Chapter – 1

Introduction to XML

Before we continue with the XML, we have to revise certain fact about the terms like HTML and JavaScript.

HTML

It stands for **HyperText Markup Language**. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A **markup language** is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

HTML is a markup language used by the browser to manipulate text, images, and other content, in order to display it in the required format. HTML was created by Tim Berners-Lee in 1991. The first-ever version of HTML was HTML 1.0, but the first standard version was HTML 2.0, published in 1999.

Elements, **tags**, and **attributes** are basic concepts in HTML.

HTML **element** is a main structural unit of a web page. HTML tags are used to define HTML elements, and attributes provide additional information about these elements.

HTML **tags** are used to structure website content (text, hyperlinks, images, media, etc). Tags are not displayed in the browsers, they only "instruct" browsers how to show the content of the web page. There are over 100 tags in HTML, and can be used for styling the web content like text, images, videos, audios etc. HTML tags are written in angle brackets (e.g <html>).

HTML **attributes** are added to an HTML element to provide additional information about it. For example, to define an image with <img/> tag, we can use attributes like: src, height, width attributes to provide information about its source, height, width correspondingly.

JavaScript

JavaScript is a **scripting or a programming language**, allowing developers to perform complex features on web pages. Initially, this language was created for making web pages alive. In JavaScript, the programs are called scripts. One can write them in the HTML of a web page, then it will automatically run once the page loads. At this point, JavaScript is completely different from another language, called Java.

Initially, when JavaScript was created, it was called "**LiveScript**". Then, as Java was extremely popular in that period, then it was decided to call it JavaScript to position it relative to Java. But, over the years, JavaScript has transformed into a completely independent language, with its specification, known as **ECMAscript**, having no relation to Java. Modern JavaScript can both work in the browser and on the server. Basically, it can run on any device that has a specific program known as the JavaScript engine.

What In-browser JavaScript Does

Modern JavaScript is considered a safe programming language. It never provides low-level access to the memory or CPU as it was made for the browsers that don't require it. The capabilities of this language highly rely on the environment it runs in. For example, Node.js includes functions allowing JavaScript to write and read arbitrary files, implement network requests, and so on. In-browser JavaScript does anything related to web page manipulation.

For example, with in-browser JavaScript, you can do the following:

1. Adding new HTML to the page, changing the content, modifying the styles.
2. Reacting to use actions, running on mouse clicks, key presses, and more.
3. Sending requests over the network to remote servers.
4. Getting and setting cookies, asking questions to visitors, sending messages.
5. Remembering the data on the client-side.

Limitations of In-browser JavaScript

The capabilities of JavaScript are limited for the purpose of keeping the user's safety. With it, an evil web page can't access private information or harm the user's data.

Here are some examples of such restrictions:

- JavaScript has no direct access to OS functions. It can't read and write arbitrary files on the hard disc, copy or execute them.
- Different windows/tabs don't recognize each other. JavaSCript from one page is not able to access the other one, in case they are from different sites. It's known as "Same Origin Policy".
- JavaScript allows communication over the net to the server from where the page comes from. But, its capability of receiving data from the other site is prohibited. That's a safety limitation.

What Makes JavaScript Specific

There are at least three perfect things about JavaScript:

1. It supports complete integration with CSS and HTML.
2. It provides straightforward ways of doing simple things.
3. It is supported by almost all the major browsers and is performed by default.

It is the exclusive browser technology that encompasses the three great things above.

In modern programming, JavaScript is the most widespread tool that helps to create browser interfaces. It also allows creating mobile applications, servers, and much more. Our JavaScript book consists of several sections that cover all the information you need to learn this unique programming language. Each of the chapters includes both the theory and practical cases to make it easier for beginners to grasp the language.

XML (eXtensible Markup Language)

XML is an independent tool for storing and transporting data other than the hardware and the software. It doesn't depend on the platform and the software (programming language). You can write a program in any language on any platform (Operating System) to send, receive or store data using XML. XML is a markup language for documents containing structured information.

Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays (for example, content in a section heading has a different –meaning from content in a footnote, which means something different than content in a figure caption or content in a database table, etc.). Almost all documents have some structure.

A markup language is a mechanism to identify structures in a document. The XML specification defines a standard way to add markup to documents.

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.

1.1 Characteristics and Use of XML

XML was designed for describing well-formed data. Also XML have some strict rules. That rules follow every XML document. Following are some characteristics of the XML.

Characteristics of XML

- **XML is extensible**

Unlike HTML, XML applications allows to have user defined tags in structured manner. User is able to add or remove the tags in the defined structure. Older one with predefined structure as well updated one, both will work perfectly.

example1.1	example1.1(updated)
<?xml version="1.0" encoding="utf-8"?> <bca> <subject>405 : Web Design-2</subject> <credit>4</credit> <hoursperweek>4 Hrs</hoursperweek> <body>Web Design requires designers to create graphics, typography as well as images which are used only on the World Wide Web. While creating any design, web designers need to maintain balance between creating a good	<?xml version="1.0" encoding="utf-8"?> <bca> <subject>405 : Web Design-2</subject> <credit>4</credit> <hoursperweek>4 Hrs</hoursperweek> <objective>To make students aware of web terminology and website designing tools. Student can understand and implement the real functions of website development. </objective>

design as well as the speed and efficiency for the webpage/website. </body> </bca>	<body>Web Design requires designers to create graphics, typography as well as images which are used only on the World Wide Web. While creating any design, web designers need to maintain balance between creating a good design as well as the speed and efficiency for the webpage/website. </body> </bca>

- **XML focuses on data rather than how it looks**
  One of the reason, XML is popular because it focuses on data rather than data presentation. The other markup language such as HTML is used for data presentation. This separates the data and its presentation part and gives us the freedom to present the data, the way we want, once we receive it using XML.
  Two or more systems can receive the same data from a same XML and present it in a different way using other markup language such as HTML.
- **XML is public standard**

  XML was developed by W3C (World Wide Web Consortium) organization as an open standard under February 1998.

- **XML is easy and efficient for data sharing**

  Since XML is software and hardware independent, it is easier to share data between different systems with different hardware and software configuration. Any system with any programming language can read and process a XML document.

- **XML is compatible with other markup languages like HTML**

  It is so much easier to read the data from XML and display it on an GUI (graphical user interface) using HTML markup language.

  When the data changes over time, we need not to make any changes in the HTML.

- **XML supports platform transition**

  The main reason why changing to new systems and platform is challenging, because it involves the headache of data conversion between incompatible formats which often results in data loss. XML simplifies this process as the data is transported on new upgraded systems without any data loss.

- **XML supports validation**

  A XML document can be validated using DTD or XML schema. This ensures that the XML document is syntactically correct and avoids any issues that may arise due to the incorrect XML.

- **XML supports Unicode**

  XML supports Unicode that allows it to communicate almost any information in any written human language.

- **XML adopts technology advancements**

  The reason why XML is popular and being used from a very long time is because, it can adapt to the new technologies because of its platform-independent nature.

Uses of XML

XML is playing a vital role in the web. XML has a variety of uses in the sectors like Web, e-business, and mobile applications. Few XML-based Languages include XHTML, RSS, SMIL, WSDL, WAP, and SOAP. XML files are used to develop database-driven types. Due to their Flexibility, they could transfer data without missing descriptive information among corporate databases. Few examples like, with XML business-to-business applications share information electronically between buyers and sellers. It's an excellent choice for exchange formats with small collections of data.

It can be described as follows:

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

1.2 XML syntax (Declaration, tags, elements)

XML syntax rule is used while writing an XML document or an XML application. It is a very simple and straight forward to learn and code.

Check out the below sample code for understanding of the XML syntax rules:

Sample Code (example 1.2)

```
<?xml version="1.0" encoding="UTF-8"?>
    <assign>
        <to>SYBCA</to>
        <from>Faculty</from>
        <subject>405 : WebDesign-1</subject>
        <info>Each Assignment of Subject carries 30 marks. </info>
    </assign>
```

XML Declaration / Prolog rules

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

- This line is called **XML Prolog** or **XML declaration**.
- This line is optional i.e., it can be either used or not in an XML document. However, it should be the very first line if used.

- The version="1.0" is the version of the XML currently used. There are various versions of XML available.
- The encoding="UTF-8" specifies the character encoding used while writing an XML document, for example, êèé is for French and so on. Its default value is "UTF-8".
- This declaration is case sensitive for example "xml" should must be in lower case in.

XML Root element rules

Every XML files should have one or more Root elements to avoid error.

- In the example 1.2 stated above the Root element is <assign> and all the remaining elements <to>, <from>,<subject>,<info> are the child elements and reside within the root element <assign>.
- It is case sensitive.

XML Element rules

- The XML elements should have a closing element for example **<info category = "internal">Sample code</info>** is correct but **<info category = "internal"> Sample code** is not correct because it does not contain the closing element and it will throw an error and vice-versa.
- The elements in XML should be nested properly otherwise it will throw an error. For example **<to> <from> VBP </from> </to>** is nested correctly but **<to> <from> VBP </to> </from>** is wrong because if <from> is opened inside the <to> element then this should also end inside of the </to> element.
- It is also case sensitive i.e., the starting and closing element should be in the same case. For example **<to>....</to>** is correct but **<to>.....</To>** is not correct and it will throw an error.

XML Attribute rules

- The XML attribute is having two part one is Name and other is its value. It resides inside of the opening of an XML element. For example: <info category = "internal"> Each Assignment of Subject carries 30 marks.</info> Here category is the attribute name and internal is its value and the attribute value should either be in a single quotation or in double quotation otherwise it will throw an error. The Attribute Name is written without any quotation.
- The XML attribute is also case sensitive.
- An XML element can have multiple attributes but cannot have the same attribute names in the same element.
- For example: **<info category="internal" sem="IV">** Each Assignment of Subject carries 30 marks. **</info>**

Above attributes is correct because of having multiple attributes with the different attribute name.

- **<info category="internal" category="IV">** Each Assignment of Subject carries 30 marks. **</info>**

  Above attribute is wrong because of having the same attribute name in a single element.

XML Tags

XML tags are the important features of XML document. It is similar to HTML but XML is more flexible then HTML. It allows to create new tags (user defined tags). The first element of XML document is called root element. The simple XML document contain opening tag and closing tag. The XML tags are case sensitive i.e. <root> and <Root> both tags are different. The XML tags are used to define the scope of elements in XML document.

XML Tags Property Rules

- Every XML document must have a root tag which enclose the XML document. It is not necessary to name of root tag is root. The name of root tag is any possible tag name.
- The XML document must have start-tag, so first starting tag is known as root tag. The opening tag started with < bracket followed by tag name or element name and close with > bracket.
- The tag which is started by start tag must end with the same tag with forward slash (end tag), or in other words every XML document must be ended with end-tag. The end tag started with < followed by / and its pair tag name ended with >
- In XML, tags are case sensitive. It means that <Root> and <root> both are different tags.
- The tag which contains no content are known as empty tags.
- XML tag must be close in appropriate order. For example, an XML tag opened inside another element must be closed before the outer element is closed.

1.3 Root Element, Case Sensitivity

XML document must have a root element. A root element can have child elements and sub-child elements. We already have discussed it in detail on page. 7(Previous page).

Some basic rules of XML are mentioned as follows:

- **XML is Case Sensitive**

  While defining the tags in XML we have to keep in mind that the tags which we defined must be strictly case sensitive. If a single letter in the tag name differs in cases for starting and ending it will be considered as an invalid XML declaration.

- **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.

  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.

- **No overlapping for elements in XML**

  All the elements in XML should be properly nested and they should not overlap.

- **White-spaces are preserved in XML**

  Unlike HTML that doesn't preserve white space, the XML document preserves white spaces.

1.4 XML Document

XML Document forms the full structure of xml formatted data composed with prolog and root element nested with other elements. There would be only one root element, where root element encloses many other elements inside the final inner element holds the data. XML Document can be divided as three components.

