What is xml

- **Xml** (eXtensible Markup Language) is a mark up language.
- XML is designed to store and transport data.
- Xml was released in late 90's. it was created to provide an easy to use and store self describing data.
- XML became a W3C Recommendation on February 10, 1998.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

Note: Self-describing data is the data that describes both its content and structure.

What is mark-up language

A **mark up language** is a modern system for highlight or underline a document.

Students often underline or highlight a passage to revise easily, same in the sense of modern mark up language highlighting or underlining is replaced by tags.

Prerequisite

Before you start to learn xml, you should know basic of HTML & JavaScript.

Why xml

Platform Independent and Language Independent: The main benefit of xml is that you can use it to take data from a program like Microsoft SQL, convert it into XML then share that XML with other programs and platforms. You can communicate between two platforms which are generally very difficult.

The main thing which makes XML truly powerful is its international acceptance. Many corporation use XML interfaces for databases, programming, office application mobile phones and more. It is due to its platform independent feature.

Features and Advantages of XML

XML is widely used in the era of web development. It is also used to simplify data storage and data sharing.

The main features or advantages of XML are given below.

1) XML separates data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.

With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

2) XML simplifies data sharing

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

This makes it much easier to create data that can be shared by different applications.

3) XML simplifies data transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

4) XML simplifies Platform change

Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost.

XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

5) XML increases data availability

Different applications can access your data, not only in HTML pages, but also from XML data sources.

With XML, your data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

6) XML can be used to create new internet languages

A lot of new Internet languages are created with XML.

Here are some examples:

- **XHTML**
- **WSDL** for describing available web services
- **WAP** and **WML** as markup languages for handheld devices
- **RSS** languages for news feeds
- **RDF** and **OWL** for describing resources and ontology
- **SMIL** for describing multimedia for the web

XML Syntax Rules

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

XML Documents Must Have a Root Element

XML documents must contain one **root** element that is the **parent** of all other elements:

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

In this example **<note>** is the root element:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
```

```
  <body>Don't forget me this weekend!</body>
</note>
```

The XML Prolog

This line is called the XML **prolog**:

```
<?xml version="1.0" encoding="UTF-8"?>
```

The XML prolog is optional. If it exists, it must come first in the document.

XML documents can contain international characters, like Norwegian øæå or French êèé.

To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.

UTF-8 is the default character encoding for XML documents.

UTF-8 is also the default encoding for HTML5, CSS, JavaScript, PHP, and SQL.

All XML Elements Must Have a Closing Tag

In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:

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

Note: The XML prolog does not have a closing tag! This is not an error. The prolog is not a part of the XML document.

XML Tags are Case Sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

Opening and closing tags must be written with the same case:

```
<message>This is correct</message>
```

"Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements:

```
<b><i>This text is bold and italic</b></i>
```

In XML, all elements **must** be properly nested within each other:

```
<b><i>This text is bold and italic</i></b>
```

In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

XML Attribute Values Must Always be Quoted

XML elements can have attributes in name/value pairs just like in HTML.

In XML, the attribute values must always be quoted:

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>salary < 1000</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

```
<message>salary &lt; 1000</message>
```

There are 5 pre-defined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand

&apos;	'	apostrophe
&quot;	"	quotation mark

Only < and & are strictly illegal in XML, but it is a good habit to replace > with > as well.

Comments in XML

The syntax for writing comments in XML is similar to that of HTML:

<!-- This is a comment -->

Two dashes in the middle of a comment are not allowed:

<!-- This is an invalid -- comment -->

White-space is Preserved in XML

XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space):

XML:	Hello Tove
HTML:	Hello Tove

XML Stores New Line as LF

Windows applications store a new line as: carriage return and line feed (CR+LF).

Unix and Mac OSX use LF.

Old Mac systems use CR.

XML stores a new line as LF.

XML Example

XML documents create a hierarchical structure looks like a tree so it is known as XML Tree that starts at "the root" and branches to "the leaves".

Example of XML Document

XML documents uses a self-describing and simple syntax:

```
<?xml version="1.0" encoding="UTF-8" standalone=”no”?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0)

The next line describes the root element of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 child elements of the root (to, from, heading, and body).

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element.

```
</note>
```

XML documents must contain a **root element.** This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have sub elements (child elements).

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

Another Example of XML: Books

File: books.xml

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root element in the example is <bookstore>. All elements in the document are contained within <bookstore>.The <book> element has 4 children: <title>,< author>, <year> and <price>.

Another Example of XML: Emails

File: emails.xml

```
<?xml version="1.0" encoding="UTF-8"?>
    <emails>
    <email>
      <to>Vimal</to>
      <from>Sonoo</from>
      <heading>Hello</heading>
      <body>Hello brother, how are you!</body>
    </email>
    <email>
      <to>Peter</to>
      <from>Jack</from>
      <heading>Birth day wish</heading>
      <body>Happy birth day Tom!</body>
    </email>
    <email>
      <to>James</to>
      <from>Jaclin</from>
      <heading>Morning walk</heading>
      <body>Please start morning walk to stay fit!</body>
    </email>
    <email>
      <to>Kartik</to>
      <from>Kumar</from>
      <heading>Health Tips</heading>
      <body>Smoking is injurious to health!</body>
    </email>
    </emails>
```

XML Attributes

XML elements can have attributes. By the use of attributes we can add the information about the element.XML attributes enhance the properties of the elements.

Note: XML attributes must always be quoted. We can use single or double quote.

Let us take an example of a book publisher. Here, book is the element and publisher is the attribute.

```
<book publisher="Tata McGraw Hill"></book>
```

Or

```
<book publisher='Tata McGraw Hill'></book>
```

Metadata should be stored as attribute and data should be stored as element.

```
<book>
<book category="computer">
<author> A & B </author>
</book>
```

Data can be stored in attributes or in child elements. But there are some limitations in using attributes, over child elements.

Why should we avoid XML attributes

- Attributes cannot contain multiple values but child elements can have multiple values.
- Attributes cannot contain tree structure but child element can.
- Attributes are not easily expandable. If you want to change in attribute's vales in future, it may be complicated.
- Attributes cannot describe structure but child elements can.
- Attributes are more difficult to be manipulated by program code.
- Attributes values are not easy to test against a DTD, which is used to define the legal elements of an XML document.

Difference between attribute and sub-element

In the context of documents, attributes are part of markup, while sub elements are part of the basic document contents.

In the context of data representation, the difference is unclear and may be confusing.

Same information can be represented in two ways:

1st way:

```
<book publisher="Tata McGraw Hill"> </book>
```

2nd way:

```
<book>
<publisher> Tata McGraw Hill </publisher>
</book>
```

In the first example publisher is used as an attribute and in the second example publisher is an element.

Both examples provide the same information but it is good practice to avoid attribute in XML and use elements instead of attributes.

XML Comments

XML comments are just like HTML comments. We know that the comments are used to make codes more understandable other developers.

XML Comments add notes or lines for understanding the purpose of an XML code. Although XML is known as self-describing data but sometimes XML comments are necessary.

Syntax

An XML comment should be written as:

```
<!-- Write your comment-->
```

You cannot nest one XML comment inside the another.

XML Comments Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--Students marks are uploaded by months-->
<students>
  <student>
    <name>Ratan</name>
    <marks>70</marks>
  </student>
  <student>
    <name>Aryan</name>
    <marks>60</marks>
  </student>
 </students>
```

Rules for adding XML comments

- Don't use a comment before an XML declaration.
- You can use a comment anywhere in XML document except within attribute value.
- Don't nest a comment inside the other comment.

XML Tree Structure

An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.

A tree structure contains root element (as parent), child element and so on. It is very easy to traverse all succeeding branches and sub-branches and leaf nodes starting from the root.

Example of an XML document

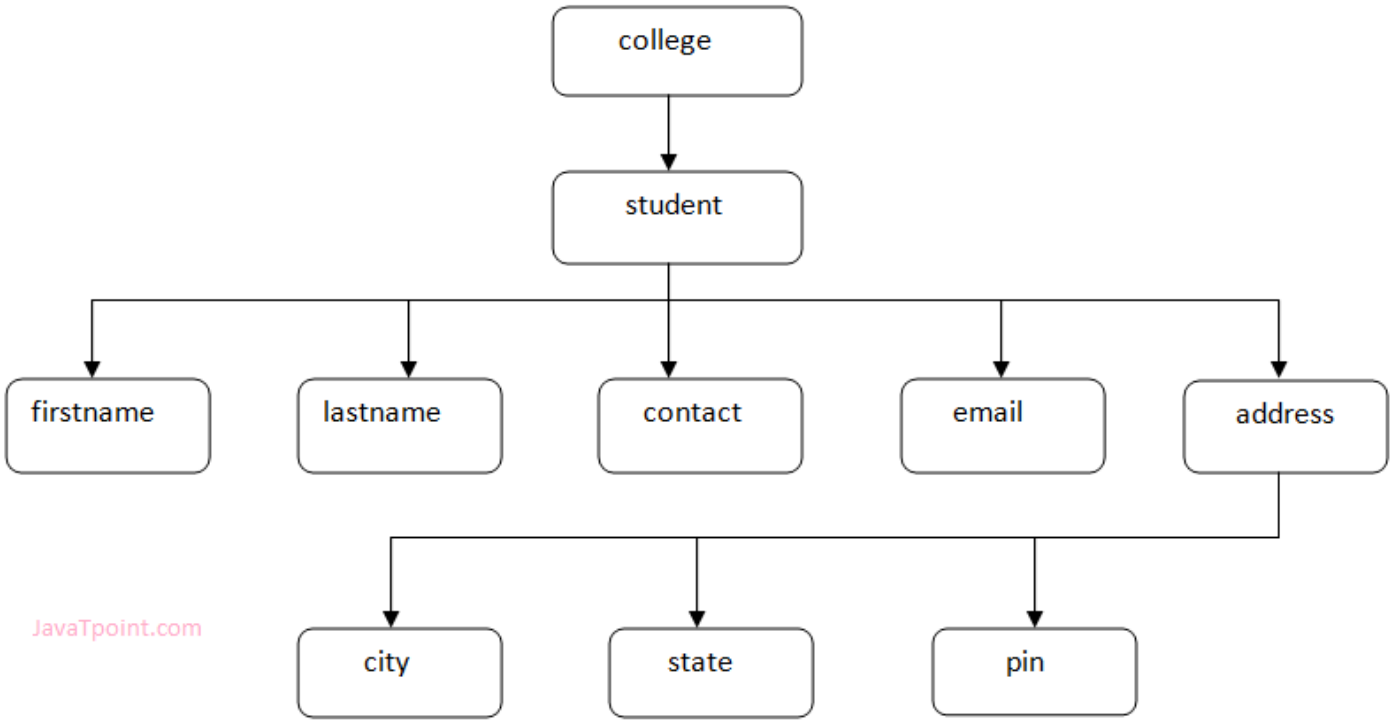
```
<?xml version="1.0"?>
<college>
  <student>
    <firstname>Tamanna</firstname>
    <lastname>Bhatia</lastname>
    <contact>09990449935</contact>
    <email>tammanabhatia@abc.com</email>
    <address>
      <city>Ghaziabad</city>
      <state>Uttar Pradesh</state>
      <pin>201007</pin>
    </address>
  </student>
</college>
```

Let's see the tree-structure representation of the above example.

In the below example, first line is the XML declaration. It defines the XML version 1.0. Next line shows the root element (college) of the document. Inside that there is one more element (student). Student element contains five branches named <firstname>, <lastname>, <contact>, <Email> and <address>.

<address> branch contains 3 sub-branches named <city>, <state> and <pin>.

Note: DOM parser represents the XML document in Tree structure.



XML DTD

What is DTD

DTD stands for **Document Type Definition**. It defines the legal building blocks of an XML document. It is used to define document structure with a list of legal elements and attributes.

Purpose of DTD

Its main purpose is to define the structure of an XML document. It contains a list of legal elements and define the structure with the help of them.

Checking Validation

Before proceeding with XML DTD, you must check the validation. An XML document is called "well-formed" if it contains the correct syntax.

A well-formed and valid XML document is one which have been validated against DTD.

Valid and well-formed XML document with DTD

Let's take an example of well-formed and valid XML document. It follows all the rules of DTD.

employee.xml

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

In the above example, the DOCTYPE declaration refers to an external DTD file. The content of the file is shown in below paragraph.

employee.dtd

```
<!ELEMENT employee (firstname,lastname,email)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Description of DTD

<!DOCTYPE employee : It defines that the root element of the document is employee.

<!ELEMENT employee: It defines that the employee element contains 3 elements "firstname, lastname and email".

<!ELEMENT firstname: It defines that the firstname element is #PCDATA typed. (parse-able data type).

<!ELEMENT lastname: It defines that the lastname element is #PCDATA typed. (parse-able data type).

<!ELEMENT email: It defines that the email element is #PCDATA typed. (parse-able data type).

Example of Valid XML Documents

A "Valid" XML document is "Well Formed", as well as it conforms to the rules of a DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
```

```
<body>Don't forget me this weekend!</body>
</note>
```

The DOCTYPE declaration above contains a reference to a DTD file. The content of the DTD file is shown and explained below.

!DOCTYPE note defines that the root element of this document is note.

Note.dtd:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

The DTD above is interpreted like this:

- !DOCTYPE note - Defines that the root element of the document is note
- !ELEMENT note - Defines that the note element must contain the elements: "to, from, heading, body"
- !ELEMENT to - Defines the to element to be of type "#PCDATA"
- !ELEMENT from - Defines the from element to be of type "#PCDATA"
- !ELEMENT heading - Defines the heading element to be of type "#PCDATA"
- !ELEMENT body - Defines the body element to be of type "#PCDATA"

Tip: #PCDATA means parseable character data.

When to Use a DTD?

With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.

With a DTD, you can verify that the data you receive from the outside world is valid.

You can also use a DTD to verify your own data.

When NOT to Use a DTD?

XML does not require a DTD.

When you are experimenting with XML, or when you are working with small XML files, creating DTDs may be a waste of time.

If you develop applications, wait until the specification is stable before you add a DTD. Otherwise, your software might stop working because of validation errors

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then the document can be linked to the DTD document to use it.

Syntax

Basic syntax of a DTD is as follows −

```
<!DOCTYPE element DTD identifier
[
  declaration1
  declaration2
  ........
]>
```

In the above syntax −

- **DTD** starts with <!DOCTYPE delimiter.
- An **element** tells the parser to parse the document from the specified root element.
- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **external subset.**
- The **square brackets [ ]** enclose an optional list of entity declarations called **internal subset**.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To reference it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means the declaration works independent of external source.

Syntax

The syntax of internal DTD is as shown −

```
<!DOCTYPE root-element [element-declarations]>
```

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>

<!DOCTYPE address [
  <!ELEMENT address (name,company,phone)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT company (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
]>

<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

Let us go through the above code −

Start Declaration − Begin the XML declaration with following statement.

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
```

DTD − Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE −

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body − The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations −

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration − Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) - it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To reference it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD −

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">

<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown −

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by either using **system identifiers** or **public identifiers**.

System Identifiers

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows −

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see it contains keyword SYSTEM and a URI reference pointing to the location of the document.

Public Identifiers

Public identifiers provide a mechanism to locate DTD resources and are written as below −

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called *Formal Public Identifiers, or FPIs.*

PCDATA

PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that WILL be parsed by a parser. **The text will be examined by the parser for entities and markup**.

Tags inside the text will be treated as markup and entities will be expanded.

However, parsed character data should not contain any &, <, or > characters; these need to be represented by the & < and > entities, respectively.

CDATA

CDATA means character data.

CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

jQuery

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.

What is jQuery

jQuery is a small, light-weight and fast JavaScript library. It is cross-platform and supports different types of browsers. It is also referred as write less do more because it takes a lot of common tasks that requires many lines of JavaScript code to accomplish, and binds them into methods that can be called with a single line of code whenever needed. It is also very useful to simplify a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

- jQuery is a small, fast, and lightweight JavaScript library.
- jQuery is platform-independent.
- jQuery means "write less do more".
- jQuery simplifies AJAX call and DOM manipulation.

jQuery Features

Following are the important features of jQuery.

- 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

Sometimes, a question can arise that what is the need of jQuery or what difference it makes on bringing jQuery instead of AJAX/ JavaScript? If jQuery is the replacement of AJAX and JavaScript? For all these questions, you can state the following answers.

- 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.

So, you can say that out of the lot of JavaScript frameworks, jQuery is the most popular and the most extendable. Many of the biggest companies on the web use jQuery.

Some of these companies are:

- Microsoft
- Google
- IBM
- Netflix

- Utilities

Tip: In addition, jQuery has plugins for almost any task out there.

Will jQuery work in all browsers?

The jQuery team knows all about cross-browser issues, and they have written this knowledge into the jQuery library. jQuery will run the same in all major browsers.

What should you know before starting to learn jQuery?

It is always advised to a fresher to learn the basics of web designing before starting to learn jQuery. He should learn HTML, CSS and JavaScript first. But, if you belong to a technical background, it is up to you.If you are a fresher and want to study these subjects first.

jQuery History

jQuery was first released in January 2006 by **John Resig** at BarCamp NYC. It is currently headed by Timmy Wilson and maintained by a team of developers.

Nowadays, jQuery is widely used technology. Most of the websites are using jQuery.

jQuery Get Started

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-3.6.0.min.js"></script>
</head>
```

Note : we will use "jquery-3.5.1.min.js" jQuery version(3.5.1))

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/3.6.0/jquery.min.js"></script>
</head>
```

One big advantage of using the hosted jQuery from Google:

Many users already have downloaded jQuery from Google when visiting another site. As a

result, it will be loaded from cache when they visit your site, which leads to faster loading time. Also, most CDN's will make sure that once a user requests a file from it, it will be served from the server closest to them, which also leads to faster loading time.

