Form elements

The `<select>` element is used to create a drop-down list.

The `<select>` element is most often used in a form, to collect user input.

The `name` attribute is needed to reference the form data after the form is submitted (if you omit the `name` attribute, no data from the drop-down list will be submitted).

The `id` attribute is needed to associate the drop-down list with a label.

The <option> tags inside the `<select>` element define the available options in the drop-down list.

```
<label for="cars">Choose a car:</label>

<select name="cars" id="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="mercedes">Mercedes</option>
  <option value="audi">Audi</option>
</select>
```

The `<option>` elements defines an option that can be selected.

By default, the first item in the drop-down list is selected.

Use the `size` attribute to specify the number of visible values:

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars" size="3">
```

The <option> tag is used to define the possible options to choose from. The tag is put into the <select> tag.

The first option from the list of options is selected by default. To change a predefined option, the selected attribute is used.

The <optgroup> tag is used to group several options into one group.The content of <optgroup> looks like heading in bold.

This tag is used to create a group of the same category options in a drop-down list.

The `<optgroup>` tag is used to group related options in a <select> element (drop-down list).

If you have a long list of options, groups of related options are easier to handle for a user.

```
<select>
        <!-- optgroup tag starts -->
            <optgroup label="Programming Languages">
                <option value="C">C</option>
                <option value="C++">C++</option>
                <option value="Java">Java</option>
            </optgroup>
            <optgroup label="Scripting Language">
       <option value="JavaScript">JavaScript</option>
                <option value="PHP">PHP</option>
                <option value="Shell">Shell</option>
            </optgroup>
        <!-- optgroup tag ends     -->
        </select>
```

The <textarea> Element

The `<textarea>` element defines a multi-line input field (a text area):

```
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

The `rows` attribute specifies the visible number of lines in a text area.

The `cols` attribute specifies the visible width of a text area.

HTML <label> Tag

The `<label>` tag defines a label for several elements:

The **<label>** tag is used to specify a label for an <input> element of a form. It adds a label to a form control such as text, email, password, textarea etc. It toggles the control when a user clicks on a text within the <label> element.

Set the id attribute **inside the** <input> **element and specify its name for the** for attribute **inside the <label> tag.**

The `<label>` element defines a label for several form elements.

The `<label>` element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focus on the input element.

The `<label>` element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the `<label>` element, it toggles the radio button/checkbox.

The `for` attribute of the `<label>` tag should be equal to the `id` attribute of the `<input>` element to bind them together.

```
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname">
```

The <fieldset> and <legend> Elements

The <fieldset> element is used to group related data in a form.

The <fieldset> tag draws a box around the related elements.

The <legend> element defines a caption for the <fieldset> element.

```
<form action="/action_page.php">
  <fieldset>
    <legend>Personalia:</legend>
    <label for="fname">First name:</label><br>
    <input type="text" id="fname" name="fname" value="John"><br>
    <label for="lname">Last name:</label><br>
    <input type="text" id="lname" name="lname" value="Doe"><br><br>
    <input type="submit" value="Submit">
  </fieldset>
</form>
```

HTML <datalist> Tag

The <datalist> tag specifies a list of pre-defined options for an <input> element.

The <datalist> tag is used to provide an "autocomplete" feature for <input> elements. Users will see a drop-down list of pre-defined options as they input data.

The <datalist> element's id attribute must be equal to the <input> element's list attribute (this binds them together).

```
<label for="browser">Choose your browser from the list:</label>
<input list="browsers" name="browser" id="browser">
<datalist id="browsers">
  <option value="Edge">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

```
<body>
      <input list = "tutorials" />

      <datalist id = "tutorials">
         <option value = "Java">
         <option value = "ASP">
         <option value = "PHP">
         <option value = "Ruby">
         <option value = "jQuery">
      </datalist>
   </body>
```

HTML action Attribute

The `action` attribute specifies where to send the form-data when a form is submitted.

On submit, send the form-data to a file named "/action_page.php" (to process the input):

```
<form action="/action_page.php">

  <label for="fname">First name:</label>

  <input type="text" id="fname" name="fname"><br><br>

  <label for="lname">Last name:</label>

  <input type="text" id="lname" name="lname"><br><br>

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

</form>
```

HTML <button> Tag

The `<button>` tag defines a clickable button.

Inside a `<button>` element you can put text (and tags like `<i>`, `<b>`, `<strong>`, `<br>`, `<img>`, etc.). That is not possible with a button created with the `<input>` element!

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

HTML input Tag

In HTML, the **input** field can be specified using *where a user can enter data*. The input tag is used within **< form>** element to declare input controls that allow users to input data. An input field can be of various types depending upon the attribute type. The Input tag is an empty element which only contains attributes. For defining labels for the input element, **< label>** can be used.

Syntax

```
<input type = "value" .... />
```

type=" "	Description
text	Defines a one-line text input field
password	Defines a one-line password input field
submit	Defines a submit button to submit the form to server
reset	Defines a reset button to reset all values in the form.
radio	Defines a radio button which allows select one option.
checkbox	Defines checkboxes which allow select multiple options form.
button	Defines a simple push button, which can be programmed to perform a task on an event.
file	Defines to select the file from device storage.
image	Defines a graphical submit button.

HTML5 added new types on <input> element. Following is the list of types of elements of HTML5

type=" "	Description
color	Defines an input field with a specific color.
date	Defines an input field for selection of date.
datetime-local	Defines an input field for entering a date without time zone.
email	Defines an input field for entering an email address.

month	Defines a control with month and year, without time zone.
number	Defines an input field to enter a number.
url	Defines a field for entering URL
week	Defines a field to enter the date with week-year, without time zone.
search	Defines a single line text field for entering a search string.
tel	Defines an input field for entering the telephone number.

1. <input type="text">:

`<input type="text">` defines a **single-line text input field**:

<input> element of type "text" are used to define a single-line input text field.

