
```

Try it Yourself »

Note

`replaceAll()` is an ES2021 feature.

`replaceAll()` does not work in Internet Explorer.

Converting to Upper and Lower Case

A string is converted to upper case with `toUpperCase()`:

A string is converted to lower case with `toLowerCase()`:

JavaScript String toUpperCase()

Example

```
let text1 = "Hello World!";
let text2 = text1.toUpperCase();
```

Try it Yourself »

JavaScript String toLowerCase()

Example

```
let text1 = "Hello World!";       // String
let text2 = text1.toLowerCase();  // text2 is text1 converted to lower
```

Try it Yourself »

JavaScript String concat()

`concat()` joins two or more strings:

Example

```
let text1 = "Hello";
let text2 = "World";
let text3 = text1.concat(" ", text2);
```

Try it Yourself »

The `concat()` method can be used instead of the plus operator. These two lines do the same:

Example

```
text = "Hello" + " " + "World!";
text = "Hello".concat(" ", "World!");
```

Note

All string methods return a new string. They don't modify the original string.

Formally said:

Strings are immutable: Strings cannot be changed, only replaced.

JavaScript String trim()

The `trim()` method removes whitespace from both sides of a string:

Example

```
let text1 = "      Hello World!      ";
let text2 = text1.trim();
```

Try it Yourself »

JavaScript String trimStart()

ECMAScript 2019 added the String method `trimStart()` to JavaScript.

The `trimStart()` method works like `trim()`, but removes whitespace only from the start of a string.

Example

```
let text1 = "     Hello World!     ";
let text2 = text1.trimStart();
```

Try it Yourself »

JavaScript String `trimStart()` is supported in all browsers since January 2020:

Chrome 66	Edge 79	Firefox 61	Safari 12	Opera 50
Apr 2018	Jan 2020	Jun 2018	Sep 2018	May 2018

JavaScript String trimEnd()

ECMAScript 2019 added the String method `trimEnd()` to JavaScript.

The `trimEnd()` method works like `trim()`, but removes whitespace only from the end of a string.

Example

```
let text1 = "     Hello World!     ";
let text2 = text1.trimEnd();
```

Try it Yourself »

JavaScript String `trimEnd()` is supported in all browsers since January 2020:

Chrome 66	Edge 79	Firefox 61	Safari 12	Opera 50
Apr 2018	Jan 2020	Jun 2018	Sep 2018	May 2018

JavaScript String Padding

ECMAScript 2017 added two String methods: `padStart()` and `padEnd()` to support padding at the beginning and at the end of a string.

JavaScript String padStart()

The `padStart()` method pads a string with another string:

Example

```
let text = "5";
let padded = text.padStart(4,"x");
```

Try it Yourself »

Example

```
let text = "5";
let padded = text.padStart(4,"0");
```

Try it Yourself »

Note

The `padStart()` method is a string method.

To pad a number, convert the number to a string first.

See the example below.

Example

```
let numb = 5;
let text = numb.toString();
let padded = text.padStart(4,"0");
```

Try it Yourself »

Browser Support

`padStart()` is an ECMAScript 2017 feature.

It is supported in all modern browsers:

Chrome	Edge	Firefox	Safari	Opera
Yes	Yes	Yes	Yes	Yes

`padStart()` is not supported in Internet Explorer.

JavaScript String padEnd()

The `padEnd()` method pads a string with another string:

Example

```
let text = "5";
let padded = text.padEnd(4,"x");
```

Try it Yourself »

Example

```
let text = "5";
let padded = text.padEnd(4,"0");
```

Try it Yourself »

Note

The `padEnd()` method is a string method.

To pad a number, convert the number to a string first.

See the example below.

Example

```
let numb = 5;
let text = numb.toString();
let padded = text.padEnd(4,"0");
```

Try it Yourself »

Browser Support

`padEnd()` is an ECMAScript 2017 feature.

It is supported in all modern browsers:

Chrome	Edge	Firefox	Safari	Opera
Yes	Yes	Yes	Yes	Yes

`padEnd()` is not supported in Internet Explorer.

Extracting String Characters

There are 3 methods for extracting string characters:

- `charAt(position)`
- `charCodeAt(position)`
- Property access [ ]

JavaScript String charAt()

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

Example

```
let text = "HELLO WORLD";
let char = text.charAt(0);
```

Try it Yourself »

JavaScript String charCodeAt()

The `charCodeAt()` method returns the unicode of the character at a specified index in a string:

The method returns a UTF-16 code (an integer between 0 and 65535).

Example

```
let text = "HELLO WORLD";
let char = text.charCodeAt(0);
```

Try it Yourself »

Property Access

ECMAScript 5 (2009) allows property access [ ] on strings:

Example

```
let text = "HELLO WORLD";
let char = text[0];
```

Try it Yourself »

Note

Property access might be a little **unpredictable:**

- It makes strings look like arrays (but they are not)
- If no character is found, [ ] returns undefined, while charAt() returns an empty string.
- It is read only. str[0] = "A" gives no error (but does not work!)

Example

```
let text = "HELLO WORLD";
text[0] = "A";    // Gives no error, but does not work
```

Try it Yourself »

Converting a String to an Array

If you want to work with a string as an array, you can convert it to an array.

JavaScript String split()

A string can be converted to an array with the `split()` method:

Example

```
text.split(",")    // Split on commas
text.split(" ")    // Split on spaces
text.split("|")    // Split on pipe
```

Try it Yourself »

If the separator is omitted, the returned array will contain the whole string in index [0].

If the separator is "", the returned array will be an array of single characters:

Example

```
text.split("")
```

Try it Yourself »

JavaScript Numbers

JavaScript has only one type of number. Numbers can be written with or without decimals.

Example

```
let x = 3.14;    // A number with decimals
let y = 3;       // A number without decimals
```

Try it Yourself »

Extra large or extra small numbers can be written with scientific (exponent) notation:

Example

```
let x = 123e5;    // 12300000
let y = 123e-5;   // 0.00123
```

Try it Yourself »

JavaScript Numbers are Always 64-bit Floating Point

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.

This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

Value (aka Fraction/Mantissa)	Exponent	Sign

52 bits (0 - 51)	11 bits (52 - 62)	1 bit (63)

Integer Precision

Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

Example

```
let x = 999999999999999;   // x will be 999999999999999
let y = 9999999999999999;  // y will be 10000000000000000
```

Try it Yourself »

The maximum number of decimals is 17.

Floating Precision

Floating point arithmetic is not always 100% accurate:

```
let x = 0.2 + 0.1;
```

Try it Yourself »

To solve the problem above, it helps to multiply and divide:

```
let x = (0.2 * 10 + 0.1 * 10) / 10;
```

Try it Yourself »

Adding Numbers and Strings

WARNING !!

JavaScript uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

Example

```
let x = 10;
let y = 20;
let z = x + y;
```

Try it Yourself »

If you add two strings, the result will be a string concatenation:

Example

```
let x = "10";
let y = "20";
let z = x + y;
```

Try it Yourself »

If you add a number and a string, the result will be a string concatenation:

Example

```
let x = 10;
let y = "20";
let z = x + y;
```

Try it Yourself »

If you add a string and a number, the result will be a string concatenation:

Example

```
let x = "10";
let y = 20;
let z = x + y;
```

Try it Yourself »

A common mistake is to expect this result to be 30:

Example

```
let x = 10;
let y = 20;
let z = "The result is: " + x + y;
```

Try it Yourself »

A common mistake is to expect this result to be 102030:

Example

```
let x = 10;
let y = 20;
let z = "30";
let result = x + y + z;
```

Try it Yourself »

The JavaScript interpreter works from left to right.

First 10 + 20 is added because x and y are both numbers.

Then 30 + "30" is concatenated because z is a string.

Numeric Strings

JavaScript strings can have numeric content:

```
let x = 100;         // x is a number

let y = "100";       // y is a string
```

JavaScript will try to convert strings to numbers in all numeric operations:

This will work:

```
let x = "100";
let y = "10";
let z = x / y;
```

Try it Yourself »

This will also work:

```
let x = "100";
let y = "10";
let z = x * y;
```

Try it Yourself »

And this will work:

```
let x = "100";
let y = "10";
let z = x - y;
```

Try it Yourself »

But this will not work:

```
let x = "100";
let y = "10";
let z = x + y;
```

Try it Yourself »

In the last example JavaScript uses the + operator to concatenate the strings.

NaN - Not a Number

`NaN` is a JavaScript reserved word indicating that a number is not a legal number.

Trying to do arithmetic with a non-numeric string will result in `NaN` (Not a Number):

Example

```
let x = 100 / "Apple";
```

Try it Yourself »

However, if the string contains a numeric value , the result will be a number:

Example

```
let x = 100 / "10";
```

Try it Yourself »

You can use the global JavaScript function `isNaN()` to find out if a value is a not a number:

Example

```
let x = 100 / "Apple";
isNaN(x);
```

Try it Yourself »

Watch out for `NaN`. If you use `NaN` in a mathematical operation, the result will also be `NaN`:

Example

```
let x = NaN;
let y = 5;
let z = x + y;
```

Try it Yourself »

Or the result might be a concatenation like NaN5:

Example

```
let x = NaN;
let y = "5";
let z = x + y;
```

Try it Yourself »

`NaN` is a number: `typeof NaN` returns `number`:

Example

```
typeof NaN;
```

Try it Yourself »

Infinity

`Infinity` (or `-Infinity`) is the value JavaScript will return if you calculate a number outside the largest possible number.

Example

```
let myNumber = 2;
// Execute until Infinity
while (myNumber != Infinity) {
  myNumber = myNumber * myNumber;
}
```

Try it Yourself »

Division by 0 (zero) also generates `Infinity`:

Example

```
let x =  2 / 0;
let y = -2 / 0;
```

Try it Yourself »

`Infinity` is a number: `typeof Infinity` returns `number`.

Example

```
typeof Infinity;
```

Try it Yourself »

Hexadecimal

JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

Example

```
let x = 0xFF;
```

Try it Yourself »

Never write a number with a leading zero (like 07).
Some JavaScript versions interpret numbers as octal if they are written with a leading zero.

By default, JavaScript displays numbers as **base 10** decimals.

But you can use the `toString()` method to output numbers from **base 2** to **base 36**.

Hexadecimal is **base 16**. Decimal is **base 10**. Octal is **base 8**. Binary is **base 2**.

Example

```
let myNumber = 32;
myNumber.toString(32);
myNumber.toString(16);
myNumber.toString(12);
myNumber.toString(10);
myNumber.toString(8);
```

```
myNumber.toString(2);
```

Try it Yourself »

JavaScript Numbers as Objects

Normally JavaScript numbers are primitive values created from literals:

```
let x = 123;
```

But numbers can also be defined as objects with the keyword `new`:

```
let y = new Number(123);
```

Example

```
let x = 123;
let y = new Number(123);
```

Try it Yourself »

Do not create Number objects.

The `new` keyword complicates the code and slows down execution speed.

Number Objects can produce unexpected results:

When using the `==` operator, x and y are **equal**:

```
let x = 500;
let y = new Number(500);
```

Try it Yourself »

When using the `===` operator, x and y are **not equal**.

```
let x = 500;
let y = new Number(500);
```

Try it Yourself »

Note the difference between `(x==y)` and `(x===y)`.

`(x == y)` true or false?

```
let x = new Number(500);
let y = new Number(500);
```

Try it Yourself »

`(x === y)` true or false?

```
let x = new Number(500);
let y = new Number(500);
```

Try it Yourself »

Comparing two JavaScript objects **always** returns **false**.

JavaScript Number Methods

These **number methods** can be used on all JavaScript numbers:

Method	Description
toString()	Returns a number as a string
toExponential()	Returns a number written in exponential notation
toFixed()	Returns a number written with a number of decimals
toPrecision()	Returns a number written with a specified length
ValueOf()	Returns a number as a number

The toString() Method

The `toString()` method returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

Example

```
let x = 123;
x.toString();
(123).toString();
(100 + 23).toString();
```

Try it Yourself »

The toExponential() Method

`toExponential()` returns a string, with a number rounded and written using exponential notation.

A parameter defines the number of characters behind the decimal point:

Example

```
let x = 9.656;
x.toExponential(2);
x.toExponential(4);
x.toExponential(6);
```

Try it Yourself »

The parameter is optional. If you don't specify it, JavaScript will not round the number.

The toFixed() Method

`toFixed()` returns a string, with the number written with a specified number of decimals:

Example

```
let x = 9.656;
x.toFixed(0);
x.toFixed(2);
```

```
x.toFixed(4);
x.toFixed(6);
```

Try it Yourself »

`toFixed(2)` is perfect for working with money.

The toPrecision() Method

`toPrecision()` returns a string, with a number written with a specified length:

Example

```
let x = 9.656;
x.toPrecision();
x.toPrecision(2);
x.toPrecision(4);
x.toPrecision(6);
```

Try it Yourself »

The valueOf() Method

`valueOf()` returns a number as a number.

Example

```
let x = 123;
x.valueOf();
(123).valueOf();
(100 + 23).valueOf();
```

Try it Yourself »

In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object).

The `valueOf()` method is used internally in JavaScript to convert Number objects to primitive values.

There is no reason to use it in your code.

All JavaScript data types have a `valueOf()` and a `toString()` method.

Converting Variables to Numbers

There are 3 JavaScript methods that can be used to convert a variable to a number:

Method	Description
Number()	Returns a number converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns a whole number

The methods above are not **number** methods. They are **global** JavaScript methods.

The Number() Method

The `Number()` method can be used to convert JavaScript variables to numbers:

Example

```
Number(true);
Number(false);
Number("10");
Number("  10");
Number("10  ");
Number(" 10  ");
Number("10.33");
Number("10,33");
Number("10 33");
Number("John");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

The Number() Method Used on Dates

`Number()` can also convert a date to a number.

Example

```
Number(new Date("1970-01-01"))
```

Try it Yourself »

Note

The `Date()` method returns the number of milliseconds since 1.1.1970.

The number of milliseconds between 1970-01-02 and 1970-01-01 is 86400000:

Example

```
Number(new Date("1970-01-02"))
```

Try it Yourself »

Example

```
Number(new Date("2017-09-30"))
```

Try it Yourself »

The parseInt() Method

`parseInt()` parses a string and returns a whole number. Spaces are allowed. Only the first number is returned:

Example

```
parseInt("-10");
parseInt("-10.33");
parseInt("10");
parseInt("10.33");
```

```
parseInt("10 20 30");
parseInt("10 years");
parseInt("years 10");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

The parseFloat() Method

`parseFloat()` parses a string and returns a number. Spaces are allowed. Only the first number is returned:

Example

```
parseFloat("10");
parseFloat("10.33");
parseFloat("10 20 30");
parseFloat("10 years");
parseFloat("years 10");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

Number Object Methods

These **object methods** belong to the **Number** object:

Method	Description
Number.isInteger()	Returns true if the argument is an integer
Number.isSafeInteger()	Returns true if the argument is a safe integer

Number.parseFloat()	Converts a string to a number
Number.parseInt()	Converts a string to a whole number

Number Methods Cannot be Used on Variables

The number methods above belong to the JavaScript **Number Object**.

These methods can only be accessed like `Number.isInteger()`.

Using X.isInteger() where X is a variable, will result in an error:

`TypeError X.isInteger is not a function`.

The Number.isInteger() Method

The `Number.isInteger()` method returns `true` if the argument is an integer.

Example

```
Number.isInteger(10);
Number.isInteger(10.5);
```

Try it Yourself »

The Number.isSafeInteger() Method

A safe integer is an integer that can be exactly represented as a double precision number.

The `Number.isSafeInteger()` method returns `true` if the argument is a safe integer.

Example

```
Number.isSafeInteger(10);
Number.isSafeInteger(12345678901234567890);
```

Try it Yourself »

Safe integers are all integers from -(2^{53} - 1) to +(2^{53} - 1).
This is safe: 9007199254740991. This is not safe: 9007199254740992.

The Number.parseFloat() Method

`Number.parseFloat()` parses a string and returns a number.

Spaces are allowed. Only the first number is returned:

Example

```
Number.parseFloat("10");
Number.parseFloat("10.33");
Number.parseFloat("10 20 30");
Number.parseFloat("10 years");
Number.parseFloat("years 10");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

Note

The **Number** methods `Number.parseInt()` and `Number.parseFloat()`

are the same as the

Global methods `parseInt()` and `parseFloat()`.

The purpose is modularization of globals (to make it easier to use the same JavaScript code outside the browser).

The Number.parseInt() Method

`Number.parseInt()` parses a string and returns a whole number.

Spaces are allowed. Only the first number is returned:

Example

```
Number.parseInt("-10");
Number.parseInt("-10.33");
Number.parseInt("10");
Number.parseInt("10.33");
Number.parseInt("10 20 30");
Number.parseInt("10 years");
Number.parseInt("years 10");
```

Try it Yourself »

If the number cannot be converted, `NaN` (Not a Number) is returned.

JavaScript Date Objects

JavaScript **Date Objects** let us work with dates:

Mon Feb 20 2023 21:46:21 GMT+0530 (India Standard Time)

Year: 2023 Month: 2 Day: 20 Hours: 21 Minutes: 46 Seconds: 21

Examples

```
const d = new Date();
```

Try it Yourself »

```
const d = new Date("2022-03-25");
```

Try it Yourself »

Note

Date objects are static. The "clock" is not "running".

The computer clock is ticking, date objects are not.

JavaScript Date Output

By default, JavaScript will use the browser's time zone and display a date as a full text string:

Mon Feb 20 2023 21:46:21 GMT+0530 (India Standard Time)

You will learn much more about how to display dates, later in this tutorial.

Creating Date Objects

Date objects are created with the `new Date()` constructor.

There are **9 ways** to create a new date object:

```
new Date()
new Date(date string)

new Date(year,month)
new Date(year,month,day)
new Date(year,month,day,hours)
new Date(year,month,day,hours,minutes)
new Date(year,month,day,hours,minutes,seconds)
new Date(year,month,day,hours,minutes,seconds,ms)

new Date(milliseconds)
```

JavaScript new Date()

`new Date()` creates a date object with the **current date and time**:

Example

```
const d = new Date();
```

Try it Yourself »

new Date(*date string*)

`new Date(date string)` creates a date object from a **date string**:

Examples

```
const d = new Date("October 13, 2014 11:13:00");
```

Try it Yourself »

```
const d = new Date("2022-03-25");
```

Try it Yourself »

Date string formats are described in the next chapter.

new Date(*year, month, ...*)

`new Date(year, month, ...)` creates a date object with a **specified date and time**.

7 numbers specify year, month, day, hour, minute, second, and millisecond (in that order):

Example

```
const d = new Date(2018, 11, 24, 10, 33, 30, 0);
```

Try it Yourself »

Note

JavaScript counts months from **0** to **11**:

January = 0.

December = 11.

Specifying a month higher than 11, will not result in an error but add the overflow to the next year:

Specifying:

```
const d = new Date(2018, 15, 24, 10, 33, 30);
```

Try it Yourself »

Is the same as:

```
const d = new Date(2019, 3, 24, 10, 33, 30);
```

Try it Yourself »

Specifying a day higher than max, will not result in an error but add the overflow to the next month:

Specifying:

```
const d = new Date(2018, 5, 35, 10, 33, 30);
```

Is the same as:

```
const d = new Date(2018, 6, 5, 10, 33, 30);
```

Try it Yourself »

Using 6, 4, 3, or 2 Numbers

6 numbers specify year, month, day, hour, minute, second:

Example

```
const d = new Date(2018, 11, 24, 10, 33, 30);
```

Try it Yourself »

5 numbers specify year, month, day, hour, and minute:

Example

```
const d = new Date(2018, 11, 24, 10, 33);
```

Try it Yourself »

4 numbers specify year, month, day, and hour:

Example

```
const d = new Date(2018, 11, 24, 10);
```

Try it Yourself »

3 numbers specify year, month, and day:

Example

```
const d = new Date(2018, 11, 24);
```

Try it Yourself »

2 numbers specify year and month:

Example

```
const d = new Date(2018, 11);
```

Try it Yourself »

You cannot omit month. If you supply only one parameter it will be treated as milliseconds.

Example

```
const d = new Date(2018);
```

Try it Yourself »

Previous Century

One and two digit years will be interpreted as 19xx:

Example

```
const d = new Date(99, 11, 24);
```

Try it Yourself »

Example

```
const d = new Date(9, 11, 24);
```

Try it Yourself »

JavaScript Stores Dates as Milliseconds

JavaScript stores dates as number of milliseconds since January 01, 1970.

Zero time is January 01, 1970 00:00:00 UTC.

One day (24 hours) is 86 400 000 milliseconds.

Now the time is: **1676909781755** milliseconds past January 01, 1970

new Date(*milliseconds*)

`new Date(milliseconds)` creates a new date object as **milliseconds** plus zero time:

Examples

01 January 1970 **plus** 100 000 000 000 milliseconds is:

```
const d = new Date(100000000000);
```

Try it Yourself »

January 01 1970 **minus** 100 000 000 000 milliseconds is:

```
const d = new Date(-100000000000);
```

Try it Yourself »

January 01 1970 **plus** 24 hours is:

```
const d = new Date(24 * 60 * 60 * 1000);
// or
const d = new Date(86400000);
```

Try it Yourself »

01 January 1970 **plus** 0 milliseconds is:

```
const d = new Date(0);
```

Try it Yourself »

Date Methods

When a date object is created, a number of **methods** allow you to operate on it.

Date methods allow you to get and set the year, month, day, hour, minute, second, and millisecond of date objects, using either local time or UTC (universal, or GMT) time.

Date methods and time zones are covered in the next chapters.

Displaying Dates

JavaScript will (by default) output dates in full text string format:

Example

```
Mon Feb 20 2023 21:46:21 GMT+0530 (India Standard Time)
```

Try it Yourself »

When you display a date object in HTML, it is automatically converted to a string, with the `toString()` method.

Example

```
const d = new Date();
d.toString();
```

Try it Yourself »

The `toDateString()` method converts a date to a more readable format:

Example

```
const d = new Date();
d.toDateString();
```

Try it Yourself »

The `toUTCString()` method converts a date to a string using the UTC standard:

Example

```
const d = new Date();
d.toUTCString();
```

Try it Yourself »

The `toISOString()` method converts a date to a string using the ISO standard:

Example

```
const d = new Date();
d.toISOString();
```

Try it Yourself »

JavaScript Date Formats

❮ Previous Next ❯

JavaScript Date Input

There are generally 3 types of JavaScript date input formats:

Type	Example
ISO Date	"2015-03-25" (The International Standard)
Short Date	"03/25/2015"
Long Date	"Mar 25 2015" or "25 Mar 2015"

The ISO format follows a strict standard in JavaScript.

The other formats are not so well defined and might be browser specific.

JavaScript Date Output

Independent of input format, JavaScript will (by default) output dates in full text string format:

```
Mon Feb 20 2023 21:47:49 GMT+0530 (India Standard Time)
```

JavaScript ISO Dates

ISO 8601 is the international standard for the representation of dates and times.

The ISO 8601 syntax (YYYY-MM-DD) is also the preferred JavaScript date format:

Example (Complete date)

```
const d = new Date("2015-03-25");
```

Try it Yourself »

The computed date will be relative to your time zone.
Depending on your time zone, the result above will vary between March 24 and March 25.

ISO Dates (Year and Month)

ISO dates can be written without specifying the day (YYYY-MM):

Example

```
const d = new Date("2015-03");
```

Try it Yourself »

Time zones will vary the result above between February 28 and March 01.

ISO Dates (Only Year)

ISO dates can be written without month and day (YYYY):

Example

```
const d = new Date("2015");
```

Try it Yourself »

Time zones will vary the result above between December 31 2014 and January 01 2015.

ISO Dates (Date-Time)

ISO dates can be written with added hours, minutes, and seconds (YYYY-MM-DDTHH:MM:SSZ):

Example

```
const d = new Date("2015-03-25T12:00:00Z");
```

Try it Yourself »

Date and time is separated with a capital T.

UTC time is defined with a capital letter Z.

If you want to modify the time relative to UTC, remove the Z and add +HH:MM or -HH:MM instead:

Example

```
const d = new Date("2015-03-25T12:00:00-06:30");
```

Try it Yourself »

UTC (Universal Time Coordinated) is the same as GMT (Greenwich Mean Time).

Omitting T or Z in a date-time string can give different results in different browsers.

Time Zones

When setting a date, without specifying the time zone, JavaScript will use the browser's time zone.

When getting a date, without specifying the time zone, the result is converted to the browser's time zone.

In other words: If a date/time is created in GMT (Greenwich Mean Time), the date/time will be converted to CDT (Central US Daylight Time) if a user browses from central US.

JavaScript Short Dates.

Short dates are written with an "MM/DD/YYYY" syntax like this:

Example

```
const d = new Date("03/25/2015");
```

Try it Yourself »

WARNINGS !

In some browsers, months or days with no leading zeroes may produce an error:

```
const d = new Date("2015-3-25");
```

The behavior of "YYYY/MM/DD" is undefined.
Some browsers will try to guess the format. Some will return NaN.

```
const d = new Date("2015/03/25");
```

The behavior of "DD-MM-YYYY" is also undefined.
Some browsers will try to guess the format. Some will return NaN.

```
const d = new Date("25-03-2015");
```

JavaScript Long Dates.

Long dates are most often written with a "MMM DD YYYY" syntax like this:

Example

```
const d = new Date("Mar 25 2015");
```

Try it Yourself »

Month and day can be in any order:

Example

```
const d = new Date("25 Mar 2015");
```

Try it Yourself »

And, month can be written in full (January), or abbreviated (Jan):

Example

```
const d = new Date("January 25 2015");
```

Try it Yourself »

Example

```
const d = new Date("Jan 25 2015");
```

Try it Yourself »

Commas are ignored. Names are case insensitive:

Example

```
const d = new Date("JANUARY, 25, 2015");
```

Try it Yourself »

Date Input - Parsing Dates

If you have a valid date string, you can use the `Date.parse()` method to convert it to milliseconds.

`Date.parse()` returns the number of milliseconds between the date and January 1, 1970:

Example

```
let msec = Date.parse("March 21, 2012");
```

Try it Yourself »

You can then use the number of milliseconds to **convert it to a date** object:

Example

```
let msec = Date.parse("March 21, 2012");
const d = new Date(msec);
```

Try it Yourself »

JavaScript Get Date Methods

The new Date() Constructor

In JavaScript, date objects are created with `new Date()`.

`new Date()` returns a date object with the current date and time.

Get the Current Time

```
const date = new Date();
```

Try it Yourself »

Date Get Methods

Method	Description
getFullYear()	Get **year** as a four digit number (yyyy)
getMonth()	Get **month** as a number (0-11)
getDate()	Get **day** as a number (1-31)

getDay()	Get **weekday** as a number (0-6)
getHours()	Get **hour** (0-23)
getMinutes()	Get **minute** (0-59)
getSeconds()	Get **second** (0-59)
getMilliseconds()	Get **millisecond** (0-999)
getTime()	Get **time** (milliseconds since January 1, 1970)

Note 1

The get methods above return **Local time**.

Universal time (UTC) is documented at the bottom of this page.

Note 2

The get methods return information from existing date objects.

In a date object, the time is static. The "clock" is not "running".

The time in a date object is NOT the same as current time.

The getFullYear() Method

The `getFullYear()` method returns the year of a date as a four digit number:

Examples

```
const d = new Date("2021-03-25");
d.getFullYear();
```

Try it Yourself »

```
const d = new Date();
d.getFullYear();
```

Try it Yourself »

Warning !

Old JavaScript code might use the non-standard method getYear().

getYear() is supposed to return a 2-digit year.

getYear() is deprecated. Do not use it!

The getMonth() Method

The `getMonth()` method returns the month of a date as a number (0-11).

Note

In JavaScript, January is month number 0, February is number 1, ...

Finally, December is month number 11.

Examples

```
const d = new Date("2021-03-25");
d.getMonth();
```

Try it Yourself »

```
const d = new Date();
d.getMonth();
```

Try it Yourself »

Note

You can use an array of names to return the month as a name:

Examples

```
const months =
["January", "February", "March", "April", "May", "June", "July", "Augus
```

```
t", "September", "October", "November", "December"];

const d = new Date("2021-03-25");
let month = months[d.getMonth()];
```

Try it Yourself »

```
const months =
["January", "February", "March", "April", "May", "June", "July", "Augus
t", "September", "October", "November", "December"];

const d = new Date();
let month = months[d.getMonth()];
```

Try it Yourself »

The getDate() Method

The `getDate()` method returns the day of a date as a number (1-31):

Examples

```
const d = new Date("2021-03-25");
d.getDate();
```

Try it Yourself »

```
const d = new Date();
d.getDate();
```

Try it Yourself »

The getHours() Method

The `getHours()` method returns the hours of a date as a number (0-23):

Examples

```
const d = new Date("2021-03-25");
d.getHours();
```

Try it Yourself »

```
const d = new Date();
d.getHours();
```

Try it Yourself »

The getMinutes() Method

The `getMinutes()` method returns the minutes of a date as a number (0-59):

Examples

```
const d = new Date("2021-03-25");
d.getMinutes();
```

Try it Yourself »

```
const d = new Date();
d.getMinutes();
```

Try it Yourself »

The getSeconds() Method

The `getSeconds()` method returns the seconds of a date as a number (0-59):

Examples

```
const d = new Date("2021-03-25");
d.getSeconds();
```

Try it Yourself »

```
const d = new Date();
d.getSeconds();
```

Try it Yourself »

The getMilliseconds() Method

The `getMilliseconds()` method returns the milliseconds of a date as a number (0-999):

Examples

```
const d = new Date("2021-03-25");
d.getMilliseconds();
```

Try it Yourself »

```
const d = new Date();
d.getMilliseconds();
```

Try it Yourself »

The getDay() Method

The `getDay()` method returns the weekday of a date as a number (0-6).

Note

In JavaScript, the first day of the week (day 0) is Sunday.

Some countries in the world consider the first day of the week to be Monday.

Examples

```
const d = new Date("2021-03-25");
d.getDay();
```

Try it Yourself »

```
const d = new Date();
d.getDay();
```

Try it Yourself »

Note

You can use an array of names, and `getDay()` to return weekday as a name:

Examples

```
const days =
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Sat
urday"];

const d = new Date("2021-03-25");
let day = days[d.getDay()];
```

Try it Yourself »

```
const days =
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Sat
urday"];
```

```
const d = new Date();
let day = days[d.getDay()];
```

Try it Yourself »

The getTime() Method

The `getTime()` method returns the number of milliseconds since January 1, 1970:

Examples

```
const d = new Date("1970-01-01");
d.getTime();
```

Try it Yourself »

```
const d = new Date("2021-03-25");
d.getTime();
```

Try it Yourself »

```
const d = new Date();
d.getTime();
```

Try it Yourself »

The Date.now() Method

`Date.now()` returns the number of milliseconds since January 1, 1970.

Examples

```
let ms = Date.now();
```

Try it Yourself »

Calculate the number of years since 1970/01/01:

```
const minute = 1000 * 60;
const hour = minute * 60;
const day = hour * 24;
const year = day * 365;

let years = Math.round(Date.now() / year);
```

Try it Yourself »

`Date.now()` is a static method of the Date object.

You cannot use it on a date object like `myDate.now()`.

The syntax is always `Date.now()`.

UTC Date Get Methods

Method	Same As	Description
getUTCDate()	getDate()	Returns the UTC date
getUTCFullYear()	getFullYear()	Returns the UTC year
getUTCMonth()	getMonth()	Returns the UTC month
getUTCDay()	getDay()	Returns the UTC day
getUTCHours()	getHours()	Returns the UTC hour
getUTCMinutes()	getMinutes()	Returns the UTC minutes
getUTCSeconds()	getSeconds()	Returns the UTC seconds
getUTCMilliseconds()	getMilliseconds()	Returns the UTC milliseconds

UTC methods use UTC time (Coordinated Universal Time).

UTC time is the same as GMT (Greenwich Mean Time).

The getTimezoneOffset() Method

The `getTimezoneOffset()` method returns the difference (in minutes) between local time an UTC time:

Example

```
let diff = d.getTimezoneOffset();
```

Try it Yourself »

JavaScript Set Date Methods

Set Date methods let you set date values (years, months, days, hours, minutes, seconds, milliseconds) for a Date Object.

Set Date Methods

Set Date methods are used for setting a part of a date:

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)

setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

The setFullYear() Method

The `setFullYear()` method sets the year of a date object. In this example to 2020:

Example

```
const d = new Date();
d.setFullYear(2020);
```

Try it Yourself »

The `setFullYear()` method can **optionally** set month and day:

Example

```
const d = new Date();
d.setFullYear(2020, 11, 3);
```

Try it Yourself »

The setMonth() Method

The `setMonth()` method sets the month of a date object (0-11):

Example

```
const d = new Date();
d.setMonth(11);
```

Try it Yourself »

The setDate() Method

The `setDate()` method sets the day of a date object (1-31):

Example

```
const d = new Date();
d.setDate(15);
```

Try it Yourself »

The `setDate()` method can also be used to **add days** to a date:

Example

```
const d = new Date();
d.setDate(d.getDate() + 50);
```

Try it Yourself »

If adding days shifts the month or year, the changes are handled automatically by the Date object.

The setHours() Method

The `setHours()` method sets the hours of a date object (0-23):

Example

```
const d = new Date();
d.setHours(22);
```

Try it Yourself »

The setMinutes() Method

The `setMinutes()` method sets the minutes of a date object (0-59):

Example

```
const d = new Date();
d.setMinutes(30);
```

Try it Yourself »

The setSeconds() Method

The `setSeconds()` method sets the seconds of a date object (0-59):

Example

```
const d = new Date();
d.setSeconds(30);
```

Try it Yourself »

Compare Dates

Dates can easily be compared.

The following example compares today's date with January 14, 2100:

Example

```
let text = "";
const today = new Date();
```

```
const someday = new Date();
someday.setFullYear(2100, 0, 14);

if (someday > today) {
  text = "Today is before January 14, 2100.";
} else {
  text = "Today is after January 14, 2100.";
}
```

JavaScript Arrays

An array is a special variable, which can hold more than one value:

```
const cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself »

Why Use Arrays?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```

It is a common practice to declare arrays with the `const` keyword.