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".

Are you familiar with CSS selectors?

jQuery uses CSS syntax to select elements..

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 Example

jQuery is developed by Google. To create the first jQuery example, you need to use JavaScript file for jQuery. You can download the jQuery file from jquery.com or use the absolute URL of jQuery file.

In this jQuery example, we are using the absolute URL of jQuery file. The jQuery example is written inside the script tag.

Let's see a simple example of jQuery.

File: firstjquery.html

```
<!DOCTYPE html>
<html>
<head>
 <title>First jQuery Example</title>
 <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
 </script>
 <script type="text/javascript" language="javascript">
 $(document).ready(function() {
 $("p").css("background-color", "cyan");
 });
```

```
 </script>
 </head>
<body>
<p>The first paragraph is selected.</p>
<p>The second paragraph is selected.</p>
<p>The third paragraph is selected.</p>
</body>
</html>
```

Output:

The first paragraph is selected.

The second paragraph is selected.

The third paragraph is selected.

$(document).ready() and $()

The code inserted between $(document).ready() is executed only once when page is ready for JavaScript code to execute.

In place of $(document).ready(), you can use shorthand notation $() only.

```
$(document).ready(function() {
$("p").css("color", "red");
});
```

The above code is equivalent to this code.

```
$(function() {
$("p").css("color", "red");
});
```

Let's see the full example of jQuery using shorthand notation $().

File: shortjquery.html

```
<!DOCTYPE html>
<html>
<head>
 <title>Second jQuery Example</title>
 <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
 </script>
```

```
<script type="text/javascript" language="javascript">
$(function() {
$("p").css("color", "red");
});
</script>
</head>
<body>
<p>The first paragraph is selected.</p>
<p>The second paragraph is selected.</p>
<p>The third paragraph is selected.</p>
</body>
</html>
```

Output:

The first paragraph is selected.

The second paragraph is selected.

The third paragraph is selected.

function() { $("p").css("background-color", "cyan"); }

It changes the background-color of all <p> tag or paragraph to cyan.

jQuery Selectors

jQuery Selectors are used to select and manipulate HTML elements. They are very important part of jQuery library.

With jQuery selectors, you can find or select HTML elements based on their id, classes, attributes, types and much more from a DOM.

In simple words, you can say that selectors are used to select one or more HTML elements using jQuery and once the element is selected then you can perform various operation on that.

All jQuery selectors start with a dollor sign and parenthesis e.g. $(). It is known as the factory function.

The $() factory function

Every jQuery selector start with thiis sign $(). This sign is known as the factory function. It uses the three basic building blocks while selecting an element in a given document.

S.No.	Selector	Description
1)	Tag Name:	It represents a tag name available in the DOM. For example: $('p') selects all paragraphs'p'in the document.
2)	Tag ID:	It represents a tag available with a specific ID in the DOM. For example: $('#real-id') selects a specific element in the document that has an ID of real-id.
3)	Tag Class:	It represents a tag available with a specific class in the DOM. For example: $('real-class') selects all elements in the document that have a class of real-class.

Let's take a simple example to see the use of Tag selector. This would select all the elements with a tag name

and the background color is set to "pink".

File: firstjquery.html

```
<!DOCTYPE html>
<html>
<head>
 <title>First jQuery Example</title>
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js">
 </script>
 <script type="text/javascript" language="javascript">
 $(document).ready(function() {
 $("p").css("background-color", "pink");
 });
 </script>
 </head>
<body>
<p>This is first paragraph.</p>
<p>This is second paragraph.</p>
<p>This is third paragraph.</p>
</body>
</html>
```

Output:

This is first paragraph.

This is second paragraph.

This is third paragraph.

Note: 1. All of the above discussed selectors can be used alone or with the combination of other selectors.

Note: 2. If you have any confliction with theuse of dollor sign $ in any JavaScript library then you can use jQuery() function instead of factory function $(). The factory function $() and the jQuery function is the same.

How to use Selectors

The jQuery selectors can be used single or with the combination of other selectors. They are required at every steps while using jQuery. They are used to select the exact element that you want from your HTML document.

S.No.	Selector	Description
1)	Name:	It selects all elements that match with the given element name.
2)	#ID:	It selects a single element that matches with the given id.
3)	.Class:	It selects all elements that matches with the given class.
4)	Universal(*)	It selects all elements available in a DOM.
5)	Multiple Elements A,B,C	It selects the combined results of all the specified selectors A,B and C.

Different jQuery Selectors

Selector	Example	Description
*	$("*")	It is used to select all elements.
#id	$("#firstname")	It will select the element with id="firstname"

.class	$(".primary")	It will select all elements with class="primary"
Element	$("p")	It will select all p elements.
:first	$("p:first")	This will select the first p element
:last	$("p:last")	This will select he last p element
:even	$("tr:even")	This will select all even tr elements
:odd	$("tr:odd")	This will select all odd tr elements
:eq(index)	$("ul li:eq(3)")	It will select the fourth element in a list (index starts at 0)
:header	$(":header")	Select all header elements h1, h2 ...
[attribute]	$("[href]")	Select all elements with a href attribute
[attribute=value]	$("[href='default.htm']")	Select all elements with a href attribute value equal to "default.htm"
:input	$(":input")	It will select all input elements
:text	$(":text")	It will select all input elements with type="text"
:password	$(":password")	It will select all input elements with type="password"
:radio	$(":radio")	It will select all input elements with type="radio"
:checkbox	$(":checkbox")	Itwill select all input elements with type="checkbox"
:submit	$(":submit")	It will select all input elements with type="submit"
:reset	$(":reset")	It will select all input elements with type="reset"
:button	$(":button")	It will select all input elements with type="button"
:image	$(":image")	It will select all input elements with type="image"

:file	$(":file")	It will select all input elements with type="file"
:enabled	$(":enabled")	Select all enabled input elements
:disabled	$(":disabled")	It will select all disabled input elements
:selected	$(":selected")	It will select all selected input elements
:checked	$(":checked")	It will select all checked input elements

Example:

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

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

$(".test").hide();

$("#p1").hide();
});
});
```

In the above example:

$("button")-> element button is selected when its event click is performed.

$(".test")->the element having class test is selected.

$("#p1")-> the element with id p1 is selected.

jQuery Event Methods

jQuery is tailor-made to respond to events in an HTML page.

What are Events?

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".

Here are some common DOM events:

Mouse Events	Keyboard Events	Form Events	Document/Window Events
Click	keypress	submit	load
Dblclick	keydown	change	resize
Mouseenter	keyup	focus	scroll
Mouseleave		blur	unload

jQuery Syntax For Event Methods

In jQuery, most DOM events have an equivalent jQuery method.

To assign a click event to all paragraphs on a page, you can do this:

```
$("p").click();
```

The next step is to define what should happen when the event fires. You must pass a function to the event:

```
$("p").click(function(){
  // action goes here!!
});
```

Commonly Used jQuery Event Methods

$(document).ready()

The $(document).ready() method allows us to execute a function when the document is fully loaded. This event is already explained .

click()

The click() method attaches an event handler function to an HTML element.

The function is executed when the user clicks on the HTML element.

The following example says: When a click event fires on a <p> element; hide the current <p> element:

Example

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

dblclick()

The dblclick() method attaches an event handler function to an HTML element.

The function is executed when the user double-clicks on the HTML element:

Example

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

mouseenter()

The mouseenter() method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer enters the HTML element:

Example

```
$("#p1").mouseenter(function(){
  alert("You entered p1!");
});
```

mouseleave()

The mouseleave() method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer leaves the HTML element:

Example

```
$("#p1").mouseleave(function(){
  alert("Bye! You now leave p1!");
});
```

mousedown()

The mousedown() method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is pressed down, while the mouse is over the HTML element:

Example

```
$("#p1").mousedown(function(){
  alert("Mouse down over p1!");
});
```

mouseup()

The mouseup() method attaches an event handler function to an HTML element.

The function is executed, when the left, middle or right mouse button is released, while the mouse is over the HTML element:

Example

```
$("#p1").mouseup(function(){
  alert("Mouse up over p1!");
});
```

hover()

The hover() method takes two functions and is a combination of the mouseenter() and mouseleave() methods.

The first function is executed when the mouse enters the HTML element, and the second function is executed when the mouse leaves the HTML element:

Example

```
$("#p1").hover(function(){
  alert("You entered p1!");
},
function(){
  alert("Bye! You now leave p1!");
});
```

focus()

The focus() method attaches an event handler function to an HTML form field.

The function is executed when the form field gets focus:

Example

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

blur()

The blur() method attaches an event handler function to an HTML form field.

The function is executed when the form field loses focus:

Example

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

The on() Method

The on() method attaches one or more event handlers for the selected elements.

Attach a click event to a <p> element:

Example

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

Attach multiple event handlers to a <p> element:

Example

```
$("p").on({
  mouseenter: function(){
    $(this).css("background-color", "lightgray");
  },
  mouseleave: function(){
    $(this).css("background-color", "lightblue");
  },
  click: function(){
```

```
    $(this).css("background-color", "yellow");
  }
});
```

jQuery keypress()

The jQuery keypress () event is occurred when a keyboard button is pressed down. This event is similar to keydown() event. The keypress() method is executed or attach a function to run when a keypress() event occurs.

Syntax:

```
$(selector).keypress()
```

It triggers the keypress event for selected elements.

```
$(selector).keypress(function)
```

It adds a function to the keypress event.

Parameters of jQuery keypress() event

Parameter	Description
Function	It is an optional parameter. It is executed itself when the keypress event is triggered.

Example of jQuery keypress() event

Let's take an example to demonstrate jQuery keypress() event.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>
<script>
i = 0;
$(document).ready(function(){
    $("input").keypress(function(){
        $("span").text (i += 1);
    });
});
</script>
</head>
<body>
```

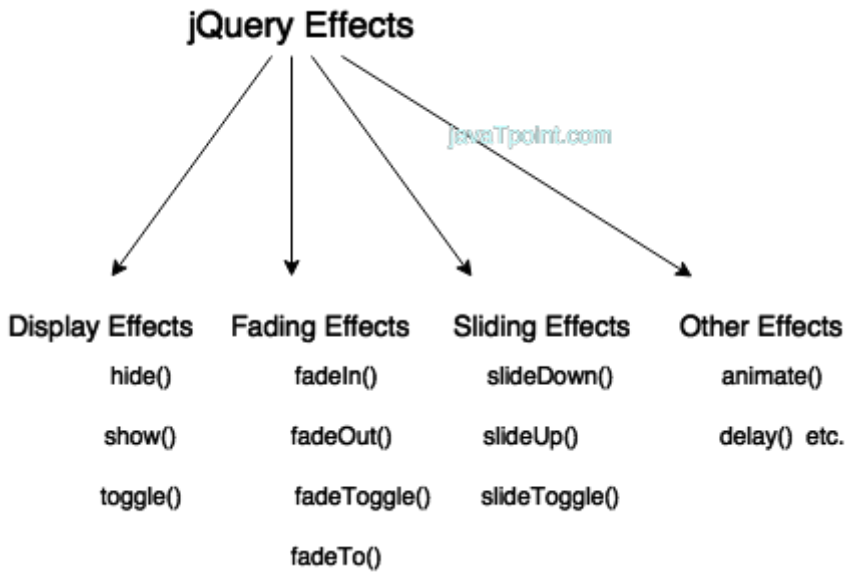
```
Write something: <input type="text">
<p>Keypresses: <span>0</span></p>
</body>
</html>
```

Write something: 0123

Keypresses: 4

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 provides many methods for effects on a web page. A complete list of jQuery effect methods are given below:

No.	Method	Description
1)	animate()	performs animation.
2	clearQueue()	It is used to remove all remaining queued functions from the selected elements.
3)	delay()	sets delay execution for all the queued functions on the selected elements.

4	dequeue()	It is used to remove the next function from the queue, and then execute the function.
5)	fadein()	shows the matched elements by fading it to opaque. In other words, it fades in the selected elements.
6)	fadeout()	shows the matched elements by fading it to transparent. In other words, it fades out the selected elements.
7)	fadeto()	adjusts opacity for the matched element. In other words, it fades in/out the selected elements.
8)	fadetoggle()	shows or hides the matched element. In other words, toggles between the fadeIn() and fadeOut() methods.
9)	finish()	It stops, removes and complete all queued animation for the selected elements.
10)	hide()	hides the matched or selected elements.
11)	queue()	shows or manipulates the queue of methods i.e. to be executed on the selected elements.
12)	show()	displays or shows the selected elements.
13)	slidedown()	shows the matched elements with slide.
14)	slidetoggle()	shows or hides the matched elements with slide. In other words, it is used to toggle between the slideUp() and slideDown() methods.
15)	slideup()	hides the matched elements with slide.
16)	stop()	stops the animation which is running on the matched elements.
17)	toggle()	shows or hides the matched elements. In other words, it toggles between the hide() and show() methods.

jQuery hide()

The jQuery hide() method is used to hide the selected elements.

Syntax:

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

Let's take an example to see the jQuery hide effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
   $("#hide").click(function(){
      $("p").hide();
   });
});
</script>
</head>
<body>
<p>
<b>This is a little poem: </b><br/>
Twinkle, twinkle, little star<br/>
How I wonder what you are<br/>
Up above the world so high<br/>
Like a diamond in the sky<br/>
Twinkle, twinkle little star<br/>
How I wonder what you are
</p>
<button id="hide">Hide</button>
</body>
</html>
```

jQuery show()

The jQuery show() method is used to show the selected elements.

Syntax:

1. $(selector).show();
2. $(selector).show(speed, callback);
3. $(selector).show(speed, easing, 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 show() effect.

Let's take an example to see the jQuery show effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#hide").click(function(){
    $("p").hide();
  });
  $("#show").click(function(){
    $("p").show();
  });
});
</script>
</head>
<body>
<p>
<b>This is a little poem: </b><br/>
Twinkle, twinkle, little star<br/>
How I wonder what you are<br/>
Up above the world so high<br/>
Like a diamond in the sky<br/>
Twinkle, twinkle little star<br/>
How I wonder what you are
</p>
<button id="hide">Hide</button>
<button id="show">Show</button>
</body>
</html>
```

jQuery show() effect with speed parameter

Let's see the example of jQuery show effect with 1500 milliseconds speed.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#hide").click(function(){
    $("p").hide(1000);
  });
  $("#show").click(function(){
    $("p").show(1500);
  });
});
</script>
</head>
<body>
<p>
<b>This is a little poem: </b><br/>
Twinkle, twinkle, little star<br/>
How I wonder what you are<br/>
Up above the world so high<br/>
Like a diamond in the sky<br/>
Twinkle, twinkle little star<br/>
How I wonder what you are
</p>
<button id="hide">Hide</button>
<button id="show">Show</button>
</body>
</html>
```

jQuery toggle()

The jQuery toggle() is a special type of method which is used to toggle between the hide() and show() method. It shows the hidden elements and hides the shown element.

Syntax:

```
$(selector).toggle();
$(selector).toggle(speed, callback);
$(selector).toggle(speed, easing, callback);
$(selector).toggle(display);
```

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 toggle() effect.

display: If true, it displays element. If false, it hides the element.

Let's take an example to see the jQuery toggle effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("div.d1").toggle();
    });
});
</script>
</head>
<body>
<button>Toggle</button>
<div class="d1" style="border:1px solid black;padding:10px;width:250px">
<p><b>This is a little poem: </b><br/>
Twinkle, twinkle, little star<br/>
How I wonder what you are<br/>
Up above the world so high<br/>
```

```
Like a diamond in the sky<br/>
Twinkle, twinkle little star<br/>
How I wonder what you are</p>
</div>
</body>
</html>
```

jQuery toggle() effect with speed parameter

Let's see the example of jQuery toggle effect with 1500 milliseconds speed.

```
$(document).ready(function(){
    $("button").click(function(){
        $("div.d1").toggle(1500);
    });
});
```

jQuery fadeIn()

jQuery fadeIn() method is used to fade in the element.

Syntax:

1. $(selector).fadein();
2. $(selector).fadeIn(speed,callback);
3. $(selector).fadeIn(speed, easing, 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 fadein() effect.

Let's take an example to demonstrate jQuery fadeIn() effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
```

```
        $("#div1").fadeIn();
        $("#div2").fadeIn("slow");
        $("#div3").fadeIn(3000);
    });
});
</script>
</head>
<body>
<p>See the fadeIn() method example with different parameters.</p>
<button>Click to fade in boxes</button><br><br>
<div id="div1" style="width:80px;height:80px;display:none;background-color:red;"></div><br>
<div id="div2" style="width:80px;height:80px;display:none;background-color:green;"></div><br>
<div id="div3" style="width:80px;height:80px;display:none;background-color:blue;"></div>
</body>
</html>
```

jQuery fadeOut()

The jQuery fadeOut() method is used to fade out the element.

Syntax:

1. $(selector).fadeOut();
2. $(selector).fadeOut(speed,callback);
3. $(selector).fadeOut(speed, easing, 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 fadeOut() effect.

Let's take an example to demonstrate jQuery fadeOut() effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
```

```
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").fadeOut();
    $("#div2").fadeOut("slow");
    $("#div3").fadeOut(3000);
  });
});
</script>
</head>
<body>
<p>See the fadeOut() method example with different parameters.</p>
<button>Click to fade out boxes</button><br><br>
<div id="div1" style="width:80px;height:80px;background-color:red;"></div><br>
<div id="div2" style="width:80px;height:80px;background-color:green;"></div><br>
<div id="div3" style="width:80px;height:80px;background-color:blue;"></div>
</body>
</html>
```

jQuery fadeToggle()

jQuery fadeToggle() method is used to toggle between the fadeIn() and fadeOut() methods. If the elements are faded in, it will make them faded out and if they are faded out it will make them faded in.