```
<form>
    <label>Enter first name</label><br>
    <input type="text" name="firstname"><br>
    <label>Enter last name</label><br>
    <input type="text" name="lastname"><br>
    <p><strong>Note:</strong>The default maximum cahracter lenght is 20.
</p>
</form>
```

<input type="password">:

`<input type="password">` defines a **password field**:

The <input> element of type "password" allow a user to enter the password securely in a webpage. The entered text in password filed converted into "*" or ".", so that it cannot be read by another user.

```
<form>
    <label>Enter User name</label><br>
    <input type="text" name="firstname"><br>
    <label>Enter Password</label><br>
    <input type="Password" name="password"><br>
    <br><input type="submit" value="submit">
</form>
```

<input type="submit">:

`<input type="submit">` defines a button for **submitting** form data to a **form-handler**.

The <input> element of type "submit" defines a submit button to submit the form to the server when the "click" event occurs.

```
<form action="https://www.javatpoint.com/html-tutorial">
    <label>Enter User name</label><br>
    <input type="text" name="firstname"><br>
    <label>Enter Password</label><br>
    <input type="Password" name="password"><br>
    <br><input type="submit" value="submit">
</form>
```

<input type="reset">:

`<input type="reset">` defines a **reset button** that will reset all form values to their default values:

The <input> type "reset" is also defined as a button but when the user performs a click event, it by default reset the all inputted values.

Example:

```
<form>
   <label>User id: </label>
    <input type="text" name="user-id" value="user">
          <label>Password: </label>
    <input type="password" name="pass" value="pass"><br><br>
    <input type="submit" value="login">
     <input type="reset" value="Reset">
</form>
```

<input type="radio">:

`<input type="radio">` defines a **radio button**.

Radio buttons let a user select ONLY ONE of a limited number of choices:

The <input> type "radio" defines the radio buttons, which allow choosing an option between a set of related options. At a time only one radio button option can be selected at a time.

```
<form>
  <p>Kindly Select your favorite color</p>
  <input type="radio" name="color" value="red"> Red <br>
  <input type="radio" name="color" value="blue"> blue <br>
  <input type="radio" name="color" value="green">green <br>
  <input type="radio" name="color" value="pink">pink <br>
  <input type="submit" value="submit">
</form>
```

<input type="checkbox">: `<input type="checkbox">` defines a **checkbox**.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

The <input> type "checkbox" are displayed as square boxes which can be checked or unchecked to select the choices from the given options.

```
<form>
    <label>Enter your Name:</label>
    <input type="text" name="name">
    <p>Kindly Select your favourite sports</p>
    <input type="checkbox" name="sport1" value="cricket">Cricket<br>
    <input type="checkbox" name="sport2" value="tennis">Tennis<br>
    <input type="checkbox" name="sport3" value="football">Football<br>
    <input type="checkbox" name="sport4" value="baseball">Baseball<br>

    <input type="checkbox" name="sport5" value="badminton">Badminton
<br><br>
    <input type="submit" value="submit">
</form>
```

<input type="file">: The `<input type="file">` defines a file-select field and a "Browse" button for file uploads.

The <input> element with type "file" is used to select one or more files from user device storage. Once you select the file, and after submission, this file can be uploaded to the server with the help of JS code and file API.

Example:

```
<form>
    <label>Select file to upload:</label>
    <input type="file" name="newfile">
    <input type="submit" value="submit">
</form>
```

<input type="image">:

The <input> type "image" is used to represent a submit button in the form of image.

```
<form>
    <label>User id:</label><br>
    <input type="text" name="name"><br><br>
    <input type="image" alt="Submit" src="login.png"  width="100px">
</form>
```

HTML5 newly added <input> types element

1. <input type="color">: The `<input type="color">` is used for input fields that should contain a color.

The <input> type "color" is used to define an input field which contains a colour. It allows a user to specify the colour by the visual colour interface on a browser.

```
<form>
    Pick your Favorite color: <br><br>
    <input type="color" name="upclick" value="#a52a2a"> Upclick<br><br>

    <input type="color" name="downclick" value="#f5f5dc"> Downclick
```

```
</form>
```

<input type="date">:

The `<input type="date">` is used for input fields that should contain a date.

The <input> element of type "date" generates an input field, which allows a user to input the date in a given format. A user can enter the date by text field or by date picker interface.

Example:

```
<form>
    Select Start and End Date: <br><br>
      <input type="date" name="Startdate"> Start date:<br><br>
      <input type="date" name="Enddate"> End date:<br><br>
     <input type="submit">
</form>
```

<input type="datetime-local">:

The <input> element of type "datetime-local" creates input filed which allow a user to select the date as well as local time in the hour and minute without time zone information.

Input Type Datetime-local

The `<input type="datetime-local">` specifies a date and time input field, with no time zone.

Depending on browser support, a date picker can show up in the input field.

Example

```
<form>
  <label for="birthdaytime">Birthday (date and time):</label>
  <input type="datetime-local" id="birthdaytime" name="birthdaytime">
</form>
```

Example:

```
<form>
    <label>
      Select the meeting schedule: <br><br>
      Select date & time: <input type="datetime-
local" name="meetingdate"> <br><br>
    </label>
     <input type="submit">
</form>
```

<input type="email">: The `<input type="email">` is used for input fields that should contain an e-mail address.

The <input> type "email" creates an input filed which allow a user to enter the e-mail address with pattern validation. The multiple attributes allow a user to enter more than one email address.