Learn more about `const` with arrays in the chapter: JS Array Const.

Example

```
const cars = ["Saab", "Volvo", "BMW"];
```

Try it Yourself »

Spaces and line breaks are not important. A declaration can span multiple lines:

Example

```
const cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

Try it Yourself »

You can also create an array, and then provide the elements:

Example

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Try it Yourself »

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example

```
const cars = new Array("Saab", "Volvo", "BMW");
```

Try it Yourself »

The two examples above do exactly the same.

There is no need to use `new Array()`.

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

Accessing Array Elements

You access an array element by referring to the **index number**:

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

Try it Yourself »

Note: Array indexes start with 0.

[0] is the first element. [1] is the second element.

Changing an Array Element

This statement changes the value of the first element in `cars`:

```
cars[0] = "Opel";
```

Example

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

Try it Yourself »

Access the Full Array

With JavaScript, the full array can be accessed by referring to the array name:

Example

```
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

Try it Yourself »

Arrays are Objects

Arrays are a special type of objects. The `typeof` operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Arrays use **numbers** to access its "elements". In this example, `person[0]` returns John:

Array:

```
const person = ["John", "Doe", 46];
```

Try it Yourself »

Objects use **names** to access its "members". In this example, `person.firstName` returns John:

Object:

```
const person = {firstName:"John", lastName:"Doe", age:46};
```

Try it Yourself »

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array:

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

Array Properties and Methods

The real strength of JavaScript arrays are the built-in array properties and methods:

```
cars.length   // Returns the number of elements
cars.sort()   // Sorts the array
```

Array methods are covered in the next chapters.

The length Property

The `length` property of an array returns the length of an array (the number of array elements).

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
```

Try it Yourself »

The `length` property is always one more than the highest array index.

Accessing the First Array Element

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0];
```

Try it Yourself »

Accessing the Last Array Element

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1];
```

Try it Yourself »

Looping Array Elements

One way to loop through an array, is using a `for` loop:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;

let text = "<ul>";
for (let i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```

Try it Yourself »

You can also use the `Array.forEach()` function:

Example

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

let text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
```

Try it Yourself »

Adding Array Elements

The easiest way to add a new element to an array is using the `push()` method:

Example

```
const fruits = ["Banana", "Orange", "Apple"];
fruits.push("Lemon");  // Adds a new element (Lemon) to fruits
```

Try it Yourself »

New element can also be added to an array using the `length` property:

Example

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[fruits.length] = "Lemon";  // Adds "Lemon" to fruits
```

Try it Yourself »

WARNING !

Adding elements with high indexes can create undefined "holes" in an array:

Example

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[6] = "Lemon";  // Creates undefined "holes" in fruits
```

Try it Yourself »

Associative Arrays

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

Example

```
const person = [];
person[0] = "John";
person[1] = "Doe";
person[2] = 46;
person.length;    // Will return 3
person[0];        // Will return "John"
```

Try it Yourself »

WARNING !!
If you use named indexes, JavaScript will redefine the array to an object.

After that, some array methods and properties will produce **incorrect results**.

Example:

```
const person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
person.length;     // Will return 0
person[0];         // Will return undefined
```

Try it Yourself »

The Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

Arrays are a special kind of objects, with numbered indexes.

When to Use Arrays. When to use Objects.

- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

JavaScript new Array()

JavaScript has a built-in array constructor `new Array()`.

But you can safely use `[]` instead.

These two different statements both create a new empty array named points:

```
const points = new Array();
const points = [];
```

These two different statements both create a new array containing 6 numbers:

```
const points = new Array(40, 100, 1, 5, 25, 10);
const points = [40, 100, 1, 5, 25, 10];
```

Try it Yourself »

The new keyword can produce some unexpected results:

```
// Create an array with three elements:
const points = new Array(40, 100, 1);
```

Try it Yourself »

```
// Create an array with two elements:
const points = new Array(40, 100);
```

Try it Yourself »

```
// Create an array with one element ???
const points = new Array(40);
```

Try it Yourself »

A Common Error

```
const points = [40];
```

is not the same as:

```
const points = new Array(40);
```

```
// Create an array with one element:
const points = [40];
```

Try it Yourself »

```
// Create an array with 40 undefined elements:
const points = new Array(40);
```

Try it Yourself »

How to Recognize an Array

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator typeof returns "object":

```
const fruits = ["Banana", "Orange", "Apple"];
let type = typeof fruits;
```

Try it Yourself »

The typeof operator returns object because a JavaScript array is an object.

Solution 1:

To solve this problem ECMAScript 5 (JavaScript 2009) defined a new method `Array.isArray()`:

```
Array.isArray(fruits);
```

Try it Yourself »

Solution 2:

The `instanceof` operator returns true if an object is created by a given constructor:

```
const fruits = ["Banana", "Orange", "Apple"];

fruits instanceof Array;
```

Try it Yourself »

JavaScript Array Methods

Converting Arrays to Strings

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Result:

Banana,Orange,Apple,Mango

Try it Yourself »

The `join()` method also joins all array elements into a string.

It behaves just like `toString()`, but in addition you can specify the separator:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Result:

Banana * Orange * Apple * Mango

Try it Yourself »

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

JavaScript Array pop()

The `pop()` method removes the last element from an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

Try it Yourself »

The `pop()` method returns the value that was "popped out":

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();
```

Try it Yourself »

JavaScript Array push()

The `push()` method adds a new element to an array (at the end):

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

Try it Yourself »

The `push()` method returns the new array length:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");
```

Try it Yourself »

Shifting Elements

Shifting is equivalent to popping, but working on the first element instead of the last.

JavaScript Array shift()

The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

Try it Yourself »

The `shift()` method returns the value that was "shifted out":

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.shift();
```

Try it Yourself »

JavaScript Array unshift()

The `unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Try it Yourself »

The `unshift()` method returns the new array length:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon");
```

Try it Yourself »

Changing Elements

Array elements are accessed using their **index number**:

Array **indexes** start with 0:

[0] is the first array element
[1] is the second
[2] is the third ...

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";
```

Try it Yourself »

JavaScript Array length

The `length` property provides an easy way to append a new element to an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";
```

Try it Yourself »

JavaScript Array delete()

Warning !

Array elements can be deleted using the JavaScript operator `delete`.

Using `delete` leaves `undefined` holes in the array.

Use pop() or shift() instead.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];
```

Try it Yourself »

Merging (Concatenating) Arrays

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Example (Merging Two Arrays)

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];

const myChildren = myGirls.concat(myBoys);
```

Try it Yourself »

The `concat()` method does not change the existing arrays. It always returns a new array.

The `concat()` method can take any number of array arguments:

Example (Merging Three Arrays)

```
const arr1 = ["Cecilie", "Lone"];
const arr2 = ["Emil", "Tobias", "Linus"];
const arr3 = ["Robin", "Morgan"];
const myChildren = arr1.concat(arr2, arr3);
```

Try it Yourself »

The `concat()` method can also take strings as arguments:

Example (Merging an Array with Values)

```
const arr1 = ["Emil", "Tobias", "Linus"];
const myChildren = arr1.concat("Peter");
```

Try it Yourself »

Splicing and Slicing Arrays

The `splice()` method adds new items to an array.

The `slice()` method slices out a piece of an array.

JavaScript Array splice()

The `splice()` method can be used to add new items to an array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

Try it Yourself »

The first parameter (2) defines the position **where** new elements should be **added** (spliced in).

The second parameter (0) defines **how many** elements should be **removed**.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.

The `splice()` method returns an array with the deleted items:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 2, "Lemon", "Kiwi");
```

Try it Yourself »

Using splice() to Remove Elements

With clever parameter setting, you can use `splice()` to remove elements without leaving "holes" in the array:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1);
```

Try it Yourself »

The first parameter (0) defines the position where new elements should be **added** (spliced in).

The second parameter (1) defines **how many** elements should be **removed**.

The rest of the parameters are omitted. No new elements will be added.

JavaScript Array slice()

The `slice()` method slices out a piece of an array into a new array.

This example slices out a part of an array starting from array element 1 ("Orange"):

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1);
```

Try it Yourself »

Note

The `slice()` method creates a new array.

The `slice()` method does not remove any elements from the source array.

This example slices out a part of an array starting from array element 3 ("Apple"):

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(3);
```

Try it Yourself »

The `slice()` method can take two arguments like `slice(1, 3)`.

The method then selects elements from the start argument, and up to (but not including) the end argument.

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1, 3);
```

Try it Yourself »

If the end argument is omitted, like in the first examples, the `slice()` method slices out the rest of the array.

Example

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(2);
```

Try it Yourself »

Automatic toString()

JavaScript automatically converts an array to a comma separated string when a primitive value is expected.

This is always the case when you try to output an array.

These two examples will produce the same result:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

Try it Yourself »

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
```

Try it Yourself »

Note

All JavaScript objects have a toString() method.

JavaScript Booleans

A JavaScript Boolean represents one of two values: **true** or **false**.

Boolean Values

Very often, in programming, you will need a data type that can only have one of two values, like

- YES / NO
- ON / OFF
- TRUE / FALSE

For this, JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.

The Boolean() Function

You can use the `Boolean()` function to find out if an expression (or a variable) is true:

Example

```
Boolean(10 > 9)
```

Try it Yourself »

Or even easier:

Example

```
(10 > 9)
10 > 9
```

Try it Yourself »

Comparisons and Conditions

The chapter JS Comparisons gives a full overview of comparison operators.

The chapter JS Conditions gives a full overview of conditional statements.

Here are some examples:

Operator	Description	Example
==	equal to	if (day == "Monday")
>	greater than	if (salary > 9000)
<	less than	if (age < 18)

The Boolean value of an expression is the basis for all JavaScript comparisons and conditions.

Everything With a "Value" is True

Examples

```
100

3.14

-15

"Hello"

"false"

7 + 1 + 3.14
```

Try it Yourself »

Everything Without a "Value" is False

The Boolean value of **0** (zero) is **false**:

```
let x = 0;
Boolean(x);
```

Try it Yourself »

The Boolean value of **-0** (minus zero) is **false**:

```
let x = -0;
Boolean(x);
```

Try it Yourself »

The Boolean value of **""** (empty string) is **false**:

```
let x = "";
Boolean(x);
```

Try it Yourself »

The Boolean value of **undefined** is **false**:

```
let x;
Boolean(x);
```

Try it Yourself »

The Boolean value of **null** is **false**:

```
let x = null;
Boolean(x);
```

Try it Yourself »

The Boolean value of **false** is (you guessed it) **false**:

```
let x = false;
Boolean(x);
```

Try it Yourself »

The Boolean value of **NaN** is **false**:

```
let x = 10 / "Hallo";
Boolean(x);
```

Try it Yourself »

JavaScript Booleans as Objects

Normally JavaScript booleans are primitive values created from literals:

```
let x = false;
```

But booleans can also be defined as objects with the keyword `new`:

```
let y = new Boolean(false);
```

Example

```
let x = false;
let y = new Boolean(false);

// typeof x returns boolean
// typeof y returns object
```

Try it yourself »

Do not create Boolean objects.

The `new` keyword complicates the code and slows down execution speed.

Boolean objects can produce unexpected results:

When using the `==` operator, x and y are **equal**:

```
let x = false;
let y = new Boolean(false);
```

Try it Yourself »

When using the `===` operator, x and y are **not equal**:

```
let x = false;
let y = new Boolean(false);
```

Try it Yourself »

Note the difference between (x==y) and (x===y).

`(x == y)` true or false?

```
let x = new Boolean(false);
let y = new Boolean(false);
```

Try it Yourself »

`(x === y)` true or false?

```
let x = new Boolean(false);
let y = new Boolean(false);
```

Try it Yourself »

Comparing two JavaScript objects **always** return **false**.

JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for `true` or `false`.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that `x = 5`, the table below explains the comparison operators:

Operator	Description	Comparing	Returns	Try it
==	equal to	x == 8	false	Try it »
		x == 5	true	Try it »
		x == "5"	true	Try it »
===	equal value and equal type	x === 5	true	Try it »
		x === "5"	false	Try it »
!=	not equal	x != 8	true	Try it »
!==	not equal value or not equal type	x !== 5	false	Try it »
		x !== "5"	true	Try it »
		x !== 8	true	Try it »
>	greater than	x > 8	false	Try it »

<	less than	x < 8	true	Try it »
>=	greater than or equal to	x >= 8	false	Try it »
<=	less than or equal to	x <= 8	true	Try it »

How Can it be Used

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age < 18) text = "Too young to buy alcohol";
```

You will learn more about the use of conditional statements in the next chapter of this tutorial.

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that `x = 6` and `y = 3`, the table below explains the logical operators:

Operator	Description	Example	Try it
&&	and	(x < 10 && y > 1) is true	Try it »

||	or	(x == 5 || y == 5) is false	Try it »
!	not	!(x == y) is true	Try it »

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

```
variablename = (condition) ? value1:value2
```

Example

```
let voteable = (age < 18) ? "Too young":"Old enough";
```

Try it Yourself »

If the variable age is a value below 18, the value of the variable voteable will be "Too young", otherwise the value of voteable will be "Old enough".

Comparing Different Types

Comparing data of different types may give unexpected results.

When comparing a string with a number, JavaScript will convert the string to a number when doing the comparison. An empty string converts to 0. A non-numeric string converts to `NaN` which is always `false`.

Case	Value	Try

2 < 12	true	Try it »
2 < "12"	true	Try it »
2 < "John"	false	Try it »
2 > "John"	false	Try it »
2 == "John"	false	Try it »
"2" < "12"	false	Try it »
"2" > "12"	true	Try it »
"2" == "12"	false	Try it »

When comparing two strings, "2" will be greater than "12", because (alphabetically) 1 is less than 2.

To secure a proper result, variables should be converted to the proper type before comparison:

```
age = Number(age);
if (isNaN(age)) {
  voteable = "Input is not a number";
} else {
  voteable = (age < 18) ? "Too young" : "Old enough";
}
```

Try it Yourself »

The Nullish Coalescing Operator (??)

The `??` operator returns the first argument if it is not **nullish** (`null` or `undefined`).

Otherwise it returns the second argument.

Example

```
let name = null;
let text = "missing";
let result = name ?? text;
```

Try it Yourself »

The nullish operator is supported in all browsers since March 2020:

Chrome 80	Edge 80	Firefox 72	Safari 13.1	Opera 67
Feb 2020	Feb 2020	Jan 2020	Mar 2020	Mar 2020

The Optional Chaining Operator (?.)

The `?.` operator returns `undefined` if an object is `undefined` or `null` (instead of throwing an error).

Example

```
// Create an object:
const car = {type:"Fiat", model:"500", color:"white"};
// Ask for car name:
document.getElementById("demo").innerHTML = car?.name;
```

Try it Yourself »

The optional chaining operator is supported in all browsers since March 2020:

Chrome 80	Edge 80	Firefox 72	Safari 13.1	Opera 67
Feb 2020	Feb 2020	Jan 2020	Mar 2020	Mar 2020

MODULE-II

Control Structures:

Conditions, Switch, Loop For, Loop While, Break

Functions:

Function Definitions, Function Parameters, Function Invocation, Function Closures.

Objects:

Object Definitions, Object Properties, Object Methods, Object Prototypes

Object Oriented Programming:

Method,Constructor,Inheritance,Encapsulation,Abstraction,Polymorphism,Javascript Validations, Document Object Model, Document and Events (DOM Manipulation).

Control Structures

Control structure actually controls the flow of execution of a program. Following are the several control structure supported by javascript.

- if … else
- switch case
- do while loop
- while loop
- for loop

If … else

The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

```
if (expression){
  Statement(s) to be executed if expression is true
}
```

Example

```
<script type="text/javascript">
  <!--
    var age = 20;
    if( age > 18 ){
      document.write("<b>Qualifies for driving</b>");
    }
  //-->
</script>
```

Switch case

The basic syntax of the switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

Syntax

```
switch (expression) {
  case condition 1: statement(s)
            break;
  case condition 2: statement(s)
            break;
  ...
  case condition n: statement(s)
            break;
  default: statement(s)
}
```

Example

```
<script type="text/javascript">
  <!--
```

```
    var grade='A';
    document.write("Entering switch block<br/>");
    switch (grade) {
      case 'A': document.write("Good job<br/>");
        break;
      case 'B': document.write("Pretty good<br/>");
        break;
      case 'C': document.write("Passed<br/>");
        break;
      case 'D': document.write("Not so good<br/>");
        break;
      case 'F': document.write("Failed<br/>");
        break;
      default:  document.write("Unknown grade<br/>")
    }
    document.write("Exiting switch block");
  //-->
</script>
```

Do while Loop

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

Syntax

```
do{
  Statement(s) to be executed;
} while (expression);
```

Example

```
<script type="text/javascript">
  <!--
    var count = 0;
    document.write("Starting Loop" + "<br/>");
    do{
      document.write("Current Count : " + count + "<br/>");
      count++;
    }while (count < 0);
    document.write("Loop stopped!");
  //-->
</script>
```

This will produce following result –

```
Starting Loop
Current Count : 0
Loop stopped!
```

While Loop

The purpose of a while loop is to execute a statement or code block repeatedly as long as expression is true. Once expression becomes false, the loop will be exited.

Syntax

```
while (expression){
  Statement(s) to be executed if expression is true
}
```

Example

```
<script type="text/javascript">
  <!--
    var count = 0;
    document.write("Starting Loop" + "<br/>");
    while (count < 10){
      document.write("Current Count : " + count + "<br/>");
      count++;
    }
    document.write("Loop stopped!");
  //-->
</script>
```

This will produce following result –

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

For Loop

The for loop is the most compact form of looping and includes the following three important parts –

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.

Syntax

```
for (initialization; test condition; iteration statement){
  Statement(s) to be executed if test condition is true
}
```

Example

```
<script type="text/javascript">
  <!--
    var count;
    document.write("Starting Loop" + "<br/>");
    for(count = 0; count < 10; count++){
      document.write("Current Count : " + count );
      document.write("<br/>");
    }
    document.write("Loop stopped!");
  //-->
</script>
```

This will produce following result which is similar to while loop –

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

Creating Sample Program

Following is the sample program that shows time, when we click in button.

```
<html>
  <body>
    <button onclick="this.innerHTML=Date()">The time is?</button>
    <p>Click to display the date.</p>
    <button onclick="displayDate()">The time is?</button>
    <script>
      function displayDate() {
        document.getElementById("demo").innerHTML = Date();
      }</script>

      <p id="demo"></p>
    </script>
  </body>
</html>
```

Output:

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

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

Example

```
// Function to compute the product of p1 and p2
function myFunction(p1, p2) {
  return p1 * p2;
}
```

Try it Yourself »

JavaScript Function Syntax

A JavaScript function is defined with the `function` keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: **(*parameter1, parameter2, ...*)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local

JavaScript Function Parameters

A JavaScript `function` does not perform any checking on parameter values (arguments).

Function Parameters and Arguments

Earlier in this tutorial, you learned that functions can have **parameters**:

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

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** passed to (and received by) the function.

Parameter Rules

JavaScript function definitions do not specify data types for parameters.

JavaScript functions do not perform type checking on the passed arguments.

JavaScript functions do not check the number of arguments received.

Default Parameters

If a function is called with **missing arguments** (less than declared), the missing values are set to `undefined`.

Sometimes this is acceptable, but sometimes it is better to assign a default value to the parameter:

Example

```
function myFunction(x, y) {
  if (y === undefined) {
    y = 2;
  }
}
```

Try it Yourself »

Default Parameter Values

ES6 allows function parameters to have default values.

Example

If y is not passed or undefined, then y = 10.

```
function myFunction(x, y = 10) {
  return x + y;
}
myFunction(5);
```

Try it Yourself »

Function Rest Parameter

The rest parameter (...) allows a function to treat an indefinite number of arguments as an array:

Example

```
function sum(...args) {
  let sum = 0;
  for (let arg of args) sum += arg;
  return sum;
}

let x = sum(4, 9, 16, 25, 29, 100, 66, 77);
```

Try it Yourself »

The Arguments Object

JavaScript functions have a built-in object called the arguments object.

The argument object contains an array of the arguments used when the function was called (invoked).

This way you can simply use a function to find (for instance) the highest value in a list of numbers:

Example

```
x = findMax(1, 123, 500, 115, 44, 88);

function findMax() {
  let max = -Infinity;
  for (let i = 0; i < arguments.length; i++) {
    if (arguments[i] > max) {
      max = arguments[i];
    }
  }
  return max;
}
```

Try it Yourself »

Or create a function to sum all input values:

Example

```
x = sumAll(1, 123, 500, 115, 44, 88);

function sumAll() {
  let sum = 0;
  for (let i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
}
```

Try it Yourself »

If a function is called with **too many arguments** (more than declared), these arguments can be reached using **the arguments object**.

Arguments are Passed by Value

The parameters, in a function call, are the function's arguments.

JavaScript arguments are passed by **value**: The function only gets to know the values, not the argument's locations.

If a function changes an argument's value, it does not change the parameter's original value.

Changes to arguments are not visible (reflected) outside the function.

Objects are Passed by Reference

In JavaScript, object references are values.

Because of this, objects will behave like they are passed by **reference:**

If a function changes an object property, it changes the original value.

Changes to object properties are visible (reflected) outside the function.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

Function Return

When JavaScript reaches a `return` statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example

Calculate the product of two numbers, and return the result:

```
let x = myFunction(4, 3);   // Function is called, return value will
end up in x

function myFunction(a, b) {
  return a * b;             // Function returns the product of a and b
}
```

The result in x will be:

```
12
```

Try it Yourself »

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Example

Convert Fahrenheit to Celsius:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

Try it Yourself »

The () Operator Invokes the Function

Using the example above, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result.

Accessing a function without () will return the function object instead of the function result.

Example

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

Try it Yourself »

Functions Used as Variable Values

Functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations.

Example

Instead of using a variable to store the return value of a function:

```
let x = toCelsius(77);
let text = "The temperature is " + x + " Celsius";
```

You can use the function directly, as a variable value:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

Try it Yourself »

You will learn a lot more about functions later in this tutorial.

Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function.

Local variables can only be accessed from within the function.

Example

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Try it Yourself »

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

JavaScript Closures

JavaScript variables can belong to the **local** or **global** scope.

Global variables can be made local (private) with **closures**.

Global Variables

A `function` can access all variables defined **inside** the function, like this:

Example

```
function myFunction() {
  let a = 4;
  return a * a;
}
```

Try it Yourself »

But a `function` can also access variables defined **outside** the function, like this:

Example

```
let a = 4;
function myFunction() {
  return a * a;
}
```

Try it Yourself »

In the last example, **a** is a **global** variable.

In a web page, global variables belong to the page.

Global variables can be used (and changed) by all other scripts in the page.

In the first example, **a** is a **local** variable.

A local variable can only be used inside the function where it is defined. It is hidden from other functions and other scripting code.

Global and local variables with the same name are different variables. Modifying one, does not modify the other.

Note

Variables created **without** a declaration keyword (`var`, `let`, or `const`) are always global, even if they are created inside a function.

Example

```
function myFunction() {
  a = 4;
}
```

Try it Yourself »

Variable Lifetime

Global variables live until the page is discarded, like when you navigate to another page or close the window.

Local variables have short lives. They are created when the function is invoked, and deleted when the function is finished.

A Counter Dilemma

Suppose you want to use a variable for counting something, and you want this counter to be available to all functions.

You could use a global variable, and a `function` to increase the counter:

Example

```
// Initiate counter
let counter = 0;

// Function to increment counter
function add() {
  counter += 1;
}

// Call add() 3 times
add();
add();
add();

// The counter should now be 3
```

Try it Yourself »

There is a problem with the solution above: Any code on the page can change the counter, without calling add().

The counter should be local to the `add()` function, to prevent other code from changing it:

Example

```
// Initiate counter
let counter = 0;

// Function to increment counter
function add() {
  let counter = 0;
  counter += 1;
}

// Call add() 3 times
add();
add();
add();
```

```
//The counter should now be 3. But it is 0
```

Try it Yourself »

It did not work because we display the global counter instead of the local counter.

We can remove the global counter and access the local counter by letting the function return it:

Example

```
// Function to increment counter
function add() {
  let counter = 0;
  counter += 1;
  return counter;
}

// Call add() 3 times
add();
add();
add();

//The counter should now be 3. But it is 1.
```

Try it Yourself »

It did not work because we reset the local counter every time we call the function.

A JavaScript inner function can solve this.

JavaScript Nested Functions

All functions have access to the global scope.

In fact, in JavaScript, all functions have access to the scope "above" them.

JavaScript supports nested functions. Nested functions have access to the scope "above" them.

In this example, the inner function `plus()` has access to the `counter` variable in the parent function:

Example

```
function add() {
  let counter = 0;
  function plus() {counter += 1;}
  plus();
  return counter;
}
```

Try it Yourself »

This could have solved the counter dilemma, if we could reach the `plus()` function from the outside.

We also need to find a way to execute `counter = 0` only once.

We need a closure.

JavaScript Closures

Remember self-invoking functions? What does this function do?

Example

```
const add = (function () {
  let counter = 0;
  return function () {counter += 1; return counter}
})();

add();
add();
add();

// the counter is now 3
```

Try it Yourself »

Example Explained

The variable `add` is assigned to the return value of a self-invoking function.

The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.

This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.

This is called a JavaScript **closure.** It makes it possible for a function to have "**private**" variables.

The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

A closure is a function having access to the parent scope, even after the parent function has closed.

JavaScript Objects

Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties
	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white

All cars have the same **properties**, but the property **values** differ from car to car.

All cars have the same **methods**, but the methods are performed **at different times**.

JavaScript Objects

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

```
let car = "Fiat";
```

Try it Yourself »

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

```
const car = {type:"Fiat", model:"500", color:"white"};
```

Try it Yourself »

The values are written as **name:value** pairs (name and value separated by a colon).

It is a common practice to declare objects with the `const` keyword.

Learn more about using `const` with objects in the chapter: JS Const.

Object Definition

You define (and create) a JavaScript object with an object literal:

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 »

Object Properties

The **name:values** pairs in JavaScript objects are called **properties**:

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

or

objectName["propertyName"]

Example1

```
person.lastName;
```

Try it Yourself »

Example2

```
person["lastName"];
```

Try it Yourself »

JavaScript objects are containers for **named values** called properties.

Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Property	Property Value
firstName	John
lastName	Doe
age	50

eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

A method is a function stored as a property.

Example

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

In the example above, `this` refers to the **person object**.

I.E. **this.firstName** means the **firstName** property of **this**.

I.E. **this.firstName** means the **firstName** property of **person**.

What is **this**?

In JavaScript, the `this` keyword refers to an **object**.

Which object depends on how `this` is being invoked (used or called).

The `this` keyword refers to different objects depending on how it is used:

In an object method, `this` refers to the **object**.
Alone, `this` refers to the **global object**.

In a function, `this` refers to the **global object**.
In a function, in strict mode, `this` is `undefined`.
In an event, `this` refers to the **element** that received the event.
Methods like `call()`, `apply()`, and `bind()` can refer `this` to **any object**.

Note

`this` is not a variable. It is a keyword. You cannot change the value of `this`.

See Also:

The JavaScript **this** Tutorial

The **this** Keyword

In a function definition, `this` refers to the "owner" of the function.

In the example above, `this` is the **person object** that "owns" the `fullName` function.

In other words, `this.firstName` means the `firstName` property of **this object**.

Learn more about `this` in The JavaScript this Tutorial.

Accessing Object Methods

You access an object method with the following syntax:

`objectName.methodName()`

Example

```
name = person.fullName();
```

Try it Yourself »

If you access a method **without** the () parentheses, it will return the **function definition**:

Example

```
name = person.fullName;
```

Try it Yourself »

JavaScript Object Prototypes

All JavaScript objects inherit properties and methods from a prototype.

In the previous chapter we learned how to use an **object constructor**:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}

const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
```

Try it Yourself »

We also learned that you can **not** add a new property to an existing object constructor:

Example

```
Person.nationality = "English";
```

Try it Yourself »

To add a new property to a constructor, you must add it to the constructor function:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.nationality = "English";
}
```

Try it Yourself »

Prototype Inheritance

All JavaScript objects inherit properties and methods from a prototype:

- `Date` objects inherit from `Date.prototype`
- `Array` objects inherit from `Array.prototype`
- `Person` objects inherit from `Person.prototype`

The `Object.prototype` is on the top of the prototype inheritance chain:

`Date` objects, `Array` objects, and `Person` objects inherit from `Object.prototype`.

Adding Properties and Methods to Objects

Sometimes you want to add new properties (or methods) to all existing objects of a given type.

Sometimes you want to add new properties (or methods) to an object constructor.

Using the **prototype** Property

The JavaScript `prototype` property allows you to add new properties to object constructors:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}

Person.prototype.nationality = "English";
```

Try it Yourself »

The JavaScript `prototype` property also allows you to add new methods to objects constructors:

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
}

Person.prototype.name = function() {
  return this.firstName + " " + this.lastName;
};
```

Try it Yourself »

Only modify your **own** prototypes. Never modify the prototypes of standard JavaScript objects.

Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "`new`", the variable is created as an object:

```
x = new String();        // Declares x as a String object
y = new Number();        // Declares y as a Number object
z = new Boolean();       // Declares z as a Boolean object
```

Avoid `String`, `Number`, and `Boolean` objects. They complicate your code and slow down execution speed.

Introduction to Object Oriented Programming in JavaScript

- Difficulty Level : Easy
- Last Updated : 08 Dec, 2022

- Read
- Discuss
- Courses
- Practice
- Video

As JavaScript is widely used in Web Development, in this article we will explore some of the **Object Oriented** mechanisms supported by **JavaScript** to get the most out of it. Some of the common interview questions in JavaScript on OOPS include:

- How is Object-Oriented Programming implemented in JavaScript?
- How does it differ from other languages?
- Can you implement Inheritance in JavaScript?

and so on...

There are certain features or mechanisms which make a Language Object-Oriented like:

OOPs Concept in JavaScript		
Object	**Classes**	**Encapsulation**
Abstraction	**Inheritance**	**Polymorphism**

Let's dive into the details of each one of them and see how they are implemented in JavaScript.

Object: An Object is a **unique** entity that contains **properties** and **methods**. For example "a car" is a real-life Object, which has some characteristics like color, type, model, and horsepower and performs certain actions like driving. The characteristics of an Object are called Properties in Object-Oriented Programming and the actions are called methods. An Object is an **instance** of a class. Objects are everywhere in JavaScript, almost every element is an Object whether it is a function, array, or string.

Note: A Method in javascript is a property of an object whose value is a function.

The object can be created in two ways in JavaScript:

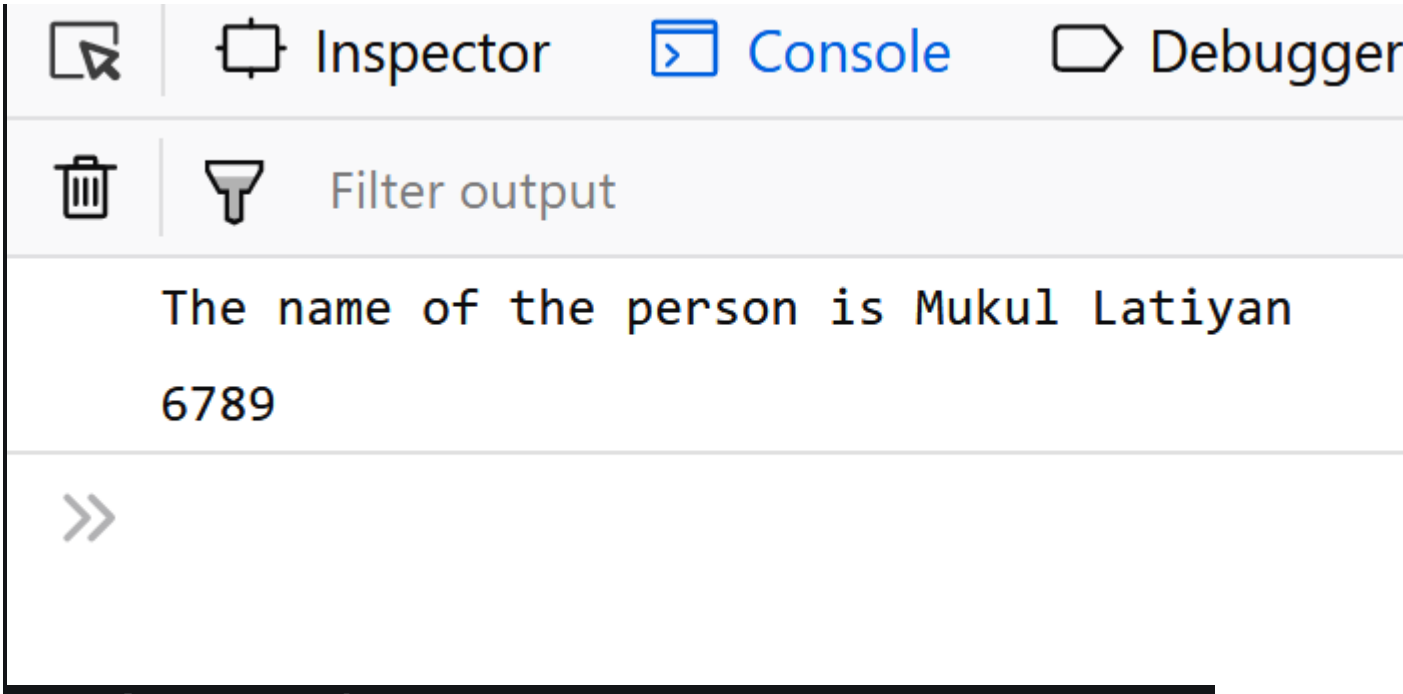
- **Object Literal**
- **Object Constructor**

Example: Using an Object Literal.

- Javascript

```
// Defining object
let person = {
    first_name:'Mukul',
    last_name: 'Latiyan',

    //method
    getFunction : function(){
        return (`The name of the person is
          ${person.first_name} ${person.last_name}`)
    },
    //object within object
    phone_number : {
        mobile:'12345',
        landline:'6789'
    }
}
console.log(person.getFunction());
console.log(person.phone_number.landline);
```

Output:

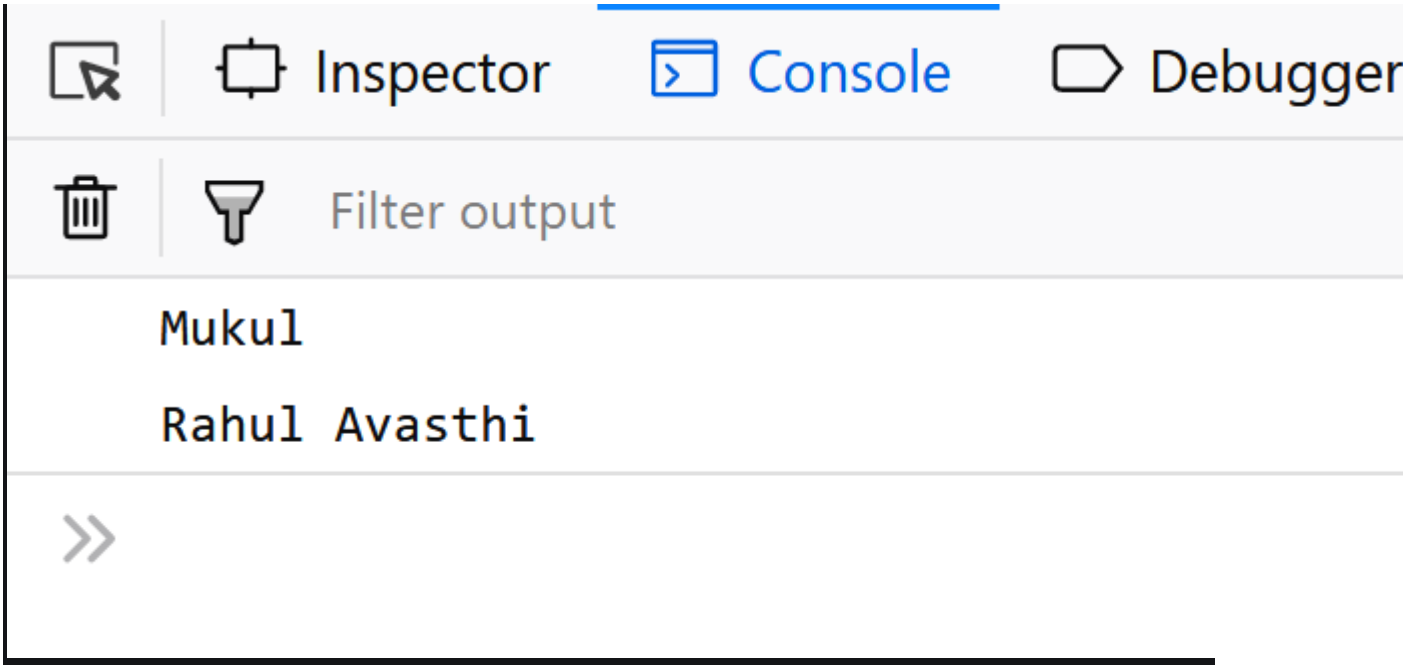


Example: Using an Object Constructor.

- Javascript

```
// Using a constructor
function person(first_name,last_name){
   this.first_name = first_name;
   this.last_name = last_name;
}
// Creating new instances of person object
let person1 = new person('Mukul','Latiyan');
let person2 = new person('Rahul','Avasthi');

console.log(person1.first_name);
console.log(`${person2.first_name} ${person2.last_name}`);
```

Output:



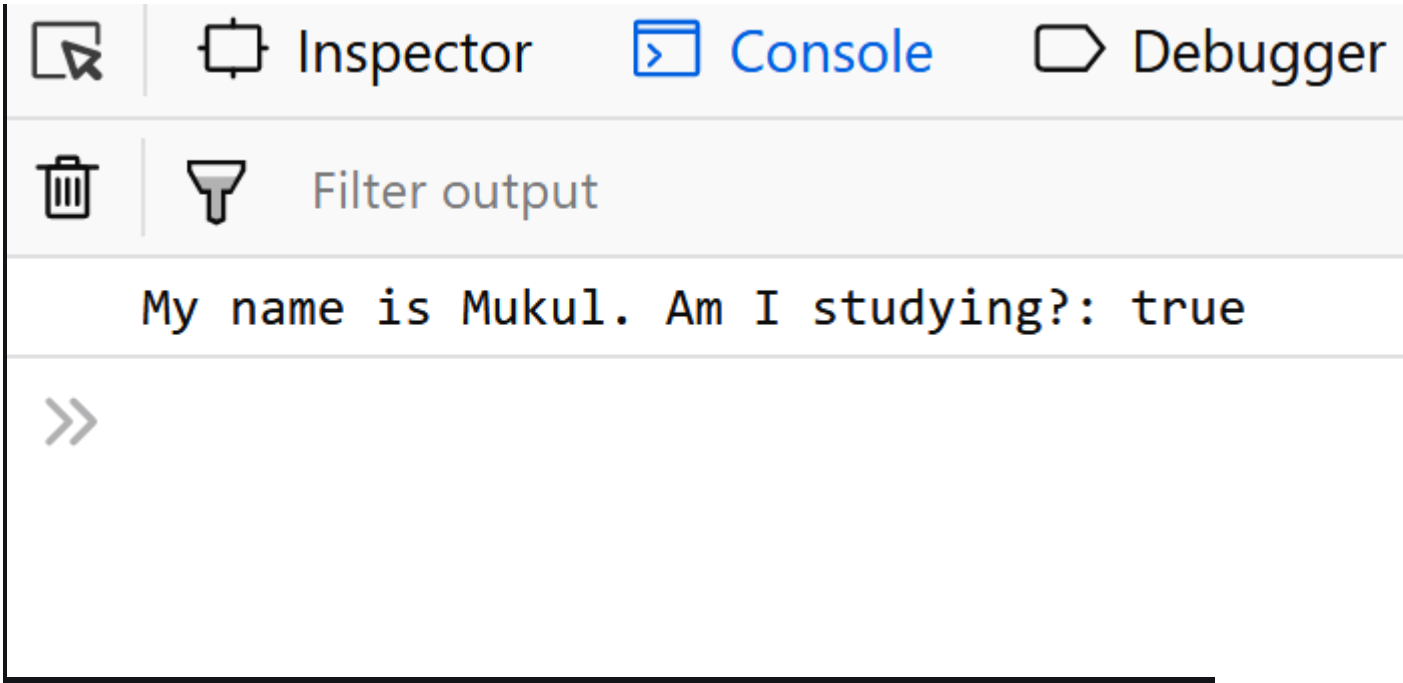
Note: The JavaScript Object.create() Method creates a new object, using an existing object as the prototype of the newly created object.

Example:

- Javascript

```
// Object.create() example a
// simple object with some properties
const coder = {
    isStudying : false,
    printIntroduction : function(){
        console.log(`My name is ${this.name}. Am I
          studying?: ${this.isStudying}.`)
    }
}
// Object.create() method
const me = Object.create(coder);

// "name" is a property set on "me", but not on "coder"
me.name = 'Mukul';

// Inherited properties can be overwritten
me.isStudying = true;

me.printIntroduction();
```

Output:



Classes: Classes are **blueprints** of an Object. A class can have many Objects because the class is a **template** while Objects are **instances** of the class or the concrete implementation.
Before we move further into implementation, we should know unlike other Object Oriented languages there are **no classes in JavaScript** we have only Object. To be more precise, JavaScript is a prototype-based Object Oriented Language, which means it doesn't have classes, rather it defines behaviors using a constructor function and then reuses it using the prototype.
Note: Even the classes provided by ECMA2015 are objects.
JavaScript classes, introduced in ECMAScript 2015, are primarily syntactical sugar over JavaScript's existing prototype-based inheritance. The class syntax is not introducing a new object-oriented inheritance model to JavaScript. JavaScript classes provide a much simpler and clearer syntax to create objects and deal with inheritance.

-Mozilla Developer Network

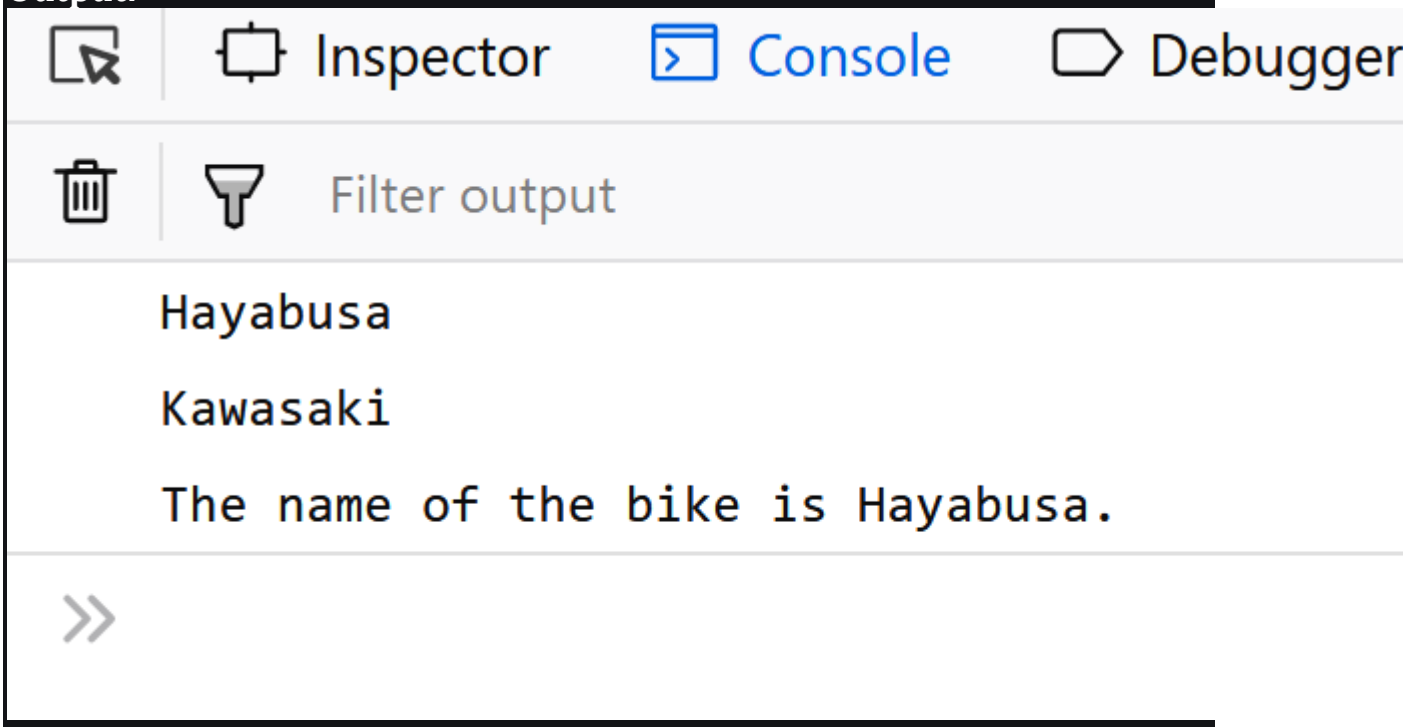
Example: Let's use ES6 classes then we will look at the traditional way of defining an Object and simulate them as classes.

- Javascript

```
// Defining class using es6
class Vehicle {
  constructor(name, maker, engine) {
    this.name = name;
    this.maker =  maker;
    this.engine = engine;
```

```
    }
    getDetails(){
        return (`The name of the bike is ${this.name}.`)
    }
}
// Making object with the help of the constructor
let bike1 = new Vehicle('Hayabusa', 'Suzuki', '1340cc');
let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');

console.log(bike1.name);    // Hayabusa
console.log(bike2.maker);   // Kawasaki
console.log(bike1.getDetails());
```

Output:



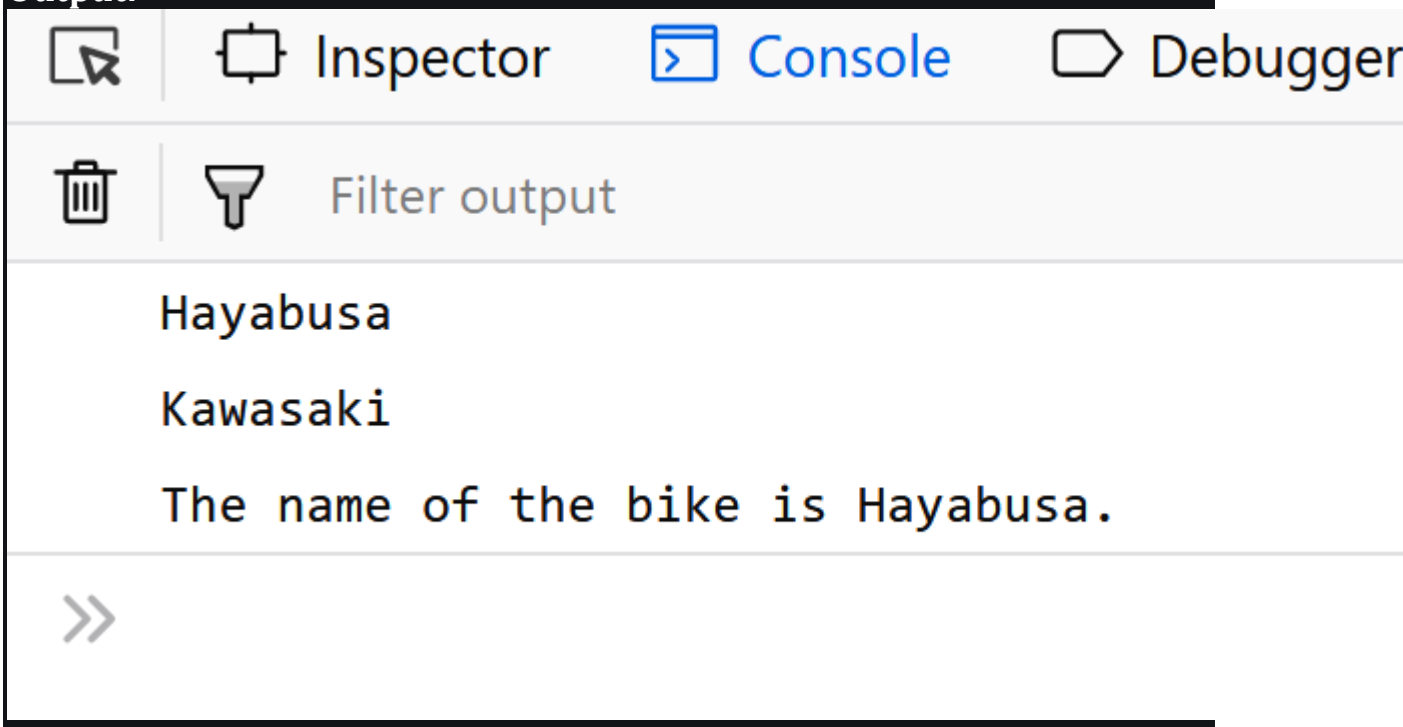
Example: Traditional Way of defining an Object and simulating them as classes.

- Javascript

```
// Defining class in a Traditional Way.
function Vehicle(name,maker,engine){
    this.name = name,
    this.maker = maker,
    this.engine = engine
};

Vehicle.prototype.getDetails = function(){
```

```
        console.log('The name of the bike is '+ this.name);
    }

    let bike1 = new Vehicle('Hayabusa','Suzuki','1340cc');
    let bike2 = new Vehicle('Ninja','Kawasaki','998cc');

    console.log(bike1.name);
    console.log(bike2.maker);
    console.log(bike1.getDetails());
```

Output:



As seen in the above example it is much simpler to define and reuse objects in ES6. Hence, we would be using ES6 in all of our examples.

Abstraction: Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Encapsulation: The process of **wrapping properties and functions** within a **single unit** is known as encapsulation.

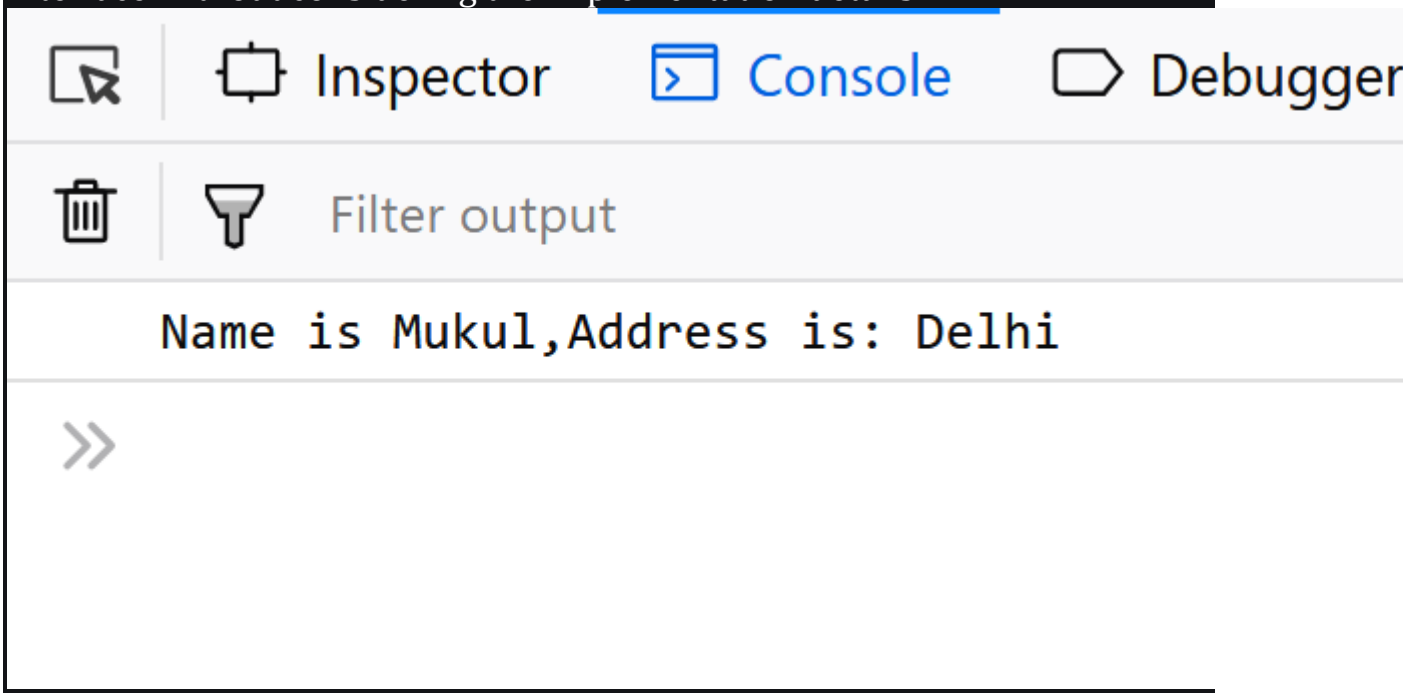
Example: Let's understand encapsulation with an example.

- Javascript

```
    // Encapsulation example
    class person{
        constructor(name,id){
```

```
            this.name = name;
            this.id = id;
        }
        add_Address(add){
            this.add = add;
        }
        getDetails(){
            console.log(`Name is ${this.name},
            Address is: ${this.add}`);
        }
    }

    let person1 = new person('Mukul',21);
    person1.add_Address('Delhi');
    person1.getDetails();
```

Output: In this example, we simply create a person Object using the constructor, Initialize its properties and use its functions. We are not bothered by the implementation details. We are working with an Object's interface without considering the implementation details.



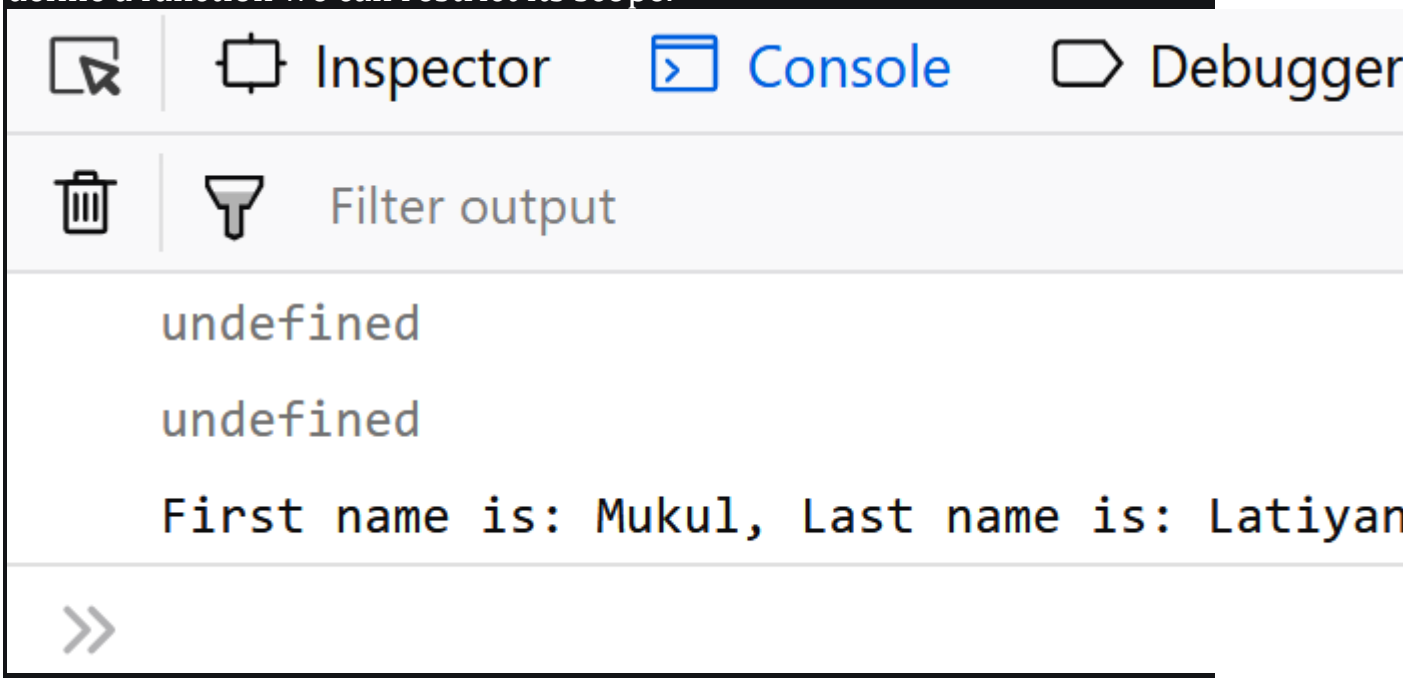
Sometimes encapsulation refers to the **hiding of data** or **data Abstraction** which means representing essential features hiding the background detail. Most of the OOP languages provide access modifiers to restrict the scope of a variable, but there are no such access modifiers in JavaScript, there are certain ways by which we can restrict the scope of variables within the Class/Object.

Example:

- Javascript

```
// Abstraction example
function person(fname,lname){
    let firstname = fname;
    let lastname = lname;

    let getDetails_noaccess = function(){
        return (`First name is: ${firstname} Last
            name is: ${lastname}`);
    }

    this.getDetails_access = function(){
        return (`First name is: ${firstname}, Last
            name is: ${lastname}`);
    }
}
let person1 = new person('Mukul','Latiyan');
console.log(person1.firstname);
console.log(person1.getDetails_noaccess);
console.log(person1.getDetails_access());
```

Output: In this example, we try to access some property(person1.firstname) and functions(person1.getDetails_noaccess) but it returns undefined while there is a method that we can access from the person object(person1.getDetails_access()). By changing the way we define a function we can restrict its scope.



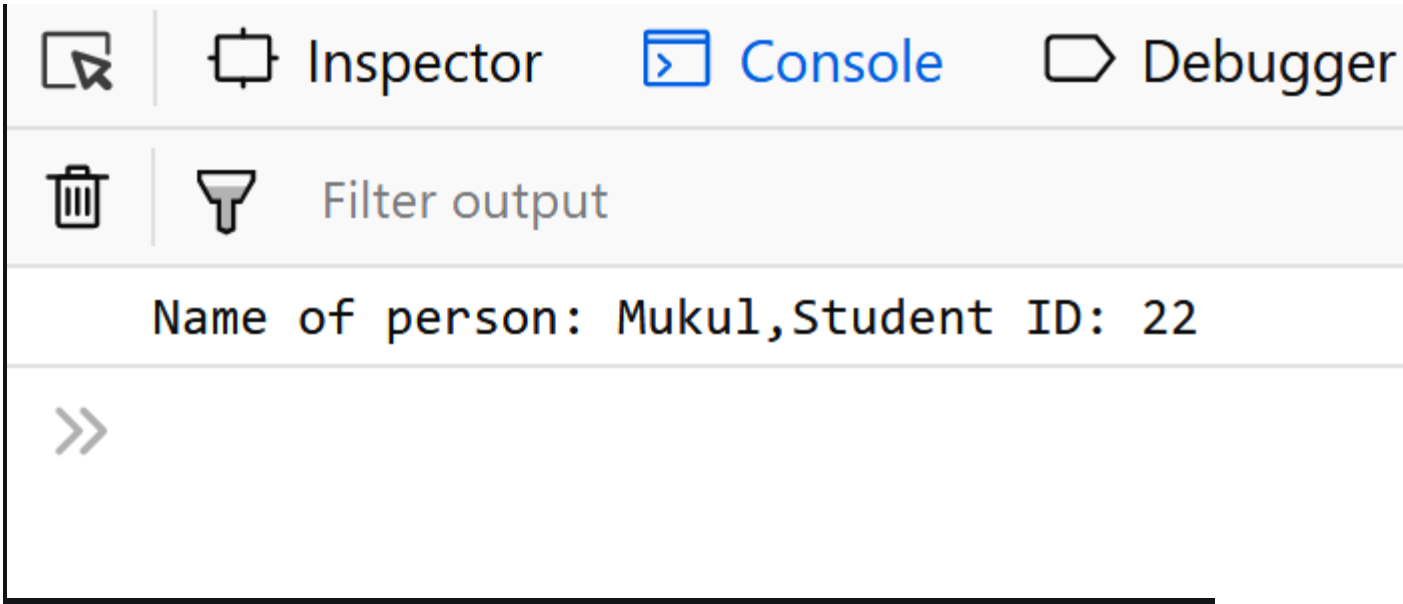
Inheritance: It is a concept in which some properties and methods of an Object are being used by another Object. Unlike most of the OOP languages where classes inherit classes, JavaScript Objects inherit Objects i.e. certain

features (property and methods) of one object can be reused by other Objects.
Example: Let's understand inheritance and polymorphism with an example.