Syntax:

1. $(selector).fadeToggle();
2. $(selector).fadeToggle(speed,callback);
3. $(selector).fadeToggle(speed, easing, 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 fadeToggle() effect.

Let's take an example to demonstrate jQuery fadeToggle() effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
```

```
<script>
$(document).ready(function(){
   $("button").click(function(){
      $("#div1").fadeToggle();
      $("#div2").fadeToggle("slow");
      $("#div3").fadeToggle(3000);
   });
});
</script>
</head>
<body>
<p>See the fadeToggle() method example with different parameters.</p>
<button>Click to fade Toggle boxes</button><br><br>
<div id="div1" style="width:80px;height:80px;background-color:red;"></div><br>
<div id="div2" style="width:80px;height:80px;background-color:green;"></div><br>
<div id="div3" style="width:80px;height:80px;background-color:blue;"></div>
</body>
</html>
```

jQuery fadeTo()

jQuery fadeTo() method is used to fading to a given opacity.

Syntax:

1. $(selector).fadeTo(speed, opacity);
2. $(selector).fadeTo(speed, opacity, callback);
3. $(selector).fadeTo(speed, opacity, easing, callback);

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

opacity:It specifies the opacity. The opacity value ranges between 0 and 1.

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 fadeToggle() effect.

Let's take an example to demonstrate jQuery fadeTo() effect.

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

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").fadeTo("slow", 0.3);
        $("#div2").fadeTo("slow", 0.4);
        $("#div3").fadeTo("slow", 0.5);
    });
});
</script>
</head>
<body>
<p>See the fadeTo() method example with different parameters.</p>
<button>Click to fade boxes</button><br><br>
<div id="div1" style="width:80px;height:80px;background-color:red;"></div><br>
<div id="div2" style="width:80px;height:80px;background-color:green;"></div><br>
<div id="div3" style="width:80px;height:80px;background-color:blue;"></div>
</body>
</html>
```

jQuery slideDown()

jQuery slideDown() method is used to slide down an element.

Syntax:

1. $(selector).slideDown(speed);
2. $(selector).slideDown(speed, callback);
3. $(selector).slideDown(speed, easing, callback);

speed: 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 slideDown() effect.

Let's take an example to demonstrate jQuery slideDown() effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
```

```
<script>
$(document).ready(function(){
    $("#flip").click(function(){
        $("#panel").slideDown("slow");
    });
});
</script>
 <style>
#panel, #flip {
    padding: 5px;
    text-align: center;
    background-color: #00FFFF;
    border: solid 1px #c3c3c3;
}
#panel {
    padding: 50px;
    display: none;
}
</style>
</head>
<body>
<div id="flip">Click to slide down panel</div>
<div id="panel">Hello javatpoint.com!
It is the best tutorial website to learn jQuery and other languages.</div>
</body>
</html>
```

jQuery slideUp()

jQuery slideDown() method is used to slide up an element.

Syntax:

1. $(selector).slideUp(speed);
2. $(selector).slideUp(speed, callback);
3. $(selector).slideUp(speed, easing, callback);

speed: 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 slideUp() effect.

Let's take an example to demonstrate jQuery slideUp() effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#flip").click(function(){
        $("#panel").slideUp("slow");
    });
});
</script>
 <style>
#panel, #flip {
    padding: 5px;
    text-align: center;
    background-color: #00FFFF;
    border: solid 1px #c3c3c3;
}
#panel {
    padding: 50px;
}
</style>
</head>
<body>
<div id="flip">Click to slide up panel</div>
<div id="panel">Hello javatpoint.com!
It is the best tutorial website to learn jQuery and other languages.</div>
</body>
</html>
```

jQuery slideToggle()

jQuery slideToggle () method is used to toggle between slideUp() and slideDown() method. If the element is slide down, it will slide up the element and if it is slide up, it will slide down.

Syntax:

1. $(selector).slideToggle(speed);
2. $(selector).slideToggle(speed, callback);
3. $(selector).slideToggle(speed, easing, callback);

speed: 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 slideToggle() effect.

Let's take an example to demonstrate jQuery slideToggle() effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("#flip").click(function(){
        $("#panel").slideToggle("slow");
    });
});
</script>
 <style>
#panel, #flip {
    padding: 5px;
    text-align: center;
    background-color: #00FFFF;
    border: solid 1px #c3c3c3;
}
#panel {
    padding: 50px;
    display:none;
}
</style>
</head>
<body>
<div id="flip">Click to slide toggle panel</div>
<div id="panel">Hello javatpoint.com!
It is the best tutorial website to learn jQuery and other languages.</div>
</body>
</html>
```

jQuery animate()

The jQuery animate() method provides you a way to create custom animations.

Syntax:

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

Here, **params** parameter defines the CSS properties to be animated.

The **speed** parameter is optional and specifies the duration of the effect. It can be set as "slow" , "fast" or milliseconds.

The **callback** parameter is also optional and it is a function which is executed after the animation completes.

Let's take a simple example to see the animation effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("div").animate({left: '450px'});
    });
});
</script>
</head>
<body>
<button>Start Animation</button>
<p>A simple animation example:</p>
<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>
</body>
</html>
```

A simple animation example:

Note: The default position of all HTML elements is static. If you want to manipulate their position, set the CSS position property to the element to relative, fixed or absolute.

jQuery animate() method using multiple properties

You can use multiple properties to animate at the same time.

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

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("div").animate({
            left: '250px',
            opacity: '0.5',
            height: '150px',
            width: '150px'
        });
    });
});
</script>
</head>
<body>
<button>Start Animation</button>
<div style="background:#125f21;height:100px;width:100px;position:absolute;"></div>
</body>
</html>
```

jQuery animate() method using relative values

You can also define relative values (it is relative to the element's current value) by putting += or -= in front of the value.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("div").animate({
            left: '250px',
            height: '+=150px',
            width: '+=150px'
        });
    });
});
</script>
</head>
```

```
<body>
<button>Start Animation</button>
<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>
</body>
</html>
```

jQuery animate() method using predefined value

You can also specify a property's animation value as "show" , "hide" , or "toggle".

In this example, we are using "toggle" value for height, it means it will show/hide the selected element.

```
<!DOCTYPE html>
<html>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("div").animate({
            height: 'toggle'
        });
    });
});
</script>
</head>
<body>
<button>Start Animation</button>
<div style="background:#98bf21;height:100px;width:100px;position:absolute;"></div>
</body>
</html>
```

jQuery Color animation

You can also animate the properties of elements between colors.

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>jQuery UI Effects - Animate demo</title>
 <link rel="stylesheet" href="http://code.jquery.com/ui/1.11.4/themes/smoothness/jquery-ui.css">
```

```
<script src="http://code.jquery.com/jquery-1.10.2.js"></script>
<script src="http://code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
<style>
 .toggler { width: 500px; height: 200px; position: relative; }
 #button { padding: .5em 1em; text-decoration: none; }
 #effect { width: 240px; height: 135px; padding: 0.4em; position: relative; background: #fff; }
 #effect h3 { margin: 0; padding: 0.4em; text-align: center; }
</style>
<script>
$(function() {
 var state = true;
 $( "#button" ).click(function() {
  if ( state ) {
   $( "#effect" ).animate({
    backgroundColor: "#aa0000",
    color: "#fff",
    width: 500
   }, 1000 );
  } else {
   $( "#effect" ).animate({
    backgroundColor: "#fff",
    color: "#000",
    width: 240
   }, 1000 );
  }
  state = !state;
 });
});
 </script>
</head>
<body>
<div class="toggler">
 <div id="effect" class="ui-widget-content ui-corner-all">
  <h3 class="ui-widget-header ui-corner-all">Animate</h3>
  <p>Javatpoint.com is the best tutorial website to learn Java and other programming language
s.</p>
 </div>
</div>
 <button id="button" class="ui-state-default ui-corner-all">Toggle Effect</button>
</body>
```

```
</html>
```

jQuery delay()

The jQuery delay() method is used to delay the execution of functions in the queue. It is a best method to make a delay between the queued jQuery effects. The jQUery delay () method sets a timer to delay the execution of the next item in the queue.

Syntax:

```
$(selector).delay (speed, queueName)
```

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

queueName: It is also an optional parameter. It specifies the name of the queue. Its default value is "fx" the standard queue effect.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").delay("slow").fadeIn();
});
});
</script>
</head>
<body>
<button>Click me</button><br>
<div id="div1" style="width:90px;height:90px;display:none;background-
color:black;"></div><br>
</body>
</html>
```

jQuery stop() Method

The jQuery `stop()` method is used to stop an animation or effect before it is finished.

The `stop()` method works for all jQuery effect functions, including sliding, fading and custom animations.

Syntax:

```
$(selector).stop(stopAll,goToEnd);
```

The optional stopAll parameter specifies whether also the animation queue should be cleared or not. Default is false, which means that only the active animation will be stopped, allowing any queued animations to be performed afterwards.

The optional goToEnd parameter specifies whether or not to complete the current animation immediately. Default is false.

So, by default, the `stop()` method kills the current animation being performed on the selected element.

The following example demonstrates the `stop()` method, with no parameters:

Example

```
$("#stop").click(function(){
  $("#panel").stop();
});
```

jQuery Callback Functions

A callback function is executed after the current effect is 100% finished.

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical syntax: **$(*selector*).hide(*speed,callback*);**

Examples

The example below has a callback parameter that is a function that will be executed after the hide effect is completed:

Example with Callback

```
$("button").click(function(){
  $("p").hide("slow", function(){
    alert("The paragraph is now hidden");
  });
});
```

The example below has no callback parameter, and the alert box will be displayed before the hide effect is completed:

Example without Callback

```
$("button").click(function(){
  $("p").hide(1000);
  alert("The paragraph is now hidden");
});
```

jQuery - Chaining

With jQuery, you can chain together actions/methods.

Chaining allows us to run multiple jQuery methods (on the same element) within a single statement.

jQuery Method Chaining

Until now we have been writing jQuery statements one at a time (one after the other).

However, there is a technique called chaining, that allows us to run multiple jQuery commands, one after the other, on the same element(s).

Tip: This way, browsers do not have to find the same element(s) more than once.

To chain an action, you simply append the action to the previous action.

The following example chains together the `css()`, `slideUp()`, and `slideDown()` methods. The "p1" element first changes to red, then it slides up, and then it slides down:

Example

```
$("#p1").css("color", "red").slideUp(2000).slideDown(2000);
```

We could also have added more method calls if needed.

Tip: When chaining, the line of code could become quite long. However, jQuery is not very strict on the syntax; you can format it like you want, including line breaks and indentations.

This also works just fine:

Example

```
$("#p1").css("color", "red")
  .slideUp(2000)
  .slideDown(2000);
```

jQuery - Get Content and Attributes

jQuery contains powerful methods for changing and manipulating HTML elements and attributes.

jQuery DOM Manipulation

One very important part of jQuery is the possibility to manipulate the DOM.

jQuery comes with a bunch of DOM related methods that make it easy to access and manipulate elements and attributes.

DOM = Document Object Model

The DOM defines a standard for accessing HTML and XML 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."

Get/set Content - text(), html(), and val() attr()

Three simple, but useful, jQuery methods for DOM manipulation are:

- `text()` - Sets or returns the text content of selected elements
- `html()` - Sets or returns the content of selected elements (including HTML markup)
- `val()` - Sets or returns the value of form fields

The following example demonstrates how to get content with the jQuery `text()` and `html()` methods:

Example

```
$("#btn1").click(function(){
  alert("Text: " + $("#test").text());
});
$("#btn2").click(function(){
  alert("HTML: " + $("#test").html());
});
```

The following example demonstrates how to get the value of an input field with the jQuery `val()` method:

Example

```
$("#btn1").click(function(){
  alert("Value: " + $("#test").val());
});
```

Get Attributes - attr()

The jQuery `attr()` method is used to get attribute values.

The following example demonstrates how to get the value of the href attribute in a link:

Example

```
$("button").click(function(){
  alert($("#w3s").attr("href"));
});
```

jQuery - Set Content and Attributes

Set Content - text(), html(), and val()

We will use the same three methods from the previous page to **set content**:

- `text()` - Sets or returns the text content of selected elements
- `html()` - Sets or returns the content of selected elements (including HTML markup)
- `val()` - Sets or returns the value of form fields

The following example demonstrates how to set content with the jQuery `text()`, `html()`, and `val()` methods:

Example

```
$("#btn1").click(function(){
  $("#test1").text("Hello world!");
});
$("#btn2").click(function(){
  $("#test2").html("<b>Hello world!</b>");
});
$("#btn3").click(function(){
  $("#test3").val("Dolly Duck");
});
```

A Callback Function for text(), html(), and val()

All of the three jQuery methods above: `text()`, `html()`, and `val()`, also come with a callback function. The callback function has two parameters: the index of the current element in the

list of elements selected and the original (old) value. You then return the string you wish to use as the new value from the function.

The following example demonstrates `text()` and `html()` with a callback function:

Example

```
$("#btn1").click(function(){
  $("#test1").text(function(i, origText){
    return "Old text: " + origText + " New text: Hello world!
    (index: " + i + ")";
  });
});

$("#btn2").click(function(){
  $("#test2").html(function(i, origText){
    return "Old html: " + origText + " New html: Hello <b>world!</b>
    (index: " + i + ")";
  });
});
```

Set Attributes - attr()

The jQuery `attr()` method is also used to set/change attribute values.

The following example demonstrates how to change (set) the value of the href attribute in a link:

Example

```
$("button").click(function(){
  $("#w3s").attr("href", "https://www.w3schools.com/jquery/");
});
```

The `attr()` method also allows you to set multiple attributes at the same time.

The following example demonstrates how to set both the href and title attributes at the same time:

Example

```
$("button").click(function(){
  $("#w3s").attr({
    "href" : "https://www.w3schools.com/jquery/",
    "title" : "W3Schools jQuery Tutorial"
  });
});
```

A Callback Function for attr()

The jQuery method `attr()`, also comes with a callback function. The callback function has two parameters: the index of the current element in the list of elements selected and the original (old) attribute value. You then return the string you wish to use as the new attribute value from the function.

The following example demonstrates `attr()` with a callback function:

Example

```
$("button").click(function(){
  $("#w3s").attr("href", function(i, origValue){
    return origValue + "/jquery/";
  });
});
```

jQuery - Add Elements

With jQuery, it is easy to add new elements/content.

Add New HTML Content

We will look at four jQuery methods that are used to add new content:

- `append()` - Inserts content at the end of the selected elements
- `prepend()` - Inserts content at the beginning of the selected elements
- `after()` - Inserts content after the selected elements
- `before()` - Inserts content before the selected elements

jQuery append() Method

The jQuery `append()` method inserts content AT THE END of the selected HTML elements.

Example

```
$("p").append("Some appended text.");
```

jQuery prepend() Method

The jQuery `prepend()` method inserts content AT THE BEGINNING of the selected HTML elements.

Example

```
$("p").prepend("Some prepended text.");
```

Add Several New Elements With append() and prepend()

In both examples above, we have only inserted some text/HTML at the beginning/end of the selected HTML elements.

However, both the `append()` and `prepend()` methods can take an infinite number of new elements as parameters. The new elements can be generated with text/HTML (like we have done in the examples above), with jQuery, or with JavaScript code and DOM elements.

In the following example, we create several new elements. The elements are created with text/HTML, jQuery, and JavaScript/DOM. Then we append the new elements to the text with the `append()` method (this would have worked for `prepend()` too) :

Example

```
function appendText() {
  var txt1 = "<p>Text.</p>";               // Create element with HTML
  var txt2 = $("<p></p>").text("Text.");   // Create with jQuery
  var txt3 = document.createElement("p");  // Create with DOM
  txt3.innerHTML = "Text.";
  $("body").append(txt1, txt2, txt3);      // Append the new elements
}
```

jQuery after() and before() Methods

The jQuery `after()` method inserts content AFTER the selected HTML elements.

The jQuery `before()` method inserts content BEFORE the selected HTML elements.

Example

```
$("img").after("Some text after");

$("img").before("Some text before");
```

Add Several New Elements With after() and before()

Also, both the `after()` and `before()` methods can take an infinite number of new elements as parameters. The new elements can be generated with text/HTML (like we have done in the example above), with jQuery, or with JavaScript code and DOM elements.

In the following example, we create several new elements. The elements are created with text/HTML, jQuery, and JavaScript/DOM. Then we insert the new elements to the text with the `after()` method (this would have worked for `before()` too) :

Example

```
function afterText() {
  var txt1 = "<b>I </b>";                    // Create element with HTML
  var txt2 = $("<i></i>").text("love ");     // Create with jQuery
  var txt3 = document.createElement("b");    // Create with DOM
  txt3.innerHTML = "jQuery!";
  $("img").after(txt1, txt2, txt3);          // Insert new elements after <img>
}
```

jQuery wrap()

jQuery wrap() method is used to wrap specified HTML elements around each selected element. The wrap () function can accept any string or object that could be passed through the $() factory function.