```
<form>
      <label><b>Enter your Email-address</b></label>
      <input type="email" name="email" required>
      <input type="submit">
```

<input type="month">:

The <input> type "month" creates an input field which allows a user to easily enter month and year in the format of "MM, YYYY" where MM defines month value, and YYYY defines the year value

```
<label>Enter your Birth Month-year: </label>
  <input type="month" name="newMonth">
  <input type="submit">
```

<input type="number">:

The <input> element type number creates input filed which allows a user to enter the numeric value. You can also restrict to enter a minimum and maximum value using min and max attribute.

Example:

```
<form>
    <label>Enter your age: </label>
    <input type="number" name="num" min="50" max="80">
    <input type="submit">
</form>
```

<input type="url">:

The <input> element of type "url" creates an input filed which enables user to enter the URL.

```
<label>Enter your website URL: </label>
    <input type="url" name="website" placeholder="http://example.com"><br>
        <input type="submit" value="send data">
```

<input type="search">:

The <input> type "search" creates an input filed which allows a user to enter a search string. These are functionally symmetrical to the text input type, but may be styled differently.

```
<label>Search here:</label>
    <input type="search" name="q">
    <input type="submit" value="search">
```

JavaScript is used to create client-side dynamic pages.

JavaScript is *an object-based scripting language* which is lightweight and cross-platform.

What is JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.

Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

JavaScript Example

```
<script>

document.write("Hello JavaScript by JavaScript");
</script>
```

The **document.write()** function is used to display dynamic content through JavaScript

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is `getElementById()`.

The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript":

Example

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

JavaScript Can Change HTML Attribute Values

In this example JavaScript changes the value of the `src` (source) attribute of an `<img>` tag:

JavaScript Can Change HTML Styles (CSS)

Changing the style of an HTML element, is a variant of changing an HTML attribute:

Example

```
document.getElementById("demo").style.fontSize = "35px";
```

JavaScript Can Hide HTML Elements

Hiding HTML elements can be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "none";
```

JavaScript Can Show HTML Elements

Showing hidden HTML elements can also be done by changing the `display` style:

Example

```
document.getElementById("demo").style.display = "block";
```

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)

---

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.

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

```
<html>
<head>
<script type="text/javascript">
function msg(){
 alert("Hello Javatpoint");
}
</script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</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.

```
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

```
<html>  <head>
<script type="text/javascript" src="message.js"></script>  </head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>  </body> </html>
```

Advantages of External JavaScript

There will be following benefits if a user creates an external javascript:

1. It helps in the reusability of code in more than one HTML file.
2. It allows easy code readability.
3. It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
4. It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
5. The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

There are the following disadvantages of external files:

1. The stealer may download the coder's code using the url of the js file.
2. If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
3. The web browser needs to make an additional http request to get the js code.
4. A tiny to a large change in the js code may cause unexpected results in all its dependent files.
5. We need to check each file that depends on the commonly created external javascript file.
6. If it is a few lines of code, then better to implement the internal javascript code.

HTML <script> Tag

The <script> tag is used to embed a client-side script (JavaScript).

The <script> element either contains scripting statements, or it points to an external script file through the src attribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

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

JavaScript Variables

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

```
<script>
var x = 10;
var y = 20;
var z=x+y;
document.write(z);
</script>
```

There are 3 ways to declare a JavaScript variable:

- Using var
- Using let
- Using const

Variables

Variables are containers for storing data (values).

- Using var

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;
```

The `let` keyword was introduced in ES6 (2015).

Variables defined with `let` cannot be Redeclared.

Variables defined with `let` must be Declared before use.

Variables defined with `let` have Block Scope.

Cannot be Redeclared

Variables defined with `let` cannot be **redeclared**.

You cannot accidentally redeclare a variable.

With `let` you can not do this:

Example

```
let x = "John Doe";

let x = 0;

// SyntaxError: 'x' has already been declared
```

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
```

Redeclaring Variables

Redeclaring a variable using the `var` keyword can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

Example

```
var x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
```

Redeclaring a variable using the `let` keyword can solve this problem.

Redeclaring a variable inside a block will not redeclare the variable outside the block:

Example

```
let x = 10;
// Here x is 10

{
let x = 2;
// Here x is 2
}

// Here x is 10
```

JavaScript Const

The `const` keyword was introduced in ES6 (2015).

Variables defined with `const` cannot be Redeclared.

Variables defined with `const` cannot be Reassigned.

Variables defined with `const` have Block Scope.

Cannot be Reassigned

A `const` variable cannot be reassigned:

Example

```
const PI = 3.141592653589793;
PI = 3.14;      // This will give an error
PI = PI + 10;   // This will also give an error
```

Must be Assigned

JavaScript `const` variables must be assigned a value when they are declared:

Correct

```
const PI = 3.14159265359;
```

Incorrect

```
const PI;
PI = 3.14159265359;
```

When to use JavaScript const?

As a general rule, always declare a variable with `const` unless you know that the value will change.

Use `const` when you declare:

- A new Array
- A new Object
- A new Function
- A new RegExp

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>
function abc(){
var x=10;//local variable
}
</script>
```

Or,

```
<script>
If(10<13){
var y=20;//JavaScript local variable
}
</script>
```

JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

```
<script>
```

```
var data=200;//gloabal variable
function a(){
document.writeln(data);
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
</script>
```

JavaScript Global Variable

A **JavaScript global variable** is declared outside the function or declared with window object. It can be accessed from any function.

Let's see the simple example of global variable in JavaScript.