- Javascript

```
// Inheritance example
class person{
    constructor(name){
        this.name = name;
    }
    // method to return the string
    toString(){
        return (`Name of person: ${this.name}`);
    }
}
class student extends person{
    constructor(name,id){
        // super keyword for calling the above
        // class constructor
        super(name);
        this.id = id;
    }
    toString(){
        return (`${super.toString()},
        Student ID: ${this.id}`);
    }
}
let student1 = new student('Mukul',22);
console.log(student1.toString());
```

Output: In this example, we define a Person Object with certain properties and methods and then we inherit the Person Object in the Student Object and use all the properties and methods of the person Object as well as define certain properties and methods for the Student Object.



Note: The Person and Student objects both have the same method (i.e toString()), this is called **Method Overriding**. Method Overriding allows a method in a child class to have the same name(polymorphism) and method signature as that of a parent class.
In the above code, the super keyword is used to refer to the immediate parent class's instance variable.

Polymorphism: Polymorphism is one of the core concepts of object-oriented programming languages. Polymorphism means the same function with different signatures is called many times. In real life, for example, a boy at the same time may be a student, a class monitor, etc. So a boy can perform different operations at the same time. Polymorphism can be achieved by method overriding and method overloading

JavaScript Forms

JavaScript Form Validation

HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

JavaScript Example

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
```

```
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

The function can be called when the form is submitted:

HTML Form Example

```
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

Try it Yourself »

JavaScript Can Validate Numeric Input

JavaScript is often used to validate numeric input:

Please input a number between 1 and 10

Submit

Try it Yourself »

Automatic HTML Form Validation

HTML form validation can be performed automatically by the browser:

If a form field (fname) is empty, the `required` attribute prevents this form from being submitted:

HTML Form Example

```
<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
```

Try it Yourself »

Automatic HTML form validation does not work in Internet Explorer 9 or earlier.

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

Server side validation is performed by a web server, after input has been sent to the server.

Client side validation is performed by a web browser, before input is sent to a web server.

HTML Constraint Validation

HTML5 introduced a new HTML validation concept called **constraint validation**.

HTML constraint validation is based on:

- Constraint validation **HTML Input Attributes**
- Constraint validation **CSS Pseudo Selectors**
- Constraint validation **DOM Properties and Methods**

Constraint Validation HTML Input Attributes

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

For a full list, go to HTML Input Attributes.

Constraint Validation CSS Pseudo Selectors

Selector	Description

:disabled	Selects input elements with the "disabled" attribute specified
:invalid	Selects input elements with invalid values
:optional	Selects input elements with no "required" attribute specified
:required	Selects input elements with the "required" attribute specified
:valid	Selects input elements with valid values

JavaScript HTML DOM

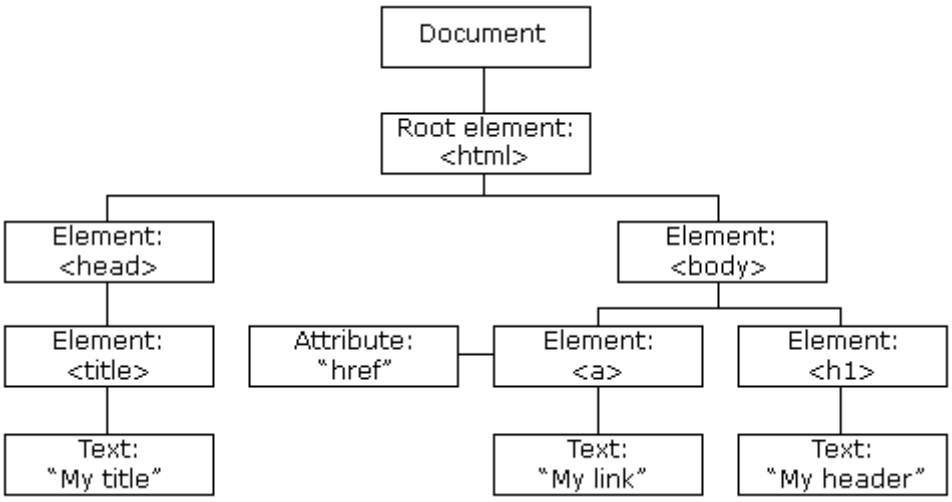
With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

What You Will Learn

In the next chapters of this tutorial you will learn:

- How to change the content of HTML elements
- How to change the style (CSS) of HTML elements
- How to react to HTML DOM events
- How to add and delete HTML elements

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

JavaScript HTML DOM Document

❮ Previous Next ❯

The HTML DOM document object is the owner of all other objects in your web page.

The HTML DOM Document Object

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	**Description**
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

---

Changing HTML Elements

Property	**Description**
element.innerHTML = *new html content*	Change the inner HTML of an element
element.*attribute* = *new value*	Change the attribute value of an HTML element

element.style.*property = new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

---

Adding and Deleting Elements

Method	**Description**
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

---

Finding HTML Objects

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

Later, in HTML DOM Level 3, more objects, collections, and properties were added.

Property	Description	DOM
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Deprecated	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1

document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete.	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3

document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

JavaScript HTML DOM Events

HTML DOM allows JavaScript to react to HTML events:

Mouse Over Me

Click Me

Reacting to Events

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

`onclick=`*`JavaScript`*

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

In this example, the content of the `<h1>` element is changed when a user clicks on it:

Example

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

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>

</body>
</html>
```

Try it Yourself »

In this example, a function is called from the event handler:

Example

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

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

Try it Yourself »

HTML Event Attributes

To assign events to HTML elements you can use event attributes.

Example

Assign an onclick event to a button element:

```
<button onclick="displayDate()">Try it</button>
```

Try it Yourself »

In the example above, a function named `displayDate` will be executed when the button is clicked.

Assign Events Using the HTML DOM

The HTML DOM allows you to assign events to HTML elements using JavaScript:

Example

Assign an onclick event to a button element:

```
<script>
document.getElementById("myBtn").onclick = displayDate;
</script>
```

Try it Yourself »

In the example above, a function named `displayDate` is assigned to an HTML element with the `id="myBtn"`.

The function will be executed when the button is clicked.

The onload and onunload Events

The `onload` and `onunload` events are triggered when the user enters or leaves the page.

The `onload` event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The `onload` and `onunload` events can be used to deal with cookies.

Example

```
<body onload="checkCookies()">
```

Try it Yourself »

The onchange Event

The `onchange` event is often used in combination with validation of input fields.

Below is an example of how to use the onchange. The `upperCase()` function will be called when a user changes the content of an input field.

Example

```
<input type="text" id="fname" onchange="upperCase()">
```

Try it Yourself »

The onmouseover and onmouseout Events

The `onmouseover` and `onmouseout` events can be used to trigger a function when the user mouses over, or out of, an HTML element:

Mouse Over Me

Try it Yourself »

The onmousedown, onmouseup and onclick Events

The `onmousedown`, `onmouseup`, and `onclick` events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered, finally, when the mouse-click is completed, the onclick event is triggered.

Thank You

Try it Yourself »

MODULE-2

HTML EVENTS &DOM

Events

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

EVENTS

An event is an action that occurs in the web browser, which the web browser feedbacks to you so that you can respond to it.

For example, when users click a button on a webpage, you may want to respond to this click event by displaying a dialog box.

Each event may have an event handler which is a block of code that will execute when the event occurs.

An event handler is also known as an event listener. It listens to the event and executes when the event occurs.

- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.
- Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.
- Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Some of the HTML events and their event handlers are:

→Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

→Keyboard events

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

→Form Events

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

→Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

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.

There are some javascript events:

1) onclick events
2) onkeyup event
3) onmouseover event
4) onmouseout event
5) onchange event
6) onload event
7) onfocus event
8) onblur event

- when events happen, you may want to do something. JavaScript lets you execute code when events are detected.

- HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

❑With single quotes:

```
<element event='some JavaScript'>
```

❑With double quotes:

```
<element event="some JavaScript">
```

- In the following example, an onclick attribute (with code), is added to a <button> element:

Example

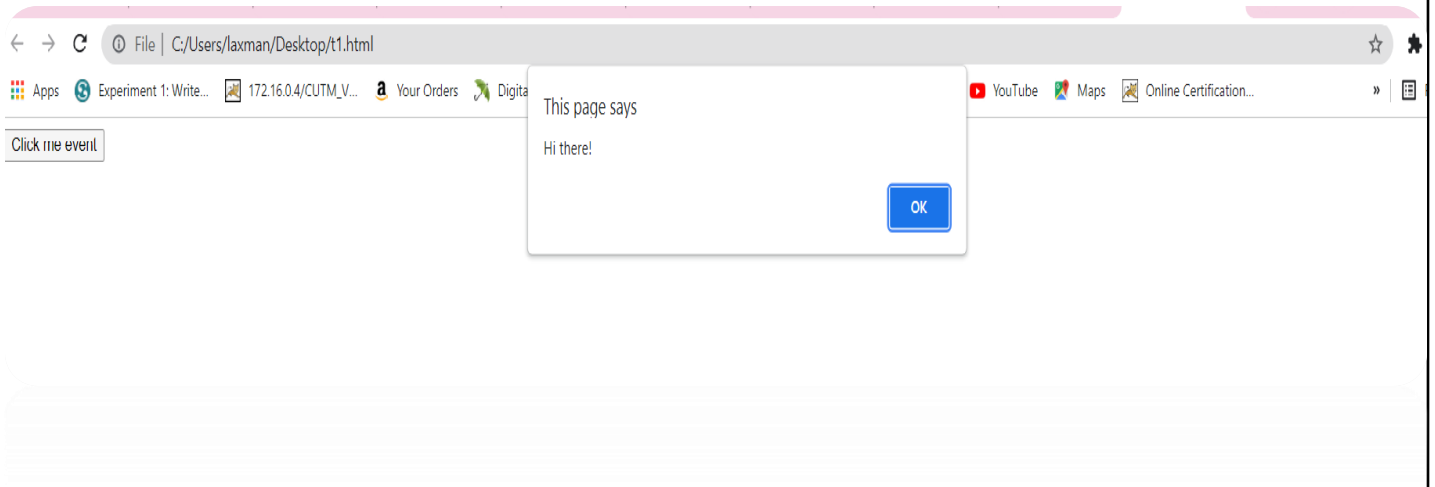
```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

1) onclick events: This is a mouse event and provokes any logic defined if the user clicks on the element it is bound to.

EXAMPLE:

```
<!doctype html>
<html>
  <head>
    <script>
      function hiThere() {
        alert('Hi there!');
      }
    </script>
  </head>
  <body>
    <button type="button" onclick="hiThere()">Click me event</button>
  </body>
</html>
```

OUTPUT:



Example : onclick event

```
<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
   <!--
   function clickevent()
   {
      document.write("This is centurion university");
   }
   //-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>
```

2) **onkeyup event:** This event is a keyboard event and executes instructions whenever a key is released after pressing.

```
<!doctype html>
<html>  <head>
   <script>
     var a = 0;
     var b = 0;
     var c = 0;
     function changeBackground() {
       var x = document.getElementById('bg');
       bg.style.backgroundColor = 'rgb('+a+', '+b+', '+c+')';
       a += 1;
       b += a + 1;
       c += b + 1;
       if (a > 255) a = a - b;
       if (b > 255) b = a;
       if (c > 255) c = b;
     }
   </script>  </head>  <body>
   <input id="bg" onkeyup="changeBackground()"
     placeholder="write something" style="color:#fff">
  </body></html>
```

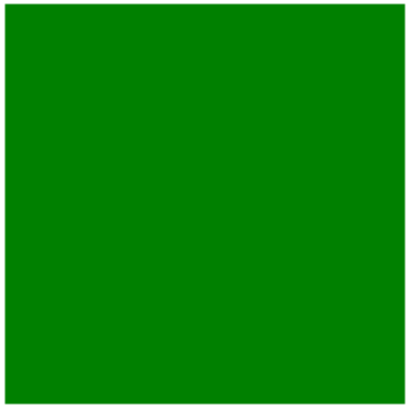
Output:
Before written of “gfg”-

After written of “gfg”-

3) onmouseover event: This event corresponds to hovering the mouse pointer over the element and its children, to which it is bound to.

```
<!doctype html>
<html>
 <head>
  <script>
    function hov() {
      var e = document.getElementById('hover');
      e.style.display = 'none';
    }
  </script>
 </head>
 <body>
  <div id="hover" onmouseover="hov()"
    style="background-
color:green;height:200px;width:200px;">
  </div>
 </body>
</html>
```

Output:
Before mouse is taken over green square-



Green square gets disappear after mouse is taken over it.

4) onmouseout event: Whenever the mouse cursor leaves the element which handles a mouseout event, a function associated with it is executed.

```
<!doctype html>
<html>
  <head>
    <script>
      function out() {
        var e = document.getElementById('hover');
        e.style.display = 'none';
      }
    </script>
  </head>
  <body>
    <div id="hover" onmouseout="out()"
      style="background-
color:green;height:200px;width:200px;">
    </div>
  </body>
</html>
```

Output:
Before mouse is taken over the green square-

Green square will disappear after mouse is taken over it and removed after some time.

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The onmouseover event triggers when you bring your mouse over any element and the onmouseout triggers when you move your mouse out from that element. Try the following example.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function over() {
          document.write ("Mouse Over");
        }
        function out() {
          document.write ("Mouse Out");
        }
      //-->
    </script>
  </head>
    <body>
    <p>Bring your mouse inside the division to see the result:</p>
    <div onmouseover = "over()" onmouseout = "out()">
      <h2> This is inside the division </h2>
    </div>
  </body></html>
```

5) onchange event: This event detects the change in value of any element listing to this event.

```
<!doctype html>
<html>
  <head></head>
  <body>
    <input
onchange="alert(this.value)"
type="number">
  </body>
</html>
```

Output:

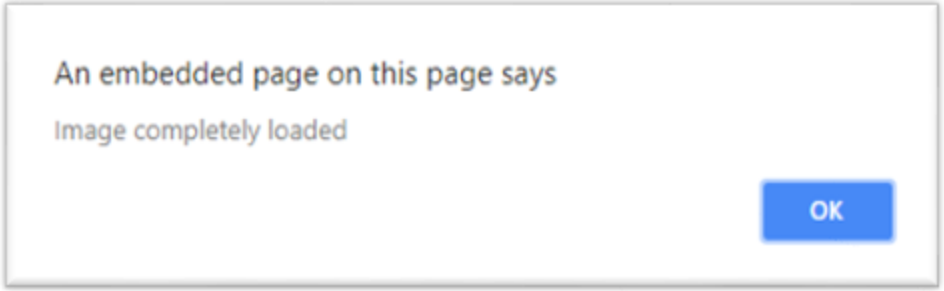
Before any key is pressed-

After 2 key is pressed-

6) onload event: When an element is loaded completely, this event is evoked.

```
<!doctype html>
<html>
  <head></head>
  <body>
    <img onload="alert('Image completely loaded')"
      alt="GFG-Logo"
      src="C:\Users\laxman\Desktop\LOGO.JPG
">
  </body>
</html>
```

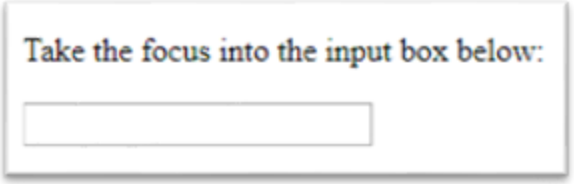
Output:



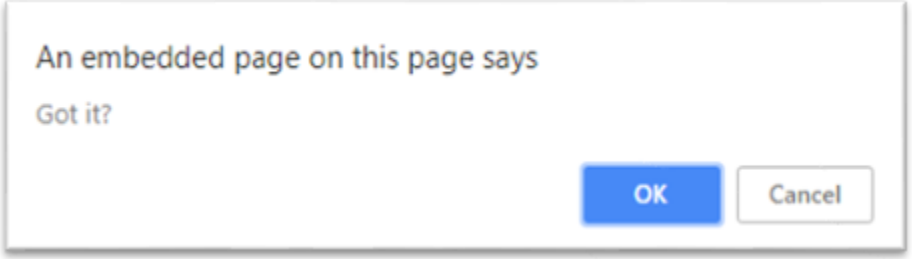
7) onfocus event: An element listing to this event executes instructions whenever it receives focus.

```
<!doctype html>
<!doctype html>
<html>
 <head>
  <script>
   function focused() {
    var e = document.getElementById('inp');
    if (confirm('Got it?')) {
     e.blur();
    }
   }
  </script>
 </head>
 <body>
  <p >Take the focus into the input box below:</p>

  <input id="inp" onfocus="focused()">
 </body>
</html>
```

Output:
Before mouse is clicked inside of the box-



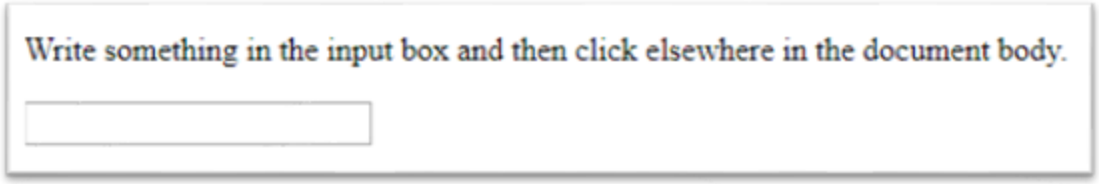
After mouse is clicked inside of the box-



8) onblur event: This event is evoked when an element loses focus.

```
<!doctype html>
<html>
  <head></head>
  <body>

<p>Write something in the input box and then click
elsewhere
      in the document body.</p>

    <input onblur="alert(this.value)">
  </body>
</html>
```

Output:
Before “gfg” is entered inside of the box-



After “gfg” is entered inside of the box and pressed enter-

PS: onmouseup event listens to left and middle mouse click, but onmousedown event listens to left, middle, and right mouse clicks whereas onclick only handles left click.

HTML 5 Standard Events

The standard HTML 5 events List. Here script indicates a Javascript function to be executed against that event.

Attribute	Value	Description
Offline	script	Triggers when the document goes offline
Onabort	script	Triggers on an abort event
onafterprint	script	Triggers after the document is printed
onbeforeonload	script	Triggers before the document loads
onbeforeprint	script	Triggers before the document is printed
onblur	script	Triggers when the window loses focus
oncanplay	script	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	script	Triggers when media can be played to the end, without stopping for buffering
onchange	script	Triggers when an element changes
onclick	script	Triggers on a mouse click

Attribute	Value	Description
oncontextmenu	script	Triggers when a context menu is triggered
ondblclick	script	Triggers on a mouse double-click
ondrag	script	Triggers when an element is dragged
ondragend	script	Triggers at the end of a drag operation
ondragenter	script	Triggers when an element has been dragged to a valid drop target
ondragleave	script	Triggers when an element is being dragged over a valid drop target
ondragover	script	Triggers at the start of a drag operation
ondragstart	script	Triggers at the start of a drag operation
ondrop	script	Triggers when dragged element is being dropped
ondurationchange	script	Triggers when the length of the media is changed
onemptied	script	Triggers when a media resource element suddenly becomes empty.
onended	script	Triggers when media has reach the end
onerror	script	Triggers when an error occur
onfocus	script	Triggers when the window gets focus
onformchange	script	Triggers when a form changes

Attribute	Value	Description
onforminput	script	Triggers when a form gets user input
onhaschange	script	Triggers when the document has change
oninput	script	Triggers when an element gets user input
oninvalid	script	Triggers when an element is invalid
onkeydown	script	Triggers when a key is pressed
onkeypress	script	Triggers when a key is pressed and released
onkeyup	script	Triggers when a key is released
onload	script	Triggers when the document loads
onloadeddata	script	Triggers when media data is loaded
onloadedmetadata	script	Triggers when the duration and other media data of a media element is loaded
onloadstart	script	Triggers when the browser starts to load the media data
onmessage	script	Triggers when the message is triggered
onmousedown	script	Triggers when a mouse button is pressed
onmousemove	script	Triggers when the mouse pointer moves

Attribute	Value	Description
onmouseout	script	Triggers when the mouse pointer moves out of an element
onmouseover	script	Triggers when the mouse pointer moves over an element
onmouseup	script	Triggers when a mouse button is released
onmousewheel	script	Triggers when the mouse wheel is being rotated
onoffline	script	Triggers when the document goes offline
onoine	script	Triggers when the document comes online
ononline	script	Triggers when the document comes online
onpagehide	script	Triggers when the window is hidden
onpageshow	script	Triggers when the window becomes visible
onpause	script	Triggers when media data is paused
onplay	script	Triggers when media data is going to start playing
onplaying	script	Triggers when media data has start playing
onpopstate	script	Triggers when the window's history changes
onprogress	script	Triggers when the browser is fetching the media data
onratechange	script	Triggers when the media data's playing rate has changed
onreadystatechange	script	Triggers when the ready-state changes

Attribute	Value	Description
onredo	script	Triggers when the document performs a redo
onresize	script	Triggers when the window is resized
onscroll	script	Triggers when an element's scrollbar is being scrolled
onseeked	script	Triggers when a media element's seeking attribute is no longer true, and the seeking has ended
onseeking	script	Triggers when a media element's seeking attribute is true, and the seeking has begun
onselect	script	Triggers when an element is selected
onstalled	script	Triggers when there is an error in fetching media data
onstorage	script	Triggers when a document loads
onsubmit	script	Triggers when a form is submitted
onsuspend	script	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate	script	Triggers when media changes its playing position
onundo	script	Triggers when a document performs an undo
onunload	script	Triggers when the user leaves the document
onvolumechange	script	Triggers when media changes the volume, also when volume is set to "mute"
onwaiting	script	Triggers when media has stopped playing, but is expected to resume

DOM

Document Object Model (DOM)

The Document Object Model (DOM) is an application programming interface (API) for manipulating HTML and XML documents.

The DOM represents a document as a tree of nodes. It provides API that allows you to add, remove, and modify parts of the document effectively.

Note that the DOM is cross-platform and language-independent ways of manipulating HTML and XML documents.

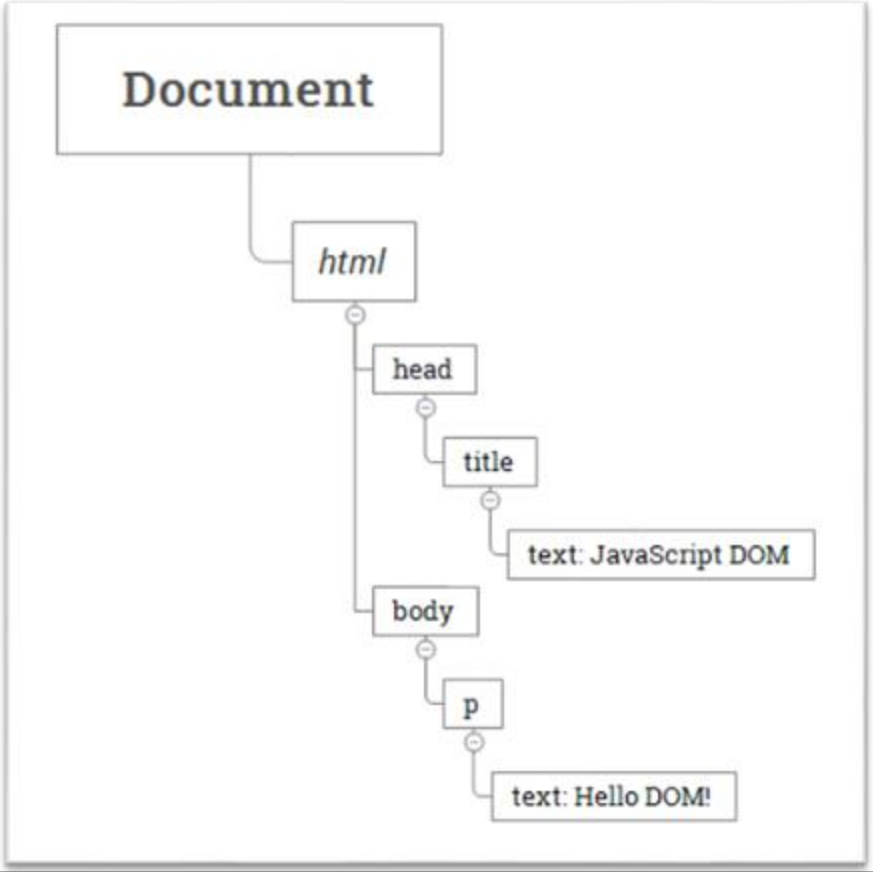
A document as a hierarchy of nodes

The DOM represents an HTML or XML document as a hierarchy of nodes. Consider the following HTML document:

```
<html>
   <head>
      <title>JavaScript DOM</title>
   </head>
   <body>
      <p>Hello DOM!</p>
   </body>
</html>
```

Code language: HTML, XML (xml)

The following tree represents the above HTML document:



In this DOM tree, the document is the **root node**. The root node has one child which is the <html> element. The <html> element is called the document element.

Each document can have only one document element. In an HTML document, the document element is the <html> element. Each markup can be represented by a **node** in the tree.

Node Types

Each node in the DOM tree is identified by a node type. JavaScript uses integer numbers to determine the node types.

The following table illustrates the node type constants:

Constant	Value	Description
Node.ELEMENT_NODE	1	An Element node like <p> or .
Node.TEXT_NODE	3	The actual Text inside an Element or Attr.
Node.CDATA_SECTION_NODE	4	A CDATASection, such as <![CDATA[…]]>.
Node.PROCESSING_INSTRUCTION_NODE	7	A ProcessingInstruction of an XML document, such as <?xml-stylesheet … ?>.
Node.COMMENT_NODE	8	A Comment node, such as <!-- … -->.
Node.DOCUMENT_NODE	9	A Document node.
Node.DOCUMENT_TYPE_NODE	10	A DocumentType node, such as <!DOCTYPE html>.
Node.DOCUMENT_FRAGMENT_NODE	11	A DocumentFragment node.

To get the type of a node, you use the nodeType property:

```
node.nodeType
```

Code language: CSS (css)

You can compare the nodeType property with the above constants to determine the node type. For example:

```
if (node.nodeType == Node.ELEMENT_NODE) {
   // node is the element node
}
```

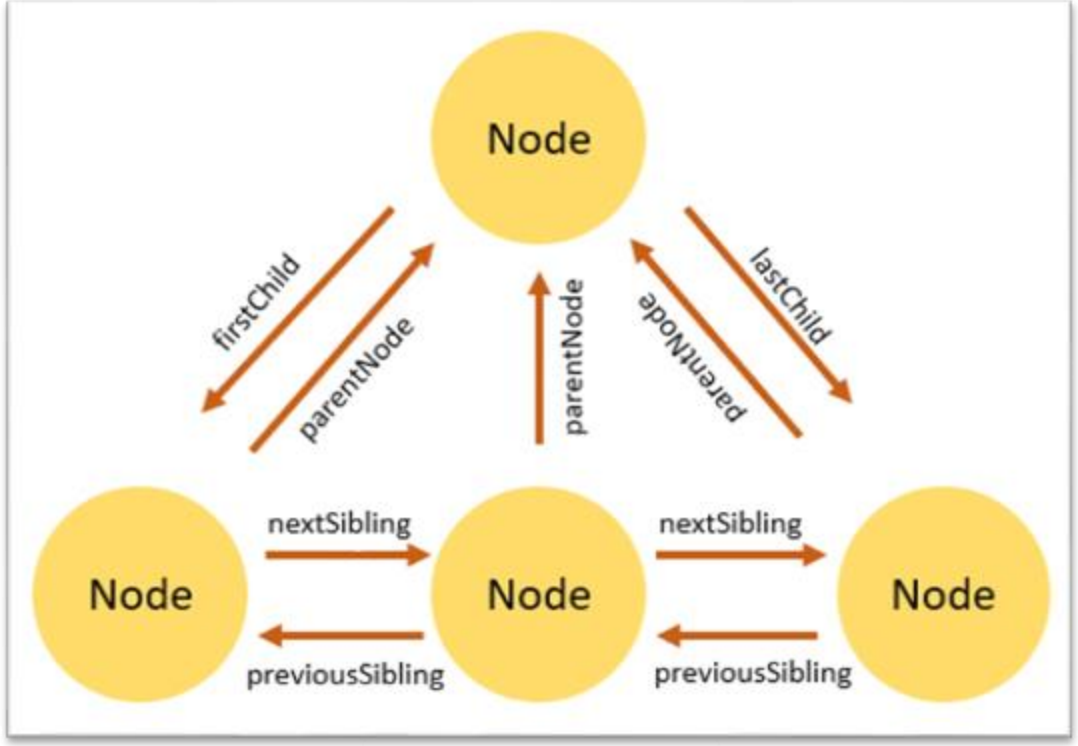
Node Relationships

Any node has relationships to other nodes in the DOM tree. The relationships are the same as the one described in a traditional family tree.

For example, <body> is a child node of the <html> node, and <html> is the parent of the <body> node.

The <body> node is the sibling of the <head> node because they share the same immediate parent, which is the <html> element.

The following picture illustrates the relationships between nodes:



- An HTML or XML document can be represented as a tree of nodes, like a traditional family tree.
- Each markup can be represented as a node with a specific node type.
- Element is a specific type of node with the node type Node.ELEMENT_NODE.
- In the DOM tree, a node has relationships with other nodes.

Selecting elements

- getElementById() – select an element by id.
- getElementsByName() – select elements by name.
- getElementsByTagName() – select elements by a tag name.
- getElementsByClassName() – select elements by one or more class names.
- querySelector() – select elements by CSS selectors.

1.Introduction to JavaScript getElementById() method

An HTML element often has an id attribute like this:

-

- The id is used to uniquely identify an HTML element within the document. By rules, the id root is unique within the document; no other elements can have this root id.
- The id is case-sensitive. For example, the 'root' and 'Root' are totally different id.
- To select the element by its id, you use the document.getElementById method.

→The following shows the syntax of the getElementById() method:

```
let element = document.getElementById(id);
```

In this syntax, the id represents the id of the element that you want to select.

- The getElementById() returns an **Element object** that describes the DOM element object with the specified id. It returns null if there is no element with that id exists.

As mentioned earlier, id is unique within a document. However, HTML is a forgiving language. If a document has more than one element with the same id, the getElementById() method returns the first one it encounters.

JavaScript getElementById() method example Consider the following HTML document:

```
<html>
  <head>
    <title>JavaScript getElementById() Method</title>
  </head>
  <body>
    <p id="message">A paragraph</p>
  </body>
</html>
```

- The document contains a <p> element that has the id attribute with the value message:

```
const p = document.getElementById('message');
console.log(p);
```

Output:

```
<p id="message">A paragraph</p>
```

Once you selected an element, you can add styles to the element, manipulate its attributes, and traversing to parent and child elements.

2.Introduction to JavaScript getElementsByName() method

Every element on an HTML document may have a name attribute:

```
<input type="radio" name="language" value="JavaScript">
```

Unlike the id attribute, multiple HTML elements can share the same value of the name attribute like this:

```
<input type="radio" name="language" value="JavaScript">
<input type="radio" name="language" value="TypeScript">
```

- To get all elements with a specified name, you use the getElementsByName() method of the document object:

```
let elements = document.getElementsByName(name);
```

The getElementsByName() accepts a name which is the value of the name attribute of elements and returns a live NodeList of elements.

The return collection of elements is live. It means that the return elements are automatically updated when elements with the same name are inserted and/or removed from the document.

Example: The following example shows a list of radio buttons that have the same name (rate).