Syntax:

$(selector).wrap(wrappingElement,function(index))

Parameters of jQuery wrap() method

Parameter	Description
WrappingElement	It is a mandatory parameter. It specifies what HTML elements to wrap around each selected element. Its possible values are: ○ HTML elements ○ jQuery objects ○ DOM elements
Function(index)	It is an optional parameter. It specifies a function that returns the wrapping element. ○ **Index:** It provides the index position of the element in the set.

Example of jQuery wrap() method

Let's take an example to demonstrate the jQuery wrap() method.

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

```
    $("button").click(function(){
      $("p").wrap("<div></div>");
    });
});
</script>
<style>
div{background-color: pink;}
</style>
</head>
<body>
<p>Hello Guys!</p>
<p>This is javatpoint.com</p>
<button>Wrap a div element around each p element</button>
</body>
</html>
```

jQuery - Remove Elements

With jQuery, it is easy to remove existing HTML elements.

Remove Elements/Content

To remove elements and content, there are mainly two jQuery methods:

- `remove()` - Removes the selected element (and its child elements)
- `empty()` - Removes the child elements from the selected element

jQuery remove() Method

The jQuery `remove()` method removes the selected element(s) and its child elements.

Example

```
$("#div1").remove();
```

jQuery empty() Method

The jQuery `empty()` method removes the child elements of the selected element(s).

Example

```
$("#div1").empty();
```

Filter the Elements to be Removed

The jQuery `remove()` method also accepts one parameter, which allows you to filter the elements to be removed.

The parameter can be any of the jQuery selector syntaxes.

The following example removes all `<p>` elements with `class="test"`:

Example

```
$("p").remove(".test");
```

This example removes all `<p>` elements with `class="test"` and `class="demo"`:

Example

```
$("p").remove(".test, .demo");
```

jQuery unwrap()

The jQuery unwrap() method is used to remove the parent element of the selected elements.

Syntax:

```
$(selector).unwrap()
```

Example of jQuery unwrap() method

Let's take an example to demonstrate the jQuery unwrap() method.

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<script>
$(document).ready(function(){
    $("button").click(function(){
        $("p").unwrap();
    });
});
</script>
<style>
div{background-color: orange;}
article{background-color: yellowgreen;}
```

```
</style>
</head>
<body>
<div>
<p>Hello Guys!</p>
</div>
<article>
<p>This is javatpoint.com</p>
</article>
<button>Click here to remove the parent element of each p element</button>
</body>
</html>
```

jQuery - css() Method

jQuery css() Method

The `css()` method sets or returns one or more style properties for the selected elements.

Return a CSS Property

To return the value of a specified CSS property, use the following syntax:

```
css("propertyname");
```

The following example will return the background-color value of the FIRST matched element:

Example

```
$("p").css("background-color");
```

Set a CSS Property

To set a specified CSS property, use the following syntax:

```
css("propertyname","value");
```

The following example will set the background-color value for ALL matched elements:

Example

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

Set Multiple CSS Properties

To set multiple CSS properties, use the following syntax:

```
css({"propertyname":"value","propertyname":"value",...});
```

The following example will set a background-color and a font-size for ALL matched elements:

Example

```
$("p").css({"background-color": "yellow", "font-size": "200%"});
```

What is JSON

JSON is an acronym for JavaScript Object Notation, is an open standard format, which is lightweight and text-based, designed explicitly for human-readable data interchange. It is a language-independent data format. It supports almost every kind of language, framework, and library.

In the early 2000s, JSON was initially specified by Douglas Crockford. In 2013, JSON was standardized as ECMA-404, and RCF 8259 was published in 2017.

JSON is an open standard for exchanging data on the web. It supports data structures like objects and arrays. So, it is easy to write and read data from JSON.

In JSON, data is represented in key-value pairs, and curly braces hold objects, where a colon is followed after each name. The comma is used to separate key-value pairs. Square brackets are used to hold arrays, where each value is comma-separated.

What is JSON

- JSON stands for JavaScript Object Notation.
- JSON is an open standard data-interchange format.
- JSON is lightweight and self-describing.
- JSON originated from JavaScript.
- JSON is easy to read and write.
- JSON is language independent.
- JSON supports data structures such as arrays and objects.
- JSON is plain text written in JavaScript object notation
- JSON is used to send data between computers

The JSON syntax is derived from JavaScript object notation, but the JSON format is text only.

Code for reading and generating JSON exists in many programming languages.

The JSON format was originally specified by Douglas Crockford.

Why Use JSON?

The JSON format is syntactically similar to the code for creating JavaScript objects. Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.

Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.

JavaScript has a built in function for converting JSON strings into JavaScript objects:

JSON.parse()

JavaScript also has a built in function for converting an object into a JSON string:

JSON.stringify()

You can receive pure text from a server and use it as a JavaScript object.

You can send a JavaScript object to a server in pure text format.

You can work with data as JavaScript objects, with no complicated parsing and translations.

Storing Data

When storing data, the data has to be a certain format, and regardless of where you choose to store it, *text* is always one of the legal formats.

JSON makes it possible to store JavaScript objects as text.

Features of JSON

- Simplicity
- Openness
- Self-Describing
- Internationalization
- Extensibility
- Interoperability

Why do we use JSON?

Since JSON is an easy-to-use, lightweight language data interchange format in comparison to other available options, it can be used for API integration. Following are the advantages of JSON:

- **Less Verbose**: In contrast to XML, JSON follows a compact style to improve its users' readability. While working with a complex system, JSON tends to make substantial enhancements.
- **Faster**: The JSON parsing process is faster than that of the XML because the DOM manipulation library in XML requires extra memory for handling large XML files. However, JSON requires less data that ultimately results in reducing the cost and increasing the parsing speed.

- **Readable**: The JSON structure is easily readable and straightforward. Regardless of the programming language that you are using, you can easily map the domain objects.
- **Structured Data**: In JSON, a map data structure is used, whereas XML follows a tree structure. The key-value pairs limit the task but facilitate the predictive and easily understandable model.

JSON vs XML

Before knowing about the differences between JSON and XML, we should be aware of the definition of json and xml.

What is json?

JSON stands for **JavaScript object notation**. JSON has been derived from javascript, where javascript is a programming language. It was originally created to hold the structured data that could be used in javascript. JSON became so popular that it is used for data for all kinds of applications. It is the most popular way of sending the data for Web APIs.

Basic data types supported by json are:

- **Strings**: Characters that are enclosed in single or double quotation marks.
- **Number**: A number could be integer or decimal, positive or negative.
- **Booleans**: The Boolean value could be either true or false without any quotation marks.
- **Null**: Here, null means nothing without any quotation marks.

In addition to basic data types, json has arrays and objects.

Arrays

Arrays are the lists that are represented by the square brackets, and the values have commas in between them. They can contain mix data types, i.e., a single array can have strings, Boolean, numbers.

For example:

Example 1: [1, 2, 7.8, 5, 9, 10];

Example 2: ["red", "yellow", "green"];

Example 3: [8, "hello", null, true];

In the above, example 1 is an array of numbers, example 2 is an array of strings, and example 3 is an array of mix data types.

Objects

Objects are JSON dictionaries that are enclosed in curly brackets. In objects, keys and values are separated by a colon ':', pairs are separated by comma. Keys and values can be of any type, but the most common type for the keys is a string.

For example: {"red" : 1, "yellow" : 2, "green" : 3};

Nesting

Nesting involves keeping the arrays and objects inside of each other. We can put the arrays inside objects, objects inside arrays, arrays inside arrays, etc. We can say that json file is a big object with lots of objects and arrays inside.

Example:

```
{
 "song" :
        {
           "title" :  "Hey Dude";
            "artist": "The Beatles";
             "musicians": ["John Lennon", "Paul McCratney", "Ringo Starr"];
         }
}
```

In the above code, the song starts with a curly bracket. Therefore, a song is an object. It contains three key-value pairs wherein title, artist and musicians are the keys.'

What is XML?

XML stands for an extensible markup language. It is like HTML, where HTML stands for Hypertext Markup language. HTML is used for creating websites, whereas XML can be used for any kind of structured data.

XML has two ways of handling data, i.e., Tags and Attributes. The tags work as HTML. The start tags start with the <_> and end with the </_>. The start and end tags must match. The names must only be letters, numbers, and underscore, and the tag name must start with a letter only.

For example:

<title> Hello World </title>

Nested Tags

When we put the tag inside of another tag that creates the nested data.

For example:

```
<color>
 <red> 1 </red>
<yellow> 2 </yellow>
<green> 3 </green>
</color>
```

As we can observe in the above code that inside the color tag, we have three more tags, i.e., red, yellow, and green.

Similarities between the json and XML.

- **Self-describing:** Both json and xml are self-describing as both xml data and json data are human-readable text.
- **Hierarchical:** Both json and xml support hierarchical structure. Here hierarchical means that the values within values.
- **Data interchange format:** JSON and XML can be used as data interchange formats by many different programming languages.
- **Parse:** Both the formats can be easily parsed.
- **Retrieve:** Both formats can be retrieved by using HTTP requests. The methods used for retrieving the data are GET, PUT, POST.

Differences between the json and XML.

JSON	XML
JSON stands for javascript object notation.	XML stands for an extensible markup language.
The extension of json file is .json.	The extension of xml file is .xml.
The internet media type is application/json.	The internet media type is application/xml or text/xml.
The type of format in JSON is data interchange.	The type of format in XML is a markup language.
It is extended from javascript.	It is extended from SGML.
It is open source means that we do not have to pay anything to use JSON.	It is also open source.
The object created in JSON has some type.	XML data does not have any type.

The data types supported by JSON are strings, numbers, Booleans, null, array.	XML data is in a string format.
It does not have any capacity to display the data.	XML is a markup language, so it has the capacity to display the content.
JSON has no tags.	XML data is represented in tags, i.e., start tag and end tag.
	XML file is larger. If we want to represent the data in XML then it would create a larger file as compared to JSON.
JSON is quicker to read and write.	XML file takes time to read and write because the learning curve is higher.
JSON can use arrays to represent the data.	XML does not contain the concept of arrays.
It can be parsed by a standard javascript function. It has to be parsed before use.	XML data which is used to interchange the data, must be parsed with respective to their programming language to use that.
It can be easily parsed and little bit code is required to parse the data.	It is difficult to parse.
File size is smaller as compared to XML.	File size is larger.
JSON is data-oriented.	XML is document-oriented.
It is less secure than XML.	It is more secure than JSON.

EXAMPLE:

Both JSON and XML can be used to receive data from a web server.

The following JSON and XML examples both define an employees object, with an array of 3 employees:

JSON Example

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

XML Example

```
<employees>
 <employee>
  <firstName>John</firstName> <lastName>Doe</lastName>
 </employee>
 <employee>
  <firstName>Anna</firstName> <lastName>Smith</lastName>
 </employee>
 <employee>
  <firstName>Peter</firstName> <lastName>Jones</lastName>
 </employee>
</employees>
```

JSON is Like XML Because

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest

JSON is Unlike XML Because

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays

The biggest difference is:

XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

Why JSON is Better Than XML

XML is much more difficult to parse than JSON.
JSON is parsed into a ready-to-use JavaScript object.

For AJAX applications, JSON is faster and easier than XML:

Using XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables

Using JSON

- Fetch a JSON string
- JSON.Parse the JSON string

JSON Syntax(**JSON Syntax Rules)**

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON Data - A Name and a Value

JSON data is written as name/value pairs (aka key/value pairs).

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

Example

"name":"John"

JSON names require double quotes.

JSON - Evaluates to JavaScript Objects

The JSON format is almost identical to JavaScript objects.

In JSON, *keys* must be strings, written with double quotes:

JSON

{"name":"John"}

In JavaScript, keys can be strings, numbers, or identifier names:

```
JavaScript
```

```
{name:"John"}
```

JSON Values

In JSON, *values* must be one of the following data types:

- a string
- a number
- an object
- an array
- a boolean
- null

In **JavaScript** values can be all of the above, plus any other valid JavaScript expression, including:

- a function
- a date
- undefined

In JSON, *string values* must be written with double quotes:

```
JSON
```

```
{"name":"John"}
```

In JavaScript, you can write string values with double *or* single quotes:

```
JavaScript
```

```
{name:'John'}
```

JSON Files

- The file type for JSON files is ".json"
- The MIME type for JSON text is "application/json"

JSON Data Types

In JSON, values must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- *null*

JSON values **cannot** be one of the following data types:

- a function
- a date
- *undefined*

JSON Strings

Strings in JSON must be written in double quotes.

Example

```
{"name":"John"}
```

JSON Numbers

Numbers in JSON must be an integer or a floating point.

Example

```
{"age":30}
```

JSON Objects

Values in JSON can be objects.

Example

```
{
"employee":{"name":"John", "age":30, "city":"New York"}
}
```

Objects as values in JSON must follow the JSON syntax.

JSON Arrays

Values in JSON can be arrays.

Example

```
{
"employees":["John", "Anna", "Peter"]
}
```

JSON Booleans

Values in JSON can be true/false.

Example

```
{"sale":true}
```

JSON null

Values in JSON can be null.

Example

```
{"middlename":null}
```

Data Type	Description	Example
String	A string is always written in double-quotes. It may consist of numbers, alphanumeric and special characters.	"student", "name", "1234", "Ver_1"
Number	Number represents the numeric characters.	121, 899
Boolean	It can be either True or False.	true
Null	It is an empty value.	

JSON Object

This is a JSON string:

'{"name":"John", "age":30, "car":null}'

Inside the JSON string there is a JSON object literal:

{"name":"John", "age":30, "car":null}

JSON object literals are surrounded by curly braces {}.

JSON object literals contains key/value pairs.

Keys and values are separated by a colon.

Keys must be strings, and values must be a valid JSON data type:

- string
- number
- object
- array
- boolean
- null

Each key/value pair is separated by a comma.

It is a common mistake to call a JSON object literal "a JSON object".

JSON cannot be an object. JSON is a string format.

The data is only JSON when it is in a string format. When it is converted to a JavaScript variable, it becomes a JavaScript object.

Let's see an example of JSON object.

```
{
  "employee": {
    "name":     "sonoo",
    "salary":    56000,
    "married":   true
  }
}
```

In the above example, employee is an object in which "name", "salary" and "married" are the key. In this example, there are string, number and boolean value for the keys.

JSON Object with Strings

The string value must be enclosed within double quote.

```
{
    "name":     "sonoo",
    "email":    "sonoojaiswal1987@gmail.com"
}
```

JSON Object with Numbers

JSON supports numbers in double precision floating-point format. The number can be digits (0-9), fractions (.33, .532 etc) and exponents (e, e+, e-,E, E+, E-).

```
{
"integer": 34,
"fraction": .2145,
"exponent": 6.61789e+0
}
```

JSON Object with Booleans

JSON also supports boolean values *true* or *false*.

```
{
"first": true,
"second": false
}
```

JSON Nested Object Example

A JSON object can have another object also. Let's see a simple example of JSON object having another object.

```
{
   "firstName": "Sonoo",
   "lastName": "Jaiswal",
   "age": 27,
   "address" : {
     "streetAddress": "Plot-6, Mohan Nagar",
```

```
        "city": "Ghaziabad",
        "state": "UP",
        "postalCode": "201007"
      }
    }
```

Because JSON syntax is derived from JavaScript object notation, very little extra software is needed to work with JSON within JavaScript.

With JavaScript you can create an object and assign data to it, like this:

Example

```
person = {name:"John", age:31, city:"New York"};
```

You can access a JavaScript object like this:

Example

```
// returns John
person.name;
```

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

<h2>Access a JavaScript object</h2>
<p id="demo"></p>

<script>
const myObj = {name:"John", age:30, city:"New York"};
document.getElementById("demo").innerHTML = myObj.name;
</script>

</body>
</html>
```

It can also be accessed like this:

Example

```
// returns John
person["name"];
```

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

```
<body>

<h2>Access a JavaScript object</h2>
<p id="demo"></p>

<script>
const myObj = {name:"John", age:30, city:"New York"};
document.getElementById("demo").innerHTML = myObj["name"];
</script>

</body>
</html>
```

Data can be modified like this:

Example

```
person.name = "Gilbert";
```

It can also be modified like this:

Example

```
person["name"] = "Gilbert";
```

JSON Array

This is a JSON string:

```
'["Ford", "BMW", "Fiat"]'
```

Inside the JSON string there is a JSON array literal:

```
["Ford", "BMW", "Fiat"]
```

Arrays in JSON are almost the same as arrays in JavaScript.

In JSON, array values must be of type string, number, object, array, boolean or *null*.

In JavaScript, array values can be all of the above, plus any other valid JavaScript expression, including functions, dates, and *undefined.*

JSON Array of Strings

Let's see an example of JSON arrays storing string values

```
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
```

JSON Array of Numbers

Let's see an example of JSON arrays storing number values.

```
[12, 34, 56, 43, 95]
```

JSON Array of Booleans

Let's see an example of JSON arrays storing boolean values.

```
[true, true, false, false, true]
```

JSON Array of Objects

Let's see a simple JSON array example having 4 objects.

```
{"employees":[
   {"name":"Ram", "email":"ram@gmail.com", "age":23},
   {"name":"Shyam", "email":"shyam23@gmail.com", "age":28},
   {"name":"John", "email":"john@gmail.com", "age":33},
   {"name":"Bob", "email":"bob32@gmail.com", "age":41}
]}
```

JSON Array of Objects is an ordered list of objects. Following is the example of JSON array having three Objects. In the example, the object "Book" is an array containing three objects Each object is a record of a book (with a title and price).

```
{"Book":
 [
 {"title":"Advanced Web Technologies", "price":200},
 {"title":"Fundamental of web", "price":140},
{"title":"Advanced concept", "price":300},
]};
```

The first title in the object array can be accessed like this: Book[0].title (remember: The array index starts with 0)

Book [0].title

OR

Book [0]["title"]

How to Assign / modify value:

To assign value use syntax:
Array_name[index].attribute=value;

Example:
Book[0].title=”Advanced Web Technologies”;
Book[0][“title”]=” Advanced Web Technologies”;

JSON Multidimensional Array

We can store array inside JSON array, it is known as array of arrays or multidimensional array.

```
[
 [ "a", "b", "c" ],
 [ "m", "n", "o" ],
 [ "x", "y", "z" ]
]
```

It is possible to store an array inside another JSON array. This concept is known as an array of arrays or a multi-dimensional array.