```
<script>
var value=50;//global variable
function a(){
alert(value);
}
function b(){
alert(value);
}
</script>
```

Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object**. For example:

```
window.value=90;
```

Now it can be declared inside any function and can be accessed from any function. For example:

```
function m(){
window.value=100;//declaring global variable by window object
}
function n(){
alert(window.value);//accessing global variable from other function
}
```

*i*nternals of global variable in JavaScript

When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also. For example:

```
var value=50;
function a(){  alert(window.value);//accessing global variable   }
```

JavaScript Operators

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

1. var sum=10+20;

Here, + is the arithmetic operator and = is the assignment operator.

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

JavaScript Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
|	Bitwise OR	(10==20 | 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

JavaScript Special Operators

The following operators are known as JavaScript special operators.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

JavaScript If-else

The **JavaScript if-else statement** is used *to execute the code whether condition is true or false*. There are three forms of if statement in JavaScript.

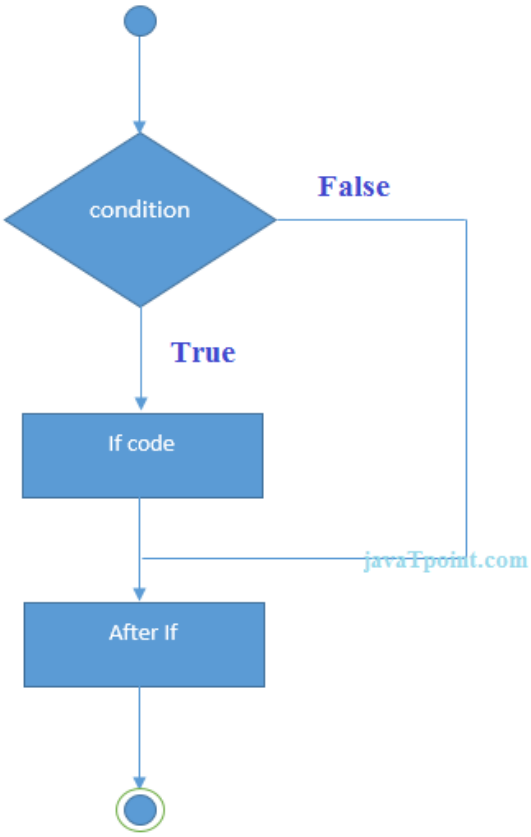
1. If Statement
2. If else statement
3. if else if statement

JavaScript If statement

It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

```
1. if(expression){
2. //content to be evaluated
3. }
```

Flowchart of JavaScript If statement



Let's see the simple example of if statement in javascript.

```
1. <script>
2. var a=20;
3. if(a>10){
4. document.write("value of a is greater than 10");
5. }
6. </script>
```

Output of the above example

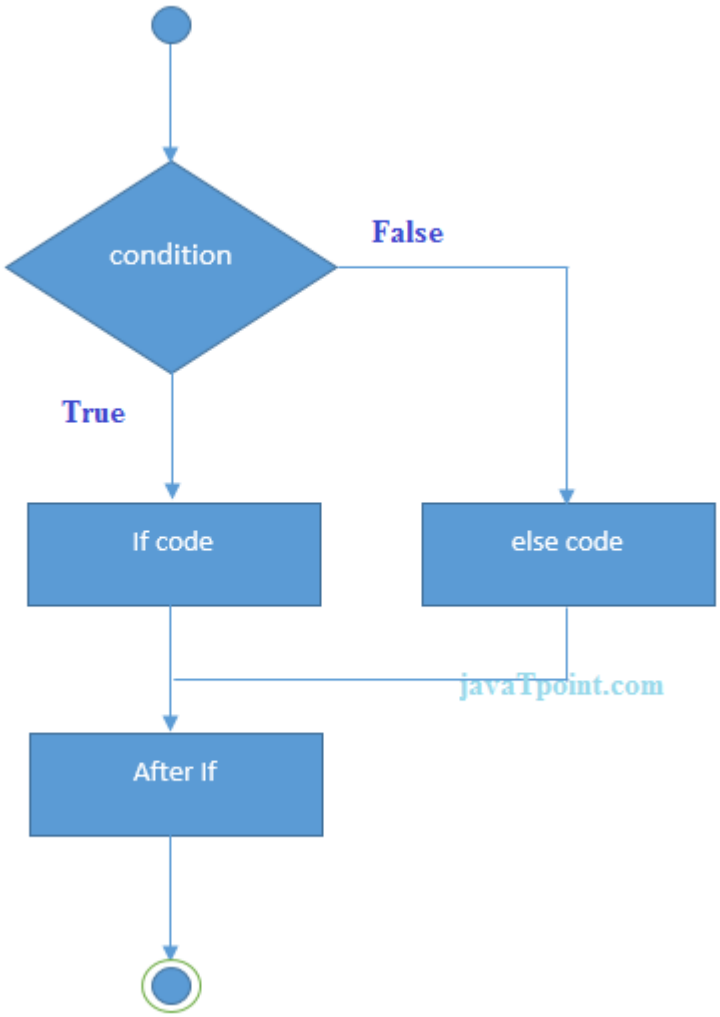
```
value of a is greater than 10
```

JavaScript If...else Statement

It evaluates the content whether condition is true of false. The syntax of JavaScript if-else statement is given below.

```
if(expression){
//content to be evaluated if condition is true
}
else{
//content to be evaluated if condition is false
}
```

Flowchart of JavaScript If...else statement



Let's see the example of if-else statement in JavaScript to find out the even or odd number.