They are:

1. Prolog
2. Elements (Root or Other)
3. Data

Below is the example of Students XML document with the root element "students".

students.xml (example 1.3)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<?xml-stylesheet type="text/css" href="/style/design"?>
<!-- This is a comment -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<students>

    <stud>

        <name> Amit</name>

        <city> Surat</city>
```

```
    </stud>
    <stud>
        <name> Manish</name>
        <city> Bharuch</city>
    </stud>
</students>
```

In the above example 1.3,

Document Prolog —

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<?xml-stylesheet type="text/css" href="/style/design"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Root Element — <students>

Other Elements — <stud> , <name> , <city>

Data — Amit, Surat, Manish, Bharuch

Let's See above concepts in descriptive manner.

1.4.1 XML Prolog section

XML Prolog is the component added in the beginning of an XML document. Otherwise, we can say that whatever it appears before the document's root element can be considered as Prolog. XML Prolog includes XML declaration, DOCTYPE and comments, processing instructions too. (See example 1.3)

DOCTYPE is used to mention the Document Type Definition and it is used for validation purpose. We could call it as a valid XML if it get validated through DTD. DTD will consist the structure of an XML document. DTD defines the elements to be used in the XML document. (See example 1.3)

```
<!DOCTYPE students
[
<!ELEMENT students (stud)>
<!ELEMENT stud (name,city)>
<!ELEMENT name (#DATA)>
<!ELEMENT city (#DATA)>
]>
```

1.4.2 XML Element section (Root, Other elements)

XML Element is the main component which holds the data from start tag till end tag. It is used as a container to store text elements, attributes, media objects etc or in case of empty elements it is delimited by an empty tag. Every XML documents contain at least one element whose scopes are delimited by start and end tags. Every XML document can have only one root element. In the above shown example 1.3 there is one root element listed as <students>. And the root element is able to hold other sub

elements. XML document consists one or more elements enclosed in root element. Elements naming represent the self-explanatory form of data inside it. Elements contains attributes which are used to specify additional information about the element. In the above shown example 3, inner elements are listed as <stud>, <name>, <city>.

Syntax

<element-name attributes> Contents...</element-name>

In the syntax above the attributes are used to define the XML element property and these attributes are separated by white space. It associates the name with a value, which is a string of characters.

Empty Elements

An element in XML document which does not contains the content is known as Empty Element. The basic syntax of empty element in XML as follows:

Syntax

<element-name attributes></element-name>

OR

<element-name attributes / >

Rules of XML elements

There are some rules to create XML elements which are given below:

- An element can contain alphanumeric values or characters. But only three special characters are required in the names these are hyphen, underscore and period.
- Names are case sensitive. It means lower case letters have different meaning and upper case characters have different meaning. For example city, City, CITY are different names.
- Both start and end tags for elements need to be same.
- An element, which is a container, can contain text or elements

Data

An XML element hold the value which is considered as data value.

1.5 XML Declaration and Rules of Declaration

XML documents can contain an XML declaration that if present, must be the first construct in the document. An XML declaration is made up of as many as three name/value pairs, syntactically identical to attributes. The three attributes are a mandatory version attribute and optional encoding and standalone attributes. The order of these attributes within an XML declaration is fixed.

Syntax

```
<?xml
  version = "version_number"
  encoding = "encoding_declaration"
  standalone = "standalone_status"
?>
```

The XML declaration begins with the character sequence **<?xml** and ends with the character sequence **?>**. Note that although this syntax is identical to that for processing instructions, the XML declaration is not considered to be a processing instruction.

All XML declarations have a **version** attribute with a value that must be **1.0**

The **character encoding** used for the document content can be specified through the encoding attribute. XML documents are inherently Unicode, even when stored in a non-Unicode character encoding. The XML recommendation defines several possible values for the encoding attribute. For example, UTF-8, UTF-16, ISO-10646-UCS-2, and ISO-10646-UCS-4 all refer to unicode/ISO-10646 encodings, whereas ISO-8859-1 and ISO-8859-2 refer to 8-bit Latin character encodings. Encodings for other character sets including Chinese, Japanese, and Korean characters are also supported. It is recommended that encodings be referred to using the encoding names registered with the Internet Assigned Numbers Authority (IANA). All XML processors are required to be able to process documents encoded using UTF-8 or UTF-16, with or without an XML declaration. The encoding of UTF-8 and UTF-16 encoded documents is detected using the Unicode byte-order-mark. The XML declaration is mandatory if the encoding of the document is anything other than UTF-8 or UTF-16. In practice, this means that documents encoded using US-ASCII can also omit the XML declaration because US-ASCII overlaps entirely with UTF-8.

Only one encoding can be used for an entire XML document. It is not possible to “redefine” the encoding part of the way through. If data in different encodings need to be represented, then external entities should be used.

If an XML document can be read with no reference to external sources, it is said to be a stand-alone document. Such documents can be annotated with a **standalone** attribute with a value of **yes** in the XML declaration. If an XML document requires external sources to be resolved to parse correctly and/or to construct the entire data tree (for example, a document with references to external general entities), then it is not a stand-alone document. Such documents may be marked **standalone='no'**, but because this is the default, such an annotation rarely appears in XML documents.

Rules of XML Declaration

- An XML declaration should abide with the following rules –

- If the XML declaration is present in the XML, it must be placed as the first line in the XML document.
- If the XML declaration is included, it must contain version number attribute.
- The Parameter names and values are case-sensitive.
- The names are always in lower case.
- The order of placing the parameters is important. The correct order is: version, encoding and standalone.
- Either single or double quotes may be used.
- The XML declaration has no closing tag i.e. </?xml>

Chapter – 2

jQuery Fundamentals

2.1 Introduction and Basics

jQuery is a JavaScript framework; which purpose is to make it much easier to use JavaScript on your website. You could also describe jQuery as an abstraction layer, since it takes a lot of the functionality that you would have to write many lines of JavaScript to accomplish and wraps it into functions that you can call with a single line of code. It's important to note that jQuery does not replace JavaScript, and while it does offer some syntactical shortcuts, the code you write when you use jQuery is still JavaScript code.

jQuery is a client-side JavaScript library that abstracts away browsers' different implementations into an easy-to-use API. What jQuery does best is to interact with the DOM (add, modify, remove elements on your page), do AJAX requests, create effects (animations) and so forth. It does not provide an application framework, it's merely a tool amongst others that should be used what it's meant to be used for. However, there's a plethora of plug-ins due to a thriving community, and there's pretty much a plug-ins for anything you can think of.

With that in mind, you should be aware that you don't need to be a JavaScript expert to use jQuery. In fact, jQuery tries to simplify a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation, so that you may do these things without knowing a lot about JavaScript.

There are a bunch of other JavaScript frameworks out there, but as of right now, jQuery seems to be the most popular and also the most extendable, proved by the fact that you can find jQuery plug-ins for almost any task out there. The power, the wide range of plug-ins and the beautiful syntax is what makes jQuery such a great framework. Keep reading to know much more about it and to see why we recommend it.

JQUERY

jQuery is a fast and concise JavaScript Library created by **John Resig** in 2006 with a nice motto –**Write less, do more**. jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code.

FEATURES OF JQUERY

- **DOM manipulation** – The jQuery made it easy to select DOM elements, traverse them and modifying their content by using cross-browser open source selector engine called **Sizzle**.
- **Event handling** – The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support** – The jQuery helps you a lot to develop a responsive and feature-rich site using AJAX technology.

- **Animations** – The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight** – The jQuery is very lightweight library - about 19KB in size Minified and gzipped.
- **Cross Browser Support** – The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Compatibility All Dynamic Languages** - jQuery script can be use with all most Dynamic Web Languages like PHP, ASP, JSP, CGI etc.
- **Latest Technology** – The jQuery supports CSS3 selectors and basic XPath syntax.

HOW TO USE JQUERY?

There are two ways to use jQuery.

- **Local Installation –** You can download jQuery library on your local machine and include it in your HTML code.
- **CDN Based Version –** You can include jQuery library into your HTML code directly from Content Delivery Network CDN.

LOCAL INSTALLATION

- Go to the https://jquery.com/download/ to download the latest version available.
- Now put downloaded jquery-2.1.3.min.js file in a directory of your website, e.g. /jquery.

Example

Now you can include jquery library in your HTML file as follows –

```
<html>
<head>
<title>The jQuery Example</title>
 <script type = "text/javascript"  src = "/jquery/jquery-2.1.3.min.js"></script>
        <script type = "text/javascript">
                $(document).ready(function()
                        {
                        document.write("Hello, World!");
                        });
        </script>
</head>
<body>
    <h1>Hello</h1>
</body>
</html>
```

CDN BASED VERSION

You can include jQuery library into your HTML code directly from Content Delivery Network CDN. Google and Microsoft provides content deliver for the latest version.

Example

Now let us rewrite above example using jQuery library from Google CDN.

```
<html>
<head>
   <title>The 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">
     $(document).ready(function(){
       document.write("Hello, World!");
     });
   </script>
 </head>
 <body>
   <h1>Hello</h1>
 </body>
</html>
```

INSTALLATION OF JQUERY

First of all we have to need jQuery library file. Download latest version of **jquery.js** file from www.jquery.com Website.

Current version is 1.8.0 is Select the **Production (32KB, Minified and Gzipped)** and click on Download **jquery-1.8.0.min.js**. Following is the list of available jQuery versions on the site

- Download jquery-1.8.0.min.js (Current Version)
- Download jquery-1.7.2.min.js
- Download jquery-1.7.1.min.js
- Download jquery-1.6.1.min.js
- Download jquery-1.6.min.js
- Download jquery-1.5.2.min.js
- Download jquery-1.5.1.min.js
- Download jquery-1.5.min.js
- Download jquery-1.4.4.min.js

You can rename your file to **jquery.js** for simplicity and put it on the root directory of your website. Following is a small example of using it.

```
<html>
<head>
<title>The jQuery Structure</title>
<script type="text/javascript" src="jquery.js"></script>
  <script type="text/javascript">
    /* add needed javascript code */
  </script>
</head>
<body>
<!-- Some HTMl code may be written here -->
</body>
</html>
```

HOW TO CALL A JQUERY LIBRARY FUNCTIONS?

As almost everything we do when using jQuery reads or manipulates the document object model (DOM), we need to make sure that we start adding events etc. as soon as the DOM is ready.

If you want an event to work on your page, you should call it inside the $(document).ready() function. Everything inside it will load as soon as the DOM is loaded and before the page contents are loaded.

To do this, we register a ready event for the document as follows –

```
$(document).ready(function() {
  // do stuff when DOM is ready
});
```

To call upon any jQuery library function, use HTML script tags as shown below –

```
<html>
  <head>
    <title>The 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() {
        $("div").click(function() {alert("Hello, world!");});
      });
    </script>
  </head>
  <body>
    <div id = "mydiv">
      Click on this to see a dialogue box.
    </div>
  </body>
</html>
```

HOW TO INCLUDE EXTERNAL SCRIPT USING JQUERY?

It is very good way to write the script in external page and then after linked in the document. It main benefit is reducing complex coding and easy to understand and another benefit is external script is use one or more documents to reduce the size of main document.

```
/* Save this Filename: external_demo.js */
  $(document).ready(function()
  {
    $("div").click(function()
            {
                    alert("This Example is External jQuery..!");
            });
  });
```

Now we can include external_demo.js file in our main html document.