Following example Create multidimensional array object

```
var books = { ".net" : [

{"title":".net complete", "price": 200},
{"title":".net black book", "price" : 300 }
],
"web" : [

{"title":"concepts of web technology ", "price": 150),
 {"title":"Advanced Web Technologies", "price": 130}]}
```

JSON Comments

JSON doesn't support comments. It is not a standard.

But you can do some tricks such as adding extra attribute for comment in JSON object, for example:

```
{
   "employee": {
      "name":     "Bob",
      "salary":     56000,
      "comments":   "He is a nice man"
   }
}
```

Here, "comments" attribute can be treated as comment.

JSON doesn't support comments by default. However, there are some tricks we can use, such as adding an extra attribute or key that work as comments in a JSON object. Consider following example. Here, key-value pair ("comments": "This is comment" serve as our comment.

```
{"book": {
"title": "Advanced Web Technologies",
"price": 200,
"comments": "This is comment"
}}
```

Consider following example. Here, attribute "comment_1" serve as comment.

```
{"book":{

"title": "Advanced Web Technologies",

"comment_1": "This is good book"
}}
```

JSON with javascript Example

Following example demonstrate creation of JSON object "book" using javascript.

```
<html>

<head>
```

```
<h3>Demo Example: JSON with JavaScript</h3>

<script>

var book = {"title":"Advanced Web Technologies", "author":"imp"};

document.write("<br>");

document.write("Title = " + book.title);

document.write("<br>");

document.write("Author = " + book.author);

</script>

</head>

<body>

</body>

</html>
```

Output:

Demo Example: JSON with JavaScript

Title Advanced Web Technologies
Author=jmp

JSON array with javascript Example:

Following example demonstrate creation of JSON object "book" and array "author

using javascript.

```
<html>

<head>

<h3>Demo Example: JSON array with JavaScript</h3>
```

```
<script>

var book = {"title":"Advanced Web Technologies",

"authors" : ["imp","mv"] };

document.write("<br>");

document.write("Title = " + book.title);

document.write("<br>");

document.write("Author name -1: " + book.authors[0]);

document.write("<br>");

document.write("Author name -2: " + book.authors[1]);

</script>

</head>

</body>

</html>
```

Output

Demo Example: JSON array with JavaScript

Title= Advanced Web Technologies

Author name -1: jmp
Author name -2: mv

EXAMPLES

JSON OBJECT

JavaScript Objects

You can create a JavaScript object from a JSON object literal:

Example

```
myObj = {"name":"John", "age":30, "car":null};
```

Normally, you create a JavaScript object by parsing a JSON string:

Example

```
myJSON = '{"name":"John", "age":30, "car":null}';
myObj = JSON.parse(myJSON);
```

Accessing Object Values

You can access object values by using dot (.) notation:

Example

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);
x = myObj.name;
```

You can also access object values by using bracket ([]) notation:

Example

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);
x = myObj["name"];
```

Looping an Object

You can loop through object properties with a for-in loop:

Example

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);

let text = "";
for (const x in myObj) {
  text += x + ", ";
}
```

In a for-in loop, use the bracket notation to access the property *values*:

Example

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);

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

JSON ARRAY

JavaScript Arrays

You can create a JavaScript array from a literal:

Example

```
myArray = ["Ford", "BMW", "Fiat"];
```

You can create a JavaScript array by parsing a JSON string:

Example

```
myJSON = '["Ford", "BMW", "Fiat"]';
myArray = JSON.Parse(myJSON);
```

Accessing Array Values

You access array values by index:

Example

```
myArray[0];
```

Arrays in Objects

Objects can contain arrays:

Example

```
{
"name":"John",
"age":30,
"cars":["Ford", "BMW", "Fiat"]
}
```

You access array values by index:

Example

```
myObj.cars[0];
```

Looping Through an Array

You can access array values by using a for in loop:

Example

```
for (let i in myObj.cars) {
  x += myObj.cars[i];
}
```

Or you can use a for loop:

Example

```
for (let i = 0; i < myObj.cars.length; i++) {
  x += myObj.cars[i];
}
```

JSON.parse()

A common use of JSON is to exchange data to/from a web server.

When receiving data from a web server, the data is always a string.

Parse the data with JSON.parse(), and the data becomes a JavaScript object.

Ajax

AJAX is an acronym for **Asynchronous JavaScript and XML**. It is a group of inter-related technologies like JavaScript, DOM, XML, HTML/XHTML, CSS, XMLHttpRequest etc.

AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.

AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster

AJAX is not a programming language.

AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Where it is used?

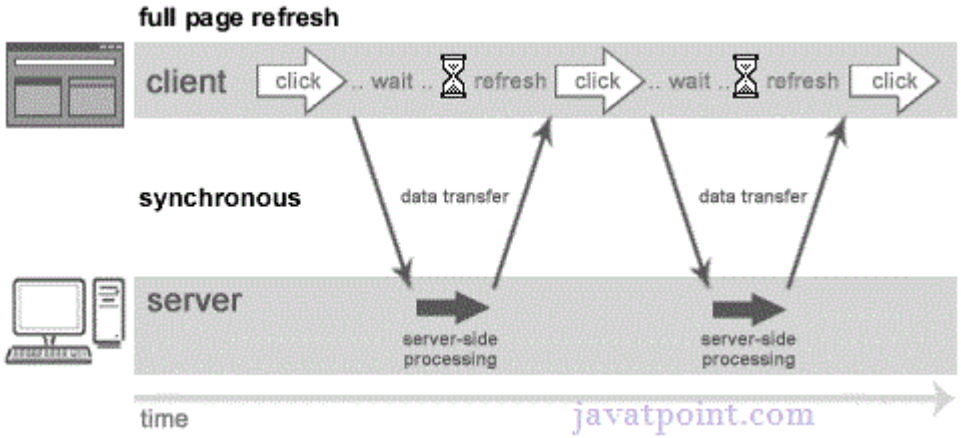
There are too many web applications running on the web that are using ajax technology like **gmail**, **facebook**,**twitter**, **google map**, **youtube** etc.

Understanding Synchronous vs Asynchronous

Before understanding AJAX, let's understand classic web application model and ajax web application model first.

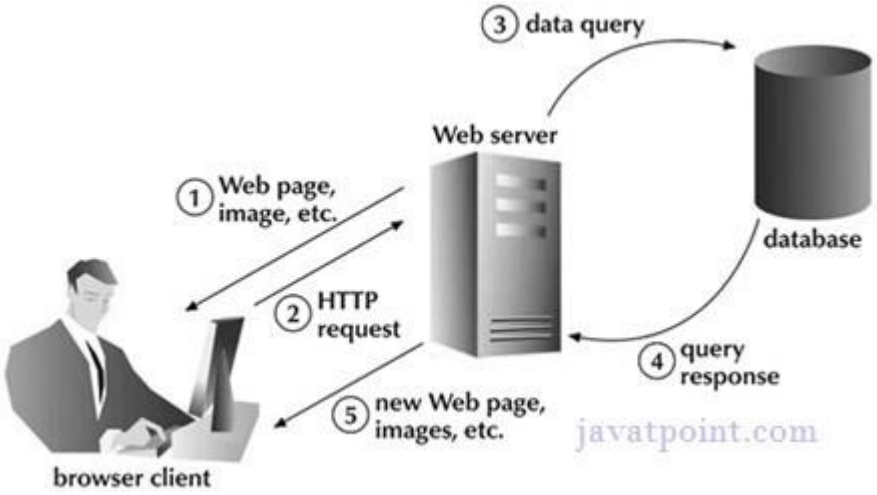
Synchronous (Classic Web-Application Model)

A synchronous request blocks the client until operation completes i.e. browser is unresponsive. In such case, javascript engine of the browser is blocked.



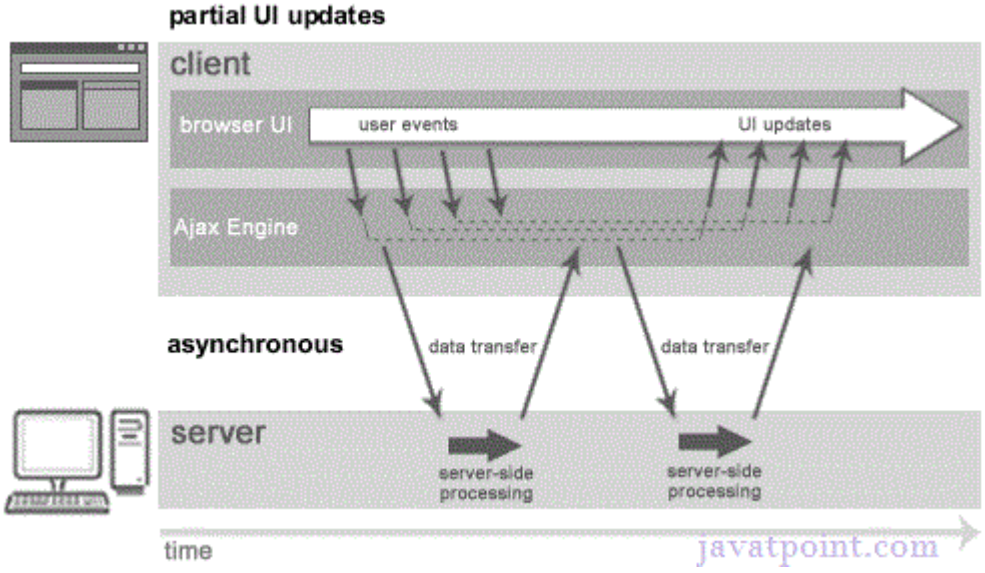
As you can see in the above image, full page is refreshed at request time and user is blocked until request completes.

Let's understand it another way.



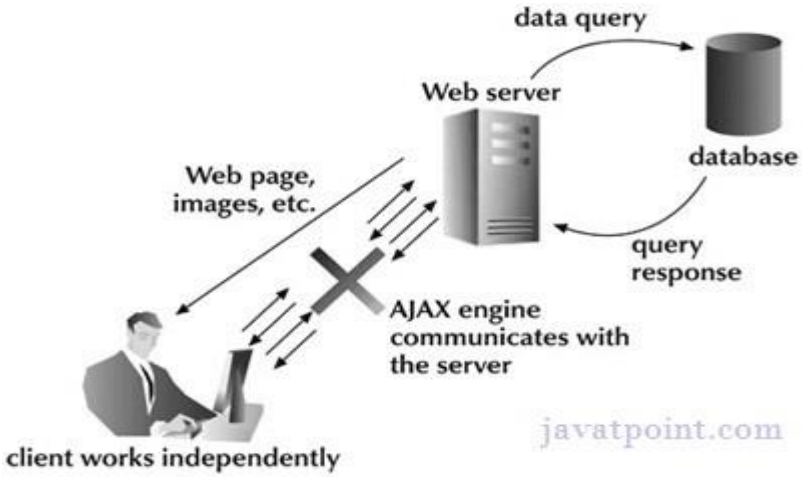
Asynchronous (AJAX Web-Application Model)

An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.



As you can see in the above image, full page is not refreshed at request time and user gets response from the ajax engine.

Let's try to understand asynchronous communication by the image given below.



Note: every blocking operation is not synchronous and every unblocking operation is not asynchronous.

AJAX Technologies

As describe earlier, ajax is not a technology but group of inter-related technologies. AJAX technologies include:

- HTML/XHTML and CSS
- DOM
- XML or JSON
- XMLHttpRequest
- JavaScript

HTML/XHTML and CSS

These technologies are used for displaying content and style. It is mainly used for presentation.

DOM

It is used for dynamic display and interaction with data.

XML or JSON

For carrying data to and from server. JSON (Javascript Object Notation) is like XML but short and faster than XML.

XMLHttpRequest

For asynchronous communication between client and server.

JavaScript

It is used to bring above technologies together.

Independently, it is used mainly for client-side validation.

Understanding XMLHttpRequest

An object of XMLHttpRequest is used for asynchronous communication between client and server.

It performs following operations:

1. Sends data from the client in the background
2. Receives the data from the server
3. Updates the webpage without reloading it.

Properties of XMLHttpRequest object

The common properties of XMLHttpRequest object are as follows:

Property	Description
onReadyStateChange	It is called whenever readystate attribute changes. It must not be used with synchronous requests.
readyState	represents the state of the request. It ranges from 0 to 4. **0** UNOPENED open() is not called. **1** OPENED open is called but send() is not called. **2** HEADERS_RECEIVED send() is called, and headers and status are available. **3** LOADING Downloading data; responseText holds the data. **4** DONE The operation is completed fully.
reponseText	returns response as text.
responseXML	returns response as XML

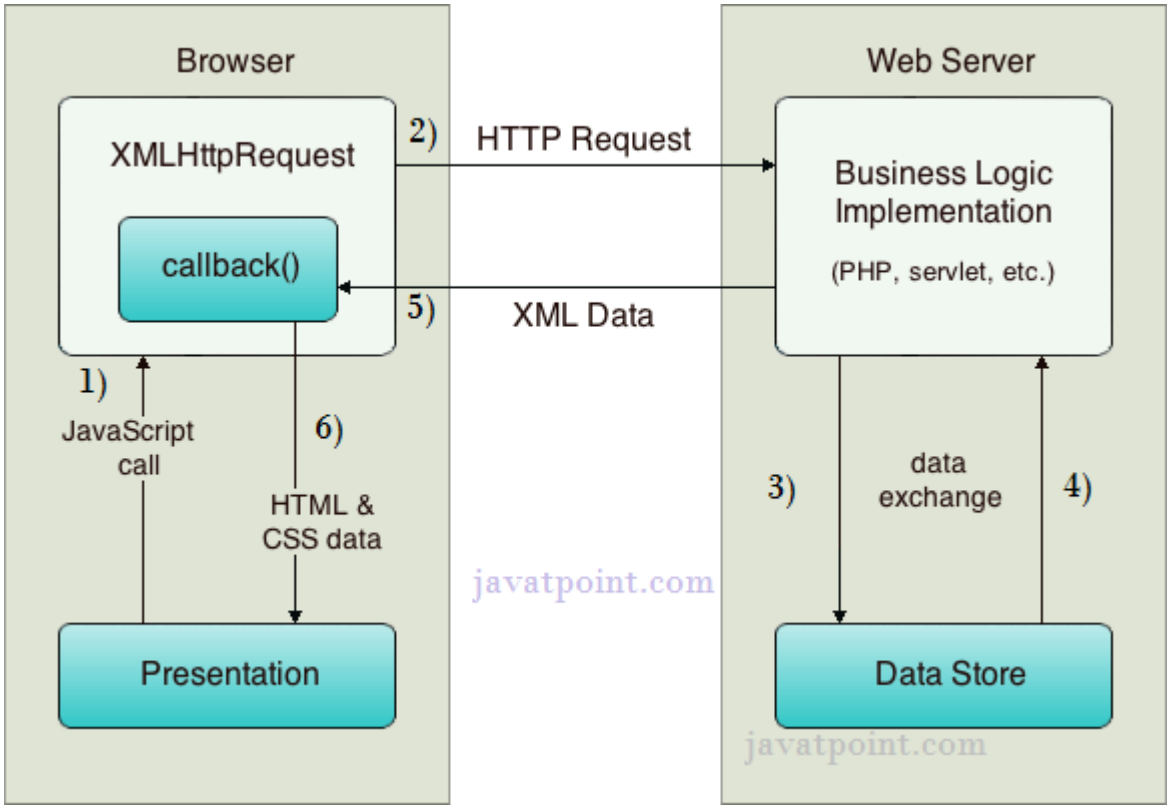
Methods of XMLHttpRequest object

The important methods of XMLHttpRequest object are as follows:

Method	Description
void open(method, URL)	opens the request specifying get or post method and url.
void open(method, URL, async)	same as above but specifies asynchronous or not.
void open(method, URL, async, username, password)	same as above but specifies username and password.
void send()	sends get request.
void send(string)	send post request.
setRequestHeader(header,value)	it adds request headers.

How AJAX works?

AJAX communicates with the server using XMLHttpRequest object. Let's try to understand the flow of ajax or how ajax works by the image displayed below.



As you can see in the above example, XMLHttpRequest object plays a important role.

1. User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.
3. Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
4. Data is retrieved.
5. Server sends XML data or JSON data to the XMLHttpRequest callback function.
6. HTML and CSS data is displayed on the browser.

AJAX is not a programming language.

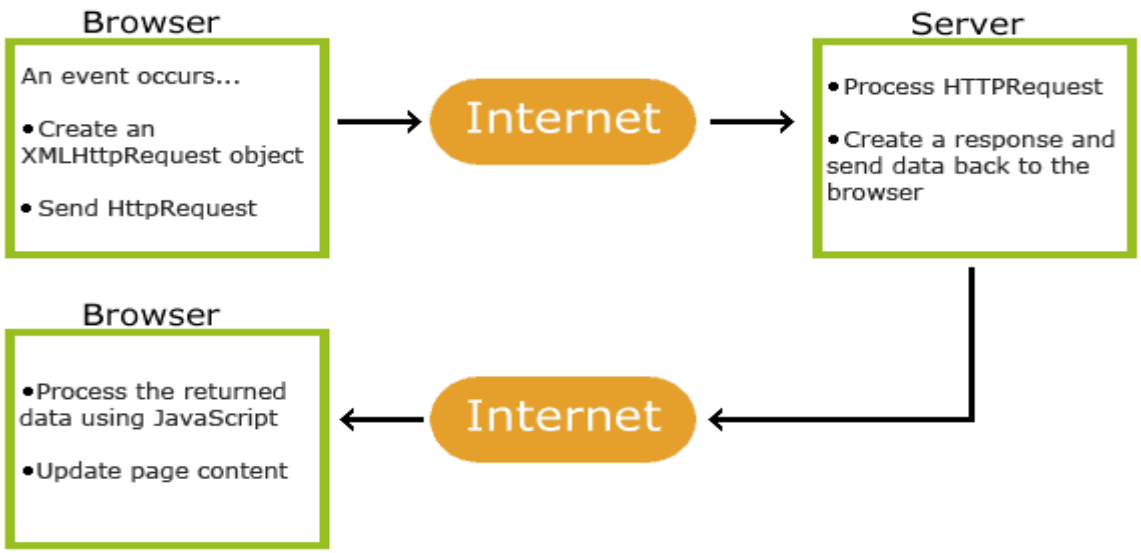
AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

How AJAX Works



- 1. An event occurs in a web page (the page is loaded, a button is clicked)
- 2. An XMLHttpRequest object is created by JavaScript
- 3. The XMLHttpRequest object sends a request to a web server
- 4. The server processes the request
- 5. The server sends a response back to the web page
- 6. The response is read by JavaScript
- 7. Proper action (like page update) is performed by JavaScript

Modern Browsers (Fetch API)

Modern Browsers can use Fetch API instead of the XMLHttpRequest Object.

The Fetch API interface allows web browser to make HTTP requests to web servers.

If you use the XMLHttpRequest Object, Fetch can do the same in a simpler way.

AJAX - The XMLHttpRequest Object

The keystone of AJAX is the XMLHttpRequest object.

1. Create an XMLHttpRequest object
2. Define a callback function
3. Open the XMLHttpRequest object
4. Send a Request to a server

The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

The XMLHttpRequest object can be used to exchange data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Create an XMLHttpRequest Object

All modern browsers (Chrome, Firefox, IE, Edge, Safari, Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

variable = new XMLHttpRequest();

Define a Callback Function

A callback function is a function passed as a parameter to another function.

In this case, the callback function should contain the code to execute when the response is ready.