```
<script>
var a=20;
if(a%2==0){
document.write("a is even number");
}
else{
document.write("a is odd number");
}
</script>
```

utput of the above example

```
a is even number
```

JavaScript If...else if statement

It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

```
1. if(expression1){
2. //content to be evaluated if expression1 is true
3. }
4. else if(expression2){
5. //content to be evaluated if expression2 is true
6. }
7. else if(expression3){
8. //content to be evaluated if expression3 is true
9. }
10. else{
11. //content to be evaluated if no expression is true
12. }
```

Let's see the simple example of if else if statement in javascript.

```
<script>
var a=20;
if(a==10){
document.write("a is equal to 10");
}
else if(a==15){
document.write("a is equal to 15");
}
else if(a==20){
document.write("a is equal to 20");
}
else{
document.write("a is not equal to 10, 15 or 20");
}
</script>
```

Output of the above example

```
a is equal to 20
```

JavaScript Switch

The **JavaScript switch statement** is used *to execute one code from multiple expressions*. It is just like else if statement that we have learned in previous page. But it is convenient than *if..else..if* because it can be used with numbers, characters etc.

The signature of JavaScript switch statement is given below.

```
switch(expression){
case value1:
 code to be executed;
 break;
case value2:
 code to be executed;
 break;
.....

default:
 code to be executed if above values are not matched;
}
```

Let's see the simple example of switch statement in javascript.

```
<script>
var grade='B';
var result;
switch(grade){
case 'A':
result="A Grade";
break;
case 'B':
result="B Grade";
break;
case 'C':
result="C Grade";
break;
default:
result="No Grade";
}
document.write(result);
</script>
```

The switch statement is fall-through i.e. all the cases will be evaluated if you don't use break statement.

JavaScript Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

---

1) JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

1. for (initialization; condition; increment)
2. {
3. code to be executed
4. }
5. Let's see the simple example of for loop in javascript.

```
<script>
for (i=1; i<=5; i++)
{
document.write(i + "<br/>")
}
</script>
```

2) JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

```
while (condition)
{
   code to be executed
```

Let's see the simple example of while loop in javascript.

```
<script>
var i=11;
while (i<=15)
{
document.write(i + "<br/>");
```

i++; } </script>

3) JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

```
1. do{
2.     code to be executed
3. }while (condition);
```

Let's see the simple example of do while loop in javascript.

```
<script>
var i=21;
do{
document.write(i + "<br/>");
i++;
}while (i<=25);
</script>
```

For In Loop

The JavaScript `for in` statement loops through the properties of an Object:

Syntax

```
for (key in object) {
  // code block to be executed
}
```

Example

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";
for (let x in person) {
  text += person[x];
}
```

Example Explained

- The **for in** loop iterates over a **person** object
- Each iteration returns a **key** (x)
- The key is used to access the **value** of the key
- The value of the key is **person[x]**

JavaScript Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don't need to write many lines of code each time to perform a common task.

JavaScript Function Syntax

The syntax of declaring function is given below.

```
1. function functionName([arg1, arg2, ...argN]){
2.  //code to be executed
3. }
```

JavaScript Functions can have 0 or more arguments.

JavaScript Function Example

Let's see the simple example of function in JavaScript that does not has arguments.

```
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="call function"/>
```

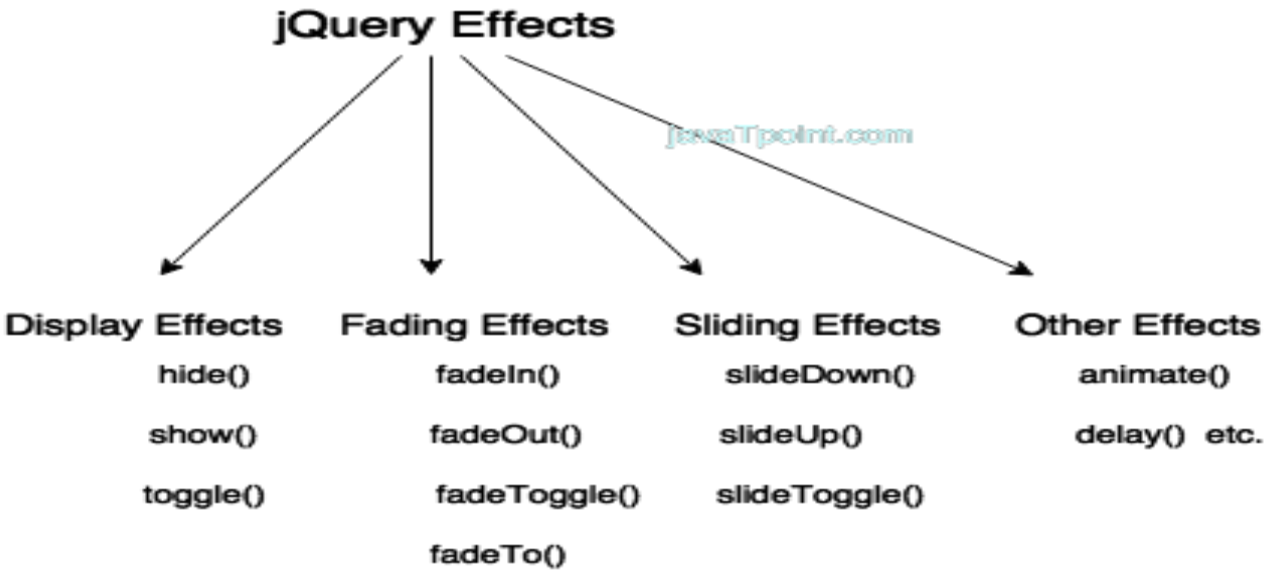
JavaScript Function Arguments

We can call function by passing arguments. Let's see the example of function that has one argument.

```
<script>
function getcube(number){
alert(number*number*number);
}
</script>
<form>
<input type="button" value="click" onclick="getcube(4)"/>
</form>
```

jQuery Effects

jQuery enables us to add effects on a web page. jQuery effects can be categorized into fading, sliding, hiding/showing and animation effects.



jQuery hide() Method

The hide() method hides the selected elements.