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>JavaScript getElementsByName Demo</title>
</head>
<body>
  <p>Please rate the service:</p>
  <p>
    <input type="radio" name="rate" value="Very poor"> Very poor
    <input type="radio" name="rate" value="Poor"> Poor
    <input type="radio" name="rate" value="OK"> OK
    <input type="radio" name="rate" value="Good"> Good
    <input type="radio" name="rate" value="Very Good"> Very Good
  </p>
  <p>
    <button id="btnRate">Submit</button>
  </p>
  <script>
    let btn = document.getElementById('btnRate');

    btn.addEventListener('click', () => {
      let rates = document.getElementsByName('rate');
      rates.forEach((rate) => {
        if (rate.checked) {
          alert(`You rated: ${rate.value}`);
        }
      })
    });
  </script>
</body>
</html>
```

When you click the Rate button, the page will show an alert dialog that displays the rating of the service such as Very Poor, Poor, OK, Good, and Very Good:

Introduction to JavaScript getElementsByTagName() method

The getElementsByTagName() is a method of the document object or a specific DOM element.

The getElementsByTagName() method accepts a tag name and returns a live HTMLCollection of elements with the matching tag name in the order which they appear in the document.

- The following illustrates the syntax of the getElementsByTagName():

```
let elements = document.getElementsByTagName(tagName);
```

The return collection of the getElementsByTagName() is live, meaning that it is automatically updated when elements with the matching tag name are added and/or removed from the document.

Note that the HTMLCollection is an array-like object, like arguments object of a function.

Example: The following example illustrates how to use the getElementsByTagName() method to get the number of H2 tags in the document. When you click the Count H2 button, the page shows the number of H2 tags:

```
<!DOCTYPE html>
<html>
<head>
   <title>JavaScript getElementsByTagName() Demo</title>
</head>
<body>
   <h1>JavaScript getElementsByTagName() Demo</h1>
   <h2>First heading</h2>
   <p>This is the first paragraph.</p>
   <h2>Second heading</h2>
   <p>This is the second paragraph.</p>
   <h2>Third heading</h2>
   <p>This is the third paragraph.</p>
   <button id="btnCount">Count H2</button>
   <script>
      let btn = document.getElementById('btnCount');
      btn.addEventListener('click', () => {
         let headings = document.getElementsByTagName('h2');
         alert(`The number of H2 tags: ${headings.length}`);
      });
   </script></body></html>
```

Introduction to the getElementsByClassName() method:

Every HTML element has an optional class attribute like this:

```
<button class="btn btn-primary">Save</button>
```

- The value of the class attribute is a space-separated list of the classes of the element. The classes are case-sensitive.

The classes allows you to use the CSS to match elements. For example:

```
.btn {
  background-color: red;
}
```

- In JavaScript, you use the getElementsByClassName() method to select elements based on their classes.

The getElementsByClassName() method is available on the document object and any HTML element.

The getElementsByClassName() method accepts a single argument which is a string that contains one or more class names:

```
let elements = document.getElementsByClassName(classNames)
let elements = parentElement.getElementsByClassName(classNames)
```

In this syntax, the classNames parameter is a string that represents a class name to match. For example:

```
let btn = document.getElementsByClassName('btn');
```

Code language: JavaScript (javascript)

If you match elements by multiple classes, you need to use whitespace to separate them like this:

```
let btn = document.getElementsByClassName('btn bt-primary');
```

Code language: JavaScript (javascript)

Note that you cannot use class selectors e.g., .btn or .btn-primary for the getElementsByClassName() method.

The getElementsByClassName() method returns a live HTMLCollection of elements, which is an array-like object.

If you call the getElementsByClassName() method on the document object, the method searches for elements with the specified class names in the whole document.

However, when you call the getElementsByClassName() method on a specific element, it returns only matching elements in the subtree of that element.

Introduction to JavaScript querySelector() and querySelectorAll() methods

The querySelector() is a method of the Element interface. The querySelector() allows you to find the first element that matches one or more CSS selectors.

You can call the querySelector() method on the document or any HTML element.

The following illustrates the syntax of the querySelector() method:

```
let element = parentNode.querySelector(selector);
```

In this syntax, the selector is a CSS selector or a group of CSS selectors to match the descendant elements of the parentNode.

If the selector is not valid CSS syntax, the method will raise a SyntaxError exception.

If no element matches the CSS selectors, the querySelector() returns null.

Besides the querySelector(), you can use the querySelectorAll() method to find all elements that match a CSS selector or a group of CSS selector:

```
let elementList = parentNode.querySelectorAll(selector);
```

Code language: JavaScript (javascript)

The querySelectorAll() method returns a static NodeList of elements that match the CSS selector. If no element found, it returns an empty NodeList.

Note that the NodeList is an array-like object, not an array object. However, in modern web browsers, you can use the forEach() method like the one in the array object.

To convert the NodeList to an array, you use the Array.from() method like this:

```
let nodeList = Array.from(document.querySelectorAll(selector));
```

Basic selector examples

Suppose that you have the following HTML document:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>querySelector() Demo</title>
</head>
<body>
  <header>
    <div id="logo">
      <img src="img/logo.jpg" alt="Logo" id="logo">
    </div>
    <nav class="primary-nav">
      <ul>
        <li class="menu-item current"><a href="#home">Home</a></li>
        <li class="menu-item"><a href="#services">Services</a></li>
        <li class="menu-item"><a href="#about">About</a></li>
        <li class="menu-item"><a href="#contact">Contact</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <h1>Welcome to the JS Dev Agency</h1>

    <div class="container">
      <section class="section-a">
        <h2>UI/UX</h2>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Autem placeat, atque accusamus voluptas
          laudantium facilis iure adipisci ab veritatis eos neque culpa id nostrum tempora tempore minima.
          Adipisci, obcaecati repellat.</p>
        <button>Read More</button>
      </section>
      <section class="section-b">
        <h2>PWA Development</h2>
        <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit. Magni fugiat similique illo nobis quibusdam
          commodi aspernatur, tempora doloribus quod, consectetur deserunt, facilis natus optio. Iure
          provident labore nihil in earum.</p>
        <button>Read More</button>
      </section>
      <section class="section-c">
        <h2>Mobile App Dev</h2>
        <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Animi eos culpa laudantium consequatur ea!
          Quibusdam, iure obcaecati. Adipisci deserunt, alias repellat eligendi odit labore! Fugit iste sit
          laborum debitis eos?</p>
        <button>Read More</button>
      </section>
    </div>
  </main>
  <script src="js/main.js"></script>
</body>
</html>
```

1) Universal selector

The universal selector denoted by * that matches all elements of any type:

```
*
```

The following example uses the querySelector() selects the first element in the document:

```
let element = document.querySelector('*');
```

Code language: JavaScript (javascript)

And this example finds all elements in the document:

```
let elements = document.querySelectorAll('*');
```

2) Type selector

To select elements by node name, you use the type selector e.g., a selects all <a> elements:

elementName

The following example finds the first h1 element in the document:

```
let firstHeading = document.querySelector('h1');
```

Code language: JavaScript (javascript)

And the following example finds all h2 elements:

```
let heading2 = document.querySelectorAll('h2');
```

3) Class selector

To find the element with a given class attribute, you use the class selector syntax:

```
.className
```

Code language: CSS (css)

The following example finds the first element with the menu-item class:

```
let note = document.querySelector('.menu-item');
```

Code language: JavaScript (javascript)

And the following example finds all elements with the menu class:

```
let notes = document.querySelectorAll('.menu-item');
```

4) ID Selector

To select an element based on the value of its id, you use the id selector syntax:

```
#id
```

Code language: CSS (css)

The following example finds the first element with the id #logo:

```
let logo = document.querySelector('#logo');
```

Code language: JavaScript (javascript)

Since the id should be unique in the document, the querySelectorAll() is not relevant.

5) Attribute selector

To select all elements that have a given attribute, you use one of the following attribute selector syntaxes:

```
[attribute]
[attribute=value]
[attribute~=value]
[attribute|=value]
[attribute^=value]
[attribute$=value]
[attribute*$*=value]
```

Code language: JSON / JSON with Comments (json)

The following example finds the first element with the attribute [autoplay] with any value:

```
let autoplay = document.querySelector('[autoplay]');
```

Code language: JavaScript (javascript)

And the following example finds all elements that have [autoplay] attribute with any value:

```
let autoplays = document.querySelectorAll('[autoplay]');
```

Grouping selectors

To group multiple selectors, you use the following syntax:

selector, selector, ...

The selector list will match any element with one of the selectors in the group.

The following example finds all and <p> elements:

```
let elements = document.querySelectorAll('<div>, <p>');
```

Combinators

1) descendant combinator

To find descendants of a node, you use the space ( ) descendant combinator syntax:

selector selector

For example p a will match all <a> elements inside the p element:

```
let links = document.querySelector('p a');
```

Code language: JavaScript (javascript)

2) Child combinator

The > child combinator finds all elements that are direct children of the first element:

selector > selector

The following example finds all li elements that are directly inside a <ul> element:

```
let listItems = document.querySelectorAll('ul > li');
```

Code language: JavaScript (javascript)

To select all li elements that are directly inside a <ul> element with the class nav:

```
let listItems = document.querySelectorAll('ul.nav > li');
```

3) General sibling combinator

The ~ combinator selects siblings that share the same parent:

selector ~ selector

For example, p ~ a will match all <a> elements that follow the p element, immediately or not:

```
let links = document.querySelectorAll('p ~ a');
```

Code language: JavaScript (javascript)

4) Adjacent sibling combinator

The + adjacent sibling combinator selects adjacent siblings:

selector + selector

For example, h1 + a matches all elements that directly follow an h1:

```
let links = document.querySelectorAll('h1 + a');
```

Code language: JavaScript (javascript)

And select the first <a> that directly follows an h1:

```
let links = document.querySelector('h1 + a');
```

Pseudo

1) Pseudo-classes

The : pseudo matches elements based on their states:

element:state

Code language: CSS (css)

For example a:visited matches all <a> elements that have been visited:

```
let visitedLinks = document.querySelectorAll('a:visited');
```

Code language: JavaScript (javascript)

2) Pseudo-elements

The :: represent entities that are not included in the document.

For example, p::first-line matches the first-line of all div elements:

```
let links = document.querySelector('p::first-line');
```

Summary

The querySelector() finds the first element that matches a CSS selector or a group of CSS selectors.

The querySelectorAll() finds all elements that match a CSS selector or a group of CSS selectors.

A CSS selector defines elements to which a CSS rule applies. https://www.javascripttutorial.net/javascript-dom/

HTML Forms

Forms are used in webpages for the user to enter their required details that further send it to the server for processing. A form is also known as a web form or HTML form. Examples of form use are prevalent in e-commerce websites, online banking, online surveys to name a few.

JavaScript Form Validation

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

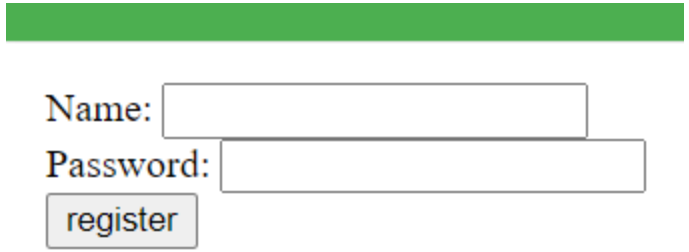
Basic Validation – First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.

Data Format Validation – Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

JavaScript Form Validation Example:

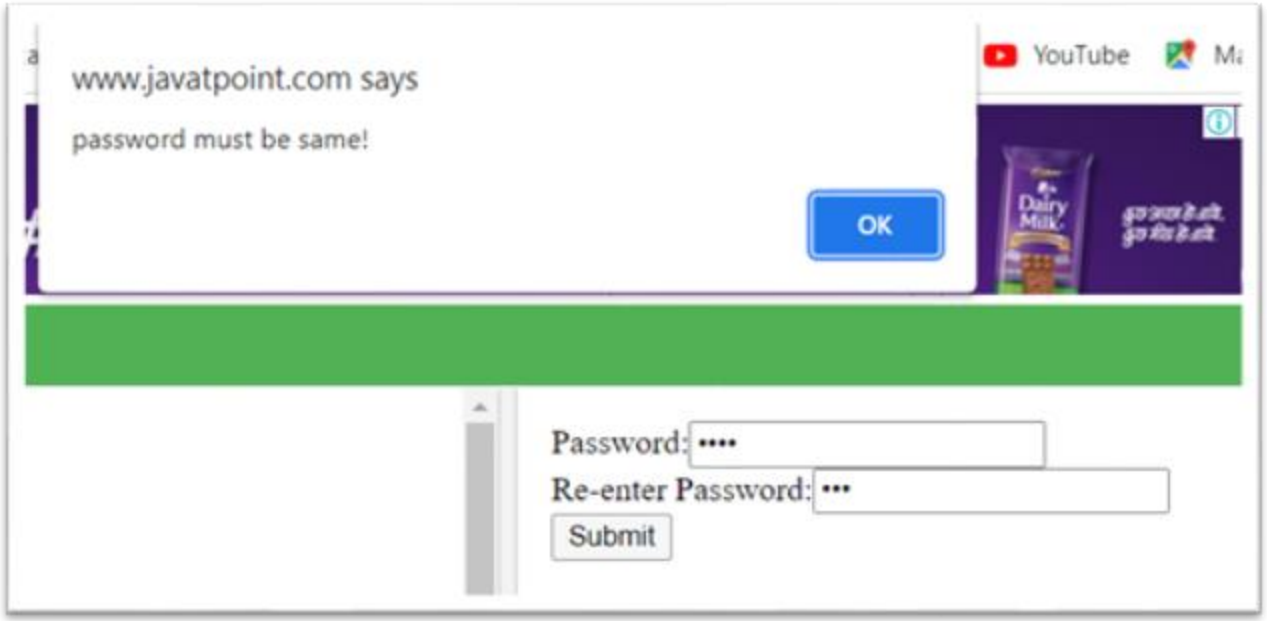
In this example, we are going to validate the name and password. The name can't be empty and password can't be less than 6 characters long.
Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

```
<html>
<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;
}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/javascriptpages/valid.jsp" onsubmit="return validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/>
<input type="submit" value="register">
</form>  </body></html>
```



JavaScript Retype Password Validation

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
function matchpass(){
var firstpassword=document.f1.password.value;
var secondpassword=document.f1.password2.value;
if(firstpassword==secondpassword){
return true;
}
else{
alert("password must be same!");
return false;
}
}
</script>
</head>
<body>
<form name="f1" action="http://www.javatpoint.com/javascriptpages/valid.jsp" onsubmit="return matchpass()">
Password:<input type="password" name="password" /><br/>
Re-enter Password:<input type="password" name="password2"/><br/>
<input type="submit">
</form></body></html>
```



JavaScript Number Validation: Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

```
<!DOCTYPE html>
<html>
<head>
<script>
function validate(){
var num=document.myform.num.value;
if (isNaN(num)){
  document.getElementById("numloc").innerHTML="Enter Numeric value only";
  return false;
}else{
  return true;
  }
}
</script>
</head>

<body>
<form name="myform" action="http://www.javatpoint.com/javascriptpages/valid.jsp" onsubmit="return validate()" >
Number: <input type="text" name="num"><span id="numloc"></span><br/>
<input type="submit" value="submit">
</form>
</body>
</html>
```

JavaScript email validation:

We can validate the email by the help of JavaScript.There are many criteria that need to be follow to validate the email id such as:

- email id must contain the @ and . character
- There must be at least one character before and after the @.
- There must be at least two characters after . (dot).

```
<html>
<body>
<script>
function validateemail()
{
var x=document.myform.email.value;
var atposition=x.indexOf("@");
var dotposition=x.lastIndexOf(".");
if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
  alert("Please enter a valid e-mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);
  return false;
  }
}
</script>
<body>
<form name="myform"  method="post" action="http://www.javatpoint.com/javascriptpages/valid.jsp" onsubmit="return
validateemail();">
Email: <input type="text" name="email"><br/>

<input type="submit" value="register">
</form>
</body>
</html>
```

jQuery
<script>
$(document).ready(function(){
$("p").text("Hello World!");
$(".btn").click(function(){
$("p").hide();
});
});
</script>

JQuery

jQuery is a fast, lightweight, and feature-rich JavaScript library that is based on the principle "write less, do more". It's easy-to-use APIs makes the things like HTML document traversal and manipulation, event handling, adding animation effects to a web page much simpler that works seamlessly across all the major browsers like Chrome, Firefox, Safari, Internet Explorer, etc.

- jQuery also gives you the ability to create an Ajax based application in a quick and simple way. Big companies like Google, Microsoft and IBM are using the jQuery for their applications. So you can easily understand how popular and powerful the jQuery is?

- jQuery was originally created by John Resig in early 2006. The jQuery project is currently run and maintained by a distributed group of developers as an open-source project. You can save a lot of time and effort with jQuery.

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

Tip: In addition, jQuery has plugins for almost any task out there.

What You Can Do with jQuery

→There are lot more things you can do with jQuery.

- You can easily select elements to perform manipulation.
- You can easily create effect like show or hide elements, sliding transition, and so on.
- You can easily create complex CSS animation with fewer lines of code.
- You can easily manipulate DOM elements and their attributes.
- You can easily implement Ajax to enable asynchronous data exchange between client and server.
- You can easily traverse all around the DOM tree to locate any element.
- You can easily perform multiple actions on an element with a single line of code.
- You can easily get or set dimensions of the HTML elements.

Advantages of Using jQuery:

- **Save lots of time** — You can save lots of time and efforts by using the jQuery inbuilt effects and selectors and concentrate on other development work.

- **Simplify common JavaScript tasks** — jQuery considerably simplifies the common JavaScript tasks. Now you can easily create feature rich and interactive web pages with fewer lines of codes, a typical example is implementing Ajax to update the content of a page without refreshing it.

- **Easy to use** — jQuery is very easy to use. Anybody with the basic working knowledge of HTML, CSS and JavaScript can start development with jQuery.

- **Compatible with browsers** — jQuery is created with modern browsers in mind and it is compatible with all major modern browsers such as Chrome, Firefox, Safari, Internet Explorer, etc.

- **Absolutely Free** — And the best part is, it is completely free to download and use.

Downloading jQuery

To get started, first download a copy of jQuery and include it in your document. There are two versions of jQuery available for downloading — compressed and uncompressed.

The uncompressed file is best suited for development or debugging; while, the minified and compressed file is recommended for production because it saves the precious bandwidth and improves the performance due to small file size.

→You can download jQuery from here: https://jquery.com/download/

Once you've downloaded the jQuery file you can see it has **.js** extension, because the jQuery is just a JavaScript library, therefore you can include the jQuery file in your HTML document with the <script> element just like you include normal JavaScript files.

Adding jQuery to Your Web Pages

There are several ways to start using jQuery on your web site. You can:

1. Download the jQuery library from jQuery.com
2. Include jQuery from a CDN, like Google

→Downloading jQuery

There are two versions of jQuery available for downloading:

1. Production version - this is for your live website because it has been minified and compressed
2. Development version - this is for testing and development (uncompressed and readable code)

Both versions can be downloaded from jQuery.com.

The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section):

```
<head>
<script src="jquery-3.5.1.min.js"></script>
</head>
```

Tip: Place the downloaded file in the same directory as the pages where you wish to use it.

jQuery CDN

1.Download the jQuery library from jQuery.com

The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag (notice that the <script> tag should be inside the <head> section):

```
<head>
<script src="jquery-3.5.1.min.js"></script>
</head>
```

Tip: Place the downloaded file in the same directory as the pages where you wish to use it.

2. Include jQuery from a CDN, like Google

- you can include jQuery in your document through freely available CDN (Content Delivery Network) links, if you don't want to download and host jQuery yourself.

- CDNs can offer a performance benefit by reducing the loading time, because they are hosting jQuery on multiple servers spread across the globe and when a user requests the file, it will be served from the server nearest to them.

- This also offers an advantage that if the visitor to your webpage has already downloaded a copy of jQuery from the same CDN while visiting other sites, it won't have to be re-downloaded since it is already there in the browser's cache.

jQuery's CDN provided by StackPath

```
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
```

You can also include jQuery through Google and Microsoft CDN's.

- Google is an example of someone who host jQuery:

Google CDN:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
```

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>My First jQuery Application</title>
   <link rel="stylesheet" type="text/css" href="/examples/css/style.css">
   <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
</head>
<body>
   <h1>Hello, World!</h1>
</body>
</html>
```

Always include the jQuery file before your custom scripts; otherwise, the jQuery APIs will not be available when your jQuery code attempts to access it.

Example: perform a simple jQuery operation by changing the color of the heading text from the default black color to red.

the <h1> element using the jQuery element selector and css() method when the document is ready which is known as document ready event

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>Example of Simple jQuery Powered Web Page</title>
   <link rel="stylesheet" type="text/css" href="/examples/css/style.css">
   <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
   <script>
      $(document).ready(function(){
         $("h1").css("color", "#0088ff");
      });
   </script>
</head>
<body>
   <h1>Hello, World!</h1>
</body>
</html>
```

Standard jQuery Syntax:

A jQuery statement typically starts with the dollar sign ($) and ends with a semicolon (;).

In jQuery, the dollar sign ($) is just an alias for jQuery. Let's consider the following example code which demonstrates the most basic statement of the jQuery.

```
<script>
   $(document).ready(function(){
      // Some code to be executed...
      alert("Hello World!");
   });
</script>
```

Explanation of code:

- **The <script> element** — Since jQuery is just a JavaScript library, so the jQuery code can be placed inside the <script> element. However, if you want to place it in an external JavaScript file, which is preferred, you just remove this part.

- **The $(document).ready(handler);** — This statement is typically known as ready event. Where the handler is basically a function that is passed to the ready() method to be executed safely as soon as the document is ready to be manipulated i.e. when the DOM hierarchy has been fully constructed.

The jQuery ready() method is typically used with an anonymous function

→The jQuery ready() method is typically used with an anonymous function

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>jQuery Shorthand Syntax</title>
   <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
            <script>
                        $(function(){
                                    // Some code to be executed...
                                    alert("Hello World!");
                        });
            </script>
</head>
<body>
            <!--Contents will be inserted here-->
</body>
</html>
```

jQuery Syntax

With jQuery you select (query) HTML elements and perform "actions" on them.

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:

1. $(this).hide() - hides the current element.

2. $("p").hide() - hides all <p> elements.

3. $(".test").hide() - hides all elements with class="test".

4. $("#test").hide() - hides the element with id="test".

The Document Ready Event:

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

  // jQuery methods go here...

});
```

- This is to prevent any jQuery code from running before the document is finished loading (is ready).
- It is good practice to wait for the document to be fully loaded and ready before working with it. This also allows you to have your JavaScript code before the body of your document, in the head section.

→Here are some examples of actions that can fail if methods are run before the document is fully loaded:

- Trying to hide an element that is not created yet
- Trying to get the size of an image that is not loaded yet

Tip: The jQuery team has also created an even shorter method for the document ready event:

```
$(function(){

  // jQuery methods go here...

});
```

Use the syntax you prefer. We think that the document ready event is easier to understand when reading the code.

inside an event handler function you can write the jQuery statements to perform any action following the basic syntax, like: $(selector).action();

Where, the **$(selector)** basically selects the HTML elements from the DOM tree so that it can be manipulated and the **action()** applies some action on the selected elements such as changes the CSS property value, or sets the element's contents, etc. Let's consider another example that sets the paragraph text after the DOM is ready:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>jQuery Document Ready Demo</title>
   <link rel="stylesheet" type="text/css" href="/examples/css/style.css">
   <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
   <script>
      $(document).ready(function(){
         $("p").text("Hello World!");
      });
   </script>
</head>
<body>
   <p>Not loaded yet.</p>
</body>
</html>
```

In the jQuery statement of the example above (line no-10) the p is a jQuery selector which select all the paragraphs i.e. the <p> elements in the document, later the text() method set the paragraph's text content to "Hello World!" text.

The paragraph text in the example above is replaced automatically when the document is ready. But what if we want the user to perform some action before executing the jQuery code to replace the paragraph text.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>jQuery Click Handler Demo</title>
   <link rel="stylesheet" type="text/css" href="/examples/css/style.css">
   <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
   <script>
      $(document).ready(function(){
         $("button").click(function(){
            $("p").text("Hello World!");
         });
      });
   </script>
</head>
<body>
   <p>Not loaded yet.</p>
   <button type="button">Replace Text</button>
</body>
</html>
```

In this example the paragraph text is replaced only when a click event is occur on the "Replace Text" <button> that simply means when a user click this button.

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: $().

- Selecting Elements by ID
- Selecting Elements by Class Name
- Selecting Elements by Name
- Selecting Elements by Attribute
- Selecting Elements by Compound CSS Selector
- jQuery Custom Selector

jQuery Selectors

Selecting Elements with jQuery

JavaScript is most commonly used to get or modify the content or value of the HTML elements on the page, as well as to apply some effects like show, hide, animations etc. But, before you can perform any action you need to find or select the target HTML element.

Selecting the elements through a typical JavaScript approach could be very painful, but the jQuery works like a magic here. The ability of making the DOM elements selection simple and easy is one of the most powerful feature of the jQuery.

- Selecting Elements by ID or **The #id Selector:**
- Selecting Elements by Class Name or **The .class Selector**
- Selecting Elements by Name or **The element Selector**
- Selecting Elements by Attribute
- Selecting Elements by Compound CSS Selector
- jQuery Custom Selector

1.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();
  });
});
```

1. Selecting Elements by ID: You can use the ID selector to select a single element with the unique ID on the page.

For example, the following jQuery code will select and highlight an element having the ID attribute id="mark", when the document is ready to be manipulated.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Select Element by ID</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   // Highlight element with id mark
   $("#mark").css("background", "yellow");
});
</script>
</head>
<body>
   <p id="mark">This is a paragraph.</p>
   <p>This is another paragraph.</p>
   <p>This is one more paragraph.</p>
   <p><strong>Note:</strong> The value of the id attribute must be unique in an HTML document.</p>
</body></html>
```

In this example, the $(document).ready() is an event that is used *to manipulate a page* safely with the jQuery. Code included inside this event will only run once the page DOM is ready.

2.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:

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

2.Selecting Elements by Class Name: The class selector can be used to *select the elements with a specific class.*

For example, the following jQuery code will select and highlight the elements having the class attribute class="mark", when the document is ready.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Select Element by Class</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   // Highlight elements with class mark
   $(".mark").css("background", "yellow");
});
</script>
</head>
<body>
   <p class="mark">This is a paragraph.</p>
   <p class="mark">This is another paragraph.</p>
   <p>This is one more paragraph.</p>
</body>
</html>
```

3. Selecting Elements by Name: The element selector can be used to select elements based on the element **name.**

For example, the following jQuery code will select and highlight all the paragraph i.e. the <p> elements of the document when it is ready.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Select Element by Name</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   // Highlight paragraph elements
   $("p").css("background", "yellow");
});
</script>
</head>
<body>
   <h1>This is heading</h1>
   <p>This is a paragraph.</p>
   <p>This is another paragraph.</p>
   <div>This is another block of text.</div>
</body>
</html>
```

3.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();
  });
});
```

4.Selecting Elements by Attribute: You can use the attribute selector to select an element by one of its HTML attributes, such as a link's target attribute or an input's type attribute, etc.

For example, the following jQuery code will select and highlight all the text inputs i.e. <input> elements with the type="text", when the document is ready.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Select Element by Attribute</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   // Highlight paragraph elements
   $('input[type="text"]').css("background", "yellow");
});
</script>
</head>
<body>
   <form>
      <label>Name: <input type="text"></label>
      <label>Password: <input type="password"></label>
      <input type="submit" value="Sign In">
   </form></body></html>
```

5.Selecting Elements by Compound CSS Selector: You can also combine the CSS selectors to make your selection even more precise.

For instance, you can combine the class selector with an element selector to find the elements in a document that has certain type and class.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Select Element by Compound Selector</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  // Highlight only paragraph elements with class mark
  $("p.mark").css("background", "yellow");

  // Highlight only span elements inside the element with ID mark
  $("#mark span").css("background", "yellow");

  // Highlight li elements inside the ul elements
  $("ul li").css("background", "yellow");

  // Highlight li elements only inside the ul element with id mark
  $("ul#mark li").css("background", "red");

  // Highlight li elements inside all the ul element with class mark
  $("ul.mark li").css("background", "green");

  // Highlight all anchor elements with target blank
  $('a[target="_blank"]').css("background", "yellow");
});
</script>
</head>
<body>
  <p>This is a paragraph.</p>
  <p class="mark">This is another paragraph.</p>
  <p>This is one more paragraph.</p>
  <ul>
    <li>List item one</li>
    <li>List item two</li>
    <li>List item three</li>
  </ul>
  <ul id="mark">
    <li>List item one</li>
    <li>List <span>item two</span></li>
    <li>List item three</li>
  </ul>
  <ul class="mark">
    <li>List item one</li>
    <li>List item two</li>
    <li>List item three</li>
  </ul>
  <p>Go to <a href="https://www.tutorialrepublic.com/"
target="_blank">Home page</a></p>
</body>
</html>
```

6.jQuery Custom Selector: In addition to the CSS defined selectors, jQuery provides its own custom selector to further enhancing the capabilities of selecting elements on a page.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Custom Selector</title>
<style>
  /* Some custom style */
  *{
    padding: 5px;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  // Highlight table rows appearing at odd places
  $("tr:odd").css("background", "yellow");

  // Highlight table rows appearing at even places
  $("tr:even").css("background", "orange");

  // Highlight first paragraph element
  $("p:first").css("background", "red");

  // Highlight last paragraph element
  $("p:last").css("background", "green");

  // Highlight all input elements with type text inside a form
  $("form :text").css("background", "purple");

  // Highlight all input elements with type password inside a form
  $("form :password").css("background", "blue");

  // Highlight all input elements with type submit inside a form
  $("form :submit").css("background", "violet");
});
</script>
</head>
<body>
  <table border="1">
    <thead>
      <tr>
        <th>No.</th>
        <th>Name</th>
        <th>Email</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>1</td>
        <td>John Carter</td>
        <td>johncarter@mail.com</td>
      </tr>
      <tr>
        <td>2</td>
        <td>Peter Parker</td>
        <td>peterparker@mail.com</td>
      </tr>
      <tr>
        <td>3</td>
        <td>John Rambo</td>
        <td>johnrambo@mail.com</td>
      </tr>
    </tbody>
  </table>
  <p>This is a paragraph.</p>
  <p>This is another paragraph.</p>
  <p>This is one more paragraph.</p>
  <form>
    <label>Name: <input type="text"></label>
    <label>Password: <input type="password"></label>
    <input type="submit" value="Sign In">
  </form>
</body>
</html>
```

Syntax	Description
$("*")	Selects all elements
$(this)	Selects the current HTML element
$("p.intro")	Selects all <p> elements with class="intro"
$("p:first")	Selects the first <p> element
$("ul li:first")	Selects the first <li> element of the first <ul>
$("ul li:first-child")	Selects the first <li> element of every <ul>
$("[href]")	Selects all elements with an href attribute
$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"
$("a[target!='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"
$(":button")	Selects all <button> elements and <input> elements of type="button"
$("tr:even")	Selects all even <tr> elements
$("tr:odd")	Selects all odd <tr> elements

jQuery Events

What are Events:

Events are often triggered by the user's interaction with the web page, such as when a *link or button is clicked*, *text is entered into an input box or textarea, selection is made in a select box, key is pressed on the keyboard, the mouse pointer is moved etc.* In some cases, the Browser itself can trigger the events, such as the page load and unload events.

jQuery enhances the basic event-handling mechanisms by offering the events methods for most native browser events, some of these methods are *ready(), click(), keypress(), focus(), blur(), change(),* etc. For example, to execute some JavaScript code when the DOM is ready, you can use the jQuery ready() method, like this:

```
<script>
$(document).ready(function(){
   // Code to be executed
   alert("Hello World!");
});
</script>
```

Note: The $(document).ready() is an event that is used to manipulate a page safely with the jQuery. Code included inside this event will only run once the page DOM is ready i.e. when the document is ready to be manipulated.