```
<html>
<head>
<title>The External jQuery Example</title>
  <script type="text/javascript"
  src="../jquery.min.js"></script>
  <script type="text/javascript"
  src="external_demo.js"></script>
</head>
<body>
<div id="ex1">
Click here to open Dialogue Box.
</div>
</body>
</html>
```

2.1.1 Advantages of jQuery and Syntax

Advantages

- **Easy to learn:** jQuery is easy to learn because it supports same JavaScript style coding.
- **Write less do more:** jQuery provides a rich set of features that increase developers' productivity by writing less and readable code.
- **Excellent API Documentation:** jQuery provides excellent online API documentation.
- **Cross-browser support:** jQuery provides excellent cross-browser support without writing extra code.
- **Unobtrusive**: jQuery is unobtrusive which allows separation of concerns by separating html and jQuery code.

Syntax

jQuery syntax is made by using HTML elements selector and perform some action on the elements are manipulation in Dot sign(.).

jQuery syntax: **$(selector).action( )**

- $ sign define the jQuery,
- Selector define the Query Elements in HTML document, and
- action() define the action performed on the elements.

JQUERY BASIC SYNTAX EXAMPLES

$("p").hide()

The jQuery hide() function, hiding all elements.

Code	Output

<html>
<head>
<script type="text/javascript" src="jquery.js">
</script>
<script type="text/javascript">
$(document).ready(function() {
$("button").click(function() {
$("p").hide();
});
});
</script>
</head>
<body>
 This is a First Paragraph.
 This is a Second Paragraph.
<button>Click Me to Hide Above All Paragraph</button>
</body>
</html>
	This is a First Paragraph.
	This is a Second Paragraph.
	Click Me to Hide Above All Paragraph

$("this").hide()

The jQuery hide() function, hiding current(this) element.

Code	Output
<html> <head> <script type="text/javascript" src="jquery.js"> </script> <script type="text/javascript"> $(document).ready(function() { $("button").click(function()	Click Me to Hide THIS button

{ $(this).hide(); }); }); </script> </head> <body> <button>Click Me to Hide THIS button</button> </body> </html>	

$("#div1").hide()

The jQuery hide() function, hiding whose id="div1" in the elements.

Code	Output
<html> <head> <script type="text/javascript" src="jquery.js"></script> <script type="text/javascript"> $(document).ready(function() { $("button").click(function() { $("#div1").hide(); }); }); </script> </head> <body> This is a Second Paragraph. <button>Click Me to Hide Above Paragraph</button> </body> </html>	This is a Second Paragraph. Click Me to Hide Above Paragraph

$(".div1").hide()

The jQuery hide() function, hiding whose class=".div1" in the elements.

Code	Output

```
<html>
<head>
  <script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function()
{
 $("button").click(function()
 {
  $(".div1").hide();
 });
});
</script>
</head>

<body>
<p>This is a First Paragraph.</p>
<p class="div1">This is a Second Paragraph.</p>
<button>Click Me to Hide Above Paragraph</button>
</body>
</html>
```

This is a First Paragraph.

This is a Second Paragraph.

Click Me to Hide Above Paragraph

2.1.2 jQuery: Selectors

jQuery selectors is most important aspects of the jQuery library. jQuery library allows you to select elements in your HTML document by wrapping them in $(" ") (also you have to use single quotes), which is the jQuery wrapper. Selectors are useful and required at every step while using jQuery. With jQuery selectors you can find elements based on their id, classes, types, attributes, values of attributes and much more. It's based on the existing CSS Selectors, and in addition, it has some own custom selectors.

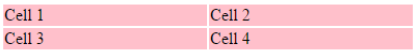
All type of selectors in jQuery, start with the dollar sign and parentheses: $( ).

jQuery Selector Syntax

Selector	Description
TagName / element	Selects all element match of given elements.
This	Selects current elements.
#ID	Selects element whose id is match of given elements.
.CLASS	Selects element whose class is match of given elements.
*	Selects all elements in the document.

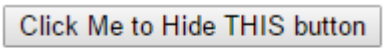
Element / Tag Selector

The jQuery element selector selects elements based on their tag names. You can select all <table> elements on a page like this: $("table").

Code	Output
<html> <head> <script type="text/javascript" src="jquery.js"></script> <script type="text/javascript"> $(document).ready(function() { $("table *").css("background-color","pink"); }); </script> </head> <body> <table border="0" width="400px"> <tr> <td>Cell 1</td> <td>Cell 2</td> </tr> <tr> <td>Cell 3</td> <td>Cell 4</td> </tr> </table> </body> </html>	

THIS SELECTOR

The jQuery this selector is used for selecting the current element.

Code	Output
<html> <head> <script type="text/javascript" src="jquery.js"> </script> <script type="text/javascript"> $(document).ready(function() { $("button").click(function() { $(this).hide(); }); }); </script> </head>	

<body> <button>Click Me to Hide THIS button</button> </body> </html>	

ID SELECTOR

The jQuery #id selector uses the id attribute of an HTML tag to find the specific element. An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.

Code	Output
<html> <head> <script type="text/javascript" src="jquery.js"></script> <script type="text/javascript"> $(document).ready(function(){ $("#name").css("background-color","pink"); }); </script> </head> <body> <table border="0" width="400px"> <tr> <td id="name">Cell 1</td> <td>Cell 2</td> </tr> <tr> <td>Cell 3</td> <td>Cell 4</td> </tr> </table> </body> </html>	Cell 1 Cell 2 Cell 3 Cell 4

CLASS SELECTOR

The jQuery class selector finds elements with a specific class. To find elements with a specific class, write a period character, followed by the name of the class: $(".test")

Code	Output