Syntax:

1. **$(selector).hide();**
2. $(selector).hide(speed, callback);
3. $(selector).hide(speed, callback);

speed: It is an optional parameter. It specifies the speed of the delay. Its possible vales are slow, fast and milliseconds.

easing: It specifies the easing function to be used for transition.

callback: It is also an optional parameter. It specifies the function to be called after completion of hide() effect.

```
<!DOCTYPE html>  <html> <head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script>

$(document).ready(function(){

 $(".btn1").click(function()
```

```
{     $("p").hide();    });

  $(".btn2").click(function(){

    $("p").show();     });  });

</script></head><body>

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

<button class="btn1">Hide</button>

<button class="btn2">Show</button>

</body>

</html>
```

jQuery Effect show() Method

The show() method shows the hidden, selected elements.

Syntax:
$(selector).show();

jQuery fadeIn() Method

The jQuery `fadeIn()` method is used to fade in a hidden element.

Syntax:`$(selector).fadeIn(speed);`

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

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

```
$("button").click(function(){
  $("#div1").fadeIn();
  $("#div2").fadeIn("slow");
  $("#div3").fadeIn(3000);
});
```

jQuery fadeOut() Method

The jQuery `fadeOut()` method is used to fade out a visible element.

Syntax:

$(*selector*).fadeOut(*speed*);

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

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

Example

```
$("button").click(function(){
  $("#div1").fadeOut();
  $("#div2").fadeOut("slow");
  $("#div3").fadeOut(3000);
});
```

<!DOCTYPE html>

<html><head> <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<script>

$(document).ready(function(){

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

$("#div1").fadeOut();

$("#div2").fadeOut("slow");

$("#div3").fadeOut(3000);

});

});</script></head><body>

Demonstrate fadeOut() with different parameters.

<button>Click to fade out boxes</button>

</body></html>

jQuery fadeToggle() Method

The jQuery `fadeToggle()` method toggles between the `fadeIn()` and `fadeOut()` methods.

If the elements are faded out, `fadeToggle()` will fade them in.If the elements are faded in, `fadeToggle()` will fade them out.

Syntax: `$(selector).fadeToggle(speed,);`

```
$("button").click(function(){
  $("#div1").fadeToggle();
  $("#div2").fadeToggle("slow");
  $("#div3").fadeToggle(3000);
});
```

jQuery fadeTo() Method

The jQuery `fadeTo()` method allows fading to a given opacity (value between 0 and 1).

Syntax:`$(selector).fadeTo( speed,opacity,callback);`

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

The required opacity parameter in the `fadeTo()` method specifies fading to a given opacity (value between 0 and 1).

```
$("button").click(function(){
  $("#div1").fadeTo("slow", 0.15);
  $("#div2").fadeTo("slow", 0.4);
  $("#div3").fadeTo("slow", 0.7);
});
```

jQuery Events

jQuery events are the actions that can be detected by your web application. They are used to create dynamic web pages. An event shows the exact moment when something happens.

All the different visitors' actions that a web page can respond to are called events.

An event represents the precise moment when something happens.

These are some examples of events.

- A mouse click , An HTML form submission ,A web page loading , A keystroke on the keyboard
- Scrolling of the web page etc.

These events can be categorized on the basis their types:

Mouse Events **Click ,dblclick ,mouseenter , mouseleave**

Keyboard Events **Keyup ,keydown ,keypress**

Form Events **Submit ,change,blur,focus**

Document/Window Events **Load ,unload ,scroll ,resize**

click()

When you click on an element, the click event occurs and once the click event occurs it execute the click () method or attaches a function to run.

It is generally used together with other events of jQuery.

Syntax: **$(selector).click()**

It is used to trigger the click event for the selected elements.
$(selector).click(function)
It is used to attach the function to the click event.

```
$("p").click(function(){
  $(this).hide();
});
```

focus()

The `focus()` method attaches an event handler function to an HTML form field.
The function is executed when the form field gets focus:

```
$("input").focus(function(){
  $(this).css("background-color", "#cccccc");
});
```

jQuery focus()

The jQuery focus event occurs when an element gains focus. It is generated by a mouse click or by navigating to it.
This event is implicitly used to limited sets of elements such as form elements like <input>, <select> etc. and links <a href>. The focused elements are usually highlighted in some way by the browsers.
The focus method is often used together with blur () method.

$(selector).focus()

It triggers the focus event for selected elements.

$(selector).focus(function)

It adds a function to the focus event.

```
<!DOCTYPE html>      <html><head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
 $("input").focus(function(){
  $(this).css("background-color", "yellow");
 });
 $("input").blur(function(){
  $(this).css("background-color", "green");  });});
```

</script></head><body>
Name: <input type="text" name="fullname">
Email: <input type="text" name="email">
</body></html>

jQuery blur()

The jQuery blur event occurs when element loses focus. It can be generated by via keyboard commands like tab key or mouse click anywhere on the page.

It makes you enable to attach a function to the event that will be executed when the element loses focus. Originally, this event was used only with form elements like <input>. In latest browsers, it has been extended to include all element types.

The blur () method is often used together with focus () method.

1. **$(selector).blur()**

It triggers the blur event for selected elements.

1. **$(selector).blur(function)**

It adds a function to the blur event.

```
$("input").blur(function(){
  $(this).css("background-color", "#ffffff");
});
```

jQuery load()

The load () method is used to load a specific element. It attaches an event handler to load event. It was deprecated in jQuery 1.8 version of jQuery library.