The events can be categorized into four main groups — mouse events, keyboard events, form events and document/window events. The following section will give you the brief overview of all these events as well as related jQuery methods one by one.

1.Mouse Events

a. click() Method
b. The dblclick() Method
c. The hover() Method
d. The mouseenter() Method
e. The mouseleave() Method

2. keyboard events

a. The keypress() Method
b. The keyup() Method
c. The keydown() Method

3. form events

a. The change() Method
b. The focus() method
c. The blur() Method
d. The submit() Method

4. document/window events

a. The ready() Method
b. The resize()
c. The scroll() method

All the different visitors' actions that a web page can respond to are called events.

An event represents the precise moment when something happens.

Examples:

- moving a mouse over an element
- selecting a radio button
- clicking on an element

→The term "fires/fired" is often used with events. Example: "The keypress event is fired, the moment you press a key".

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

1.Mouse Events: A mouse event is fired when the user click some element, move the mouse pointer etc. Here're some commonly used jQuery methods to handle the mouse events.

a. The click() Method: The jQuery click() method attach an event handler function to the selected elements for "click" event. The attached function is executed when the user clicks on that element. The following example will hide the <p> elements on a page when they are clicked.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Click Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   p{
      padding: 20px;
      font: 20px sans-seri
      background: khaki;
   }
</style>
<script>
$(document).ready(function(){
   $("p").click(function(){
      $(this).slideUp();
   });
});
</script></head><body>
   <p>Click on me and I'll disappear.</p>
   <p>Click on me and I'll disappear.</p>
   <p>Click on me and I'll disappear.</p>
</body></html>
```

Note: The this keyword inside the jQuery event handler function is a reference to the element where the event is currently being delivered.

b. The dblclick() Method:

The jQuery dblclick() method attach an event handler function to the selected elements for "dblclick" event. The attached function is executed when the user double-clicks on that element. The following example will hide the <p> elements when they are double-clicked.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Double-click Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 20px;
    font: 20px sans-serif;
    background: khaki;
  }
</style>
<script>
$(document).ready(function(){
  $("p").dblclick(function(){
    $(this).slideUp();
  });
});
</script>
</head>
<body>
  <p>Double-click on me and I'll disappear.</p>
  <p>Double-click on me and I'll disappear.</p>
  <p>Double-click on me and I'll disappear.</p>
</body></html>
```

c. The hover() Method:

The jQuery hover() method attach one or two event handler functions to the selected elements that is executed when the mouse pointer enters and leaves the elements. The first function is executed when the user place the mouse pointer over an element, whereas the second function is executed when the user removes the mouse pointer from that element.

The following example will highlight <p> elements when you place the cursor on it, the highlighting will be removed when you remove the cursor.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Hover Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 20px;
    font: 20px sans-serif;
    background: #f2f2f2;
  }
  p.highlight{
    background: yellow;
  }
</style>
<script>
$(document).ready(function(){
  $("p").hover(function(){
    $(this).addClass("highlight");
  }, function(){
    $(this).removeClass("highlight");
  });
});
</script>
</head>
<body>
  <p>Place mouse pointer on me.</p>
  <p>Place mouse pointer on me.</p>
  <p>Place mouse pointer on me.</p>
</body>
</html>
```

Tip: You can consider the hover() method is a combination of the jQuery mouseenter() and mouseleave() methods.

d. The mouseenter() Method:

The jQuery mouseenter() method attach an event handler function to the selected elements that is executed when the mouse enters an element. The following example will add the class highlight to the <p> element when you place the cursor on it.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Mouseenter Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 20px;
    font: 20px sans-serif;
    background: #f2f2f2;
  }
  p.highlight{
    background: yellow;
  }
</style>
<script>
$(document).ready(function(){
  $("p").mouseenter(function(){
    $(this).addClass("highlight");
  });
  $("p").mouseleave(function(){
    $(this).removeClass("highlight");
  });
});
</script>
</head>
<body>
  <p>Place mouse pointer on me.</p>
  <p>Place mouse pointer on me.</p>
  <p>Place mouse pointer on me.</p>
</body>
</html>
```

e.The mouseleave() Method:

The jQuery mouseleave() method attach an event handler function to the selected elements that is executed when the mouse leaves an element. The following example will remove the class highlight from the <p> element when you remove the cursor from it.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Mouseleave Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 20px;
    font: 20px sans-serif;
    background: #f2f2f2;
  }
  p.highlight{
    background: yellow;
  }
</style>
<script>
$(document).ready(function(){
  $("p").mouseenter(function(){
    $(this).addClass("highlight");
  });
  $("p").mouseleave(function(){
    $(this).removeClass("highlight");
  });
});
</script>
</head>
<body>
  <p>Place mouse pointer on me.</p>
  <p>Place mouse pointer on me.</p>
  <p>Place mouse pointer on me.</p>
</body>
</html>
```

2.Keyboard Events:

A keyboard event is fired when the user press or release a key on the keyboard. Here're some commonly used jQuery methods to handle the keyboard events.

a. The keypress() Method:

The jQuery keypress() method attach an event handler function to the selected elements (typically form controls) that is executed when the browser receives keyboard input from the user. The following example will display a message when the kypress event is fired and how many times it is fired when you press the key on the keyboard.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Keypress Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 10px;
    background: lightgreen;
    display: none;
  }
  div{
    margin: 20px 0;
  }
</style>
<script>
$(document).ready(function(){
  var i = 0;
  $('input[type="text"]').keypress(function(){
    $("span").text(i += 1);
    $("p").show().fadeOut();
  });
});
</script>
</head>
<body>
  <input type="text">
  <div>Keypress: <span>0</span></div>
          <div><strong>Note:</strong> Enter something inside the input box and see the result.</div>
  <p>Keypress is triggered.</p>
</body>
</html>
```

Note: The keypress event is similar to the keydown event, except that modifier and non-printing keys such as Shift, Esc, Backspace or Delete, Arrow keys etc. trigger keydown events but not keypress events.

b.The keydown() Method

The jQuery keydown() method attach an event handler function to the selected elements (typically form controls) that is executed when the user first presses a key on the keyboard. The following example will display a message when the keydown event is fired and how many times it is fired when you press the key on the keyboard.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Keydown Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 10px;
    background: lightgreen;
    display: none;
  }
  div{
    margin: 20px 0;
  }
</style>
<script>
$(document).ready(function(){
  var i = 0;
  $('input[type="text"]').keydown(function(){
    $("span").text(i += 1);
    $("p").show().fadeOut();
  });
});
</script>
</head>
<body>
  <input type="text">
  <div>Keydown: <span>0</span></div>
          <div><strong>Note:</strong> Enter something inside the input box and see the result.</div>
  <p>Keydown is triggered.</p>
</body>
</html>
```

c. The keyup() Method:

The jQuery keyup() method attach an event handler function to the selected elements (typically form controls) that is executed when the user releases a key on the keyboard. The following example will display a message when the keyup event is fired and how many times it is fired when you press and release a key on the keyboard.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Keyup Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 10px;
    background: lightgreen;
    display: none;
  }
  div{
    margin: 20px 0;
  }
</style>
<script>
$(document).ready(function(){
  var i = 0;
  $('input[type="text"]').keyup(function(){
    $("span").text(i += 1);
    $("p").show().fadeOut();
  });
});
</script>
</head>
<body>
  <input type="text">
  <div>Keyup: <span>0</span></div>
          <div><strong>Note:</strong> Enter something inside the input box and see the result.</div>
  <p>Keyup is triggered.</p>
</body>
</html>
```

Tip: The keyboard events can be attached to any element, but the event is only sent to the element that has the focus. That's why the keyboard events generally attached to the form controls such as text input box or textarea.

3.Form Events

A form event is fired when a form control receive or loses focus or when the user modify a form control value such as by typing text in a text input, select any option in a select box etc. Here're some commonly used jQuery methods to handle the form events.

a. The change() Method

The jQuery change() method attach an event handler function to the <input>, <textarea> and <select> elements that is executed when its value changes. The following example will display an alert message when you select any option in dropdown select box.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Change Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("select").change(function(){
    var selectedOption = $(this).find(":selected").val();
    alert("You have selected - " + selectedOption);
  });
});
</script>
</head>
<body>
  <form>
    <label>City:</label>
    <select>
      <option>London</option>
      <option>Paris</option>
      <option>New York</option>
    </select>
  </form>
        <p><strong>Note:</strong> Select any value from the dropdown select and see the result.</p>
</body>
</html>
```

Note: For select boxes, checkboxes, and radio buttons, the event is fired immediately when the user makes a selection with the mouse, but for the text input and textarea the event is fired after the element loses focus.

b. The focus() Method

The jQuery focus() method attach an event handler function to the selected elements (typically form controls and links) that is executed when it gains focus. The following example will display a message when the text input receive focus.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Focus Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  label{
    display: block;
    margin: 5px 0;
  }
  label span{
    display: none;
  }
</style>
<script>
$(document).ready(function(){
  $("input").focus(function(){
    $(this).next("span").show().fadeOut("slow");
  });
});
</script>
</head>
<body>
  <form>
    <label>Email: <input type="text"> <span>focus fire</span></label>
    <label>Password: <input type="password"> <span>focus fire</span></label>
    <label><input type="submit" value="Sign In"> <span>focus fire</span></label>
  </form>
  <p><strong>Note:</strong> Click on the form control or press the "Tab" key to set focus.</p>
</body>
</html>
```

c. The blur() Method

The jQuery blur() method attach an event handler function to the form elements such as <input>, <textarea>, <select> that is executed when it loses focus. The following example will display a message when the text input loses focus.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Blur Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  label{
    display: block;
    margin: 5px 0;
  }
  label span{
    display: none;
  }
</style>
<script>
$(document).ready(function(){
  $("input").blur(function(){
    $(this).next("span").show().fadeOut("slow");
  });
});
</script>
</head>
<body>
  <form>
    <label>Email: <input type="text"> <span>blur fire</span></label>
    <label>Password: <input type="password"> <span>blur fire</span></label>
    <label><input type="submit" value="Sign In"> <span>blur fire</span></label>
  </form>
  <p><strong>Note:</strong> Click away from the form control or press the "Tab" key to remove focus.</p>
</body>
</html>
```

d. The submit() Method:

The jQuery submit() method attach an event handler function to the <form> elements that is executed when the user is attempting to submit a form. The following example will display a message depending on the value entered when you try to submit the form.

Tip: A form can be submitted either by clicking a submit button, or by pressing Enter when certain form elements have focus.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Form Submit Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  .error{
    color: red;
  }
</style>
<script>
$(document).ready(function(){
  $("form").submit(function(event){
    var regex = /^[a-zA-Z]+$/;
    var currentValue = $("#firstName").val();
    if(regex.test(currentValue) == false){
      $("#result").html('<p class="error">Not valid!</p>').show().fadeOut(1000);
      // Preventing form submission
      event.preventDefault();
    }
  });
});
</script>
</head>
<body>
  <p><strong>Note:</strong> If try to submit any invalid value. It will produce an error.</p>
  <form action="/examples/html/action.php" method="post" id="users">
    <label for="firstName">First Name:</label>
    <input type="text" name="first-name" id="firstName">
    <input type="submit" value="Submit">
    <div id="result"></div>
  </form>
</body>
</html>
```

4.Document/Window Events

Events are also triggered in a situation when the page DOM (Document Object Model) is ready or when the user *resize or scrolls* the browser window, etc.
Here're some commonly used jQuery methods to handle such kind of events.

a. The ready() Method

The jQuery ready() method specify a function to execute when the DOM is fully loaded.

The following example will replace the paragraphs text as soon as the DOM hierarchy has been fully constructed and ready to be manipulated.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Ready Event in
jQuery</title>
<script src="https://code.jquery.com/jquery-
3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   $("p").text("The DOM is now loaded and can be
manipulated.");
});
</script>
</head>
<body>
   <p>Not loaded yet.</p>
</body>
</html>
```

b. The resize() Method

The jQuery resize() method attach an event handler function to the window element that is executed when the size of the browser window changes.

The following example will display the current width and height of the browser window when you try to resize it by dragging its corners.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Resize Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 20px;
    font: 20px sans-serif;
    background: #f0e68c;
  }
</style>
<script>
$(document).ready(function(){
  $(window).resize(function() {
    $(window).bind("resize", function(){
      $("p").text("Window width: " + $(window).width() + ", " + "Window height: " + $(window).height());
    });
  });
});
</script>
</head>
<body>
  <p>Open the output in a new tab and resize the browser window by dragging the corners.</p>
</body>
</html>
```

c. The scroll() Method

The jQuery scroll() method attach an event handler function to the window or scrollable iframes and elements that is executed whenever the element's scroll position changes.

The following example will display a message when you scroll the browser window.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Executing a Function on Scroll Event in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    width: 100%;
    padding: 50px 0;
    text-align: center;
    font: bold 34px sans-serif;
    background: #f0e68c;
    position: fixed;
    top: 50px;
    display: none;
  }
  .dummy-content{
    height: 600px;
    font: 34px sans-serif;
    text-align: center;
  }
</style>
<script>
$(document).ready(function(){
  $(window).scroll(function() {
    $("p").show().fadeOut("slow");
  });
});
</script>
</head>
<body>
  <p>Scroll Happened!</p>
  <div class="dummy-content">Scroll the viewport.</div>
  <div class="dummy-content">Scroll the viewport.</div>
  <div class="dummy-content">Scroll the viewport.</div>
  <div class="dummy-content">Scroll the viewport.</div>
  <div class="dummy-content">Scroll the viewport.</div>
</body>
</html>
```

Category	Description	Imp Methods
DOM Manipulation	These methods manipulate DOM elements in some manner e.g. changing attribute, style attribute, adding and removing elements etc.	after(),append(),attr(),before(), more..
Traversing	These methods help in navigating from DOM element to another element in a parent child hierarchy e.g. finding ancestors, descendants or sibling element of a specified element.	children(),closest(),each(),first(),next(),filter(),parent(),siblings(), more..
CSS	These methods get and set css related properties of elements.	addClass(),css(),hasClass(),removeClass(),toggleClass(), more..
Attributes	These methods get and set DOM attributes of elements.	attr(),html(),removeAttr(),prop(),val(), more..
Events	These methods are used to handle DOM or JavaScript events.	bind(),blur(),change(),click(),dblclick(),focus(),keyup(),keydown(), more..
Effects	These methods are used to add animation to elements.	animate(),fadeIn(),fadeOut(),hide(),show(),stop(), more..
Dimensions	These methods are used to get and set the CSS dimensions for the various properties.	height(),width(),innerHeight(),innerWidth(),more..
Forms	These methods and event handlers handle forms and their various elements.	blur(),change(),val(),submit(),more..
Ajax	These methods allow Ajax functionalities with jQuery e.g.	get(),getJson(),post(),load(),more..
Core	These methods are core methods in jQuery API.	jQuery(),holdReady(),when(),more..
Data	These methods allow us to associate arbitrary data with specific DOM elements.	data(),removeData(),queue(), dequeue(),clearQueue(),more..
Miscellaneous	These methods are useful in various tasks e.g. traversing elements, converting to array etc.	each(),index(),get(),toArray(),more..
Utilities	Utility methods are helpful in getting information on various things e.g. browser, function, array, window etc.	inArray(),isArray(),isFunction(),isNumeric(), isWindow(),isXmlDoc(),more..

```
<!DOCTYPE html>

<html>
<head>
   <meta name="viewport" content="width=device-width" />
   <title>Index</title>
   <script type="text/javascript" src="~/Scripts/jquery-3.3.1.js"></script>
   <script type="">
      $(document).ready(function () {

         $('p').wrap('<div class="myCls">'); @* wrap all p with div *@
         $('#myDiv').hide(); @* hides div whose id is myDiv  *@
         $('span').attr(  @*sets style and width attribute on all span *@
            {
               'style': 'border:solid',
               'width': '100%'
            });
         $('p').append('This is p.'); @* append text to <p> *@

         $('span').before('<p>This is another p</p>'); @* insert <p> before span  *@

      });
   </script>
</head>

<body>
   <div id="myDiv"></div>
   <p></p>
   <span></span>
</body>
</html>
```

JQUERY EFFECTS

jQuery provides a trivially simple interface for doing various kind of amazing effects. jQuery methods allow us to quickly apply commonly used effects with a minimum configuration.

1. **Show and Hide Effects**
2. **Fading Effects**
3. **Sliding Effects**
4. **Animation Effects**
5. **Stop Animations**

SNo	Methods & Description
1	animate(params, [duration, easing, callback])A function for making custom animations.
2	fadeIn(speed, [callback])Fade in all matched elements by adjusting their opacity and firing an optional callback after completion.
3	fadeOut(speed, [callback])Fade out all matched elements by adjusting their opacity to 0, then setting display to "none" and firing an optional callback after completion.
4	fadeTo(speed, opacity, callback)Fade the opacity of all matched elements to a specified opacity and firing an optional callback after completion.
5	hide( )Hides each of the set of matched elements if they are shown.
6	hide(speed, [callback])Hide all matched elements using a graceful animation and firing an optional callback after completion.
7	show( )Displays each of the set of matched elements if they are hidden.
8	show(speed, [callback])Show all matched elements using a graceful animation and firing an optional callback after completion.
9	slideDown(speed, [callback])Reveal all matched elements by adjusting their height and firing an optional callback after completion.
10	slideToggle(speed, [callback])Toggle the visibility of all matched elements by adjusting their height and firing an optional callback after completion.
11	slideUp(speed, [callback])Hide all matched elements by adjusting their height and firing an optional callback after completion.
12	stop([clearQueue, gotoEnd])Stops all the currently running animations on all the specified elements.
13	toggle( )Toggle displaying each of the set of matched elements.
14	toggle(speed, [callback])Toggle displaying each of the set of matched elements using a graceful animation and firing an optional callback after completion.
15	toggle(switch)Toggle displaying each of the set of matched elements based upon the switch (true shows all elements, false hides all elements).
16	jQuery.fx.offGlobally disable all animations.

1.Show and Hide Effects

a.show() and hide() Methods:

You can show and hide HTML elements using the jQuery show() and hide() methods.

The hide() method simply sets the inline style display: none for the selected elements. Conversely, the show() method restores the display properties of the matched set of elements to whatever they initially were—typically block, inline, or inline-block—before the inline style display: none was applied to them. Here's is an example.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Show Hide Effects</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #F0E68C;
  }
</style>
<script>
$(document).ready(function(){
  // Hide displayed paragraphs
  $(".hide-btn").click(function(){
    $("p").hide();
  });

  // Show hidden paragraphs
  $(".show-btn").click(function(){
    $("p").show();
  });
});
</script>
</head>
<body>
  <button type="button" class="hide-btn">Hide Paragraphs</button>
  <button type="button" class="show-btn">Show Paragraphs</button>
  <p>This is a paragraph.</p>
  <p>This is another paragraph.</p>
</body>
</html>
```

You can optionally specify the duration (also referred as speed) parameter for making the jQuery show hide effect animated over a specified period of time.

Durations can be specified either using one of the predefined string 'slow' or 'fast', or in a number of milliseconds, for greater precision; higher values indicate slower

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Animated Show Hide Effects</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #F0E68C;
  }
</style>
<script>
$(document).ready(function(){
  // Hide displayed paragraphs with different speeds
  $(".hide-btn").click(function(){
    $("p.normal").hide();
    $("p.fast").hide("fast");
    $("p.slow").hide("slow");
    $("p.very-fast").hide(50);
    $("p.very-slow").hide(2000);
  });

  // Show hidden paragraphs with different speeds
  $(".show-btn").click(function(){
    $("p.normal").show();
    $("p.fast").show("fast");
    $("p.slow").show("slow");
    $("p.very-fast").show(50);
    $("p.very-slow").show(2000);
  });
});
</script>
</head>
<body>
  <button type="button" class="hide-btn">Hide Paragraphs</button>
  <button type="button" class="show-btn">Show Paragraphs</button>
  <p class="very-fast">This paragraph will show/hide with very fast speed.</p>
  <p class="normal">This paragraph will show/hide with default speed.</p>
  <p class="fast">This paragraph will show/hide with fast speed.</p>
  <p class="slow">This paragraph will show/hide with slow speed.</p>
  <p class="very-slow">This paragraph will show/hide with very slow speed.</p>
</body>
</html>
```

You can also specify a callback function to be executed after the show() or hide() method completes.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Show Hide Effects with Callback</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   p{
      padding: 15px;
      background: #F0E68C;
   }
</style>
<script>
$(document).ready(function(){
   // Display alert message after hiding paragraphs
   $(".hide-btn").click(function(){
      $("p").hide("slow", function(){
         // Code to be executed
         alert("The hide effect is completed.");
      });
   });

   // Display alert message after showing paragraphs
   $(".show-btn").click(function(){
      $("p").show("slow", function(){
         // Code to be executed
         alert("The show effect is completed.");
      });
   });
});
</script>
</head>
<body>
   <button type="button" class="hide-btn">Hide Paragraphs</button>
   <button type="button" class="show-btn">Show Paragraphs</button>
   <p>This is a paragraph.</p>
</body>
</html>
```

b. jQuery toggle() Method

The jQuery toggle() method show or hide the elements in such a way that if the element is initially displayed, it will be hidden; if hidden, it will be displayed (i.e. toggles the visibility).

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Toggle Effect</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #F0E68C;
  }
</style>
<script>
$(document).ready(function(){
  // Toggles paragraphs display
  $(".toggle-btn").click(function(){
    $("p").toggle();
  });
});
</script>
</head>
<body>
  <button type="button" class="toggle-btn">Toggle Paragraphs</button>
  <p>This is a paragraph.</p>
  <p>This is another paragraph.</p>
</body>
</html>
```

Similarly, you can specify the duration parameter for the toggle() method to make it animated like the show() and hide() methods.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Animated Toggle Effect</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   p{
      padding: 15px;
      background: #F0E68C;
   }
</style>
<script>
$(document).ready(function(){
   // Toggles paragraphs with different speeds
   $(".toggle-btn").click(function(){
      $("p.normal").toggle();
      $("p.fast").toggle("fast");
      $("p.slow").toggle("slow");
      $("p.very-fast").toggle(50);
      $("p.very-slow").toggle(2000);
   });
});
</script>
</head>
<body>
   <button type="button" class="toggle-btn">Toggle Paragraphs</button>
   <p class="very-fast">This paragraph will show/hide with very fast speed.</p>
   <p class="normal">This paragraph will show/hide with default speed.</p>
   <p class="fast">This paragraph will show/hide with fast speed.</p>
   <p class="slow">This paragraph will show/hide with slow speed.</p>
   <p class="very-slow">This paragraph will show/hide with very slow speed.</p>
</body>
</html>
```

2.jQuery Fading Effects

a.jQuery fadeIn() and fadeOut() Methods:

You can use the jQuery fadeIn() and fadeOut() methods to display or hide the HTML elements by gradually increasing or decreasing their opacity.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Fade-In and Fade-Out Effects</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #DDA0DD;
  }
</style>
<script>
$(document).ready(function(){
  // Fadeing out displayed paragraphs
  $(".out-btn").click(function(){
    $("p").fadeOut();
  });

  // Fading in hidden paragraphs
  $(".in-btn").click(function(){
    $("p").fadeIn();
  });
});
</script>
</head>
<body>
  <button type="button" class="out-btn">Fade Out Paragraphs</button>
  <button type="button" class="in-btn">Fade In Paragraphs</button>
  <p>This is a paragraph.</p>
  <p>This is another paragraph.</p>
</body>
</html>
```

Like other jQuery effects methods, you can optionally specify the duration or speed parameter for the fadeIn() and fadeOut() methods to control how long the fading animation will run. Durations can be specified either using one of the predefined string 'slow' or 'fast', or in a number of milliseconds; higher values indicate slower animations.

Note: The effect of fadeIn()/fadeOut() method looks similar to show()/hide(), but unlike show()/hide() method the fadeIn()/fadeOut() method only animates the opacity of the target elements and does not animates their dimensions.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Fade-In and Fade-Out Effects with Different Speeds</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #DDA0DD;
  }
</style>
<script>
$(document).ready(function(){
  // Fading out displayed paragraphs with different speeds
  $(".out-btn").click(function(){
    $("p.normal").fadeOut();
    $("p.fast").fadeOut("fast");
    $("p.slow").fadeOut("slow");
    $("p.very-fast").fadeOut(50);
    $("p.very-slow").fadeOut(2000);
  });

  // Fading in hidden paragraphs with different speeds
  $(".in-btn").click(function(){
    $("p.normal").fadeIn();
    $("p.fast").fadeIn("fast");
    $("p.slow").fadeIn("slow");
    $("p.very-fast").fadeIn(50);
    $("p.very-slow").fadeIn(2000);
  });
});
</script>
</head>
<body>
  <button type="button" class="out-btn">Fade Out Paragraphs</button>
  <button type="button" class="in-btn">Fade In Paragraphs</button>
  <p class="very-fast">This paragraph will fade in/out with very fast speed.</p>
  <p class="normal">This paragraph will fade in/out with default speed.</p>
  <p class="fast">This paragraph will fade in/out with fast speed.</p>
  <p class="slow">This paragraph will fade in/out with slow speed.</p>
```

You can also specify a callback function to be executed after the fadeIn() or fadeOut() method completes.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Fade-In and Fade-Out Effects with Callback</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #DDA0DD;
  }
</style>
<script>
$(document).ready(function(){
  // Display alert message after fading out paragraphs
  $(".out-btn").click(function(){
    $("p").fadeOut("slow", function(){
      // Code to be executed
      alert("The fade-out effect is completed.");
    });
  });

  // Display alert message after fading in paragraphs
  $(".in-btn").click(function(){
    $("p").fadeIn("slow", function(){
      // Code to be executed
      alert("The fade-in effect is completed.");
    });
  });
});
</script>
</head>
<body>
  <button type="button" class="out-btn">Fade Out Paragraphs</button>
  <button type="button" class="in-btn">Fade In Paragraphs</button>
  <p>This is a paragraph.</p>
</body>
```

b. jQuery fadeToggle() Method

The jQuery fadeToggle() method display or hide the selected elements by animating their opacity in such a way that if the element is initially displayed, it will be fade out; if hidden, it will be fade in (i.e. toggles the fading effect).

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Fade-Toggle Effect</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #DDA0DD;
  }
</style>
<script>
$(document).ready(function(){
  // Toggles paragraphs display with fading
  $(".toggle-btn").click(function(){
    $("p").fadeToggle();
  });
});
</script>
</head>
<body>
  <button type="button" class="toggle-btn">Fade Toggle Paragraphs</button>
  <p>This is a paragraph.</p>
  <p>This is another paragraph.</p>
</body>
</html>
```

Similarly, you can specify the duration parameter for the fadeToggle() method like fadeIn()/fadeOut() method to control the duration or speed of the fade toggle animation.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Fade-Toggle Effect with Different Speeds</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   p{
      padding: 15px;
      background: #DDA0DD;
   }
</style>
<script>
$(document).ready(function(){
   // Fade Toggles paragraphs with different speeds
   $(".toggle-btn").click(function(){
      $("p.normal").fadeToggle();
      $("p.fast").fadeToggle("fast");
      $("p.slow").fadeToggle("slow");
      $("p.very-fast").fadeToggle(50);
      $("p.very-slow").fadeToggle(2000);
   });
});
</script>
</head>
<body>
   <button type="button" class="toggle-btn">Fade Toggle Paragraphs</button>
   <p class="very-fast">This paragraph will fade in/out with very fast speed.</p>
   <p class="normal">This paragraph will fade in/out with default speed.</p>
   <p class="fast">This paragraph will fade in/out with fast speed.</p>
   <p class="slow">This paragraph will fade in/out with slow speed.</p>
   <p class="very-slow">This paragraph will fade in/out with very slow speed.</p>
</body>
</html>
```

Similarly, you can also specify a callback function for the fadeToggle() method.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Fade-Toggle Effect with Callback</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   p{
      padding: 15px;
      background: #DDA0DD;
   }
</style>
<script>
$(document).ready(function(){
   // Display alert message after fade toggling paragraphs
   $(".toggle-btn").click(function(){
      $("p").fadeToggle(1000, function(){
         // Code to be executed
         alert("The fade-toggle effect is completed.");
      });
   });
});
</script>
</head>
<body>
   <button type="button" class="toggle-btn">Fade Toggle Paragraphs</button>
   <p>This is a paragraph.</p>
</body>
</html>
```

c. jQuery fadeTo() Method

The jQuery fadeTo() method is similar to the .fadeIn() method, but unlike .fadeIn() the fadeTo() method lets you fade in the elements to a certain opacity level.