```
<html>
<head>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
 $(".name").css("background-color","pink");
});
</script>
</head>
<body>
<table border="0" width="400px">
<tr>
<td class="name">Cell 1</td>
<td>Cell 2</td>
</tr>
<tr>
<td>Cell 3</td>
<td>Cell 4</td>
</tr>
</table>
</body>
</html>
```

jQuery Custom Selectors

Syntax	Description
$(":animated")	Selects elements currently being animated.
$(":button")	Selects any button elements (inputs or buttons tag).
$(":radio")	Selects radio buttons.
$(":checkbox")	Selects checkboxes.
$(":header")	Selects header elements (h1, h2, h3, etc..).

jQuery Selector References

Following some jQuery Selector References...

Selector	Example	Description
*	$("table *")	Select All Elements.
#id	$("#name")	Selected element with id="name".
.class	$(".name")	Selected elements with class="name".
Tag	$("p")	Selected All p elements.
Selector	**Example**	**Description**
:first	$("p:first")	first <p> element.

:last	$("p:last")	last <p> element.
:even	$("p:even")	Perform all even <tr> elements.
:odd	$("p:odd")	Perform all odd <tr> elements.
Selector	**Example**	**Description**
:enabled	$(":enabled")	All enabled elements.
:disabled	$(":disabled")	All disabled elements.
:selected	$(":selected")	All selected elements.
:checked	$(":checked")	All checked elements.
Selector	**Example**	**Description**
:input	$(":input")	selected All input elements.
:text	$(":text")	selected All input elements with type="text".
:button	$(":button")	selected All input elements with type="button".
:password	$(":password")	selected All input elements with type="password".
:radio	$(":radio")	selected All input elements with type="radio".
:checkbox	$(":checkbox")	selected All input elements with type="checkbox".
:image	$(":image")	selected All input elements with type="image".
:file	$(":file")	selected All input elements with type="file".
:submit	$(":submit")	selected All input elements with type="submit".
:reset	$(":reset")	selected All input elements with type="reset".
Selector	**Example**	**Description**
:header	$(":header")	Selected All header elements h1...h6.
:animated	$(":animated")	Selected All animated elements.
:hidden	$("p:hidden")	Selected All hidden p elements.
:visible	$("tr:visible")	Selected All visible table rows.
:empty	$(":empty")	Selected All elements with no child of the elements.
:contains(text)	$(":contains('Viral Polishwala')")	Select All elements which contains is text.
Selector	**Example**	**Description**
[attribute]	$("[href]")	Select All elements with a href attribute.
[attribute$=value]	$("a:[href$=.org]")	Selected elements with a href attribute value ending with ".org".
[attribute=value]	$("a:[href=#]")	Selected elements with a href attribute value equal to "#".
[attribute!=value]	$("a:[href!=#]")	Selected elements with a href attribute value not equal to "#".

2.1.3 jQuery Events

JavaScript has its own ability to create interaction with Users. Events is a perform action in Dynamic Web Page.

Following are the examples events –

- A mouse click
- A web page loading
- Taking mouse over an element
- Submitting an HTML form
- A keystroke on your keyboard

EVENT TYPES

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:

S.N.	Event Type & Description
1	**Blur:** Occurs when the element loses focus.
2	**Change:** Occurs when the element changes.
3	**Click:** Occurs when a mouse click.
4	**Dblclick:** Occurs when a mouse double-click.
5	**Error:** Occurs when there is an error in loading or unloading etc.
6	**Focus:** Occurs when the element gets focus.
7	**Keydown:** Occurs when key is pressed.
8	**Keypress:** Occurs when key is pressed and released.
9	**Keyup:** Occurs when key is released.
10	**Load:** Occurs when document is loaded.
11	**Mousedown:** Occurs when mouse button is pressed.
12	**Mouseenter:** Occurs when mouse enters in an element region.
13	**Mouseleave:** Occurs when mouse leaves an element region.

14 **Mousemove:** Occurs when mouse pointer moves.

15 **Mouseout:** Occurs when mouse pointer moves out of an element.

16 **Mouseover:** Occurs when mouse pointer moves over an element.

17 **Mouseup:** Occurs when mouse button is released.

18 **Resize:** Occurs when window is resized.

19 **Scroll:** Occurs when window is scrolled.

20 **Select:** Occurs when a text is selected.

21 **Submit:** Occurs when form is submitted.

22 **Unload:** Occurs when documents is unloaded.

THE EVENT OBJECT

When these events are triggered you can then use a custom function to do pretty much whatever you want with the event. These custom functions call Event Handlers.

The callback function takes a single parameter; when the handler is called the JavaScript event object will be passed through it.

The event object is often unnecessary and the parameter is omitted, as sufficient context is usually available when the handler is bound to know exactly what needs to be done when the handler is triggered, however there are certain attributes which you would need to be accessed.

THE EVENT ATTRIBUTES

The following event properties/attributes are available and safe to access in a platform independent manner –

Property	Description
altKey	Set to true if the Alt key was pressed when the event was triggered, false if not. The Alt key is labeled Option on most Mac keyboards.
ctrlKey	Set to true if the Ctrl key was pressed when the event was triggered, false if not.
Data	The value, if any, passed as the second parameter to the bind command when the handler was established.
keyCode	For keyup and keydown events, this returns the key that was pressed.
metaKey	Set to true if the Meta key was pressed when the event was triggered, false if not. The Meta key is the Ctrl key on PCs and the Command key on Macs.

pageX	For mouse events, specifies the horizontal coordinate of the event relative from the page origin.
pageY	For mouse events, specifies the vertical coordinate of the event relative from the page origin.
relatedTarget	For some mouse events, identifies the element that the cursor left or entered when the event was triggered.
Screen	For mouse events, specifies the horizontal coordinate of the event relative from the screen origin.
Screen	For mouse events, specifies the vertical coordinate of the event relative from the screen origin.
shiftKey	Set to true if the Shift key was pressed when the event was triggered, false if not.
Target	Identifies the element for which the event was triggered.
Timestamp	The timestamp inmillisecondsinmilliseconds when the event was created.
Type	For all events, specifies the type of event that was triggered forexample,clickforexample,click.
Which	For keyboard events, specifies the numeric code for the key that caused the event, and for mouse events, specifies which button was pressed 1for left, 2 for middle, 3 for right

THE EVENT METHODS

bind( )

Use	Event occur when One ore more event handlers attach the selected match elements.
Syntax	$(selector).bind(event,[data],function)
Event	event is Required parameter. event define the one or more events attach to the elements.
Data	data is Optional parameter. data define the addition data pass to the function.
Function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script src="jquery.js" type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").bind("click",function(){
	$("div").fadeTo("slow",0.20);
	});
	});
	</script>
	</head>
	<body>
	<button>Click to Show Bind Event</button>
	
	</body>

	</html>

blur( )

Use	Event occur when element lost focus.
Syntax	$(selector).blur(function) (Bind a Function to blur event) $(selector).blur() (Trigger the blur event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").focus(function(){
	$("input").css("background-color","#FFFF99");
	});
	$("input").blur(function(){
	$("input").css("background-color","pink");
	});
	});
	</script>
	</head>
	<body>
	<form action="">
	Name:
	<input type="text" name="name" />
	</form>
	</body>
	</html>

change( )

Use	Event occurs when change the elements.
Syntax	$(selector).change(function) (Bind a Function to change event) $(selector).change() (Trigger the change event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").change(function(){
	$(this).css("background-color","pink");
	});
	});

	</script> </head> <body> <form action=""> Name: <input type="text" name="name" /> <input type="submit" name="submit" /> </form> </body> </html>

click( )

Use	Event Occurs when the mouse click.
Syntax	$(selector).click(function) (Bind a Function to click event) $(selector).click() (Trigger the click event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").click(function(){
	$(this).css("background-color","pink");
	});
	});
	</script>
	</head>
	<body>
	<form action="">
	Name:
	<input type="text" name="name" />
	
	<input type="submit" name="submit" />
	</form>
	</body>
	</html>

dblclick( )

Use	Event Occurs when mouse perform double click.
Syntax	$(selector).dblclick(function) (Bind a Function to dblclick event) $(selector).dblclick() (Trigger the dblclick event)
Function	Function is Optional parameter. Function define the ready to run when event occur.
Example	<html> <head>

	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	 $("input").dblclick(function(){
	 $(this).css("background-color","pink");
	 });
	});
	</script>
	</head>
	<body>
	<form action="">
	Name:
	<input type="text" name="name" />
	
	<input type="submit" name="submit" />
	</form>
	</body>
	</html>

delegate( )

Use	Event Occurs when add one or more event handlers, specific child element of matching elements.
Syntax	$(selector).delegate(ChildSelector,event,[data],function)
Child Selector	ChildSelector is Required parameter. ChildElement define add one or more childelement add to handle event.
event	event is Required parameter. event define add one or more event add to handle event.
data	data is Optional parameter. data define the addition data pass to the function.
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	 $("div").delegate("button","click",function(){
	 $("form").slideToggle();
	 });
	});
	</script>
	</head>
	<body>
	
	<button>Click Delegate Event Run</button>
	<form action="">

	Name: <input type="text" name="name" /> <input type="submit" name="submit" /> </form> </body> </html>

die( )

Use	Event Occurs when remove all event handler along with live() event.
Syntax	$(selector).die([event],[function])
event	event is Optional parameter. event define add one or more event add to handle event.
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	 $("p").live("click",function(){
	 $(this).slideToggle();
	 });
	 $("button").click(function(){
	 $("p").die();
	 });
	});
	</script>
	</head>
	<body>
	 This Me to Shoe Live or Die Event Example
	 This Me and occur slideToogle effect with die or live event
	 This Me to Shoe Live or Die Event Example
	<button>Click to Run Die Event</button>
	</body>
	</html>

error( )

Use	Event error Occurs when selected element not loaded successfully.
Syntax	$(selector).error(function) (Bind a Function to error event) $(selector).error() (Trigger the error event)
function	Function is Optional parameter. Function define the ready to run when event occur.
Example	<html> <head> <script type="text/javascript" src="jquery.js"></script>

	<script type="text/javascript">
	$(document).ready(function(){
	 $("img").error(function(){
	 $("img").replaceWith("Error Loading Image Not Open..!");
	 });
	});
	</script>
	</head>
	<body>
	 error event occur, when image is not open
	
	
	</body>
	</html>

e.pageX( )

Use	Event Occurs when mouse position in Left Side.
Syntax	event.pageX
event	event is Required parameter. event define specific event binding to the function.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	 $(document).mousemove(function(e){
	 $("span").text("X: " + e.pageX + ", Y: " + e.pageY);
	 });
	});
	
	</script>
	</head>
	<body>
	 Mouse Position is:
	</body>
	</html>

e.pageY( )

Use	Event Occurs when mouse position in Top Side.
Syntax	event.pageY
event	event is Required parameter. event define specific event binding to the function.
Example	<html> <head> <script type="text/javascript" src="jquery.js"></script> <script type="text/javascript">

	$(document).ready(function(){
	$(document).mousemove(function(e){
	$("span").text("X: " + e.pageX + ", Y: " + e.pageY);
	});
	});
	
	</script>
	</head>
	<body>
	 Mouse Position is:
	</body>
	</html>

e.timestamp( )

Use	Event Occurs on content the number of stared milliseconds since Jan 1, 1970 to current time.
Syntax	event.timeStamp
event	event is Required parameter. event define specific event binding to the function.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(e){
	$("span").text(e.timeStamp);
	});
	});
	</script>
	</head>
	<body>
	 The click event for the button below occurred <u>_____</u> milliseconds after January 1, 1970.
	<button>Click to Update MilliSeconds</button>
	</body>
	</html>

e.which( )

Use	Event Occurs, when key press on your keyboard and return key number.
Syntax	event.which
event	event is Required parameter. event define specific event binding to the function.
Example	<html> <head> <script type="text/javascript" src="jquery.js"></script>

	<script type="text/javascript">
	$(document).ready(function(){
	  $("input").keydown(function(e){
	  $("span").text("Pressed Key: " + e.which);
	  });
	});
	</script>
	</head>
	<body>
	 press key in textbox to show which number of key is press in your keyboard
	<input type="text" name="inputtext">
	
	</body>
	</html>

focus( )

Use	Event Occurs when the element gets focus.
Syntax	$(selector).focus(function) (Bind a Function to focus event) $(selector).focus() (Trigger the focus event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	  $("input").focus(function(){
	  $("input").css("background-color","#FFFF99");
	  });
	  $("input").blur(function(){
	  $("input").css("background-color","pink");
	  });
	  $("#btn1").click(function(){
	  $("input").focus();
	  });
	  $("#btn2").click(function(){
	  $("input").blur();
	  });
	});
	</script>
	</head>
	<body>
	<button id="btn1">Click Trigger Focus Event</button>

	<button id="btn2">Click Trigger Blur Event</button> <form action=""> Name: <input type="text" name="name" /> </form> </body> </html>

focusin( )

Use	Event Occurs when the element gets focus.
Syntax	$(selector).focusin(function())
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("div").focusin(function(){
	$(this).css("background-color","pink");
	});
	$("div").focusout(function(){
	$(this).css("background-color","#CCCCCC");
	});
	});
	</script>
	</head>
	<body>
	 Click textbox to occur focusin or focusout event
	
	<form action="">
	Name:
	<input type="text" name="name" />
	</form>
	
	</body>
	</html>

focusout( )

Use	Event Occurs when the element focus out.
Syntax	$(selector).focusout(function())
function	Function is Optional parameter. Function define the ready to run when event occur.
Example	<html> <head>

	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	 $("div").focusin(function(){
	 $(this).css("background-color","pink");
	 });
	 $("div").focusout(function(){
	 $(this).css("background-color","#CCCCCC");
	 });
	});
	</script>
	</head>
	<body>
	 Click textbox to occur focusin or focusout event
	
	<form action="">
	Name:
	<input type="text" name="name" />
	</form>
	
	</body>
	</html>

hover( )

Use	Event Occurs when the hover on the selected element.
Syntax	$(selector).hover(FocusIn,FocusOut)
focusIn	FocusIn is Required parameter. FocusIn define mouse in the selected element.
focusOut	FocusOut is Required parameter. FocusOut define mouse out the selected element.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	 $("div").hover(function(){
	 $("div").css("background-color","pink");
	 },function(){
	 $("div").css("background-color","#CCCCCC");
	 });
	});
	</script>
	</head>
	<body>
	 Click textbox to occur focusin or focusout event

	 <form action=""> Name: <input type="text" name="name" /> </form> </body> </html>

keydown( )

Use	Event Occurs when key is pressed.
Syntax	$(selector).keydown(function) (Bind a Function to keydown event) $(selector).keydown() (Trigger the keydown event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").keydown(function(){
	$("input").css("background-color","#FFFF99");
	});
	$("input").keyup(function(){
	$("input").css("background-color","pink");
	});
	$("#btn1").click(function(){
	$("input").keydown();
	});
	$("#btn2").click(function(){
	$("input").keyup();
	});
	});
	</script>
	</head>
	<body>
	<button id="btn1">Function keyDown Event Run</button>
	<button id="btn2">Function keyUp Event Run</button>
	<form action="">
	Name:
	<input type="text" name="name" />
	</form>
	</body>
	</html>

keypress( )

Use	Event Occurs when key is pressed count the pressed key.
Syntax	$(selector).keypress(function) (Bind a Function to keypress event) $(selector).keypress() (Trigger the keypress event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	i=0;
	$(document).ready(function(){
	$("input").keypress(function(){
	$("span").text(i=i+1);
	});
	$("button").click(function(){
	$("input").keypress();
	});
	});
	</script>
	</head>
	<body>
	<button>Function keypress Event Run</button>
	<form action="">
	Name:
	<input type="text" name="name" />
	</form>
	 No of Keypressed:
	</body>
	</html>

keyup( )

Use	Occurs when key is released.
Syntax	$(selector).keyup(function) (Bind a Function to keyup event) $(selector).keyup() (Trigger the keyup event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").keydown(function(){
	$("input").css("background-color","#FFFF99");

	});
	$("input").keyup(function(){
	$("input").css("background-color","pink");
	});
	$("#btn1").click(function(){
	$("input").keydown();
	});
	$("#btn2").click(function(){
	$("input").keyup();
	});
	});
	</script>
	</head>
	<body>
	<button id="btn1">Function keyDown Event Run</button>
	<button id="btn2">Function keyUp Event Run</button>