```
xhttp.onload = function() {
  // What to do when the response is ready
}
```

Send a Request

To send a request to a server, you can use the open() and send() methods of the XMLHttpRequest object:

```
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

Example

```
// Create an XMLHttpRequest object
const xhttp = new XMLHttpRequest();
```

```
// Define a callback function
xhttp.onload = function() {
  // Here you can use the Data
}

// Send a request
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

Access Across Domains

For security reasons, modern browsers do not allow access across domains.

This means that both the web page and the XML file it tries to load, must be located on the same server.

The examples on W3Schools all open XML files located on the W3Schools domain.

If you want to use the example above on one of your own web pages, the XML files you load must be located on your own server.

XMLHttpRequest Object Methods

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information

open(*method, url, async, user, psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

XMLHttpRequest Object Properties

Property	**Description**
onload	Defines a function to be called when the request is recieved (loaded)
onreadystatechange	Defines a function to be called when the readyState property changes

readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The onload Property

With the XMLHttpRequest object you can define a callback function to be executed when the request receives an answer.

The function is defined in the onload property of the XMLHttpRequest object:

Example

```
xhttp.onload = function() {
  document.getElementById("demo").innerHTML = this.responseText;
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

Multiple Callback Functions

If you have more than one AJAX task in a website, you should create one function for executing the XMLHttpRequest object, and one callback function for each AJAX task.

The function call should contain the URL and what function to call when the response is ready.

Example

```
loadDoc("url-1", myFunction1);

loadDoc("url-2", myFunction2);

function loadDoc(url, cFunction) {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {cFunction(this);}
  xhttp.open("GET", url);
  xhttp.send();
}

function myFunction1(xhttp) {
  // action goes here
}
function myFunction2(xhttp) {
  // action goes here
}
```

The onreadystatechange Property

The readyState property holds the status of the XMLHttpRequest.

The onreadystatechange property defines a callback function to be executed when the readyState changes.

The status property and the statusText properties hold the status of the XMLHttpRequest object.

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready

status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The onreadystatechange function is called every time the readyState changes.

When readyState is 4 and status is 200, the response is ready:

Example

```
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
```

The onreadystatechange event is triggered four times (1-4), one time for each change in the readyState.

AJAX - XMLHttpRequest

The XMLHttpRequest object is used to request data from a server.

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Method	Description

open(*method, url, async*)	Specifies the type of request *method*: the type of request: GET or POST *url*: the server (file) location *async*: true (asynchronous) or false (synchronous)
send()	Sends the request to the server (used for GET)
send(*string*)	Sends the request to the server (used for POST)

The url - A File On a Server

The url parameter of the open() method, is an address to a file on a server:

```
xhttp.open("GET", "ajax_test.asp", true);
```

The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

Asynchronous - True or False?

Server requests should be sent asynchronously.

The async parameter of the open() method should be set to true:

```
xhttp.open("GET", "ajax_test.asp", true);
```

By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:

- execute other scripts while waiting for server response
- deal with the response after the response is ready

The default value for the async parameter is async = true.

You can safely remove the third parameter from your code.

Synchronous XMLHttpRequest (async = false) is not recommended because the JavaScript will stop executing until the server response is ready. If the server is busy or slow, the application will hang or stop.

GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

- A cached file is not an option (update a file or database on the server).
- Sending a large amount of data to the server (POST has no size limitations).
- Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

GET Requests

A simple GET request:

Example

```
xhttp.open("GET", "demo_get.asp");
xhttp.send();
```

In the example above, you may get a cached result. To avoid this, add a unique ID to the URL:

Example

```
xhttp.open("GET", "demo_get.asp?t=" + Math.random());
xhttp.send();
```

If you want to send information with the GET method, add the information to the URL:

Example

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford");
xhttp.send();
```

How the server uses the input and how the server responds to a request, is explained in a later chapter.

POST Requests

A simple POST request:

Example

```
xhttp.open("POST", "demo_post.asp");
xhttp.send();
```

To POST data like an HTML form, add an HTTP header with setRequestHeader(). Specify the data you want to send in the send() method:

Example

```
xhttp.open("POST", "ajax_test.asp");
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");
```

Method	Description
setRequestHeader(*header, value*)	Adds HTTP headers to the request *header*: specifies the header name *value*: specifies the header value

Synchronous Request

To execute a synchronous request, change the third parameter in the open() method to false:

```
xhttp.open("GET", "ajax_info.txt", false);
```

Sometimes async = false are used for quick testing. You will also find synchronous requests in older JavaScript code.

Since the code will wait for server completion, there is no need for an onreadystatechange function:

Example

```
xhttp.open("GET", "ajax_info.txt", false);
xhttp.send();
document.getElementById("demo").innerHTML = xhttp.responseText;
```

Synchronous XMLHttpRequest (async = false) is not recommended because the JavaScript will stop executing until the server response is ready. If the server is busy or slow, the application will hang or stop.

Modern developer tools are encouraged to warn about using synchronous requests and may throw an InvalidAccessError exception when it occurs.

AJAX - Server Response

Server Response Properties

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

The responseText Property

The responseText property returns the server response as a JavaScript string, and you can use it accordingly:

Example

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

The responseXML Property

The XMLHttpRequest object has an in-built XML parser.

The responseXML property returns the server response as an XML DOM object.

Using this property you can parse the response as an XML DOM object:

Example

Request the file cd_catalog.xml and parse the response:

```
const xmlDoc = xhttp.responseXML;
const x = xmlDoc.getElementsByTagName("ARTIST");

let txt = "";
for (let i = 0; i < x.length; i++) {
  txt += x[i].childNodes[0].nodeValue + "<br>";
}
document.getElementById("demo").innerHTML = txt;
```

```
xhttp.open("GET", "cd_catalog.xml");
xhttp.send();
```

Server Response Methods

Method	Description
getResponseHeader()	Returns specific header information from the server resource
getAllResponseHeaders()	Returns all the header information from the server resource

The getAllResponseHeaders() Method

The getAllResponseHeaders() method returns all header information from the server response.

Example

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  document.getElementById("demo").innerHTML =
  this.getAllResponseHeaders();
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

The getResponseHeader() Method

The getResponseHeader() method returns specific header information from the server response.

Example

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  document.getElementById("demo").innerHTML =
  this.getResponseHeader("Last-Modified");
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

AJAX XML Example

AJAX can be used for interactive communication with an XML file.

AJAX XML Example

The following example will demonstrate how a web page can fetch information from an XML file with AJAX:

Example

Get CD info

Example Explained

When a user clicks on the "Get CD info" button above, the loadDoc() function is executed.

The loadDoc() function creates an XMLHttpRequest object, adds the function to be executed when the server response is ready, and sends the request off to the server.

When the server response is ready, an HTML table is built, nodes (elements) are extracted from the XML file, and it finally updates the element "demo" with the HTML table filled with XML data:

```
function loadDoc() {
 const xhttp = new XMLHttpRequest();
 xhttp.onload = function() {myFunction(this);}
 xhttp.open("GET", "cd_catalog.xml");
 xhttp.send();
}
function myFunction(xml) {
 const xmlDoc = xml.responseXML;
 const x = xmlDoc.getElementsByTagName("CD");
 let table="<tr><th>Artist</th><th>Title</th></tr>";
 for (let i = 0; i <x.length; i++) {
  table += "<tr><td>" +
  x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "</td><td>" +
  x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "</td></tr>";
 }
 document.getElementById("demo").innerHTML = table;
}
```

The XML File

```
<CATALOG>
<CD>
<TITLE>Empire Burlesque</TITLE>
```

```
<ARTIST>Bob Dylan</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
<CD>
<TITLE>Greatest Hits</TITLE>
<ARTIST>Dolly Parton</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>RCA</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1982</YEAR>
</CD>
<CD>
<TITLE>Still got the blues</TITLE>
<ARTIST>Gary Moore</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Virgin records</COMPANY>
<PRICE>10.20</PRICE>
<YEAR>1990</YEAR>
</CD>
</CATALOG>
```

XML Applications

This chapter demonstrates some HTML applications using XML, HTTP, DOM, and JavaScript.

The XML Document Used

In this chapter we will use the XML file called "cd_catalog.xml".

Display XML Data in an HTML Table

This example loops through each <CD> element, and displays the values of the <ARTIST> and the <TITLE> elements in an HTML table:

Example

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

<script>
function loadXMLDoc() {
```

```
 const xhttp = new XMLHttpRequest();
 xhttp.onload = function() {
  const xmlDoc = xhttp.responseXML;
  const cd = xmlDoc.getElementsByTagName("CD");
  myFunction(cd);
 }
 xhttp.open("GET", "cd_catalog.xml");
 xhttp.send();
}

function myFunction(cd) {
 let table="<tr><th>Artist</th><th>Title</th></tr>";
 for (let i = 0; i < cd.length; i++) {
  table += "<tr><td>" +
  cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "</td><td>" +
  cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "</td></tr>";
 }
 document.getElementById("demo").innerHTML = table;
}
</script>

</body>
</html>
```

Display the First CD in an HTML div Element

This example uses a function to display the first CD element in an HTML element with id="showCD":

Example

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
 const xmlDoc = xhttp.responseXML;
 const cd = xmlDoc.getElementsByTagName("CD");
 myFunction(cd, 0);
}
xhttp.open("GET", "cd_catalog.xml");
xhttp.send();

function myFunction(cd, i) {
 document.getElementById("showCD").innerHTML =
 "Artist: " +
 cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
 "<br>Title: " +
 cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
 "<br>Year: " +
 cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```

Navigate Between the CDs

To navigate between the CDs in the example above, create a next() and previous() function:

Example

```
function next() {
  // display the next CD, unless you are on the last CD
  if (i < len-1) {
    i++;
    displayCD(i);
  }
}

function previous() {
  // display the previous CD, unless you are on the first CD
  if (i > 0) {
    i--;
    displayCD(i);
  }
}
```

Show Album Information When Clicking On a CD

The last example shows how you can show album information when the user clicks on a CD:

Example

```
function displayCD(i) {
  document.getElementById("showCD").innerHTML =
  "Artist: " +
  cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "<br>Title: " +
  cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "<br>Year: " +
  cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```

5. Node.js

Node.js is a cross-platform environment and library for running JavaScript applications, which is used to create networking and server-side applications.

Concept, Working, and Features

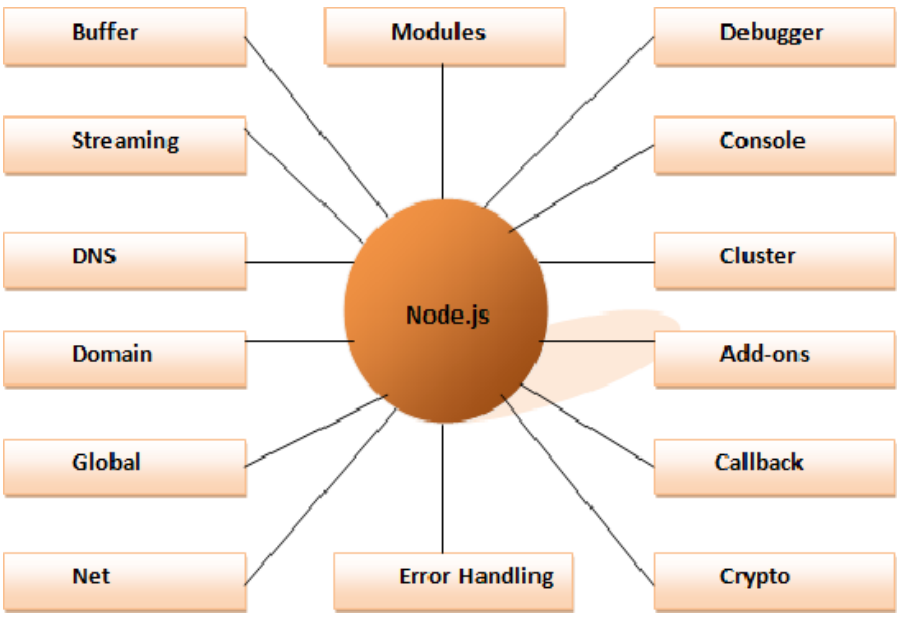
Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser. It is used for creating server-side and networking web applications. It is open-source and free to use. It can be downloaded from this link: https://nodejs.org/en/.

Many of the basic modules of Node.js are written in JavaScript. Node.js is mostly used to run real-time server applications.

The definition given by its official documentation is as follows:

“Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

Node.js also provides a rich library of various JavaScript modules to simplify the development of web applications.



Features of Node.js

Following is a list of some important features of Node.js that make it the first choice of software architects:

1. **Extremely Fast**: Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
2. **I/O is Asynchronous and Event Driven**: All APIs of the Node.js library are asynchronous (i.e., non-blocking). A Node.js-based server never waits for an API to return data. The server moves to the next API after calling it, and a notification mechanism (Events) in Node.js helps the server get a response from the previous API call. This is also why Node.js is very fast.
3. **Single-Threaded**: Node.js follows a single-threaded model with event looping.
4. **Highly Scalable**: Node.js is highly scalable because the event mechanism helps the server respond in a non-blocking way.
5. **No Buffering**: Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data; they simply output the data in chunks.
6. **Open Source**: Node.js has an open-source community that has produced many excellent modules to add additional capabilities to Node.js applications.
7. **License**: Node.js is released under the MIT license.

Downloading Node.js

To install and set up an environment for Node.js, you need the following two software programs available on your computer:

1. **Text Editor**
2. **Node.js Binary Installable**

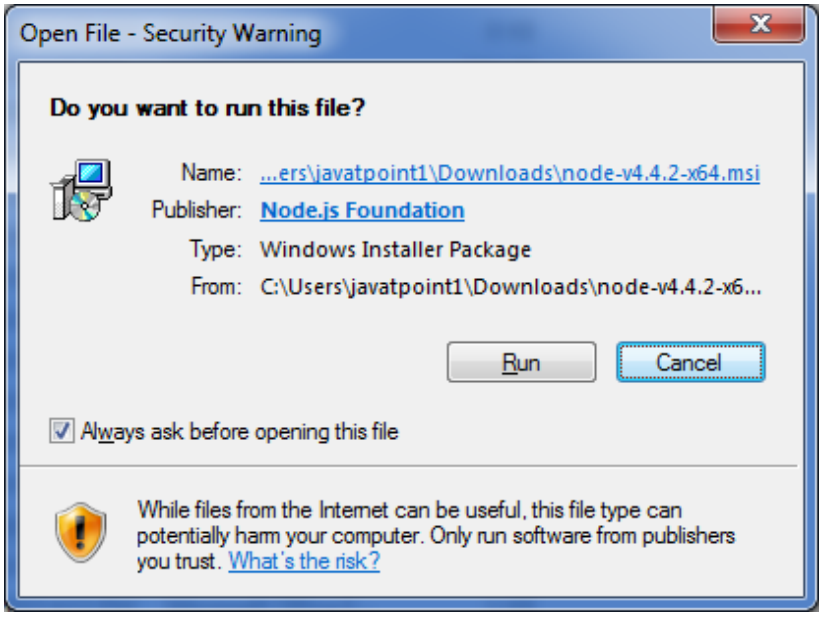
You can download the latest version of the Node.js installable archive file from https://nodejs.org/en/.

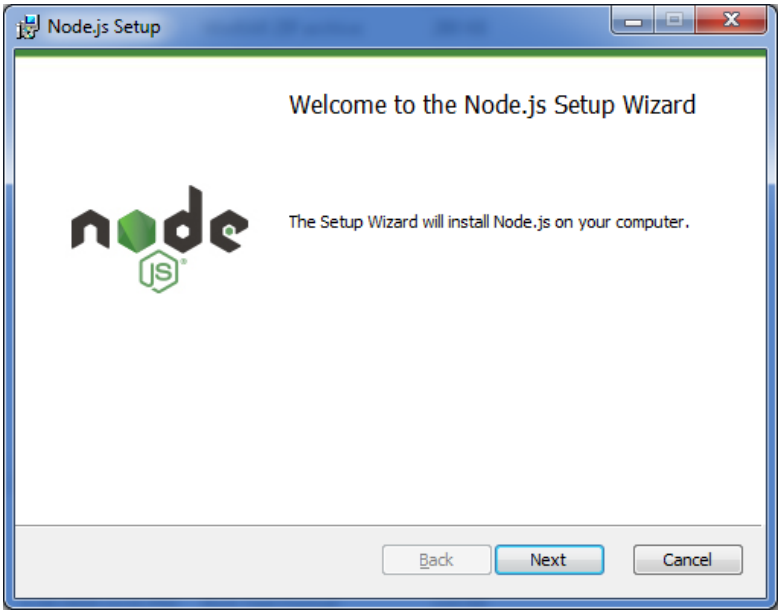
Setting up Node.js Server (HTTP Server)

Installing on Windows

There are various installers available on https://nodejs.org/en/. The installation files are available for Windows, Linux, Solaris, and macOS.

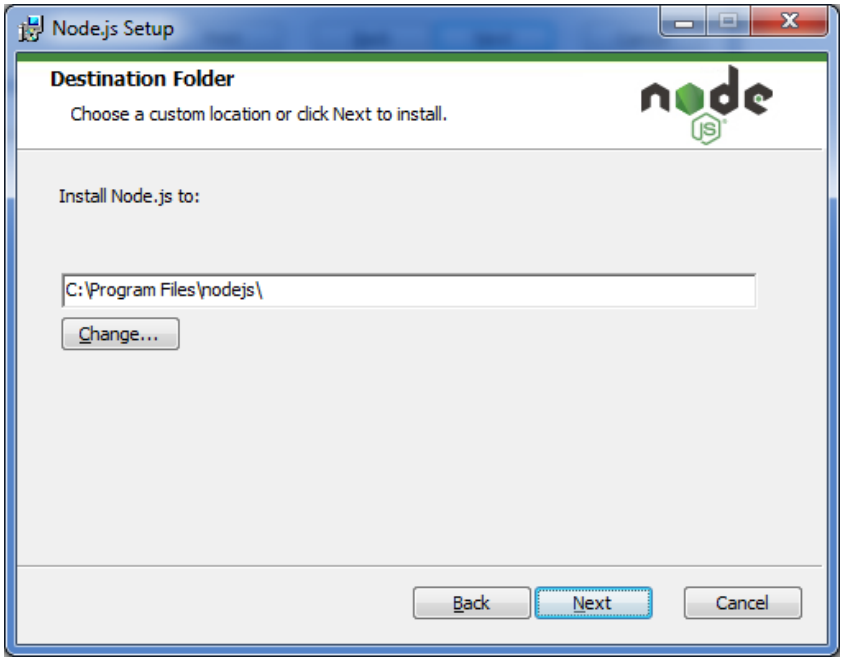
Download the installer for Windows by clicking on the **LTS** or **Current version** button. Here, we will install the latest version **LTS** for Windows that has long-time support. However, you can also install the **Current version**, which will have the latest features.
After you download the MSI file, double-click on it to start the installation, as shown below.

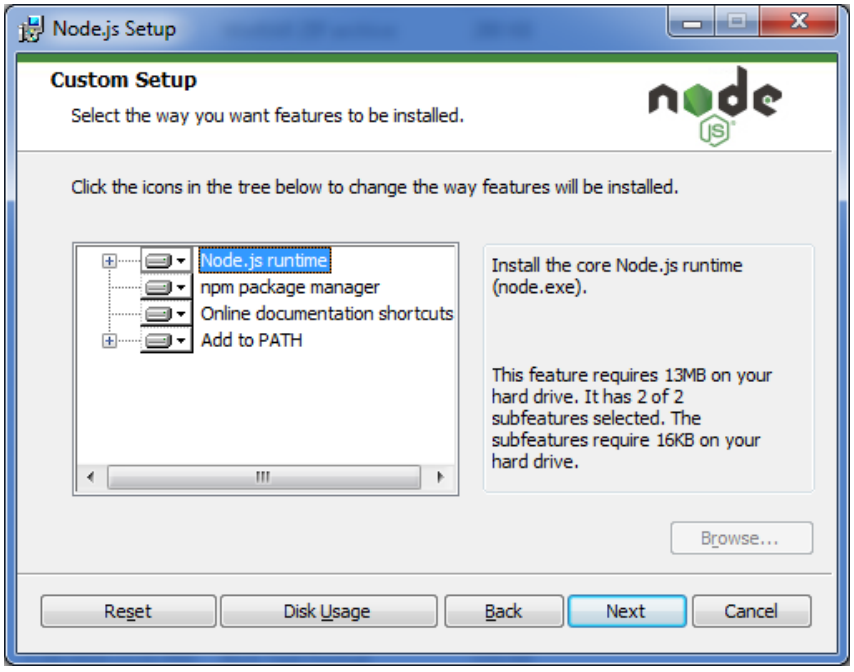




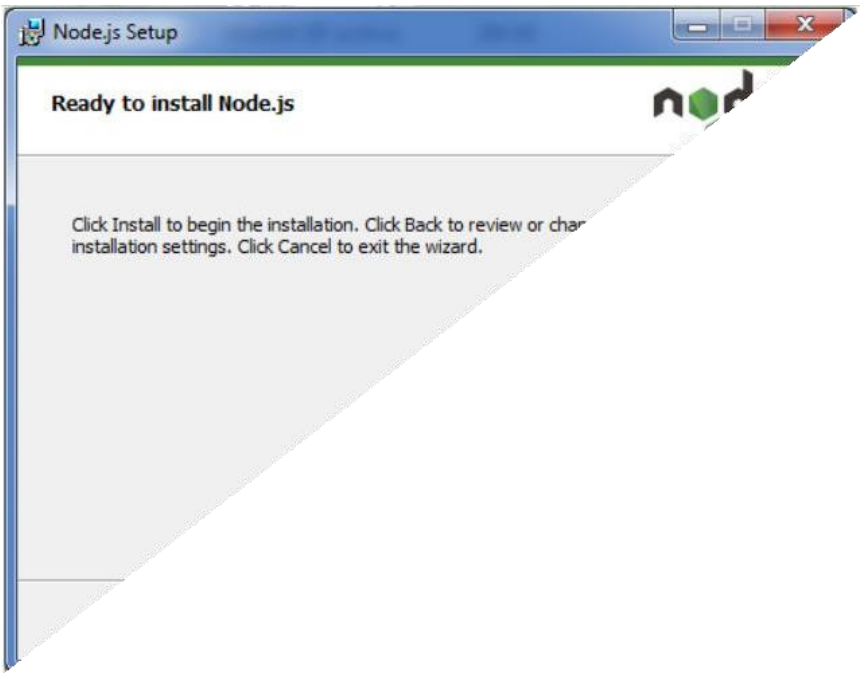
Accept the terms of license agreement.

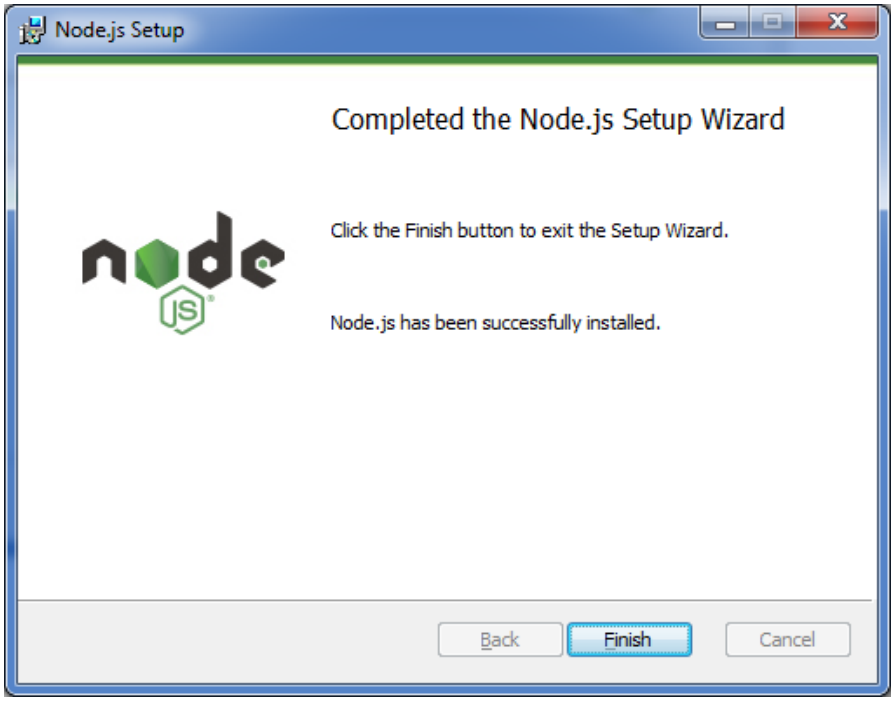
Choose the location where you want to install.





Ready to install:





Verify Installation

Once you install Node.js on your computer, you can verify it by opening the command prompt and typing node -v. If Node.js is installed successfully, it will display the version of Node.js installed on

your machine, as shown below

Components

A Node.js application consists of the following three important components:

1. **Import required modules**: The require directive is used to load a Node.js module.
2. **Create server**: You have to establish a server that will listen to client requests, similar to the Apache HTTP Server.
3. **Read request and return response**: The server created in the second step will read the HTTP request made by the client (which can be a browser or console) and return the response.

Required Modules, Create Server (http.createServer())

Require Modules:
The method require() is used to consume modules. It allows you to include modules in your app. You can add built-in core Node.js modules, community-based modules (node_modules), and local modules as well.
Node.js follows the CommonJS module system, and the built-in require function is the easiest way to include modules that exist in separate files. The basic functionality of require() is that it reads a JavaScript file, executes the file, and then proceeds to return the exports object.

```
var module = require('module_name');
```

As per the above syntax, specify the module name in the require() function. The require() function will return an object, function, property, or any other JavaScript type, depending on what the specified module returns.

We use the require directive to load the http module and store the returned HTTP instance into an http variable as follows:

```
var http = require("http");
```

In the above example, the require() function returns an object because the http module returns its functionality as an object. You can then use its properties and methods using dot notation, e.g., http.createServer().

Create Server

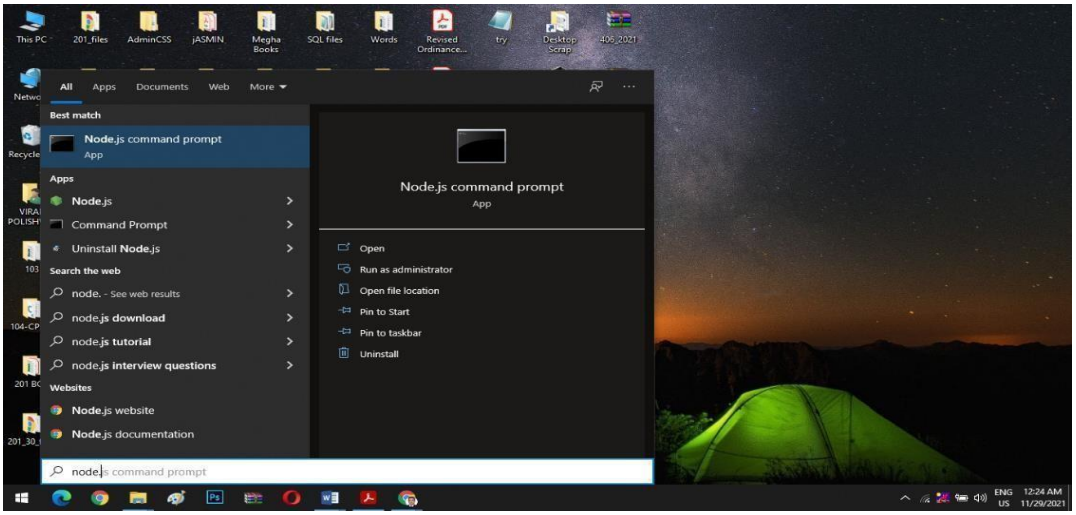
In the second component, you have to use the created http instance and call the http.createServer() method to create the server instance. Then, bind it to port **8081** using the listen method associated with the server instance. Pass it a function with request and response parameters and write the sample implementation to return "Hello World." Check the example below:

Example: Block 2

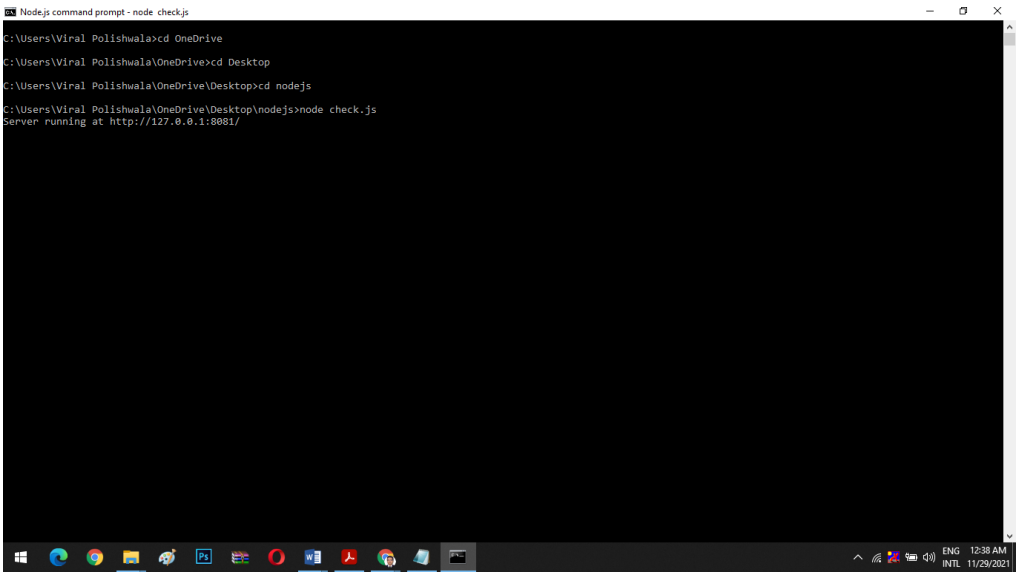
```
http.createServer(function(request, response) {
  // Send the HTTP header
  // HTTP Status: 200 OK
  // Content-Type: text/plain
  response.writeHead(200, {'Content-Type': 'text/plain'});

  // Send the response body as "Hello World"
  response.end('Hello World\n');
}).listen(8081);

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

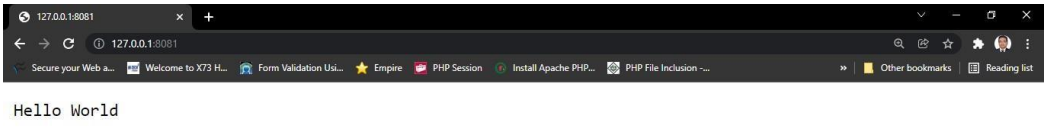
Request and Response

Now, combine Block 1 and Block 2, and save the file as **check.js**. When executing the above code, it will create an HTTP server that listens on port **8081** on the local machine.


All
Apps
Documents
Web
More
Best match
Node.js command prompt
App
Apps
Node.js
Command Prompt
Uninstall Node.js
Search the web
node. - See web results
node.js download
node.js tutorial
node.js interview questions
Websites
Node.js website
Node.js documentation
node.js command prompt
Node.js command prompt
App
Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
Uninstall

For starting the server, we need to start the command prompt as follows: Click the **Start** menu and type Node.js Command Prompt. Once the command prompt is open, just move to the location where the **check.js** file is stored.





Once you get the message like "**Server is running at...**", we can just open any browser and check for **http://127.0.0.1:8081/**.
Now, if you make any changes in the "**check.js**" file, you need to again run the "**node check.js**" command.

This is how the HTTP server is created and requests which are sent over this server are responded to from port **8081**.

Built-In Modules

In Node.js, **modules** are blocks of encapsulated code that communicate with an external application based on their related functionality. Modules can be a single file or a collection of multiple files/folders. The reason programmers are heavily reliant on modules is because of their reusability as well as the ability to break down a complex piece of code into manageable chunks.