The load event occurs when a specific element is loaded. It is generally used with a URL (image, script, frame, iframe), and the window object.

$(selector).load(function)

It adds a function to the load event.

```
<!DOCTYPE html>  <html>  <head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("img").load(function(){
```

```
        alert("Image loaded.");      });  });
    </script>  </head>  <body>
    <img src="good-morning.jpg" alt="good morning">
    <p><b>Note:</b> On some browsers, the load event did not trigger if the image is cached.</p>  </body>  </html>
```

The on() Method

Attach a click event to a <p> element:

```
<script>
$(document).ready(function(){
  $("p").on("click", function(){
    $(this).hide();
  });
});
</script>
```

jQuery ready() Method

The ready event occurs when the DOM (document object model) has been loaded.Because this event occurs after the document is ready, it is a good place to have all other jQuery events and functions. Like in the example above.

The ready() method specifies what happens when a ready event occurs.

Use ready() to make a function available after the document is loaded:

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

jQuery off() Method

The off() method is most often used to remove event handlers attached with the on() method.

Remove the click event for all <p> elements:

```
$("button").click(function(){
  $("p").off("click");
});
```

jQuery

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

jQuery is a fast, small, cross-platform and feature-rich JavaScript library. It is designed to simplify the client-side scripting of HTML. It makes things like HTML document traversal and manipulation, animation, event handling, and AJAX very simple with an easy-to-use API that works on a lot of different type of browsers.

The main purpose of jQuery is to provide an easy way to use JavaScript on your website to make it more interactive and attractive. It is also used to add animation.

- Query is a small, fast and lightweight JavaScript library.
- jQuery is platform-independent.
- jQuery means "write less do more".

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

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

The jQuery library contains the following features:

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

jQuery Features

- HTML manipulation
- DOM manipulation
- DOM element selection
- CSS manipulation
- Effects and Animations
- Utilities

- AJAX
- HTML event methods
- JSON Parsing
- Extensibility through plug-ins

Why jQuery is required

- It is very fast and extensible.
- It facilitates the users to write UI related function codes in minimum possible lines.
- It improves the performance of an application.
- Browser's compatible web applications can be developed.
- It uses mostly new features of new browsers.

Adding jQuery to Your Web Pages

There are several ways to start using jQuery on your web site. You can:

- Download the jQuery library from jQuery.com
- Include jQuery from a CDN, like Google

Downloading jQuery

There are two versions of jQuery available for downloading:

- Production version - this is for your live website because it has been minified and compressed
- 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

If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).

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>
```

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:

`$(this).hide()` - hides the current element.

`$("p").hide()` - hides all <p> elements.

`$(".test").hide()` - hides all elements with class="test".

`$("#test").hide()` - hides the element with id="test".

The Document Ready Event

You might have noticed that all jQuery methods in our examples, are inside a 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.

jQuery Selectors

jQuery selectors are one of the most important parts of the jQuery library.

---

jQuery Selectors

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

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

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

The element Selector

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

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

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

Example

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

Example

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

The #id Selector

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

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

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

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

Example

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

Example

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

The .class Selector

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

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

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

Example

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

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

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p").hide();
  });
});
</script>
</head>
<body>

<h2>This is a heading</h2>

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

<button>Click me to hide paragraphs</button>

</body>
</html>
```

Linking external JavaScript file with an HTML document:

Create external JavaScript file with the extension .js. After creating, add it to the HTML file in the script tag. The src attribute is used to include that external JavaScript file. The `src` attribute specifies the URL of an external script file.

If you have more than one external JavaScript file, then add it in the same web page to increase performance of the page.

You can place an external script reference in `<head>` or `<body>`

Creating an external JavaScript file − new.js

```
function display() {
   alert("Hello World!");
}
```

Now add the external JavaScript file to the HTML web page −

```
<html>
   <body>
      <form>
      <input type="button" value="Result"
onclick="display()"/>
      </form>
      <script src="new.js">
      </script>
   </body>
</html>
```

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

JavaScript Built-in Functions

- Function is a reusable code-block that will be executed whenever it is called.
- Function is a great time saver.
- It is used for performing repetitive tasks where you can call the same function multiple times to get the same effect.
- It allows code reusability.
- JavaScript provides number of built-in functions.
-

1. isNaN()

`NaN` is short for Not a Number.

The `isNaN()` function checks if a value is **NaN (Not-a-Number)** or not.

isNan(6453); //false

isNaN("210AA"); // true

2. isFinite() function checks if the passed value is a finite number.

The syntax of the `isFinite()` function is: isFinite(testvalue)

isFinite(643511)); // true

isFinite(Infinity)); // false

3. parseFloat()

The `parseFloat()` function parses an argument and returns a floating-point number.

console.log(parseFloat(" 10 ")); // 10

console.log(parseFloat(" 3.14seconds")); // 3.14

4. parseInt() The `parseInt()` function parses a string argument and returns an integer.

console.log(parseInt("875.99", 10)); // 875

User-defined Functions:

- User-defined function means you can create a function for your own use.
- In JavaScript, these functions are written in between the <HEAD> tag of the HTML page.