	<form action="">
	Name:
	<input type="text" name="name" />
	</form>
	</body>
	</html>

live( )

Use	Event Occurs when live all event handler.
Syntax	$(selector).live([event],[function])
event	event is Optional parameter. event define add one or more event add to handle event.
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("p").live("click",function(){
	$(this).slideToggle();
	});
	$("button").click(function(){
	$("p").die();
	});
	});
	</script>
	</head>
	<body>

	This Me to Shoe Live or Die Event Example This Me and occur slideToogle effect with die or live event This Me to Shoe Live or Die Event Example <button>Click to Run Die Event</button> </body> </html>

load( )

Use	Event occurs when document is load.
Syntax	$(selector).load(function)
function	Function is Required parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	 $("img").load(function(){
	 $("div").text("Current Load Event is Run");
	 });
	});
	</script>
	</head>
	<body>
	
	 Image is Loading
	</body>
	</html>

mousedown( )

Use	Event Occurs when mouse button is pressed.
Syntax	$(selector).mousedown(function) (Bind Function to mousedown) $(selector).mousedown() (Trigger the mousedown event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	 $("input").mousedown(function(){
	 $("input").css("background-color","#FFFF99");
	 });
	 $("input").mouseup(function(){
	 $("input").css("background-color","pink");

	});
	$("#btn1").click(function(){
	$("input").mousedown();
	});
	$("#btn2").click(function(){
	$("input").mouseup();
	});
	});
	</script>
	</head>
	<body>
	<button id="btn1">Function MouseDown Event Run</button>
	<button id="btn2">Function MouseUp Event Run</button>
	<form action="">
	Name:
	<input type="text" name="name" />
	</form>
	</body>
	</html>

mouseenter( )

Use	Event Occurs when mouse enters in an element area.
Syntax	$(selector).mouseenter(function) (Bind Function to mouseenter) $(selector).mouseenter() (Trigger the mouseenter event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").mouseenter(function(){
	$("input").css("background-color","#FFFF99");
	});
	$("input").mouseleave(function(){
	$("input").css("background-color","pink");
	});
	$("#btn1").click(function(){
	$("input").mouseenter();
	});
	$("#btn2").click(function(){
	$("input").mouseleave();
	});
	});

	</script> </head> <body> <button id="btn1">Function MouseEnter Event Run</button> <button id="btn2">Function MouseLeave Event Run</button> <form action=""> Name: <input type="text" name="name" /> </form> </body> </html>

mouseleave( )

Use	Event Occurs when mouse leaves an element area.
Syntax	$(selector).mouseleave(function) (Bind Function to mouseleave) $(selector).mouseleave() (Trigger the mouseleave event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").mouseenter(function(){
	$("input").css("background-color","#FFFF99");
	});
	$("input").mouseleave(function(){
	$("input").css("background-color","pink");
	});
	$("#btn1").click(function(){
	$("input").mouseenter();
	});
	$("#btn2").click(function(){
	$("input").mouseleave();
	});
	});
	</script>
	</head>
	<body>
	<button id="btn1">Function MouseEnter Event Run</button>
	<button id="btn2">Function MouseLeave Event Run</button>
	<form action="">
	Name:
	<input type="text" name="name" />

	</form> </body> </html>

mousemove( )

Use	Event Occurs when mouse pointer moves.
Syntax	$(selector).mousemove(function) (Bind Function to mousemove)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$(document).mousemove(function(e){
	$("span").text("X: " + e.pageX + ", Y: " + e.pageY);
	});
	});
	
	</script>
	</head>
	<body>
	 Mouse Position is:
	</body>
	</html>

mouseout( )

Use	Event Occurs when mouse pointer out an element.
Syntax	$(selector).mouseout(function) (Bind Function to mouseout) $(selector).mouseout() (Trigger the mouseout event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").mouseover(function(){
	$("input").css("background-color","#FFFF99");
	});
	$("input").mouseout(function(){
	$("input").css("background-color","pink");
	});
	$("#btn1").click(function(){
	$("input").mouseover();

	}); $("#btn2").click(function(){ $("input").mouseout(); }); }); </script> </head> <body> <button id="btn1">Function Mouseover Event Run</button> <button id="btn2">Function Mouseout Event Run</button> <form action=""> Name: <input type="text" name="name" /> </form> </body> </html>

mouseover( )

Use	Event Occurs when mouse pointer moves over an element.
Syntax	$(selector).mouseover(function) (Bind Function to mouseover) $(selector).mouseover() (Trigger the mouseover event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").mouseover(function(){
	$("input").css("background-color","#FFFF99");
	});
	$("input").mouseout(function(){
	$("input").css("background-color","pink");
	});
	$("#btn1").click(function(){
	$("input").mouseover();
	});
	$("#btn2").click(function(){
	$("input").mouseout();
	});
	});
	</script>
	</head>
	<body>

	<button id="btn1">Function Mouseover Event Run</button> <button id="btn2">Function Mouseout Event Run</button> <form action=""> Name: <input type="text" name="name" /> </form> </body> </html>

mouseup( )

Use	Event Occurs when mouse button is released.
Syntax	$(selector).mouseup(function) (Bind Function to mouseup) $(selector).mouseup() (Trigger the mouseup event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").mousedown(function(){
	$("input").css("background-color","#FFFF99");
	});
	$("input").mouseup(function(){
	$("input").css("background-color","pink");
	});
	$("#btn1").click(function(){
	$("input").mousedown();
	});
	$("#btn2").click(function(){
	$("input").mouseup();
	});
	});
	</script>
	</head>
	<body>
	<button id="btn1">Function MouseDown Event Run</button>
	<button id="btn2">Function MouseUp Event Run</button>
	<form action="">
	Name:
	<input type="text" name="name" />
	</form>
	</body>
	</html>

ready( )

Use	Event Occurs when function is ready to document.
Syntax	$(document).ready(function) (Bind Function to ready event)
Function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script src="jquery.js" type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("p").hide();
	$("#btn1").click(function () {
	$("p").toggle("slow");
	});
	});
	</script>
	
	</head>
	<body>
	<button id="btn1">Click To Show Paragraph</button>
	 This Effect is Toggle Effect with ready event work.
	</body>
	</html>

select( )

Use	Event Occurs when a text is selected.
Syntax	$(selector).select(function) (Bind Function to select event) $(selector).select() (Trigger the select event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("input").select(function(){
	$("span").text("Text Selected");
	});
	$("button").click(function(){
	$("input").select();
	});
	});
	</script>

	</head> <body> <input type="text" value="Click to Select" /> <button>Select All Textbox text with Select event.</button> </body> </html>

submit( )

Use	Event Occurs when form is submitted.
Syntax	$(selector).submit(function) (Bind Function to submit event) $(selector).select() (Trigger the submit event)
function	Function is Optional parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("form").submit(function(e){
	alert("Successfully Submitted");
	});
	$("button").click(function(){
	$("form").submit();
	});
	});
	</script>
	</head>
	<body>
	<form action="#" method="get">
	Name:
	<input type="text" name="name" />
	
	Password:
	<input type="password" name="password"/>
	
	<input type="submit" name="submit"/>
	</form>
	</body>
	</html>

unload( )

Use	Event Occurs when documents is unloaded.
Syntax	$(selector).unload(function)

function	Function is Required parameter. Function define the ready to run when event occur.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$(window).unload(function(){
	alert("Close this window to show Me Unload Event occur..!");
	});
	});
	</script>
	</head>
	<body>
	
	 Image is Loading
	</body>
	</html>

2.2 jQuery : Effects

jQuery provides a trivially simple interface for doing various kind of amazing effects. jQuery methods allow us to quickly apply commonly used effects with a minimum configuration.

HIDE

Use	The **hide** method simply hides each of the set of matched elements if they are shown. There is another form of this method which controls the speed of the animation.
Syntax	selector.hide( );
Parameter	NA

Example	<html>
	<head>
	<script src="jquery.js" type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("#btn1").hide();
	$("p").hide();
	$("#btn2").click(function () {
	$("p").show("slow");
	$("#btn2").hide();
	$("#btn1").show();
	});
	$("#btn1").click(function () {
	$("p").hide("slow");
	});

	}); </script> </head> <body> <button id="btn1">Click To Hide Paragraph</button> <button id="btn2">Click To Show Paragraph</button> This Paragraph Will Be Hide After Click... </body> </html>

SHOW

Use	The **show( )** method simply shows each of the set of matched elements if they are hidden. There is another form of this method which controls the speed of the animation.
Syntax	selector.show( );
Parameter	NA

Example	<html>
	<head>
	<script src=" jquery.js " type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("#btn1").hide();
	$("p").hide();
	$("#btn2").click(function () {
	$("p").show("slow");
	$("#btn2").hide();
	$("#btn1").show();
	});
	$("#btn1").click(function () {
	$("p").hide("slow");
	});
	});
	</script>
	</head>
	<body>
	<button id="btn1">Click To Hide Paragraph</button>
	<button id="btn2">Click To Show Paragraph</button>
	 This Paragraph Will Be Hide After Click...
	</body>
	</html>

FADE

Fadein()	
Use	The fadeIn( ) method fades in all matched elements by adjusting their opacity and firing an optional callback after completion.
Syntax	selector.fadeIn(speed, [callback]);
Parameter	• **speed** – A string representing one of the three predefined speeds "slow","def",or"fast""slow","def",or"fast" or the number of milliseconds to run the animation e.g.1000e.g.1000. • **callback** – This is optional parameter representing a function to call once the animation is complete.

Example	<html>
	<head>
	<script src=" jquery.js " type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("#btn1").click(function(){
	$("div").fadeOut(5000);
	});
	$("#btn2").click(function(){
	$("div").fadeIn(5000);
	});
	});
	</script>
	</head>
	
	<body>
	 My Effect is fadeOut Effect
	<button id="btn1">Fade Out (5 Second)</button>
	<button id="btn2">Fade In (5 Second)</button>
	</body>
	</html>

Fadeout()	
Use	The **fadeOut( )** method fades out all matched elements by adjusting their opacity to 0, then setting display to "none" and firing an optional callback after completion.
Syntax	selector.fadeOut(speed, [callback]);
Parameter	• **speed** – A string representing one of the three predefined speeds "slow","def",or"fast""slow","def",or"fast" or the number of milliseconds to run the animation e.g.1000e.g.1000. • **callback** – This is optional parameter representing a function to call once the animation is complete.
Example	<html> <head>

	<script src=" jquery.js " type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("#btn1").click(function(){
	$("div").fadeOut(5000);
	});
	$("#btn2").click(function(){
	$("div").fadeIn(5000);
	});
	});
	</script>
	</head>
	<body>
	 My Effect is fadeOut Effect
	<button id="btn1">Fade Out (5 Second)</button>
	<button id="btn2">Fade In (5 Second)</button>
	</body>
	</html>

Fadeto()	
Use	The **fadeto()** method fades the opacity of all matched elements to a specified opacity and firing an optional callback after completion.
Syntax	selector.fadeTo(speed, opacity[, callback]);
Parameter	• **speed –** A string representing one of the three predefined speeds "slow","def",or"fast""slow","def",or"fast" or the number of milliseconds to run the animation e.g.1000e.g.1000. • **opacity –** A number between 0 and 1 denoting the target opacity. • **callback –** This is optional parameter representing a function to call once the animation is complete.

Example	<html>
	<head>
	<script src=" jquery.js " type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("div").fadeTo("slow",0.20);
	});
	});
	</script>
	</head>
	<body>
	<button>Click to Show FadeTo Effect</button>

	</body> </html>

SLIDE

jQuery slide methods use to change element height is visible or hidden in a Selected elements. jQuery Slide Method support main three methods..

slidedown()	
Use	The slideDown() method reveals all matched elements by adjusting their height and firing an optional callback after completion.
Syntax	$(selector).slideDown(speed,[callback]);
Parameter	• **speed:** Elements is a represent by three predefined String speed ("slow", "normal", "fast"). otherwise number represent by milliseconds (Ex. 3000). • **callback:** Callback is optional parameter. It is use to represents a function to be executed whenever effect is completed.

Example	<html>
	<head>
	<script src="jquery.js" type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("#click").click(function(){
	$("#slide").slideDown("slow");
	});
	});
	</script>
	<style type="text/css">
	#slide, #click
	{
	font-family:Verdana;
	font-size:14px;
	background:#CCCCCC;
	border:solid 1px #c3c3c3;
	text-align:center;
	}
	#slide
	{
	display:none;
	height:60px;
	}
	</style>
	</head>
	<body>

	This is a jQuery Effect Example you are realy enjoy this to see this effect. Show Slide Down Panel </body></html>
slideup()	
Use	The slideUp() method hides all matched elements by adjusting their height and firing an optional callback after completion.
Syntax	$(selector).slideUp(speed,[callback]);
Parameter	• **speed:** Elements is a represent by three predefined String speed ("slow", "normal", "fast"). otherwise number represent by milliseconds (Ex. 3000). • **callback:** Callback is optional parameter. It is use to represents a function to be executed whenever effect is completed.

Example	<html>
	<head>
	<script src="jquery.js" type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("#click").click(function(){
	$("#slide").slideUp("slow");
	});
	});
	</script>
	<style type="text/css">
	#slide, #click
	{
	font-family:Verdana;
	font-size:14px;
	background:#CCCCCC;
	border:solid 1px #c3c3c3;
	text-align:center;
	}
	#slide
	{
	height:60px;
	}
	</style>
	</head>
	
	<body>
	
	 This is a jQuery Effect Example you are realy enjoy this to see this effect.
	
	 Show Slide Up Panel

<table>
<tr><td></td><td></body>
</html></td></tr>
<tr><td colspan="2">slidetoggle()</td></tr>
<tr><td>Use</td><td>The slideToggle() method toggles the visibility of all matched elements by adjusting their height and firing an optional callback after completion.</td></tr>
<tr><td>Syntax</td><td>$(selector).slideToggle(speed,[callback]);</td></tr>
<tr><td>Parameter</td><td>• speed: Elements is a represent by three predefined String speed ("slow", "normal", "fast"). otherwise number represent by milliseconds (Ex. 3000).
• callback: Callback is optional parameter. It is use to represents a function to be executed whenever effect is completed.</td></tr>
<tr><td>Example</td><td><html>
<head>
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function(){
$("#click").click(function(){
$("#slide").slideToggle("slow");
});
});
</script>
<style type="text/css">
#slide, #click
{
font-family:Verdana;
font-size:14px;
background:#CCCCCC;
border:solid 1px #c3c3c3;
text-align:center;
}
#slide
{
height:60px;
display:none;
}
</style>
</head>
<body>
<div id="slide">
<p>This is a jQuery Effect Example you are realy enjoy this to see this effect.
</p>
</div>
<p id="click">Show Slide Toogle Panel</p>
</body>
</html></td></tr>
</table>

ANIMATE

Use	The animate( ) method performs a custom animation of a set of CSS properties.
Syntax	selector.animate(params, [duration, easing, callback]);
Parameter	• **params** – A map of CSS properties that the animation will move toward. • **duration** – This is optional parameter representing how long the animation will run. • **easing** – This is optional parameter representing which easing function to use for the transition. • **callback** – This is optional parameter representing a function to call once the animation is complete.

Example	<html>
	<head>
	<script src="jquery.js" type="text/javascript"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("div").animate({fontSize:"60px"},"slow");
	});
	});
	</script>
	</head>
	<body style="overflow:auto;">
	<button>Start Animation</button>
	 My Animate Effect
	</body></html>

STOP

Use	jQuery stop() effect method stop the running animation of selected elements.
Syntax	$(selector).stop([stopAll])
Parameter	• **stopall** – stopAll is Optional parameter. A Boolean value specifying stop the queued animate. Default value is false.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery-1.8.0.min.js"></script>
	<script type="text/javascript">
	$(document).ready(function()
	{
	$("#btn1").click(function(){
	$("#divanimate").animate({height:"300px"},8000);
	});
	$("#btn2").click(function(){
	$("#divanimate").stop();

	}); }); </script> </head> <body> <button id="btn1">Animate Start</button> <button id="btn2">Stop Efeect</button> <div id="divanimate" style="background: pink;margin:10px;height:100px;width:100px;" > </body> </html>

CALLBACK AND FUNCTIONS

Use	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.
Syntax	$(selector).hide(speed,callback);
Parameter	• **speed:** Elements is a represent by three predefined String speed ("slow", "normal", "fast"). otherwise number represent by milliseconds (Ex. 3000). • **callback:** Callback is optional parameter. It is use to represents a function to be executed whenever effect is completed.

Example	<html>
	<head>
	<script src=" jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("button").click(function(){
	$("p").hide("slow", function(){
	alert("The paragraph is now hidden");
	});
	});
	});
	</script>
	</head>
	<body>
	<button>Hide</button>
	 This is a paragraph with little content. </body></html>