Node.js has many built-in modules that are part of the platform and come with the Node.js installation. **Node.js Core Modules** come with its installation by default. You can use them as per application requirements. These modules can be loaded into the program by using the require function.

require() Function

The require() function will return a JavaScript type depending on what the particular module returns.

javascript
Copy code

```
var module = require('module_name');
```

The following example demonstrates how to use the Node.js **http** module to create a web server:

javascript
Copy code

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('Welcome to this page!');
  res.end();
}).listen(3000);
```

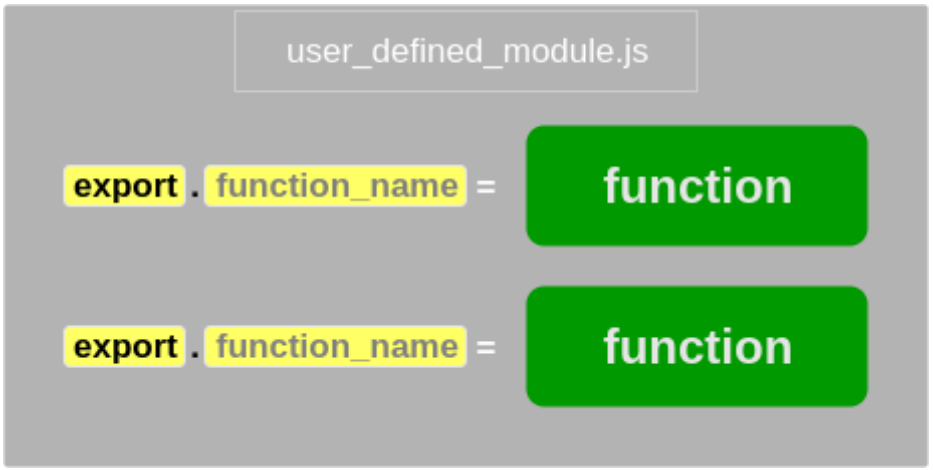
In the above example, the require() function returns an object because the **http** module returns its functionality as an object. The http.createServer() method will be executed when someone tries to access the computer on port **3000**. The res.writeHead() method is the status code where **200** means it is OK, while the second argument is an object containing the response headers.

Module	**Description**
OS Module	Provides basic operating-system related utility functions.
	varos= require("os")
Path Module	Provides utilities for handling and transforming file paths. varpath= require("path")
Net Module	Provides both servers and clients as streams. Acts as a network wrapper. varnet =require("net")
DNS Module	Provides functions to do actual DNS lookup as well as to use underlying operating system name resolution functionalities. vardns=require("dns")

User Defined Modules: Create and Include

Modules are a collection of JavaScript code in a separate logical file that can be used in external applications based on their related functionality.

Modules are popular because they are easy to use and are reusable. Sometimes, when you are implementing a Node.js application for a use case, you might want to keep your business logic separately. In such cases, you create a Node.js module with all the required functions in it.



Create Modules (exports)

The module.exports is a special object that is included in every JavaScript file in the Node.js application by default. The module is a variable that represents the current module

Create Modules (exports)

The module.exports is a special object that is included in every JavaScript file in the Node.js application by default. The module is a variable that represents the current module, and exports is an object that will be exposed as a module. So, whatever you assign to module.exports will be exposed as a module.

To create a module in Node.js, the exports keyword is used. This keyword tells Node.js that the function can be used outside the module. A Node.js module is a .js file with one or more functions.

The syntax to define a function in a Node.js module is:

```
exports.<function_name> = function(argument_1, argument_2, ..., argument_N) {
  /** function body */
};
```

- **exports** – is a keyword which tells Node.js that the function is available outside the module.
- **function_name** – is the function name that will be used to access this function in other programs.

As mentioned above, exports is an object. So, it exposes whatever you assign to it as a module. For example, if you assign a string literal, then it will expose that string literal as a module.

The following example exposes a simple string message as a module in **Message.js**:

```
module.exports = 'Hello world';
```

Include Modules (require)

Node.js follows the CommonJS module system, and the built-in require function is the easiest way to include modules that exist in separate files. The basic functionality of require() is that it reads a JavaScript file, executes the file, and then proceeds to return the exports object.

Here’s an example module:

Now, import this message module and use it as shown below:

```
var msg = require('./Messages.js');
console.log(msg);
```

Run the above example and see the result, as shown below:

```
C:\> node app.js
Hello World
```

Export Object

The exports is an object, so you can attach properties or methods to it. The following example exposes an object with a string property in the **Message.js** file:

```
exports.SimpleMessage = 'Hello world';
// or
module.exports.SimpleMessage = 'Hello world';
```

In the above example, we have attached a property **SimpleMessage** to the exports object.

Now, import and use this module, as shown below in **app.js**:

```
var msg = require('./Messages.js');
console.log(msg.SimpleMessage);
```

In the above example, the require() function will return an object {SimpleMessage: 'Hello World'} and assign it to the msg variable. So, now you can use msg.SimpleMessage.

Run the above example by writing node app.js in the command prompt and see the output, as shown below:

```
C:\> node app.js
Hello World
```

In the same way as above, you can expose an object with a function. The following example exposes an object with the log function as a module: **Log.js**:

```
module.exports.log = function(msg) {
  console.log(msg);
};
```

The above module will expose an object {log: function(msg) {console.log(msg);}}. Use the above module as shown below in **app.js**:

```
var msg = require('./Log.js');
msg.log('Hello World');
```

Run and see the output in the command prompt, as shown below:

```
C:\> node app.js
Hello World
```

You can also attach an object to module.exports, as shown below:
data.js:

```
module.exports = {
  // object properties and methods here
};
```

data.js

```
firstName: 'James',
lastName: 'Bond'
}
```

app.js

```
var person = require('./data.js');
console.log(person.firstName + ' ' + person.lastName);
```

```
C:\> node app.js
James Bond
```

HTTP Module

To make HTTP requests in Node.js, there is a built-in module **HTTP** in Node.js to transfer data over HTTP. To use the HTTP server in Node, we need to require the **HTTP** module. The HTTP module creates an HTTP server that listens to server ports and gives a response back to the

client. The HTTP core module is a key module to Node.js networking. It is designed to support many features of the HTTP protocol.

We can create an HTTP server with the help of the http.createServer() method. (Ex: **testing.js**)

```
var http = require('http');

// Create a server
http.createServer((request, response) => {
  // Sends a chunk of the response body
  response.write('Hello World!');

  // Signals the server that all of the response headers and body have been sent
  response.end();
})
.listen(3000); // Server listening on port 3000
```

The above code will start the web server by running the command:

```
node testing.js
var http = require('http');
var options = {
  host: 'www.amrolicollege.org',
  path: '/sports2122',
};
```

```
method: 'GET'

};
```

Making a GET request to 'www.amrolicollege.org'

```
http.request(options, (response) => {
  // Printing the statusCode
  console.log(`STATUS: ${response.statusCode}`);
}).end();
```

The HTTP Module

The **HTTP** module provides some properties, methods, and classes.

http.METHODS

This property lists all the HTTP methods supported:

```
require('http').METHODS
[
  'ACL', 'BIND', 'CHECKOUT', 'CONNECT', 'COPY', 'DELETE', 'GET',
'HEAD',
  'LINK', 'LOCK', 'M-SEARCH', 'MERGE', 'MKACTIVITY',
'MKCALENDAR', 'MKCOL',
  'MOVE', 'NOTIFY', 'OPTIONS', 'PATCH', 'POST', 'PROPFIND',
'PROPPATCH', 'PURGE',
  'PUT', 'REBIND', 'REPORT', 'SEARCH', 'SUBSCRIBE', 'TRACE',
'UNBIND', 'UNLINK',
  'UNLOCK', 'UNSUBSCRIBE'
]
```

http.STATUS_CODES

This property lists all the HTTP status codes and their descriptions:

```
require('http').STATUS_CODES
```

```
{
  '100': 'Continue',
  '101': 'Switching Protocols',
  '102': 'Processing',
  '103': 'Early Hints',
  '200': 'OK',
  '201': 'Created',
  '202': 'Accepted',
  '203': 'Non-Authoritative Information',
}
```

'204': 'No Content',
'205': 'Reset Content',
'206': 'Partial Content',
'207': 'Multi-Status',
'208': 'Already Reported',
'226': 'IM Used',
'300': 'Multiple Choices',
'301': 'Moved Permanently',
'302': 'Found',
'303': 'See Other',
'304': 'Not Modified',
'305': 'Use Proxy',
'307': 'Temporary Redirect',
'308': 'Permanent Redirect',
'400': 'Bad Request',
'401': 'Unauthorized',
'402': 'Payment Required',
'403': 'Forbidden',
'404': 'Not Found',
'405': 'Method Not Allowed',
'406': 'Not Acceptable',
'407': 'Proxy Authentication Required',
'408': 'Request Timeout',
'409': 'Conflict',
'410': 'Gone',
'411': 'Length Required',
'412': 'Precondition Failed',
'413': 'Payload Too Large',
'414': 'URI Too Long',
'415': 'Unsupported Media Type',
'416': 'Range Not Satisfiable',
'417': 'Expectation Failed',
'418': "I'm a Teapot",
'421': 'Misdirected Request',
'422': 'Unprocessable Entity',
'423': 'Locked',

'424': 'Failed Dependency',
'425': 'Too Early',
'426': 'Upgrade Required',
'428': 'Precondition Required',
'429': 'Too Many Requests',
'431': 'Request Header Fields Too Large',
'451': 'Unavailable For Legal Reasons',
'500': 'Internal Server Error',
'501': 'Not Implemented',
'502': 'Bad Gateway',
'503': 'Service Unavailable',
'504': 'Gateway Timeout',
'505': 'HTTP Version Not Supported',
'506': 'Variant Also Negotiates',
'507': 'Insufficient Storage',
'508': 'Loop Detected',
'509': 'Bandwidth Limit Exceeded',
'510': 'Not Extended',
'511': 'Network Authentication Required'

---

http.globalAgent

The http.globalAgent is a global object of the http.Agent class, which is utilized for all HTTP client requests by default. It is used to control connection persistence and reuse for HTTP clients. Moreover, it is a significant constituent in Node.js HTTP networking.

Node.js as a Web Server

When you view a webpage in your browser, you are making a request to another computer on the internet, which then provides you with the webpage as a response. That computer you are talking to via the internet is a web server. A web server receives HTTP requests from a client, like your browser, and provides an HTTP response, such as an HTML page or JSON from an API.

To access the webpages of any web application, you need a web server. The web server will handle all the HTTP requests for the web application. For example, IIS is a web server for ASP.NET web applications, and Apache is a web server for PHP or Java web applications.

A lot of software is involved for a server to return a webpage. This software generally falls into two categories: frontend and backend. Frontend code is concerned with how the content is presented, such as the color of a navigation bar and the text styling. Backend code is concerned with how data is exchanged, processed, and stored. Code that handles network requests from your browser or communicates with the database is primarily managed by backend code.

Node.js allows developers to use JavaScript to write backend code, even though traditionally it was used in the browser to write frontend code. Having both frontend and backend together like this reduces the effort it takes to make a web server, which is a major reason why Node.js is a popular choice for writing backend code.

There are a variety of modules, such as the http and request modules, which help in processing server-related requests in the web server space. We will have a look at how we can create a basic web server application using Node.js.

Node.js provides capabilities to create your own web server, which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web applications, but it is recommended to use the Node.js web server.

createServer(), writeHead() Method

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP). The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

Refer to the code below to understand the concept. Filename: helloserver.js

```
var http = require('http');
var server = http.createServer((function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.end("Hello World\n");
}));

server.listen(8081);
```

1. The basic functionality of the require() function is that it reads a JavaScript file, executes the file, and then proceeds to return the exports object. So, in our case, since we want to use the functionality of the http module, we use the require() function to get the desired functions from the http module so that they can be used in our application.
2. In this line of code, we are creating a server application which is based on a simple function. This function is called whenever a request is made to our server application. The request object can be used to get information about the current HTTP request (e.g., URL, request header, and data). The response object can be used to send a response for the current HTTP request.
3. When a request is received, we are saying to send a response with a header type of 200. This number is the normal response that is sent in an HTTP header when a successful response is sent to the client.
4. In the response itself, we are sending the string Hello World.
5. We are then using the server.listen() function to make our server application listen to client requests on port number 8081. You can specify any available port here.

Run the above web server by writing node helloserver.js in the command prompt or terminal window, and it will display the message that Node.js is running on port number 8081.

Reading Query String, Split Query String

In Node.js, functionality to aid in accessing URL query string parameters is built into the standard library. The built-in url.parse() method takes care of most of the heavy lifting for us. Here's an example script using this handy function and an explanation of how it works:

```
const http = require('http');
const url = require('url');

http.createServer(function(req, res) {
  const queryObject = url.parse(req.url, true).query;
  console.log(queryObject);
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Feel free to add query parameters to the end of the URL');
}).listen(8080);
```

To test this code, run node app.js (assuming app.js is the name of the file) in the terminal, and then go to your browser and type http://localhost:8080/app.js?foo=bad&baz=foo in the URL bar.

The key part of this whole script is this line:

```
const queryObject = url.parse(req.url, true).query;
```

Let's take a look at this from the inside out. First, req.url will look like /app.js?foo=bad&baz=foo. This is the part that appears in the URL bar of the browser. Next, it gets passed to url.parse(), which parses out the various elements of the URL (NOTE: the second parameter is a boolean stating whether the method should parse the query string, so we set it to true). Finally, we access the .query property, which returns us a nice, friendly JavaScript object with our query string data.

The url.parse() method returns an object, which has many key-value pairs, one of which is the query object. Some other handy information returned by the method includes host, pathname, and search keys.

In the above code:

- url.parse(req.url, true).query returns { foo: 'bad', baz: 'foo' }

- url.parse(req.url, true).host returns 'localhost:8080'
- url.parse(req.url, true).pathname returns /app.js
- url.parse(req.url, true).search returns '?foo=bad&baz=foo'

Parsing with Query String

Another way to access query string parameters is by parsing them using the querystring built-in Node.js module. However, this method must be passed just the query string portion of a URL. Passing it the whole URL, like you did in the url.parse() example, won't parse the query strings.

```
const querystring = require('querystring');
const url =
"http://example.com/index.html?code=string&key=12&id=false";
const qs = "code=string&key=12&id=false";

console.log(querystring.parse(qs));
//>{code: 'string', key: '12', id: 'false'}

console.log(querystring.parse(url));
//>{'http://example.com/index.html?code': 'string', key: '12', id: 'false'}
```

File System Module

This module, as the name suggests, enables developers to work with the file system on their computers. The require() method is used to include the File System module. Some of the most common uses of this module are to create, read, update, delete, rename, or read files. For example, the function fs.readFile() is used to read files on the computer.

Read Files (readFile())

The Node.js filesystem module (fs module) allows you to work with files. The fs.readFile() method is used to read from a file.

There are two ways to read a file:

1. **Read File Asynchronously**:

```
var fs = require('fs');
fs.readFile('my_file.txt', (err, data) => {
  if (err) {
    throw err;
  } else {
    console.log("Content of file is: " + data);
  }
});
```

2. **Read File Synchronously**:

```
var fs = require('fs');
var filename = 'my_file.txt';
var content = fs.readFileSync(filename);
console.log('Content of file is: ' + content);
```

Create Files (appendFile(), open(), writeFile())

There are a bunch of methods used for creating new files: fs.appendFile(), fs.open(), fs.writeFile().

- The appendFile() method is used to append to a file. If the file doesn't exist, a new file will be created. The syntax of fs.appendFile() looks like this:

```
fs.appendFile('newfile.txt', 'Hello Konfinity!', function(err) {
  if (err) throw err;
  console.log('Done!');
});
```

- The next method is fs.open(). This method takes a "flag" as the second argument. If the flag is "w", the specified file is opened for writing. If the file doesn't exist, it will create an empty file. The syntax of fs.open() looks like this:

```
fs.open('newfile2.txt', 'w', function(err, file) {
  if (err) throw err;
  console.log('Done!');
});
```

- If you want to replace a particular file and its contents, the fs.writeFile() method is used. If the file doesn't already exist, a new one will be created.

```
fs.writeFile('newfile3.txt', 'Hello Konfinity!', function(err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Update Files (appendFile(), writeFile())

The methods generally used to update files in a system are fs.appendFile() and fs.writeFile().

- fs.appendFile() is a method that appends particular content at the end of the file. The syntax for the method fs.appendFile() looks like this:

```
fs.appendFile('newfile1.txt', 'This is my text.', function(err) {
  if (err) throw err;
  console.log('Updated!');
});
```

This code would append the text "This is my text." to the end of the file named "newfile1.txt".

- Another method is fs.writeFile(), which replaces the file and its content as mentioned in the code. The syntax of this method is:

```
fs.writeFile('mynewfile3.txt', 'This is my text', function(err) {
  if (err) throw err;
  console.log('Replaced!');
});
```

This code above replaces the content of the file "newfile3.txt".

Delete Files (unlink())

The fs.unlink() method is used to delete a particular file with the File System module. The syntax of this method looks like this:

```
fs.unlink('newfile2.txt', function(err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

This code basically deletes the file "newfile2.txt".

Rename Files (rename())

In Node.js, developers can also upload files to their computer. Apart from this, the file system module can also be used to rename files. The fs.rename() method is used to rename a file. This method renames the file mentioned in the code. The syntax for the same looks like this:

```
fs.rename('vbp.txt', 'vbp1.txt', function(err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

The code above renames the file "vbp.txt" to "vbp1.txt".

Also, keep in mind that all the methods mentioned above are prefixed with a variable fs. In these examples, we took the variable to be fs, and while executing your code, you would have to explicitly mention it in your code. The code will look something like this:

```
var fs = require('fs');
```