Syntax:

```
function function_name()
{
    //Code;
}
```

Example : Function Declaration

```
function add()
{
    var a, b;
    var sum = 0;
    sum = a + b;
    document.write("Addition : "+sum);
}
```

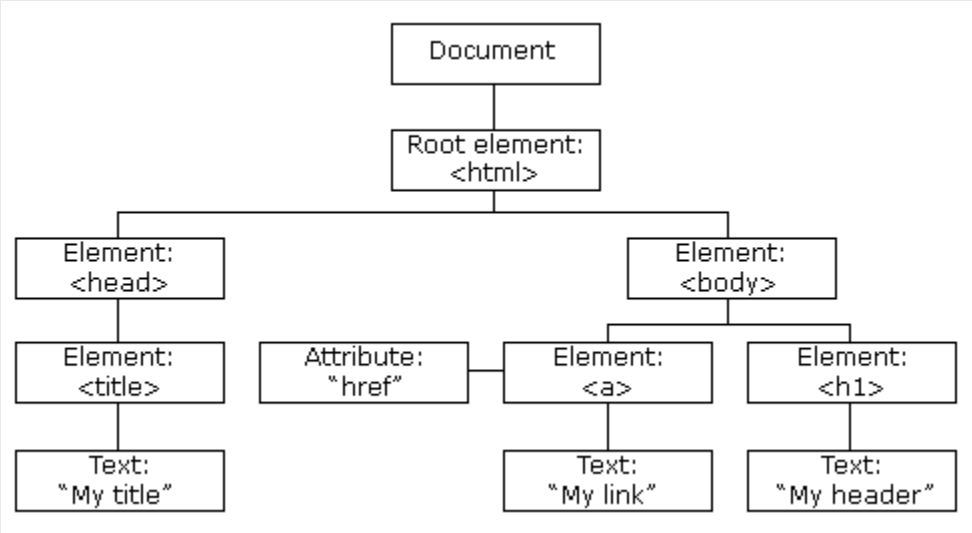
With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

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
- 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.

What is the HTML DOM?

The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

The innerHTML Property

The easiest way to get the content of an element is by using the `innerHTML` property.

The `innerHTML` property is useful for getting or replacing the content of HTML elements.

The `innerHTML` property can be used to get or change any HTML element, including `<html>` and `<body>`.

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.

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

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

Methods of document object

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucment with newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given 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

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are several ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with `id="intro"`:

```
const element = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in myElement).

If the element is not found, myElement will contain `null`.

Finding HTML Elements by Tag Name

This example finds all `<p>` elements:

```
const element = document.getElementsByTagName("p");
```

This example finds the element with `id="main"`, and then finds all `<p>` elements inside `"main"`:

```
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");
```

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use `getElementsByClassName()`.

This example returns a list of all elements with `class="intro"`.

```
const x = document.getElementsByClassName("intro");
```

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the `querySelectorAll()` method.

This example returns a list of all `<p>` elements with `class="intro"`.

```
const x = document.querySelectorAll("p.intro");
```

Changing HTML Content

The easiest way to modify the content of an HTML element is by using the `innerHTML` property.

To change the content of an HTML element, use this syntax:

`document.getElementById(id).innerHTML = new HTML`

This example changes the content of a `<p>` element:

```
<html><body>

<p id="p1">Hello World!</p>

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

- The HTML document above contains a `<p>` element with `id="p1"`
- We use the HTML DOM to get the element with `id="p1"`
- A JavaScript changes the content (`innerHTML`) of that element to "New text!"

This example changes the content of an `<h1>` element:

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

<h1 id="id01">Old Heading</h1>

<script>
const element = document.getElementById("id01");
element.innerHTML = "New Heading";</script></body>
</html>
```

Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:

```
document.getElementById(id).attribute = new value
```

This example changes the value of the src attribute of an `<img>` element:

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

<img id="myImage" src="smiley.gif">

<script>
```

```
document.getElementById("myImage").src = "landscape.jpg";
</script></body></html>
```

- The HTML document above contains an `<img>` element with `id="myImage"`
- We use the HTML DOM to get the element with `id="myImage"`
- A JavaScript changes the `src` attribute of that element from "smiley.gif" to "landscape.jpg"

Dynamic HTML content

JavaScript can create dynamic HTML content:

Date : Wed Dec 15 2021 14:37:21 GMT+0530 (India Standard Time)

```
<!DOCTYPE html><html><body><script>
document.getElementById("demo").innerHTML = "Date : " + Date(); </script>

</body></html>
```

document.write()

In JavaScript, `document.write()` can be used to write directly to the HTML output stream:

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

<p>Bla bla bla</p><script>
document.write(Date());
</script><p>Bla bla bla</p></body></html>
```

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

Here, **document** is the root element that represents the html document.
form1 is the name of the form.
name is the attribute name of the input text.
value is the property, that returns the value of the input text.
Let's see the simple example of document object that prints name with welcome message.

```
<script type="text/javascript">
function printvalue(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>

<form name="form1">
Enter Name:<input type="text" name="name"/>
<input type="button" onclick="printvalue()" value="print name"/>
</form>
```

The **document.getElementById()** method returns the element of specified id.

In the previous page, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

```
<script type="text/javascript">
```

```
function getcube(){
var number=document.getElementById("number").value;
alert(number*number*number);
}
</script>
<form>
Enter No:<input type="text" id="number" name="number"/><br/>
<input type="button" value="cube" onclick="getcube()"/>
</form>
```

The **document.getElementsByName()** method returns all the element of specified name.

The syntax of the getElementsByName() method is given below:

document.getElementsByName("name")

Here, name is required.

```
<script type="text/javascript">
function totalelements()
{
var allgenders=document.getElementsByName("gender");
alert("Total Genders:"+allgenders.length);
}
</script>
<form>
Male:<input type="radio" name="gender" value="male">
Female:<input type="radio" name="gender" value="female">

<input type="button" onclick="totalelements()" value="Total Genders">
</form>
```

The document.getElementsByTagName() method returns all the element of specified tag name.
The syntax of the getElementsByTagName() method is given below:
document.getElementsByTagName("name")