```
<html>
<head>
<script src=" jquery.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p").hide(1000);
    alert("The paragraph is now hidden");
  });
});
</script>
</head>
<body>
<button>Hide</button>
<p>This is a paragraph with little content.</p>
</body>
</html>
```

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.

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).

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:

```
<html>
<head>
<script src=" jquery.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#p1").css("color", "red").slideUp(2000).slideDown(2000);
  });
});
</script>
</head>
<body>
<p id="p1">jQuery is fun!!</p>
<button>Click me</button>
```

```
</body>
</html>
```

We could also have added more method calls if needed. 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.

```
<html>
<head>
<script src=" jquery.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#p1").css("color", "red")
      .slideUp(2000)
      .slideDown(2000);
  });
});
</script>
</head>
<body>
<p id="p1">jQuery is fun!!</p>
<button>Click me</button>
</body>
</html>
```

2.3 jQuery : HTML Manipulation Methods

jQuery HTML Methods are performed for changing and manipulate the HTML elements or attributes.

jQuery Get

One very important part of jQuery is the possibility to manipulate the DOM (Document Object Model). jQuery comes with a bunch of DOM related methods that make it easy to access and manipulate elements and attributes.

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

- **text( )** - Returns the text content of selected elements
- **html( )** - Returns the content of selected elements (including HTML markup)
- **val( )** - Returns the value of form fields

Example (Get content with HTML and TEXT)

```
<html>
```

```
<head>
<script src=" jquery.js"></script>
<script>
$(document).ready(function(){
   $("#btn1").click(function(){
      alert("Text: " + $("#test").text());
   });
   $("#btn2").click(function(){
      alert("HTML: " + $("#test").html());
   });
});
</script>
</head>
<body>
<p id="test">This is some <b>bold</b> text in a paragraph.</p>
<button id="btn1">Show Text</button>
<button id="btn2">Show HTML</button>
</body>
</html>
```

Example (Manipulating element content)

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
 $("button").click(function(){
   $("p").html("Congratulation, Viral Polishwala");
 });
});
</script>
</head>
<body>
<p>This is a paragraph.</p>
<button>Change content of p elements</button>
</body>
</html>
```

Example (Getting value of input field with val( ))

```
<html>
<head>
<script src=" jquery. js"></script>
<script>
```

```
$(document).ready(function(){
  $("button").click(function(){
    alert("Value: " + $("#test").val());
  });
});
</script>
</head>
<body>
<p>Name: <input type="text" id="test" value="Mickey Mouse"></p>
<button>Show Value</button>
</body>
</html>
```

Example (Getting value of attribute with attr( ))

```
<html>
<head>
<script src=" jquery.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    alert($("#vbp").attr("href"));
  });
});
</script>
</head>
<body>
<p>Contact : <a href="http://www.viralpolishwala.co.in" id="vbp">Viral Polishwala</a></p>
<button>Show href Value</button>
</body>
</html>
```

jQuery Set

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

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

Example (Use of text( ), html( ) and val( ) for setting new content)

```
<html>
<head>
<script src=" jquery.js "></script>
```

```
<script>
$(document).ready(function(){
  $("#btn1").click(function(){
    $("#test1").text("Hello world!");
  });
  $("#btn2").click(function(){
    $("#test2").html("<b>Hello world!</b>");
  });
  $("#btn3").click(function(){
    $("#test3").val("Dolly Duck");
  });
});
</script>
</head>
<body>
<p id="test1">This is a paragraph.</p>
<p id="test2">This is another paragraph.</p>
<p>Input field: <input type="text" id="test3" value="Mickey Mouse"></p>
<button id="btn1">Set Text</button>
<button id="btn2">Set HTML</button>
<button id="btn3">Set Value</button>
</body>
</html>
```

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.

Example (Use of text( ) and html( ) with a callback function)

```
<html>
<head>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
  $("#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 + ")";
      });
   });
});
</script>
</head>
<body>
<p id="test1">This is a <b>bold</b> paragraph.</p>
<p id="test2">This is another <b>bold</b> paragraph.</p>
<button id="btn1">Show Old/New Text</button>
<button id="btn2">Show Old/New HTML</button>
</body>
</html>
```

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

Example (Use of attr( ) for setting new vale to attribute)

```
<html>
<head>
<script src=" jquery.js"></script>
<script>
$(document).ready(function(){
   $("button").click(function(){
      $("#vbp").attr("href", "http://www.viralpolishwala.co.in/downloads");
   });
});
</script>
</head>
<body>
<p><a href="http://www.viralpolishwala.co.in" id="vbp">Viral B. Polishwala</a></p>
<button>Change href Value</button>
<p>Mouse over the link (or click on it) to see that the value of the href attribute has changed.</p>
</body>
</html>
```

jQuery Add

With jQuery, it is easy to add new elements/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

Use	jQuery append() method Inserts content into inside end of the selected elements.
Syntax	$(selector).append(content)
Parameter	• **Content:** content is Required parameter. Insert content end into selected element.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("p").append(" Online Web Development Tutorial");
	});
	});
	</script>
	</head>
	<body>
	 Viral Polishwala
	<button>Insert append end of p element</button>
	</body></html>

	<html>
	<head>
	<script src="jquery.js"></script>
	<script>
	function appendText() {
	var txt1 = " Text. "; // Create text with HTML
	var txt2 = $(" ").text("Text."); // Create text with jQuery
	var txt3 = document.createElement("p");
	txt3.innerHTML = "Text."; // Create text with DOM
	$("body").append(txt1, txt2, txt3); // Append new elements
	}
	</script>
	</head>
	<body>
	 This is a paragraph.
	<button onclick="appendText()">Append text</button>
	</body>
	</html>

- **prepend( )** - Inserts content at the beginning of the selected elements

Use	jQuery prepend() method Inserts content into inside beggining of the selected elements.
Syntax	$(selector).prepend(content)
Parameter	• **Content:** content is Required parameter. Insert content inside first into selected element.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("p").prepend(" Online Web Development Tutorial");
	});
	});
	</script>
	</head>
	<body>
	 Viral Polishwala
	<button>Insert prepend end of p element</button>
	</body>
	</html>

- **after( )** - Inserts content after the selected elements

Use	jQuery after() method add into end of selected elements.
Syntax	$(selector).after(content)
Parameter	• **Content:** content is Required parameter. Insert content inside first into selected element.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("p").after(" Online Web Development Tutorial");
	});
	});
	</script>
	</head>
	<body>
	 Viral Polishwala
	
	<button>Insert after the p element</button>
	</body>
	</html>

- **before( )** - Inserts content before the selected elements

Use	jQuery before() method add into start of selected elements.
Syntax	$(selector).before(content)
Parameter	• **Content:** content is Required parameter. Insert content end into selected element.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("p").before(" Viral Polishwala");
	});
	});
	</script>
	</head>
	<body>
	 Online Web Development Tutorial
	<button>Insert after the p element</button>
	</body>
	</html>

JQUERY REMOVE

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

- **remove() -** Removes the selected element (and its child elements)

Use	jQuery remove() method Removes all selected element content like child elements or text.
Syntax	$(selector).remove()
Parameter	NA

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("div").remove();
	});
	});
	</script>
	</head>
	<body>
	 This is a paragraph.
	<button>Click to emply p elements</button>
	</body>
	</html>

- **empty() -** Removes the child elements from the selected element

Use	jQuery empty() method Removes all child elements from selected elements.
Syntax	$(selector).empty()
Parameter	NA

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("div").empty();
	});
	});
	</script>
	</head>
	<body>
	 This is a paragraph.
	<button>Click to emply p elements</button>
	</body>
	</html>

CSS: Styling and Dimension

jQuery has several methods for CSS manipulation. We will look at the following methods:

- **addClass( ) -** Adds one or more classes to the selected elements

Use	jQuery CSS addClass() method add one or more CSS class to selected elements.
Syntax	$(selector).addClass(classname)
Parameter	• **ClassName:** classname is Required parameter. add one or more css class names.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("p").addClass("param");
	});
	});
	</script>
	<style type="text/css">
	.param {
	font: 16px Verdana, Arial;
	font-weight:bold;
	color:pink;

	} </style> </head> <body> This is a first paragraph. This is a second paragraph. <button>Add class into first P element</button> </body> </html>

- **removeClass( ) -** Removes one or more classes from the selected elements

Use	jQuery removeClass() method removes one or all class from selected elements.
Syntax	$(selector).removeClass(classname)
Parameter	• **ClassName:** classname is Required parameter. Remove one or more css class names.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("p").removeClass("param");
	});
	});
	</script>
	<style type="text/css">
	.param {
	font: 16px Verdana, Arial;
	font-weight:bold;
	color:pink;
	}
	</style>
	</head>
	<body>
	 This is a first paragraph.
	 This is a second paragraph.
	<button>Add class into first P element</button>
	</body>
	</html>

- **toggleClass( ) -** Toggles between adding/removing classes from the selected elements

Use	jQuery toggleClass() method toggles add or remove one or more class from selected elements.