$(selector).fadeTo(speed, opacity, callback);

The required opacity parameter specifies the final opacity of the target elements that can be a number between 0 and 1. The duration or speed parameter is also required for this method that specifies the duration of the fade to animation.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Fade-To Effect</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    display: none;
    padding: 15px;
    background: #DDA0DD;
  }
</style>
<script>
$(document).ready(function(){
  // Fade to paragraphs with different opacity
  $(".to-btn").click(function(){
    $("p.none").fadeTo("fast", 0);
    $("p.partial").fadeTo("slow", 0.5);
    $("p.complete").fadeTo(2000, 1);
  });
});
</script>
</head>
<body>
  <button type="button" class="to-btn">Fade To Hidden Paragraphs</button>
  <p class="none">This is a paragraph.</p>
  <p class="partial">This is another paragraph.</p>
  <p class="complete">This is one more paragraph.</p>
</body>
</html>
```

3. jQuery sliding Effects

a. slideUp() and slideDown() Methods

The jQuery slideUp() and slideDown() methods is used to hide or show the HTML elements by gradually decreasing or increasing their **height** (i.e. by sliding them up or down).

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Slide-Up and Slide-Down Effects</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #B0C4DE;
  }
</style>
<script>
$(document).ready(function(){
  // Slide up displayed paragraphs
  $(".up-btn").click(function(){
    $("p").slideUp();
  });

  // Slide down hidden paragraphs
  $(".down-btn").click(function(){
    $("p").slideDown();
  });
});
</script>
</head>
<body>
  <button type="button" class="up-btn">Slide Up Paragraphs</button>
  <button type="button" class="down-btn">Slide Down Paragraphs</button>
  <p>This is a paragraph.</p>
  <p>This is another paragraph.</p>
</body>
</html>
```

Like other jQuery effects methods, you can optionally specify the duration or speed parameter for the slideUp() and slideDown() methods to control how long the slide animation will run. Durations can be specified either using one of the predefined string 'slow' or 'fast', or in a number of milliseconds; higher values indicate slower animations.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Slide-Up and Slide-Down Effects with Different Speeds</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #B0C4DE;
  }
</style>
<script>
$(document).ready(function(){
  // Sliding up displayed paragraphs with different speeds
  $(".up-btn").click(function(){
    $("p.normal").slideUp();
    $("p.fast").slideUp("fast");
    $("p.slow").slideUp("slow");
    $("p.very-fast").slideUp(50);
    $("p.very-slow").slideUp(2000);
  });

  // Sliding down hidden paragraphs with different speeds
  $(".down-btn").click(function(){
    $("p.normal").slideDown();
    $("p.fast").slideDown("fast");
    $("p.slow").slideDown("slow");
    $("p.very-fast").slideDown(50);
    $("p.very-slow").slideDown(2000);
  });
});
</script>
</head>
<body>
  <button type="button" class="up-btn">Slide Up Paragraphs</button>
  <button type="button" class="down-btn">Slide Down Paragraphs</button>
  <p class="very-fast">This paragraph will fade in/out with very fast speed.</p>
  <p class="normal">This paragraph will fade in/out with default speed.</p>
  <p class="fast">This paragraph will fade in/out with fast speed.</p>
  <p class="slow">This paragraph will fade in/out with slow speed.</p>
  <p class="very-slow">This paragraph will fade in/out with very slow speed.</p>
</body>
</html>
```

You can also specify a callback function to be executed after the slideUp() or slideDown() method completes. We'll learn more about the callback function in upcoming chapters.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Slide-Up and Slide-Down Effects with Callback</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #B0C4DE;
  }
</style>
<script>
$(document).ready(function(){
  // Display alert message after sliding up paragraphs
  $(".up-btn").click(function(){
    $("p").slideUp("slow", function(){
      // Code to be executed
      alert("The slide-up effect is completed.");
    });
  });

  // Display alert message after sliding down paragraphs
  $(".down-btn").click(function(){
    $("p").slideDown("slow", function(){
      // Code to be executed
      alert("The slide-down effect is completed.");
    });
  });
});
</script>
</head>
<body>
  <button type="button" class="up-btn">Slide Up Paragraphs</button>
  <button type="button" class="down-btn">Slide Down Paragraphs</button>
  <p>This is a paragraph.</p>
</body>
</html>
```

jQuery slideToggle() Method

The jQuery slideToggle() method show or hide the selected elements by animating their height in such a way that if the element is initially displayed, it will be slide up; if hidden, it will be slide down i.e. toggles between the slideUp() and slideDown() methods.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Slide-Toggle Effect</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  p{
    padding: 15px;
    background: #B0C4DE;
  }
</style>
<script>
$(document).ready(function(){
  // Toggles paragraphs display with sliding
  $(".toggle-btn").click(function(){
    $("p").slideToggle();
  });
});
</script>
</head>
<body>
  <button type="button" class="toggle-btn">Slide Toggle Paragraphs</button>
  <p>This is a paragraph.</p>
  <p>This is another paragraph.</p>
</body>
</html>
```

Similarly, you can specify the duration parameter for the slideToggle() method like slideUp() and slideDown() methods to control the speed of the slide toggle animation.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Slide-Toggle Effects with Different Speeds</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   p{
      padding: 15px;
      background: #B0C4DE;
   }
</style>
<script>
$(document).ready(function(){
   // Slide Toggles paragraphs with different speeds
   $(".toggle-btn").click(function(){
      $("p.normal").slideToggle();
      $("p.fast").slideToggle("fast");
      $("p.slow").slideToggle("slow");
      $("p.very-fast").slideToggle(50);
      $("p.very-slow").slideToggle(2000);
   });
});
</script>
</head>
<body>
   <button type="button" class="toggle-btn">Fade Toggle Paragraphs</button>
   <p class="very-fast">This paragraph will slide up/down with very fast speed.</p>
   <p class="normal">This paragraph will slide up/down with default speed.</p>
   <p class="fast">This paragraph will slide up/down with fast speed.</p>
   <p class="slow">This paragraph will slide up/down with slow speed.</p>
   <p class="very-slow">This paragraph will slide up/down with very slow speed.</p>
</body>
</html>
```

Similarly, you can also specify a callback function for the slideToggle() method.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Slide-Toggle Effects with Callback</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   p{
      padding: 15px;
      background: #B0C4DE;
   }
</style>
<script>
$(document).ready(function(){
   // Display alert message after slide toggling paragraphs
   $(".toggle-btn").click(function(){
      $("p").slideToggle(1000, function(){
         // Code to be executed
         alert("The slide-toggle effect is completed.");
      });
   });
});
</script>
</head>
<body>
   <button type="button" class="toggle-btn">Fade Toggle Paragraphs</button>
   <p>This is a paragraph.</p>
</body>
</html>
```

4.jQuery animate() Method

The jQuery animate() method is used to create custom animations. The animate() method is typically used to animate numeric CSS properties, for example, width, height, margin, padding, opacity, top, left, etc. but the non-numeric properties such as color or background-color cannot be animated using the basic jQuery functionality.

Note: Not all CSS properties are animatable. In general, any CSS property that accepts values that are numbers, lengths, percentages, or colors is animatable. However, the color animation is not support by the core jQuery library. To manipulate and animate the color use the jQuery color plugin.

Syntax: The basic syntax of the jQuery animate() method can be given with:

```
$(selector).animate({ properties }, duration, callback);
```

The parameters of the animate() method have the following meanings:

- The required properties parameter defines the CSS properties to be animated.
- The optional duration parameter specifies how long the animation will run. Durations can be specified either using one of the predefined string 'slow' or 'fast', or in a number of milliseconds; higher values indicate slower animations.
- The optional callback parameter is a function to call once the animation is complete.

a simple example of the jQuery animate() method that animates an image from its original position to the right by 300 pixels on click of the button.

Note: All HTML elements have static position by default. Since the static element cannot be moved, so you must set the CSS position property for the element to relative, fixed, or absolute to manipulate or animate its position.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Animation Effects</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  img{
    position: relative; /* Required to move element */
  }
</style>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("img").animate({
      left: 300
    });
  });
});
</script>
</head>
<body>
  <button type="button">Start Animation</button>
  <p>
            <img src="/examples/images/mushroom.jpg" alt="Mushroom">
  </p>
</body>
</html>
```

Animate Multiple Properties At Once

You can also animate multiple properties of an element together at the same time using the animate() method. All the properties animated simultaneously without any delay.

Note: The CSS properties names must be camel-cased when using with the animate() method, e.g. if you want to animate the font size you need to write 'fontSize' rather than 'font-size'. Similarly, write 'marginLeft' instead of 'margin-left', 'borderWidth' instead of 'border-width', and so on.

Tip: You must set the border-style property for the element before animating its border-width property. An element must have borders before you can animate the border width, because

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Multiple Properties Animation</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  .box{
    width: 100px;
    height: 100px;
    background: #9d7ede;
    margin-top: 30px;
    border-style: solid; /* Required to animate border width */
    border-color: #6f40ce;
  }
</style>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $(".box").animate({
      width: "300px",
      height: "300px",
      marginLeft: "150px",
      borderWidth: "10px",
      opacity: 0.5
    });
  });
});
</script>
</head>
<body>
  <button type="button">Start Animation</button>
  <div class="box"></div>
</body>
</html>
```

Animate Multiple Properties One by One or Queued Animations

You can also animate the multiple properties of an element one by one individually in a queue using the jQuery's chaining feature. We'll learn more about chaining in next chapter.

The following example demonstrates a jQuery queued or chained animation, where each animation will start once the previous animation on the element has completed.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Queued Animation</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  .box{
    width: 100px;
    height: 100px;
    background: #9d7ede;
    margin-top: 30px;
    border-style: solid; /* Required to animate border width */
    border-color: #6f40ce;
  }
</style>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $(".box")
      .animate({width: "300px"})
      .animate({height: "300px"})
      .animate({marginLeft: "150px"})
      .animate({borderWidth: "10px"})
      .animate({opacity: 0.5});
  });
});
</script>
</head>
<body>
  <button type="button">Start Animation</button>
  <div class="box"></div>
</body>
</html>
```

Animate Properties with Relative Values

You can also define the relative values for the animated properties. If a value is specified with a leading += or -= prefix, then the target value is calculated by adding or subtracting the given number from the current value of the property.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Animation with Relative Values</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  .box{
    width: 100px;
    height: 100px;
    background: #9d7ede;
    margin-top: 30px;
    position: relative; /* Required to move element */
  }
</style>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $(".box").animate({
      top: "+=50px",
      left: "+=50px",
      width: "+=50px",
      height: "+=50px"
    });
  });
});
</script>
</head>
<body>
  <button type="button">Start Animation</button>
  <p><strong>Note:</strong> Click the "Start Animation" button multiple times to see how the relative value works.</p>
  <div class="box"></div>
</body>
</html>
```

Animate Properties with Pre-defined Values

In addition to the numeric values, each property can take the strings 'show', 'hide', and 'toggle'. It will be very helpful in a situation when you simply want to animate the property from its current value to the initial value and vice versa.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Animation with Pre-defined Values</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  .box{
    width: 80%;
    height: 200px;
    background: #9d7ede;
    margin-top: 30px;
  }
</style>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $(".box").animate({
      width: 'toggle'
    });
  });
});
</script>
</head>
<body>
  <button type="button">Toggle Animation</button>
  <div class="box"></div>
</body>
</html>
```

a. jQuery stop() Method

The jQuery stop() method is used to stop the jQuery animations or effects currently running on the selected elements before it completes.

The basic syntax of the jQuery stop() method can be given with:

```
$(selector).stop(stopAll, goToEnd);
```

→The parameters in the above syntax have the following meanings:

- The optional stopAll Boolean parameter specifies whether to remove queued animation or not. Default value is false, that means only the current animation will be stopped, rest of the animations in the queue will run afterwards.
- The optional goToEnd Boolean parameter specifies whether to complete the current animation immediately. Default value is false.

Here's a simple example that demonstrates the jQuery stop() method in real action in which you can start and stop the animation on click of the button.

Note: The jQuery stop() method works for all jQuery effects such as fading, sliding, animated show and hide effects as well as custom animations.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Stop Currently Running Animations</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  img{
    position: relative; /* Required to move element */
  }
</style>
<script>
$(document).ready(function(){
  // Start animation
  $(".start-btn").click(function(){
   $("img").animate({left: "+=150px"}, 2000);
  });

  // Stop running animation
  $(".stop-btn").click(function(){
   $("img").stop();
  });

  // Start animation in the opposite direction
  $(".back-btn").click(function(){
   $("img").animate({left: "-=150px"}, 2000);
  });

  // Reset to default
  $(".reset-btn").click(function(){
   $("img").animate({left: "0"}, "fast");
  });
});
</script>
</head>
<body>
  <button type="button" class="start-btn">Start</button>
  <button type="button" class="stop-btn">Stop</button>
  <button type="button" class="back-btn">Back</button>
  <button type="button" class="reset-btn">Reset</button>
  <p>
                <img src="/examples/images/mushroom.jpg" alt="Mushroom">
  </p>
</body>
```

Here's one more example of this method in which, if you click the "Slide Toggle" button again after starting the animation but before it is completed, the animation will begin in the opposite direction immediately from the saved starting point.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Stop Current Animation and Play Next Animation in Queue</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  .box{
    width: 300px;
    height: 200px;
    background: #9d7ede;
    margin-top: 20px;
    border: 3px solid #6f40ce;
  }
</style>
<script>
$(document).ready(function(){
  // Kill and toggle the current sliding animation
  $(".toggle-btn").on("click", function(){
    $(".box").stop().slideToggle(1000);
  });
});
</script>
</head>
<body>
            <p><strong>Note:</strong> Click the "Slide Toggle" button to start the animation, then click again before the animation is completed to understand this example.</p>
  <button type="button" class="toggle-btn">Slide Toggle</button>
  <div class="box"></div>
</body>
</html>
```

Creating Smooth Hover Effect

While creating the animated hover effect one of the common problem you may face is multiple queued animations, when you place and remove the mouse cursor rapidly. Because, in this situation mouseenter or mouseleave events are triggered quickly before the animation complete. To avoid this problem and create a nice and smooth hover effect you can add the stop(true, true) to the method chain, like this:

Note: The jQuery method stop(true, true) clears all the queued animations and jumps the current animation to the final value.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Stop Current Animation and Play Next Animation in Queue</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  .box{
    width: 300px;
    height: 200px;
    background: #9d7ede;
    margin-top: 20px;
    border: 3px solid #6f40ce;
  }
</style>
<script>
$(document).ready(function(){
  // Kill and toggle the current sliding animation
  $(".toggle-btn").on("click", function(){
    $(".box").stop().slideToggle(1000);
  });
});
</script>
</head>
<body>
        <p><strong>Note:</strong> Click the "Slide Toggle" button to start the animation, then click again before the animation is completed to understand this example.</p>
  <button type="button" class="toggle-btn">Slide Toggle</button>
  <div class="box"></div>
</body>
```

6. jQuery Method Chaining

The jQuery provides another robust feature called method chaining that allows us to perform multiple action on the same set of elements, all within a single line of code.

This is possible because most of the jQuery methods return a jQuery object that can be further used to call another method. Here's an example.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Method Chaining</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
/* Some custom styles to beautify this example */
p {
   width: 200px;
                  padding: 40px 0;
                  font: bold 24px sans-serif;
                  text-align: center;
   background: #aaccaa;
   border: 1px solid #63a063;
   box-sizing: border-box;
}
</style>
<script>
$(document).ready(function(){
   $(".start").click(function(){
      $("p").animate({width: "100%"}).animate({fontSize: "46px"}).animate({borderWidth: 30});
   });
   $(".reset").click(function(){
      $("p").removeAttr("style");
   });
});
</script>
</head>
<body>
   <p>Hello World!</p>
   <button type="button" class="start">Start Chaining</button>
   <button type="button" class="reset">Reset</button>
</body>
</html>
```

The this example demonstrate the chaining of three animate() method. When a user click the trigger button, it expands the <p> to 100% width. Once the width change is complete the font-size is start animating and after its completion, the border animation will begin.

Tip: The method chaining not only helps you to keep your jQuery code concise, but it also can improve your script's performance since browser doesn't have to find the same elements multiple times to do something with them.

You can also break a single line of code into multiple lines for greater readability. For example, the sequence of methods in the above example could also be written as:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Method Chaining</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
/* Some custom styles to beautify this example */
p {
   width: 200px;
                    padding: 40px 0;
                    font: bold 24px sans-serif;
                    text-align: center;
   background: #aaccaa;
   border: 1px solid #63a063;
   box-sizing: border-box;
}
</style>
<script>
$(document).ready(function(){
   $(".start").click(function(){
      $("p")
         .animate({width: "100%"})
         .animate({fontSize: "46px"})
         .animate({borderWidth: 30});
   });
   $(".reset").click(function(){
      $("p").removeAttr("style");
   });
});
</script>
</head>
<body>
   <p>Hello World!</p>
   <button type="button" class="start">Start Chaining</button>
   <button type="button" class="reset">Reset</button>
</body>
</html>
```

Some jQuery methods doesn't return the jQuery object. In general, setters i.e. methods that assign some value on a selection return a jQuery object, that allows you to continue calling jQuery methods on your selection. Whereas, getters return the requested value, so you can't continue to call jQuery methods on the value returned by the getter.

A typical example of this scenario is the html() method. If no parameters are passed to it, the HTML contents of the selected element is returned instead of a jQuery object.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Method Chaining Exceptions</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  .test{
                                    padding: 15px;
    background: yellow;
  }
</style>
<script>
$(document).ready(function(){
  $("button").click(function(){
    // This will work
    $("h1").html("Hello World!").addClass("test").fadeOut(1000);

    // This will NOT work
    $("p").html().addClass("test");
  });
});
</script>
</head>
<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph.</p>
  <button type="button">Start Chaining</button>
</body>
</html>
```

6. **jQuery Callback**

jQuery Callback Functions

JavaScript statements are executed line by line. But, since jQuery effect takes some time to finish the next line code may execute while the previous effect is still running. To prevent this from happening jQuery provides a callback function for each effect method.

A callback function is a function that is executed once the effect is complete. The callback function is passed as an argument to the effect methods and they typically appear as the last argument of the method. For example, the basic syntax of the jQuery slideToggle() effect method with a callback function can be given with:

```
$(selector).slideToggle(duration, callback);
```

Consider the following example in which we've placed the slideToggle() and alert() statements next to each other. If you try this code the alert will be displayed immediately once you click the trigger button without waiting for slide toggle effect to complete.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Effect Method without Callback Function</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   p{
      background:yellow;
      font-size: 24px;
      padding:20px;
   }
</style>
<script>
$(document).ready(function(){
   $("button").click(function(){
      $("p").slideToggle("slow");
      alert("The slide toggle effect has completed.");
   });
});
</script>
</head>
<body>
   <p>This is paragraph.</p>
   <button type="button">Click me</button>
</body>
</html>
```

And, here's the modified version of the pevious example in which we've placed the alert() statement inside a callback function for the slideToggle() method. If you try this code the alert message will be displayed once the slide toggle effect has completed.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Effect Method with Callback Function</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   p{
      background:yellow;
      font-size: 24px;
      padding:20px;
   }
</style>
<script>
$(document).ready(function(){
   $("button").click(function(){
      $("p").slideToggle("slow", function(){
         alert("The slide toggle effect has completed.");
      });
   });
});
</script>
</head>
<body>
   <p>This is paragraph.</p>
   <button type="button">Click me</button>
</body>
</html>
```

Similarly, you can define the callback functions for the other jQuery effect methods, like show(), hide(), fadeIn(), fadeOut(), animate(), etc.

Note: If the effect method is applied to multiple elements, then the callback function is executed once for each selected element, not once for all.

If you try this example code, it will display the same alert message two times once per <h1> and <p> element, upon clicking the trigger button.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Example of jQuery Callback Executed Multiple Times</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
  h1{
    display:none;
    background:red;
    padding:20px;
  }
  p{
    background:yellow;
    font-size: 24px;
    padding:20px;
  }
</style>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("h1, p").slideToggle("slow", function(){
      alert("The slide toggle effect has completed.");
    });
  });
});
</script>
</head>
<body>
  <h1>This is heading</h1>
  <p>This is paragraph.</p>
  <button type="button">Click me</button>
</body>
</html>
```

UI Library Based Effects

To use these effects you can either download latest jQuery UI Library jquery-ui-1.11.4.custom.zip from jQuery UI Library or use Google CDN to use it in the similar way as we have done for jQuery.

We have used Google CDN for jQuery UI using following code snippet in the HTML page so we can use jQuery UI –

```
<head>
  <script src = "https://ajax.googleapis.com/ajax/libs/jqueryui/1.11.3/jquery-ui.min.js">
  </script>
</head>
```

SNo	Methods & Description
1	BlindBlinds the element away or shows it by blinding it in.
2	BounceBounces the element vertically or horizontally n-times.
3	ClipClips the element on or off, vertically or horizontally.
4	DropDrops the element away or shows it by dropping it in.
5	ExplodeExplodes the element into multiple pieces.
6	FoldFolds the element like a piece of paper.
7	HighlightHighlights the background with a defined color.
8	PuffScale and fade out animations create the puff effect.
9	PulsatePulsates the opacity of the element multiple times.
10	ScaleShrink or grow an element by a percentage factor.
11	ShakeShakes the element vertically or horizontally n-times.
12	SizeResize an element to a specified width and height.
13	SlideSlides the element out of the viewport.
14	TransferTransfers the outline of an element to another.

JQUERY MANIPULATION

- jQuery Get/Set
- jQuery Insert
- jQuery Remove
- jQuery CSS Classes
- jQuery Style Properties
- jQuery Dimensions

1. jQuery Getters & Setter

jQuery Get or Set Contents and Values:

Some jQuery methods can be used to ***either assign or read some value on a selection***. A few of these methods are *text(), html(), attr(),* and val().

When these methods are called with no argument, it is referred to as a getters, because it gets (or reads) the value of the element. When these methods are called with a value as an argument, it's referred to as a setter because it sets (or assigns) that value.

a. jQuery text() Method

The jQuery text() method is either used to get the combined text contents of the selected elements, including their descendants, or set the text contents of the selected elements.

Get Contents with text() Method

The following example will show you how to get the text contents of paragraphs:

Note: The jQuery text() retrieves the values of all the selected elements (i.e. combined text), whereas the other getters such as html(), attr(), and val() returns the value only from the first element in the selection.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Get Text Contents of the Elements</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   $(".btn-one").click(function(){
      var str = $("p").text();
      alert(str);
   });
   $(".btn-two").click(function(){
     var str = $("p:first").text();
     alert(str);
   });
   $(".btn-three").click(function(){
     var str = $("p.extra").text();
     alert(str);
   });
});
</script>
</head>
<body>
   <button type="button" class="btn-one">Get All Paragraph's Text</button>
   <button type="button" class="btn-two">Get First Paragraph's Text</button>
                <button type="button" class="btn-three">Get Last Paragraph's Text</button>
   <p>This is a paragraph.</p>
   <p>This is another paragraph.</p>
                <p class="extra">This is one more paragraph.</p>
</body>
</html>
```

ii. Set Contents with text() Method

The following example will show you how to set the text contents of a paragraph:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set Text Contents of the Elements</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $(".btn-one").click(function(){
    $("p").text("This is demo text.");
  });
  $(".btn-two").click(function(){
    $("p:first").text("This is another demo text.");
  });
  $(".btn-three").click(function(){
    $("p.empty").text("This is one more demo text.");
  });
});
</script>
</head>
<body>
  <button type="button" class="btn-one">Set All Paragraph's Text</button>
  <button type="button" class="btn-two">Set First Paragraph's Text</button>
  <button type="button" class="btn-three">Set Empty Paragraph's Text</button>
  <p>This is a test paragraph.</p>
  <p>This is another test paragraph.</p>
  <p class="empty"></p>
</body>
</html>
```

Note: When the jQuery text(), html(), attr(), and val() methods are called with a value as an argument it sets that value to every matched element.

b. jQuery html() Method

The jQuery html() method is used to get or set the HTML contents of the elements.

i. Get HTML Contents with html() Method:

The following example will show you how to get the HTML contents of the paragraph elements as well as a element container:

Note: If multiple elements are selected, the html() method only returns the HTML contents of the first element from the set of matched elements.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Get HTML Contents of an Element</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $(".btn-one").click(function(){
    var str = $("p").html();
    alert(str);
  });
  $(".btn-two").click(function(){
    var str = $("#container").html();
    alert(str);
  });
});
</script>
</head>
<body>
  <button type="button" class="btn-one">Get Paragraph's HTML Contents</button>
  <button type="button" class="btn-two">Get Container's HTML Contents</button>
  <div id="container">
    <h1>Hello World!</h1>
    <p>The quick <b>brown fox</b> jumps over the lazy dog.</p>
  </div>
</body>
</html>
```

ii. Set HTML Contents with html() Method

The following example will show you how to set the HTML contents of the <body> element:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set HTML Contents of the Element</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   $("button").click(function(){
      $("body").html("<p>Hello World!</p>");
   });
});
</script>
</head>
<body>
   <button type="button">Write Message</button>
</body>
</html>
```

3. jQuery attr() Method

You can use the jQuery attr() method to either get the value of an element's attribute or set one or more attributes for the selected element.

i. Get Attribute Value with attr() Method:

The following example will show you how get the href attribute of the hyperlink i.e. the <a> element as well as the alt attribute of an <img> element:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Get an Element's Attribute Value</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   $(".btn-one").click(function(){
      var str = $("a").attr("href");
      alert(str);
   });
   $(".btn-two").click(function(){
      var str = $("img#sky").attr("alt");
      alert(str);
   });
});
</script>
</head>
<body>
   <button type="button" class="btn-one">Get Link's HREF Attribute</button>
   <button type="button" class="btn-two">Get Image ALT Attribute</button>
   <p><a href="https://www.tutorialrepublic.com/">Tutorial Republic</a></p>
   <img id="sky" src="/examples/images/sky.jpg" alt="Cloudy Sky">
</body>
</html>
```

Note: If multiple elements are selected, the attr() method only returns the attribute value of the first element from the set of matched elements.

ii. Set Attributes with attr() Method:

The following example will show you how to set the checked attribute of the checkbox.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set Element's Attribute Value</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $('input[type="checkbox"]').attr("checked", "checked");
  });
});
</script>
</head>
<body>
  <p><label><input type="checkbox"></label> I agree with terms and conditions.</p>
  <button type="button">Check</button>
</body>
</html>
```

The attr() method also allows you to set multiple attributes at a time. The following example will show you how to set the class and title attribute for the <img> elements.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set Multiple Attribute for the Elements</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("img").attr({
      "class" : "frame",
      "title" : "Hot Air Balloons"
    });
  });
});
</script>
<style>
  .frame{
    border: 6px solid #000;
  }
</style>
</head>
<body>
  <button type="button">Set Attributes for Image</button>
  <p>
    <img src="/examples/images/balloons.jpg"  alt="Hot Air Balloons">
  </p>
</body>
</html>
```

4. jQuery val() Method

The jQuery val() method is mainly used to get or set the current value of the HTML form elements such as <input>, <select> and <textarea>.

i.Get the Values of Form Fields with val() Method

The following example will show you how to get the values of form controls:

Note: If multiple form elements are selected, the val() method only returns the value of the first element from the set of matched elements.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Get a Form Field Value</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button.get-name").click(function(){
    var name = $("#name").val();
    alert(name);
  });
  $("button.get-comment").click(function(){
    var comment = $("#comment").val();
    alert(comment);
  });
  $("button.get-city").click(function(){
    var city = $("#city").val();
    alert(city);
  });
});
</script>
</head>
<body>
  <form>
    <table>
      <tr>
        <td>Name:</td>
        <td>
          <input type="text" id="name">
        </td>
      </tr>
      <tr>
        <td>Comments:</td>
        <td>
          <textarea rows="4" cols="30" id="comment"></textarea>
        </td>
      </tr>
      <tr>
        <td>City:</td>
        <td>
          <select id="city">
            <option>London</option>
            <option>Paris</option>
            <option>New York</option>
          </select>
        </td>
      </tr>
    </table>
  </form>
  <p><strong>Note:</strong> Fill the above form and click the following button to get the value.</p>
  <button type="button" class="get-name">Get Name</button>
  <button type="button" class="get-comment">Get Comment</button>
  <button type="button" class="get-city">Get City</button>
</body>
```

ii.Set the Values of Form Fields with val() Method

The following example will show you how to set the values of the form controls:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set Form Fields Values</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   $("button").click(function(){
      var text = $(this).text();
      $('input[type="text"]').val(text);
   });
});
</script>
</head>
<body>
   <button type="button">Discovery</button>
   <button type="button">Atlantis</button>
   <button type="button">Endeavour</button>
   <p><strong>Note:</strong> Click the above buttons to set the value of
following input box.</p>
   <p>
      <input type="text">
   </p>
</body>
</html>
```

2.jQuery Insert Content

a. jQuery Insert New Content

jQuery provides several methods, like *append(), prepend(), html(), text(), before(), after(), wrap()* etc. that allows us to insert new content inside an existing element.

a. jQuery append() Method

The jQuery append() method is used to insert content to the **end** of the selected elements.

The following example will append some HTML to all the paragraphs on document ready, whereas append some text to the container element on button click.

Note: The contents or elements inserted using the jQuery append() and prepend() methods is added inside of the selected elements.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Inserting HTML Contents At the End of the Elements in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  // Append all paragraphs on document ready
  $("p").append(' <a href="#">read more...</a>');

  // Append a div container on button click
  $("button").click(function(){
    $("#container").append("This is demo text.");
  });
});
</script>
</head>
<body>
  <button type="button">Insert Text</button>
  <div id="container">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante.</p>
    <p>Quis quam ut magna consequat faucibus. Pellentesque eget nisi a mi suscipit tincidunt. Ut tempus dictum risus. Pellentesque viverra sagittis quam at mattis. Suspendisse potenti.</p>
  </div>
</body>
</html>
```

b. jQuery prepend() Method

The prepend() method is used to insert content to the **beginning** of the selected elements.

The following example will prepend some HTML to all the paragraphs on document ready, whereas prepend some text to the container element on button click.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Inserting HTML Contents At the Start of the Elements in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  // Prepend all paragraphs on document ready
  $("p").prepend("<strong>Note:</strong> ");

  // Prepend a div container on button click
  $("button").click(function(){
    $("#container").prepend("This is demo text.");
  });
});
</script>
</head>
<body>
  <button type="button">Insert Text</button>
  <div id="container">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante.</p>
    <p>Quis quam ut magna consequat faucibus. Pellentesque eget nisi a mi suscipit tincidunt. Ut tempus dictum risus. Pellentesque viverra sagittis quam at mattis. Suspendisse potenti.</p>
  </div>
</body>
</html>
```

Insert Multiple Elements with append() & prepend() Method

The jQuery append() and prepend() also supports passing in multiple arguments as input.

The jQuery code in the following example will insert a <h1>, <p> and an <img> element inside the <body> element as a last three child nodes.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Append or Prepend Multiple Elements in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    var newHeading = "<h1>Important Note:</h1>";
    var newParagraph = document.createElement("p");
    newParagraph.innerHTML = "<em>Lorem Ipsum is dummy text...</em>";
    var newImage = $('<img src="/examples/images/smiley.png" alt="Symbol">');
    $("body").append(newHeading, newParagraph, newImage);
  });
});
</script>
</head>
<body>
  <button type="button">Insert Contents</button>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante, metus ac nisl bibendum.</p>
</body>
</html>
```

c. jQuery before() Method

The jQuery before() method is used to insert content before the selected elements.

The following example will insert a paragraph before the container element on document ready, whereas insert an image before the <h1> element on button click.

Note: The contents or elements inserted using the jQuery before() and after() methods is added outside of the selected elements.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Inserting HTML Contents Before or After the Elements in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  // Add content after a div container on document ready
  $("#container").after("<p>&mdash; The End &mdash;</p>");

  // Add content before a div container on document ready
  $("#container").before("<p>&mdash; Demo Text &mdash;</p>");

  // Add content after heading on button click
  $("button.insert-after").click(function(){
    $("h1").after('<img src="/examples/images/marker-right.gif" alt="Symbol">');
  });

  // Add content before heading on button click
  $("button.insert-before").click(function(){
    $("h1").before('<img src="/examples/images/marker-left.gif" alt="Symbol">');
  });
});
</script>
<style>
  h1{
    display: inline-block; /* To place marker image and heading in one line */
  }
  body{
    text-align: center;
  }
</style>
</head>
<body>
  <h1>Hello World</h1>
  <hr>
  <button type="button" class="insert-before">Insert Before</button>
  <button type="button" class="insert-after">Insert After</button>
  <hr>
  <div id="container">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante. Vestibulum id metus ac nisl bibendum scelerisque non non purus. Suspendisse varius nibh non aliquet sagittis. In tincidunt orci sit amet elementum vestibulum. Vivamus fermentum in arcu in aliquam. Quisque aliquam porta odio in fringilla. Vivamus nisl leo, blandit at bibendum eu, tristique eget risus. Integer aliquet quam ut elit suscipit, id interdum neque porttitor. Integer faucibus ligula.</p>
    <p>Quis quam ut magna consequat faucibus. Pellentesque eget nisi a mi suscipit tincidunt. Ut tempus dictum risus. Pellentesque viverra sagittis quam at mattis. Suspendisse potenti. Aliquam sit amet gravida nibh, facilisis gravida odio. Phasellus auctor velit at lacus blandit, commodo iaculis justo viverra. Etiam vitae est arcu. Mauris vel congue dolor. Aliquam eget mi mi. Fusce quam tortor, commodo ac dui quis, bibendum viverra erat. Maecenas mattis lectus enim, quis tincidunt dui molestie euismod. Curabitur et diam tristique, accumsan nunc eu, hendrerit tellus.</p>
  </div>
</body>
</html>
```

d. jQuery after() Method

The jQuery after() method is used to insert content *after the selected elements.*

The following example will insert a paragraph after the container element on document ready, whereas insert an image after the <h1> element on button click.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Inserting HTML Contents Before or After the Elements in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  // Add content after a div container on document ready
  $("#container").after("<p>&mdash; The End &mdash;</p>");

  // Add content before a div container on document ready
  $("#container").before("<p>&mdash; Demo Text &mdash;</p>");

  // Add content after heading on button click
  $("button.insert-after").click(function(){
    $("h1").after('<img src="/examples/images/marker-right.gif" alt="Symbol">');
  });

  // Add content before heading on button click
  $("button.insert-before").click(function(){
    $("h1").before('<img src="/examples/images/marker-left.gif" alt="Symbol">');
  });
});
</script>
<style>
  h1{
    display: inline-block; /* To place marker image and heading in one line */
  }
  body{
    text-align: center;
  }
</style>
</head>
<body>
  <h1>Hello World</h1>
  <hr>
  <button type="button" class="insert-before">Insert Before</button>
  <button type="button" class="insert-after">Insert After</button>
  <hr>
  <div id="container">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante. Vestibulum id metus ac nisl bibendum scelerisque non non purus. Suspendisse varius nibh non aliquet sagittis. In tincidunt orci sit amet elementum vestibulum. Vivamus fermentum in arcu in aliquam. Quisque aliquam porta odio in fringilla. Vivamus nisl leo, blandit at bibendum eu, tristique eget risus. Integer aliquet quam ut elit suscipit, id interdum neque porttitor. Integer faucibus ligula.</p>
    <p>Quis quam ut magna consequat faucibus. Pellentesque eget nisi a mi suscipit tincidunt. Ut tempus dictum risus. Pellentesque viverra sagittis quam at mattis. Suspendisse potenti. Aliquam sit amet gravida nibh, facilisis gravida odio. Phasellus auctor velit at lacus blandit, commodo iaculis justo viverra. Etiam vitae est arcu. Mauris vel congue dolor. Aliquam eget mi mi. Fusce quam tortor, commodo ac dui quis, bibendum viverra erat. Maecenas mattis lectus enim, quis tincidunt dui molestie euismod. Curabitur et diam tristique, accumsan nunc eu, hendrerit tellus.</p>
  </div>
</body>
</html>
```

Insert Multiple Elements with before() & after() Method

The jQuery before() and after() also supports passing in multiple arguments as input. The following example will insert a <h1>, <p> and an <img> element before the <p> elements.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Inserting Multiple Elements Before or After the Elements in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   $("button").click(function(){
      var newHeading = "<h2>Important Note:</h2>";
      var newParagraph = document.createElement("p");
      newParagraph.innerHTML = "<em>Lorem Ipsum is dummy text...</em>";
      var newImage = $('<img src="/examples/images/smiley.png" alt="Symbol">');
      $("p").before(newHeading, newParagraph, newImage);
   });

});
</script>
</head>
<body>
   <button type="button">Insert Contents</button>
   <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante, metus ac nisl bibendum.</p>
</body>
</html>
```

e. jQuery wrap() Method

The jQuery wrap() method is used to wrap an HTML structure around the selected elements.