Syntax	$(selector).toggleClass(classname)
Parameter	• **ClassName:** classname is Required parameter. Replaces one or more css class names.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("p").toggleClass("param");
	});
	});
	</script>
	<style type="text/css">
	.param {
	font: 16px Verdana, Arial;
	font-weight:bold;
	color:pink;
	}
	</style>
	</head>
	
	<body>
	 This is a first paragraph.
	 This is a second paragraph.
	<button>ToogleClass add or remove class into P element</button>
	</body></html>

- **css( ) -** Sets or returns the style attribute

Use	jQuery css() method set one or more style properties value into selected elements.
Syntax	$(selector).class(name,value) OR $(selector).css({property:value,property:value, ...})
Parameter	• **Name:** name is Required parameter. It is CSS property. • **Value:** value is required parameter. It is respective property value.

Example	<html>
	<head>
	<script type="text/javascript" src="jquery.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	$("button").click(function(){
	$("p").css("background-color","pink");
	});
	});
	</script>

	</head> <body> This is CSS Refrence. <button>Set the css properties value into p elements</button> </body> </html>

	<html>
	<head>
	<script type="text/javascript" src="jquery-1.8.0.min.js"></script>
	<script type="text/javascript">
	$(document).ready(function(){
	  $("button").click(function(){
	  $("p").css({"background-color":"pink","font-size":"20px"});
	  });
	});
	</script>
	</head>
	<body>
	 This is CSS Refrence.
	<button>Set multiple css properties value into p elements</button>
	</body>
	</html>

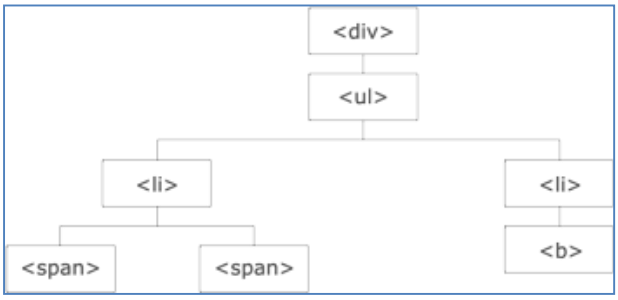
Traversing

jQuery is a very powerful tool which provides a variety of DOM traversal methods to help us select elements in a document randomly as well as in sequential method. jQuery traversing, which means "move through", are used to "find" (or select) HTML elements based on their relation to other elements. Start with one selection and move through that selection until you reach the elements you desire.

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

```
<div>
        <ul>
                <li>list <span> item 1</span>and<span> item 2</span></li>
              <li>list <b>item 3</b></li>
        </ul>
</div>
```

The image below illustrates a family tree. With jQuery traversing, you can easily move up (ancestors), down (descendants) and sideways (siblings) in the family tree, starting from the selected (current) element. This movement is called traversing - or moving through - the DOM.



Explanantion

- The element is the parent of <ul>, and an ancestor of everything inside of it
- The <ul> element is the parent of both <li> elements, and a child of
- The left <li> element is the parent of , child of and a descendant of
- The element is a child of the left <li> and a descendant of <ul> and
- The two <li> elements are siblings (they share the same parent)
- The right <li> element is the parent of <b>, child of <ul> and a descendant of
- The <b> element is a child of the right <li> and a descendant of <ul> and

ANCESTOR

An ancestor is a parent, grandparent, great-grandparent, and so on. With jQuery you can traverse up the DOM tree to find ancestors of an element.

Three useful jQuery methods for traversing up the DOM tree are: parent( ), parents( ), parentsUntil( ).

parent( )

Use	The **parent([selector])** method gets the direct parent of an element. If called on a set of elements, parent returns a set of their unique direct parent elements.
Syntax	selector.parent([selector])
Parameter	• **selector** – This is optional selector to filter the parent with.
Example	<html> <head> <style> .ancestors * { display: block; border: 2px solid lightgrey; color: lightgrey; padding: 5px; margin: 15px;

```
}
</style>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
   $("span").parent().css({"color": "red", "border": "2px solid red"});
});
</script>
</head>
<body>
<div class="ancestors">
 <div style="width:500px;">div (great-grandparent)
  <ul>ul (grandparent)
   <li>li (direct parent)
    <span>span</span>
   </li>
  </ul>
 </div>
 <div style="width:500px;">div (grandparent)
  <p>p (direct parent)
    <span>span</span>
  </p>
 </div>
</div>
</body>
</html>
```

```
<html>
  <head>
    <title>The jQuery Example</title>
    <script type = "text/javascript"
      src = " jquery.js "></script>
    <script type = "text/javascript" language = "javascript">
      $(document).ready(function(){
        $("p").parent().addClass('hilight');
      });
    </script>
    <style>
      .hilight { background:yellow; }
    </style>
  </head>
  <body>
    <scan>Top Element</scan>
    <div>
```

	sibling child sibling <scan>sibling</scan> </body> </html>

parents( )

Use	The **parents([selector])** method gets a set of elements containing the unique ancestors of the matched set of elements except for the root element.
Syntax	*selector*.parents([selector])
Parameter	• **selector** – This is optional selector to filter the ancestors with.

Example	<html>
	<head>
	<style>
	.ancestors * {
	display: block;
	border: 2px solid lightgrey;
	color: lightgrey;
	padding: 5px;
	margin: 15px;
	}
	</style>
	<script src="jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("span").parents().css({"color": "red", "border": "2px solid red"});
	});
	</script>
	</head>
	<body class="ancestors">body (great-great-grandparent)
	 div (great-grandparent)
	  ul (grandparent)
	  li (direct parent)
	span
	
	</body>
	<!-- The outer red border, before the body element, is the html element (also an ancestor)
	-->
	</html>

	<html>
	<head>
	<title>The jQuery Example</title>
	<script type = "text/javascript"
	src = " jquery.js "></script>
	<script type = "text/javascript" language = "javascript">
	$(document).ready(function(){
	var parentEls = $("p").parents()
	map(function () {
	return this.tagName;
	}).get().join(", ");
	$("b").append("" + parentEls + "");
	});
	</script>
	</head>
	<body>
	<scan>Top Element</scan>
	
	 Top division
	 First Sibling
	<scan>Second sibling</scan>
	 Third sibling
	
	Parents of elements are:
	
	</body>
	</html>

parentsUntil( )

Use	The parentsUntil() method returns all ancestor elements between two given arguments.
Syntax	selector.parentsUntil((selector))
Parameter	• **selector** – Two selectors values need to be given between which the jQuery may fired

Example	<html>
	<head>
	<style>
	.ancestors * {
	display: block;
	border: 2px solid lightgrey;
	color: lightgrey;
	padding: 5px;
	margin: 15px;
	}
	</style>
	<script src="jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("span").parentsUntil("div").css({"color": "red", "border": "2px solid red"});
	});
	</script>
	</head>
	<body class="ancestors"> body (great-great-grandparent)
	 div (great-grandparent)
	  ul (grandparent)
	  li (direct parent)
	span
	
	</body>
	</html>

DESCENDANTS

A descendant is a child, grandchild, great-grandchild, and so on. With jQuery you can traverse down the DOM tree to find descendants of an element.

Two useful jQuery methods for traversing down the DOM tree are:children( ) and find( )

children( )

Use	The **children([selector])** method gets a set of elements containing all of the unique immediate children of each of the matched set of elements.
Syntax	Selector.children([selector])
Parameter	• **selector** – This is an optional argument to filter out all the childrens. If not supplied then all the childrens are selected.

Example	<html><head><style>
	.descendants * {
	display: block;
	border: 2px solid lightgrey;
	color: lightgrey;
	padding: 5px;
	margin: 15px;
	}
	</style>
	<script src="jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("div").children().css({"color": "red", "border": "2px solid red"});
	});
	</script></head>
	<body>
	 div (current element)
	 p (child)
	span (grandchild)
	
	 p (child)
	span (grandchild)
	
	</body>
	</html>

	<html> <head> <title>The jQuery Example</title> <script type = "text/javascript" src = "jquery.js"></script> <script> $(document).ready(function(){ $("div").children(".selected").addClass("blue"); }); </script> <style> .blue { color:blue; } </style> </head> <body> Hello

	Hello Again And Again And One Last Time </body> </html>

find( )

Use	The **find(selector)** method searches for descendant elements that match the specified selector.
Syntax	*selector*.find(selector)
Parameter	• **selector** – The selector can be written using CSS 1-3 selector syntax.

Example	<html>
	<head>
	<style>
	.descendants * {
	display: block;
	border: 2px solid lightgrey;
	color: lightgrey;
	padding: 5px;
	margin: 15px;
	}
	</style>
	<script src="jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("div").find("span").css({"color": "red", "border": "2px solid red"});
	});
	</script>
	</head>
	<body>
	 div (current element)
	 p (child)
	span (grandchild)
	
	 p (child)
	span (grandchild)
	
	</body>
	</html>

	<html> <head> <title>The jQuery Example</title> <script type = "text/javascript" src = "jquery.js"></script> <script type = "text/javascript" language = "javascript"> $(document).ready(function() { $("p").find("span").addClass("selected"); }); </script> <style> .selected { color:red; } </style> </head> <body> Hello, how are you? Me? I'm good. </body> </html>

SIBLINGS

Siblings share the same parent. With jQuery you can traverse sideways in the DOM tree to find siblings of an element.

There are many useful jQuery methods for traversing sideways in the DOM tree: siblings() , next( ), nextAll( ), nextUntil( ), prev( ), prevAll( ), prevUntil( )

siblings( )

Use	The **siblings([selector])** method gets a set of elements containing all of the unique siblings of each of the matched set of elements.
Syntax	*selector*.siblings([selector])
Parameter	• **selector** – This is optional selector to filter the sibling Elements with.
Example	<html> <head> <style> .siblings * { display: block; border: 2px solid lightgrey; color: lightgrey; padding: 5px;

	margin: 15px;
	}
	</style>
	<script src="jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("h2").siblings().css({"color": "red", "border": "2px solid red"});
	});
	</script>
	</head>
	<body class="siblings">
	 div (parent)
	 p
	span
	 h2
	 h3
	 p
	
	</body></html>

	<html> <head> <title>The jQuery Example</title> <script type = "text/javascript" src = " jquery.js "></script> <script type = "text/javascript" language = "javascript"> $(document).ready(function(){ $("p").siblings('.selected').addClass("hilight"); }); </script> <style> .hilight { background:yellow; } </style> </head> <body> Hello Hello Again And Again </body></html>

next( )

Use	The **next([selector])** method gets a set of elements containing the unique next siblings of each of the given set of elements.
Syntax	*selector*.next([selector])

Parameter	• **selector** – The optional selector can be written using CSS 1-3 selector syntax. If we supply a selector expression, the element is unequivocally included as part of the object. If we do not supply one, the element would be tested for a match before it was included.

Example	<html>
	<head>
	<style>
	.siblings * {
	display: block;
	border: 2px solid lightgrey;
	color: lightgrey;
	padding: 5px;
	margin: 15px;
	}
	</style>
	<script src="jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("h2").next().css({"color": "red", "border": "2px solid red"});
	});
	</script>
	</head>
	<body class="siblings">
	 div (parent)
	 p
	span
	 h2
	 h3
	 p
	
	</body>
	</html>

	<html> <head> <title>The jQuery Example</title> <script type = "text/javascript" src = "jquery.js"></script> <script type = "text/javascript" language = "javascript"> $(document).ready(function(){ $("p").next(".selected").addClass("hilight"); }); </script> <style>

	.hilight { background:yellow; } </style> </head> <body> Hello Hello Again And Again </body> </html>

nextAll( )

Use	The **nextAll([selector])** method finds all sibling elements after the current element.
Syntax	selector.nextAll([selector])
Parameter	• **selector** – The optional selector can be written using CSS 1-3 selector syntax. If we supply a selector then result would be filtered out.