The jQuery code in the following example will wrap the container elements with a element with the class .wrapper on document ready, whereas wrap all the inner content of the paragraph elements first with the <b> and again with element.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Wrapping HTML Around the Elements in jQuery</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  // Wrap div container with another div on document ready
  $(".container").wrap('<div class="wrapper"></div>');

  // Wrap paragraph's content on button click
  $("button").click(function(){
    $("p").contents().wrap("<em><b></b></em>");
  });
});
</script>
<style>
  .wrapper{
    padding: 20px;
    background: #f0e68c;
    margin: 10px 0;
  }
  .container{
    padding: 15px;
    background: #fff;
    font-size: 24px;
  }
</style>
</head>
<body>
  <button type="button">Wrap Demo Text</button>
  <div class="container">
    <p>This is demo text.</p>
  </div>
</body>
</html>
```

3.jQuery Remove Elements & Attribute

jQuery Remove Elements or Contents

jQuery provides handful of methods, such as *empty(), remove(), unwrap()* etc. to remove existing HTML elements or contents from the document.

a. jQuery empty() Method

The jQuery empty() method removes all child elements as well as other descendant elements and the text content within the selected elements from the DOM.

The following example will remove all the content inside of the elements with the class .container on click of the button.

Note: According to the W3C (World Wide Web Consortium) DOM specification, any string of text within an element is considered a child node of that element.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Removing the Contents of the Elements in jQuery</title>
<style>
.container{
   padding: 10px;
   background: #f0e68C;
   border: 1px solid #bead18;
}
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   // Empty container div on button click
   $("button").click(function(){
     $(".container").empty();
   });
});
</script>
</head>
<body>
   <div class="container">
      <h1>Hello World!</h1>
      <p class="hint"><strong>Note:</strong> If you click the following button it will remove all the contents of
the container div including the button.</p>
      <button type="button">Empty Container</button>
   </div>
</body>
</html>
```

b. jQuery remove() Method

The jQuery remove() method removes the selected elements from the DOM as well as everything inside it. In addition to the elements themselves, all bound events and jQuery data associated with the elements are removed.

The following example will remove all the <p> elements with the class .hint from the DOM on button click. Nested elements inside these paragraphs will be removed, too.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Removing the Elements from DOM in jQuery</title>
<style>
.container{
   padding: 10px;
   background: #f0e68C;
   border: 1px solid #bead18;
}
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   // Removes paragraphs with class "hint" from DOM on button click
   $("button").click(function(){
     $("p.hint").remove();
   });
});
</script>
</head>
<body>
   <div class="container">
      <h1>Hello World!</h1>
      <p class="hint"><strong>Note:</strong> If you click the following button it will remove this
paragraph.</p>
      <button type="button">Remove Hint Paragraph</button>
   </div>
</body>
</html>
```

The jQuery remove() method can also include a selector as an optional parameter, that allows you to filter the elements to be removed. For instance, the previous example's jQuery DOM removal code could be rewritten as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Removing the Elements from DOM in jQuery</title>
<style>
.container{
   padding: 10px;
   background: #f0e68C;
   border: 1px solid #bead18;
}
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   // Removes paragraphs with class "hint" from DOM on button click
   $("button").click(function(){
     $("p").remove(".hint");
   });
});
</script>
</head>
<body>
   <div class="container">
      <h1>Hello World!</h1>
      <p class="hint"><strong>Note:</strong> If you click the following button it will
remove this paragraph.</p>
      <button type="button">Remove Hint Paragraph</button>
   </div>
</body>
</html>
```

Note: You can also include selector expression as a parameter within the jQuery remove() method, like remove(".hint, .demo") to filter multiple elements.

c. jQuery unwrap() Method

The jQuery unwrap() method removes the ***parent elements*** of the selected elements from the DOM. This is typically the inverse of the wrap() method.

The following example will remove the parent element of <p> elements on button click.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Removing the Parents of the Elements from DOM in jQuery</title>
<style>
.container{
   padding: 10px;
   background: #f0e68C;
   border: 1px solid #bead18;
}
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   // Removes the paragraph's parent element on button click
   $("button").click(function(){
     $("p").unwrap();
   });
});
</script>
</head>
<body>
   <div class="container">
      <h1>Hello World!</h1>
      <p class="hint"><strong>Note:</strong> If you click the following button it will remove the parent
element of this paragraph.</p>
      <button type="button">Remove Paragraph's Parent</button>
   </div>
</body>
</html>
```

d. jQuery removeAttr() Method

The jQuery removeAttr() method removes an attribute from the selected elements.

The example below will remove the href attribute form the <a> elements on button click.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Removing an Attribute from the Elements in jQuery</title>
<style>
                    a{
      font-size: 18px;
                                        margin-right: 20px;
                    }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   // Removes the hyperlink's href attribute on button click
   $("button").click(function(){
      $("a").removeAttr("href");
   });
});
</script>
</head>
<body>
   <div class="container">
      <p>
                                                  <a href="https://www.cutm.ac.in/">Home</a>
                                                  <a href="https://www.tutorialrepublic.com/about/">About</a>
                                                  <a href="https://www.tutorialrepublic.com/contact">Contact</a>
                                        </p>
      <button type="button">Remove Attribute</button>
   </div>
</body>
</html>
```

4.jQuery CSS Classes Manipulation

jQuery provides several methods, such as addClass(), removeClass(), toggleClass(), etc. to manipulate the CSS classes assigned to HTML elements.

a. jQuery addClass() Method

The jQuery addClass() method adds one or more classes to the selected elements.

The following example will add the class .page-header to the <h1> and the class .highlight to the <p> elements with class .hint on button click.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Adding a Class to the Elements in jQuery</title>
<style>
   .page-header{
      color: red;
      text-transform: uppercase;
   }
   .highlight{
      background: yellow;
   }
                     .hint{
      font-style: italic;
   }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   $("button").click(function(){
      $("h1").addClass("page-header");
      $("p.hint").addClass("highlight");
   });
});
</script>
</head>
<body>
   <h1>Demo Text</h1>
   <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit...</p>
   <p class="hint"><strong>Tip:</strong> Lorem Ipsum is dummy text.</p>
   <button type="button">Add Class</button>
</body>
</html>
```

b. jQuery removeClass() Method

Similarly, you can remove the classes from the elements using the jQuery removeClass() method. The removeClass() method can remove a single class, multiple classes, or all classes at once from the selected elements.

The following example will remove the class .page-header from the <h1> and the class .hint and .highlight from the <p> elements on button click.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Removing Classes from the Elements in jQuery</title>
<style>
  .page-header{
    color: red;
    text-transform: uppercase;
  }
  .highlight{
    background: yellow;
  }
                .hint{
    font-style: italic;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("h1").removeClass("page-header");
    $("p").removeClass("hint highlight");
  });
});
</script>
</head>
<body>
  <h1 class="page-header">Demo Text</h1>
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit...</p>
  <p class="hint highlight"><strong>Tip:</strong> Lorem Ipsum is dummy text.</p>
  <button type="button">Remove Class</button>
</body>
</html>
```

c. jQuery toggleClass() Method

The jQuery toggleClass() add or remove one or more classes from the selected elements in such a way that if the selected element already has the class, then it is removed; if an element does not have the class, then it is added i.e. toggle classes.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Toggle the Classes of the Elements in jQuery</title>
<style>
  p{
    padding: 10px;
    cursor: pointer;
    font: bold 16px sans-serif;
  }
  .highlight{
    background: yellow;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("p").click(function(){
    $(this).toggleClass("highlight");
  });
});
</script>
</head>
<body>
  <p>Click on me to toggle highlighting.</p>
  <p class="highlight">Click on me to toggle highlighting.</p>
  <p>Click on me to toggle highlighting.</p>
</body>
</html>
```

5.jQuery Get and Set CSS Properties

a. jQuery css() Method

The jQuery css() method is used to get the computed value of a CSS property or set one or more CSS properties for the selected elements.

This method provides a quick way to apply the styles directly to the HTML elements (i.e. inline styles) that haven't been or can't easily be defined in a stylesheet.

i.Get a CSS Property Value

You can get the computed value of an element's CSS property by simply passing the property name as a parameter to the css() method. Here's the basic syntax:

$(selector).css("propertyName");

The following example will retrieve and display the computed value of the CSS background-color property of a element, when it is clicked.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Get the Value of a CSS Property</title>
<style>
  div{
    width: 100px;
    height: 100px;
    margin: 10px;
    cursor: pointer;
    display: inline-block;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("div").click(function(){
    var color = $(this).css("background-color");
    $("#result").html(color);
  });
});
</script>
</head>
<body>
  <div style="background-color:orange;"></div>
  <div style="background-color:#ee82ee;"></div>
  <div style="background-color:rgb(139,205,50);"></div>
  <div style="background-color:#f00;"></div>
  <p>The computed background-color property value of this DIV element is: <b id="result"></b></p>
</body>
</html>
```

b. Set a Single CSS Property and Value

The css() method can take a property name and value as separate parameters for setting a single CSS property for the elements. The basic syntax can be given with:

```
$(selector).css("propertyName", "value");
```

The following example will set the CSS background-color property of the elements, to the color value blue, when it is clicked.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set the Value of a CSS Property</title>
<style>
   .box{
      width: 100px;
      height: 100px;
      margin: 10px;
      cursor: pointer;
      border: 1px solid #cdcdcd;
      display: inline-block;
   }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
   $(".box").click(function(){
      $(this).css("background-color", "blue");
   });
});
</script>
</head>
<body>
   <div class="box"></div>
   <div class="box"></div>
   <div class="box"></div>
   <div class="box"></div>
                  <p><strong>Note:</strong> Click inside the empty box to fill the background color.</p>
</body>
</html>
```

Set Multiple CSS Properties and Values

You can also set multiple CSS properties with the css() method. The basic syntax for setting the more than one property for the elements can be given with:

```
$(selector).css({"propertyName":"value",
"propertyName":"value", ...});
```

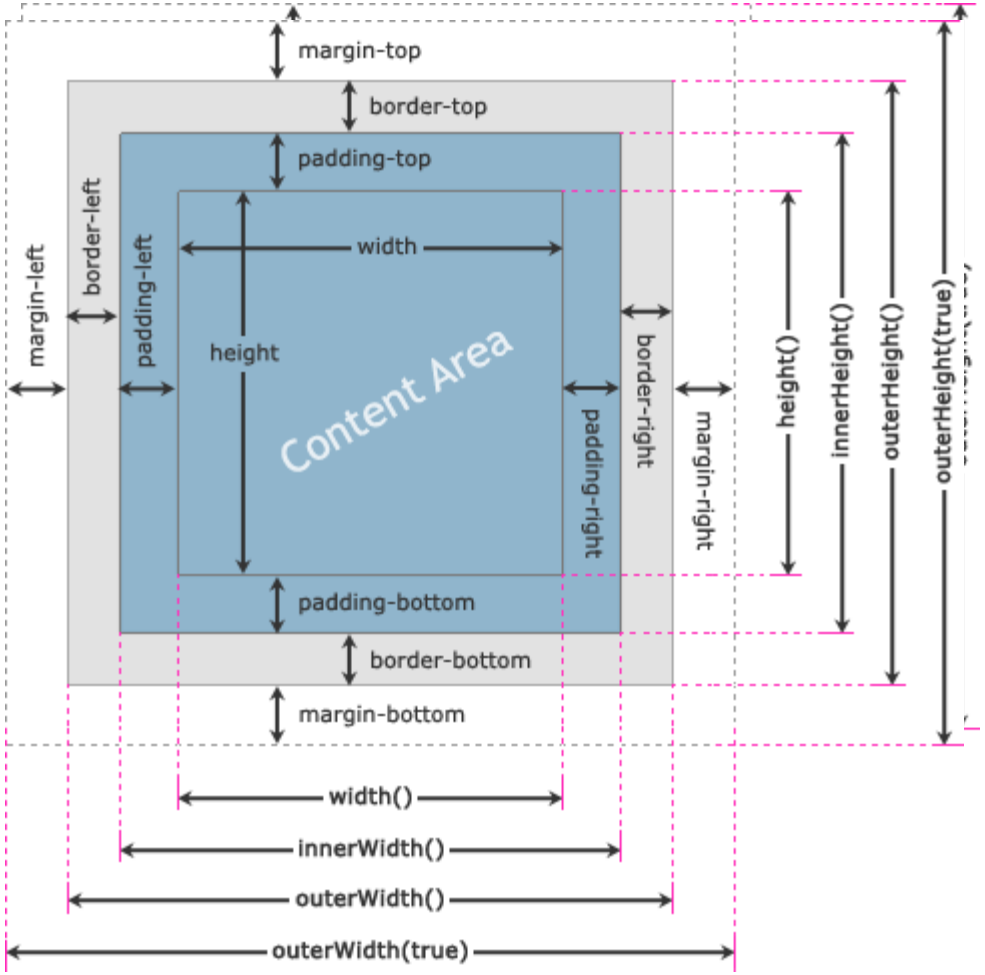
The following example will set the background-color as well as the padding CSS property for the selected elements at the same time.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set the Values of Multiple CSS Properties</title>
<style>
  p{
    font-size: 18px;
    font-family: Arial, sans-serif;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p").css({"background-color": "yellow", "padding": "20px"});
  });
});
</script>
</head>
<body>
  <h1>This is a heading</h1>
  <p style="background-color:orange;">This a paragraph.</p>
  <p style="background-color:#ee82ee;">This is another paragraph.</p>
  <p style="background-color:rgb(139,205,50);">This is none more paragraph.</p>
  <p>This is one last paragraph.</p>
  <button type="button">Add CSS Styles</button>
</body>
</html>
```

6. jQuery Dimensions

Understanding the jQuery Dimensions

jQuery provides several methods, such as height(), innerHeight(), outerHeight(), width(), innerWidth() and outerWidth() to get and set the CSS dimensions for the elements. Check out the following illustration to understand how these methods are calculating the dimensions of an element's box.



a. jQuery width() and height() Methods

The jQuery width() and height() methods get or set the width and the height of the element respectively. This width and height doesn't include padding, border and margin on the element. The following example will return the width and height of a element.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Get Width and Height of an Element</title>
<style>
  #box{
    width: 300px;
    height: 200px;
    padding: 25px;
    text-align: justify;
    border: 10px solid #c6b51a;
    background: #f0e68c;
    margin: 15px;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    var divWidth = $("#box").width();
    var divHeight = $("#box").height();
    $("#result").html("Width: " + divWidth + ", " + "Height: " + divHeight);
  });
});
</script>
</head>
<body>
  <div id="box">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante. Vestibulum id metus ac nisl bibendum scelerisque non non purus. Suspendisse varius nibh non aliquet sagittis. In tincidunt orci sit amet elementum vestibulum. Vivamus fermentum in arcu in aliquam. Quisque aliquam porta odio in fringilla non purus nisld Dapibus nec turpis vel, semper malesuada ant.</div>
  <button type="button">Get Width and Height</button>
  <p id="result"></p>
</body>
</html>
```

Similarly, you can **set the width and height** of the element by including the value as a parameter within the width() and height() method. The value can be either a string (number and unit e.g. 100px, 20em, etc.) or a number. The following example will set the width of a element to 400 pixels and height to 300 pixels respectively.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set Width and Height of the Elements</title>
<style>
  #box{
    width: 300px;
    height: 200px;
    padding: 25px;
    text-align: justify;
    border: 10px solid #c6b51a;
    background: #f0e68c;
    margin: 15px;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#box").width(400).height(300);
  });
});
</script>
</head>
<body>
  <div id="box">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante. Vestibulum id metus ac nisl bibendum scelerisque non non purus. Suspendisse varius nibh non aliquet sagittis. In tincidunt orci sit amet elementum vestibulum. Vivamus fermentum in arcu in aliquam. Quisque aliquam porta odio in fringilla non purus nisld Dapibus nec turpis vel, semper malesuada ant.</div>
  <button type="button">Set Width and Height</button>
</body>
</html>
```

b. jQuery innerWidth() and innerHeight() Methods

The jQuery innerWidth() and innerHeight() methods **get or set** the inner width and the inner height of the element respectively. This inner width and height includes the padding but excludes border and margin on the element. The following example will return the inner width and height of a element on the click of a button.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Get Inner-Width and Inner-Height of an Element</title>
<style>
  #box{
    width: 300px;
    height: 200px;
    padding: 25px;
    text-align: justify;
    border: 10px solid #c6b51a;
    background: #f0e68c;
    margin: 15px;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    var divWidth = $("#box").innerWidth();
    var divHeight = $("#box").innerHeight();
    $("#result").html("Inner Width: " + divWidth + ", " + "Inner Height: " + divHeight);
  });
});
</script>
</head>
<body>
  <div id="box">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante. Vestibulum id metus ac nisl bibendum scelerisque non non purus. Suspendisse varius nibh non aliquet sagittis. In tincidunt orci sit amet elementum vestibulum. Vivamus fermentum in arcu in aliquam. Quisque aliquam porta odio in fringilla non purus nisld Dapibus nec turpis vel, semper malesuada ant.</div>
  <button type="button">Get innerWidth and innerHeight</button>
  <p id="result"></p>
                <hr>
                <p><strong>Note:</strong> jQuery <b>innerWidth()</b> includes the CSS properties (<b>width</b> + <b>padding-left</b> + <b>padding-right</b>), whereas the <b>innerHeight()</b> includes (<b>height</b> + <b>padding-top</b> + <b>padding-bottom</b>).</p>
</body>
```

Similarly, you can **set the element's inner width and height** by passing the value as a parameter to the innerWidth() and innerHeight() method. These methods only alter the width or height of the element's content area to match the specified value.

For example, if the current width of the element is 300 pixels and the sum of the left and right padding is equal to 50 pixels than the new width of the element after setting the inner width to 400 pixels is 350 pixels i.e. New Width = Inner Width - Horizontal Padding. Similarly, you can estimate the change in height while setting the inner height.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set Inner-Width and Inner-Height of the Elements</title>
<style>
  #box{
    width: 300px;
    height: 200px;
    padding: 25px;
    text-align: justify;
    border: 10px solid #c6b51a;
    background: #f0e68c;
    margin: 15px;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#box").innerWidth(400).innerHeight(300);
  });
});
</script>
</head>
<body>
  <div id="box">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at,
luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante. Vestibulum id metus ac nisl
bibendum scelerisque non non purus. Suspendisse varius nibh non aliquet sagittis. In tincidunt orci sit amet
elementum vestibulum. Vivamus fermentum in arcu in aliquam. Quisque aliquam porta odio in fringilla non
purus nisld Dapibus nec turpis vel, semper malesuada ant.</div>
  <button type="button">Set innerWidth and innerHeight</button>
</body>
</html>
```

c. jQuery outerWidth() and outerHeight() Methods

The jQuery outerWidth() and outerHeight() methods get or set the outer width and the outer height of the element respectively. This outer width and height includes padding and border but excludes the margin on the element. The following example will return the outer width and height of a element on the click of a button.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Get Outer-Width and Outer-Height of an Element</title>
<style>
  #box{
    width: 300px;
    height: 200px;
    padding: 25px;
    text-align: justify;
    border: 10px solid #c6b51a;
    background: #f0e68c;
    margin: 15px;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    var divWidth = $("#box").outerWidth();
    var divHeight = $("#box").outerHeight();
    $("#result").html("Outer Width: " + divWidth + ", " + "Outer Height: " + divHeight);
  });
});
</script>
</head>
<body>
  <div id="box">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante. Vestibulum id metus ac nisl bibendum scelerisque non non purus. Suspendisse varius nibh non aliquet sagittis. In tincidunt orci sit amet elementum vestibulum. Vivamus fermentum in arcu in aliquam. Quisque aliquam porta odio in fringilla non purus nisld Dapibus nec turpis vel, semper malesuada ant.</div>
  <button type="button">Get outerWidth and outerHeight</button>
  <p id="result"></p>
                <hr>
                <p><strong>Note:</strong> jQuery <b>outerWidth()</b> includes the CSS properties (<b>width</b> + <b>padding-left</b> + <b>padding-right</b> + <b>border-left</b> + <b>border-right</b>), whereas the <b>outerHeight()</b> includes (<b>height</b> + <b>padding-top</b> + <b>padding-bottom</b> + <b>border-top</b> + <b>border-bottom</b>).</p>
</body>
</html>
```

You can also *get the outer width and height that includes padding and border* as well as the margin of the element. For that just specify the true parameter for the outer width methods, like outerWidth(true) and outerHeight(true).

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Get Outer-Width and Outer-Height of an Element including Margin</title>
<style>
  #box{
    width: 300px;
    height: 200px;
    padding: 25px;
    text-align: justify;
    border: 10px solid #c6b51a;
    background: #f0e68c;
    margin: 15px;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    var divWidth = $("#box").outerWidth(true);
    var divHeight = $("#box").outerHeight(true);
    $("#result").html("Outer Width: " + divWidth + ", " + "Outer Height: " + divHeight);
  });
});
</script>
</head>
<body>
  <div id="box">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante. Vestibulum id metus ac nisl bibendum scelerisque non non purus. Suspendisse varius nibh non aliquet sagittis. In tincidunt orci sit amet elementum vestibulum. Vivamus fermentum in arcu in aliquam. Quisque aliquam porta odio in fringilla non purus nisld Dapibus nec turpis vel, semper malesuada ant.</div>
  <button type="button">Get outerWidth and outerHeight with margin</button>
  <p id="result"></p>
                <hr>
                <p><strong>Note:</strong> jQuery <b>outerWidth(true)</b> includes the CSS properties (<b>width</b> + <b>padding-left</b> + <b>padding-right</b> + <b>border-left</b> + <b>border-right</b> + <b>margin-left</b> + <b>margin-right</b>), whereas the <b>outerHeight(true)</b> includes (<b>height</b> + <b>padding-top</b> + <b>padding-bottom</b> + <b>border-top</b> + <b>border-bottom</b> + <b>margin-top</b> + <b>margin-bottom</b>).</p>
</body>
```

Similarly, you can set the element's outer width and height by passing the value as a parameter to the outerWidth() and outerHeight() methods. These methods only *alter the width or height* of the element's content area to match the specified value, like the innerWidth() and innerHeight() methods.

For example, if the current width of the element is 300 pixels, and the sum of the left and right padding is equal to 50 pixels, and the sum of the width of the left and right border is 20 pixels than the new width of the element after setting the outer width to 400 pixels is 330 pixels i.e. New Width = Outer Width - (Horizontal Padding + Horizontal Border). Similarly, you can estimate the change in height while setting the outer height.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery Set Outer-Width and Outer-Height of the Elements</title>
<style>
  #box{
    width: 300px;
    height: 200px;
    padding: 25px;
    text-align: justify;
    border: 10px solid #c6b51a;
    background: #f0e68c;
    margin: 15px;
  }
</style>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#box").outerWidth(400).outerHeight(300);
  });
});
</script>
</head>
<body>
  <div id="box">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam eu sem tempor, varius quam at, luctus dui. Mauris magna metus, dapibus nec turpis vel, semper malesuada ante. Vestibulum id metus ac nisl bibendum scelerisque non non purus. Suspendisse varius nibh non aliquet sagittis. In tincidunt orci sit amet elementum vestibulum. Vivamus fermentum in arcu in aliquam. Quisque aliquam porta odio in fringilla non purus nisld Dapibus nec turpis vel, semper malesuada ant.</div>
  <button type="button">Set outerWidth and outerHeight</button>
</body>
</html>
```

jQuery Ajax

Ajax

Ajax stands for Asynchronous Javascript And Xml. Ajax is just a means of loading data from the server to the web browser without reloading the whole page.

Basically, what Ajax does is make use of the JavaScript-based XMLHttpRequest object to send and receive information to and from a web server asynchronously, in the background, without interfering with the user's experience.

Ajax has become so popular that you hardly find an application that doesn't use Ajax to some extent. The example of some large-scale Ajax-driven online applications are: Gmail, Google Maps, Google Docs, YouTube, Facebook, Flickr, etc.

Note: Ajax is not a new technology, in fact, Ajax is not even really a technology at all. Ajax is just a term to describe the process of exchanging data from a web server asynchronously through JavaScript, without a page refresh.

Ajax with jQuery

Different browsers implement the Ajax differently that means if you're adopting the typical JavaScript way to implement the Ajax you have to write the different code for different browsers to ensure that Ajax would work cross-browser.

But, fortunately jQuery simplifies the process of implementing Ajax by taking care of those browser differences. It offers simple methods such as load(), $.get(), $.post(), etc. to implement the Ajax that works seamlessly across all the browsers.

In the upcoming chapters you will learn how to load data from the server as well as how to send and receive data using HTTP GET and POST method through jQuery Ajax.

Tip: Ajax requests are triggered by the JavaScript code; your code sends a request to a URL, and when the request completes, a callback function can be triggered to handle the response. Further, since the request is asynchronous, the rest of your code continues to execute while the request is being processed.

jQuery Traversing

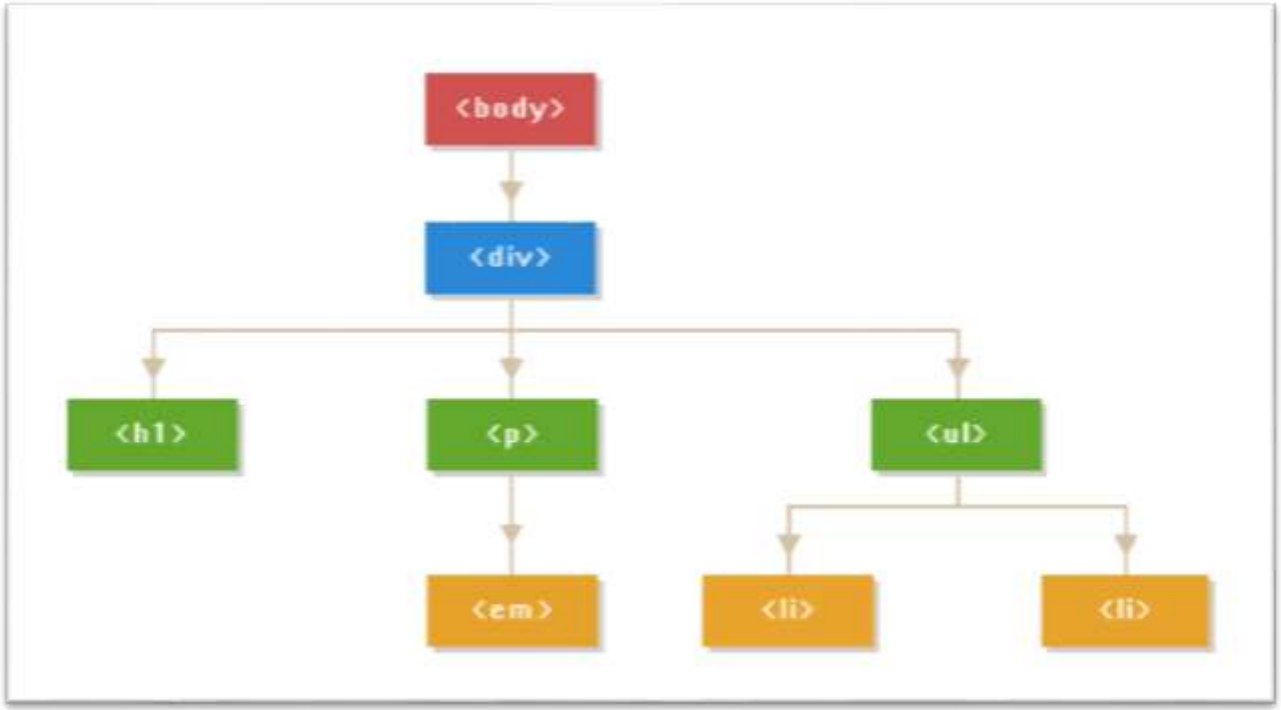
Traversing

The jQuery selectors we've seen so far only allow us to select the elements down the DOM tree. But there are many occasions when you need to select a parent or ancestor element; that is where jQuery's DOM traversal methods come into play. With these traversal methods, we can go up, down, and all around the DOM tree very easily.

DOM traversing is one of the prominent features of the jQuery. To make the most it you need to understand the relationships between the elements in a DOM tree.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>HTML DOM Tree Sample</title>
<script
src="https://code.jquery.com/jquery-
3.5.1.min.js"></script>
</head>
<body>
   <div class="container">
      <h1>Hello World</h1>
      <p>This is a <em>simple
paragraph</em>.</p>
      <ul>
         <li>Item One</li>
         <li>Item Two</li>
      </ul>
   </div>
</body>
</html>
```

The HTML code in this can be represented by the following DOM tree:



The above diagram showing the *parent/child* relationships between the elements:

- The <body> element is the parent of the element, and an ancestor of everything inside of it. The enclosed element is the parent of <h1>, <p> and <ul> elements, and a child of the <body> element.
- The elements <h1>, <p> and <ul> are siblings, since they share the same parent.
- The <h1> element is a child of the element and a descendant of the <body> element. This element does not have any children.
- The <p> element is the parent of element, child of the element and a descendant of the <body> element. The containing element is a child of this <p> element and a descendant of the and <body> element.
- Similarly, the <ul> element is the parent of the <li> elements, child of the element and a descendant of the <body> element. The containing <li> elements are the child of this <ul> element and a descendant of the and <body> element. Also, both the <li> elements are siblings.

Note: In logical relationships, an ancestor is a parent, grandparent, great-grandparent, and so on. A descendant is a child, grandchild, great-grandchild, and so on. Sibling elements are those which share the same parent.