Example	<html>
	<head>
	<style>
	.siblings * {
	display: block;
	border: 2px solid lightgrey;
	color: lightgrey;
	padding: 5px;
	margin: 15px;
	}
	</style>
	<script src=" jquery.js "></script>
	<script>
	$(document).ready(function(){
	$("h2").nextAll().css({"color": "red", "border": "2px solid red"});
	});
	</script>
	</head>
	<body class="siblings">
	 div (parent)
	 p
	span
	 h2
	 h3
	 p
	
	</body>

	</html>
	<html> <head> <title>The jQuery Example</title> <script type = "text/javascript" src = "jquery.js"></script> <script type = "text/javascript" language = "javascript"> $(document).ready(function(){ $("div:first").nextAll().addClass("hilight"); }); </script> <style> .hilight { background:yellow; } </style> </head> <body> first sibling child sibling sibling </body> </html>

nextUntill( )

Use	The nextUntil() method returns all next sibling elements between two given arguments.
Syntax	selector.nextUntil([selector])
Parameter	• **selector** – name of selector to start with and and work until the second selector.

Example	<html>
	<head>
	<style>
	.siblings * {
	display: block;
	border: 2px solid lightgrey;
	color: lightgrey;
	padding: 5px;
	margin: 15px;
	}
	</style>
	<script src="jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("h2").nextUntil("h6").css({"color": "red", "border": "2px solid red"});

	}); </script> </head> <body class="siblings"> div (parent) p span h2 h3 h4 h5 h6 p </body> </html>

prev( )

Use	The **prev([selector])** method gets the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.
Syntax	selector.prev([selector])
Parameter	• **selector** – This is optional selector to filter the previous Elements with.
Example	<html> <head> <title>The jQuery Example</title> <script type = "text/javascript" src = "jquery.js"></script> <script type = "text/javascript" language = "javascript"> $(document).ready(function(){ $("p").prev(".selected").addClass("hilight"); }); </script> <style> .hilight { background:yellow; } </style> </head> <body> Hello Hello Again And Again </body> </html>

prevAll( )

<table>
<tr><td>Use</td><td>The prev([selector]) method gets the immediately preceding sibling of each element in the set of matched elements, optionally filtered by a selector.</td></tr>
<tr><td>Syntax</td><td>$(selector).prevAll([filter])</td></tr>
<tr><td>Parameter</td><td>• selector – This is optional selector to filter the previous Elements with.
• Filter - Optional. Specifies a selector expression to narrow down the search for previous siblings</td></tr>
<tr><td>Example</td><td><html>
<head>
<style>
.siblings * {
display: block;
border: 2px solid lightgrey;
color: lightgrey;
padding: 5px;
margin: 15px;
}
</style>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
$("li.start").prevAll().css({"color": "red", "border": "2px solid red"});
});
</script>
</head>
<body>
<div style="width:500px;" class="siblings">
ul (parent)
li (the previous sibling of li with class name "start")
li (the previous sibling of li with class name "start")
li (the previous sibling of li with class name "start")
<li class="start">li (sibling with class name "start")
li (sibling)
li (sibling)

</div>
<p>In this example, we return all elements that are previous siblings of the li element with class name "start".</p>
</body>
</html></td></tr>
<tr><td></td><td><html>
<head>
<title>The jQuery Example</title>
<script type = "text/javascript" src = " jquery.js"></script></td></tr>
</table>

	<script type = "text/javascript" language = "javascript"> $(document).ready(function(){ $("div:last").prevAll().addClass("hilight"); }); </script> <style> .hilight { background:yellow; } </style> </head> <body> first sibling child sibling sibling </body> </html>

prevUntil( )

Use	The prevUntil() method returns all previous sibling elements between the selector and stop.
Syntax	$(selector).prevUntil(stop,filter)
Parameter	• **selector** – This is optional selector to filter the previous Elements with. • **Filter** - Optional. Specifies a selector expression to narrow down the search for sibling elements between the selector and stop • **Stop** – Optional. A selector expression, element or jQuery object indicating where to stop the search for previous matching siblings elements

Example	<html>
	<head>
	<style>
	.siblings * {
	display: block;
	border: 2px solid lightgrey;
	color: lightgrey;
	padding: 5px;
	margin: 15px;
	}
	</style>
	<script src="jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("li.start").prevUntil("li.stop").css({"color": "red", "border": "2px solid red"});
	});
	</script>

	</head> <body> ul (parent) <li class="stop">li (sibling with class name "stop") - li (the previous sibling of li with class name "start") - li (the previous sibling of li with class name "start") - li (the previous sibling of li with class name "start") <li class="start">li (sibling with class name "start") - li (sibling) - li (sibling) In this example, we return all previous sibling elements between the li element with class name "start" and the li element with class name "stop". </body> </html>

FILTERING

The three most basic filtering methods are first(), last() and eq(), which allow you to select a specific element based on its position in a group of elements. Other filtering methods, like filter() and not() allow you to select elements that match, or do not match, a certain criteria.

first( )

Use	The first() method returns the first element of the selected elements.
Syntax	$(selector).first()
Parameter	NA
Example	<html> <head> <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script> <script> $(document).ready(function(){ $("div p").first().css("background-color", "yellow"); }); </script> </head> <body> Welcome to My Homepage This is a paragraph in a div. This is a paragraph in a div.

	</div> <div style="border:1px solid black"> <p>This is a paragraph in another div.</p> <p>This is a paragraph in another div.</p> </div> <p>This is also a paragraph.</p> </body></html>

last( )

Use	The last() method returns the last element of the selected elements.
Syntax	$(selector).last()
Parameter	NA
Example	<html> <head> <script src="jquery.js"></script> <script> $(document).ready(function(){ $("div p").last().css("background-color", "yellow"); }); </script> </head> <body> Welcome to My Homepage This is a paragraph in a div. This is a paragraph in a div. This is a paragraph in another div. This is a paragraph in another div. This is also a paragraph. </body> </html>

eq( )

Use	The eq() method returns an element with a specific index number of the selected elements.
Syntax	$(selector).eq(index)
Parameter	• **Index -** Required. Specifies the index of the element. Can either be a positive or negative number.
Example	<html> <head> <script src="jquery.js"></script>

	<script> $(document).ready(function(){ $("p").eq(1).css("background-color", "yellow"); }); </script> </head> <body> Welcome to My Homepage My name is Donald (index 0). Donald Duck (index 1). I live in Duckburg (index 2). My best friend is Mickey (index 3). </body> </html>

not( )

Use	The not() method returns elements that do not match a certain criteria. This method lets you specify a criteria. Elements that do not match the criteria are returned from the selection, and those that match will be removed.This method is often used to remove one or more elements from a group of selected elements.
Syntax	*$(*selector).not*(*criteria,function(index)*)*
Parameter	• **criteria** – Optional. Specifies a selector expression, a jQuery object or one or more elements to be removed from a group of selected elements. • **Function(index)**- Optional. Specifies a function to run for each element in a group. If it returns true, the element is removed. Otherwise, the element is kept.
Example	<html> <head> <script src="jquery.js"></script> <script> $(document).ready(function(){ $("p").not(".intro").css("background-color", "yellow"); }); </script> </head> <body> Welcome to My Homepage My name is Donald. I live in Duckburg. I love Duckburg. My best friend is Mickey. </body> </html>

find( )

Use	The find() method returns descendant elements of the selected element. A descendant is a child, grandchild, great-grandchild, and so on.
Syntax	$(*selector*).find(*filter*)
Parameter	• **Filter** - Required. A selector expression, element or jQuery object to filter the search for descendants.

Example	<html>
	<head>
	<style>
	.ancestors * {
	display: block;
	border: 2px solid lightgrey;
	color: lightgrey;
	padding: 5px;
	margin: 15px;
	}
	</style>
	<script src="jquery.js"></script>
	<script>
	$(document).ready(function(){
	$("ul").find("span").css({"color": "red", "border": "2px solid red"});
	});
	</script>
	</head>
	<body class="ancestors">body (great-grandparent)
	 div (grandparent)
	  ul (direct parent)
	  li (child)
	span (grandchild)
	
	</body></html>

filter( )

Use	The filter() method returns elements that match a certain criteria. This method lets you specify criteria. Elements that do not match the criteria are removed from the selection, and those that match will be returned. This method is often used to narrow down the search for an element in a group of selected elements.
Syntax	$(*selector*).filter(*criteria,*function*(index)*)
Parameter	• **criteria** – Optional. Specifies a selector expression, a jQuery object or one or more elements to be removed from a group of selected elements.

	• **Function(index)**- Optional. Specifies a function to run for each element in a group. If it returns true, the element is removed. Otherwise, the element is kept.
Example	<html> <head> <script src="jquery.js"></script> <script> $(document).ready(function(){ $("p").filter(".intro").css("background-color", "yellow"); }); </script> </head> <body> Welcome to My Homepage My name is Donald. I live in Duckburg. I love Duckburg. My best friend is Mickey. </body> </html>

Chapter – 3

JSON : (Java Script Object Notation)

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.

3.1 Concept and Features of JSON

Concepts

JSON is a lightweight text-based open standard data-interchange format. It is human readable. JSON is derived from a subset of JavaScript programming language (Standard ECMA-262 3rd Edition—December 1999). It is entirely language independent and can be used with most of the modern programming languages.

JSON is often used to serialize and transfer data over a network connection, for example between the web server and a web application. In computer science, serialization is a process to transforming data structures and objects in a format suitable to be stored in a file or memory buffer or transmitted over a network connection. Later on, this data can be retrieved. Because of the very nature of the JSON, it is useful for storing or representing semi structured data

JSON is a standard and is specified on RFC4627 on IETF (International Engineering Task Force). The specification is made by **Doglus Crockford** on July 2006.

JSON files are saved with **.json** extension. Internet media type of JSON is "application/json".

Features

- JSON is light-weight
- JSON is language independent which works with most of the modern days programming languages.
- It is easy to read and write
- Text-based, human readable data exchanged format
- It is scalable.

Some other features can be narrated as simplicity, openness, self-describing, internationalization, extensibility and interoperability.

3.2 Similarities and Differences between JSON and XML

JSON stands for JavaScript Object Notation, whereas XML stands for Extensive Markup Language. Nowadays, JSON and XML are widely used as data interchange formats, and both have been acquired by applications as a technique to store structured

data. The purpose of the comparison it's definitely not in the line of which is better, rather we will try to understand which one is suitable for storing specific kind of data.

Similarities between JSON and XML

- Both JSON and XML are human readable language.
- Both JSON and XML support unicode, thus any human written language can be written in JSON and XML documents.
- Both JSON and XML can be parsed.
- The data in both JSON and XML can be fetched using XMLHttpRequest.

Differences between JSON and XML

JSON (JavaScript Object Notation)	XML (eXtensible Markup Language)
JSON is simple and easier to read and write	XML is verbose and less readable
JSON doesn't use end tag	In XML, the end tag is mandatory
JSON supports array thus it is easy to transfer a big chunk of homogeneous data items using JSON	XML doesn't support array
JSON is easier to parse and can be parsed to ready-to-use JavaScript object	XML is difficult to parse than JSON
JSON is short	XML document is lengthy, verbose and redundant
JSON is less secure than XML	XML is more secured than JSON
JSON file is more readable than XML because it is short and to the point.	XML file is big and filled with user-defined tags, thus less-readable
JSON is data-oriented	XML is document-oriented
It doesn't provide display capabilities.	It provides the display capability because it is a markup language.
Example: ["student" : { "sname" : "Ashish", "rno" : "101", "div" : "I" },	<student> <sname>Ashish</sname> <rno>101</rno> I </student>

{
"sname" : "Binita",
"rno" : "102",
"div" : "I"
}
]	

3.3 JSON objects (with strings and numbers)

JSON objects are formed using the curly braces '{' which surrounds its data. These are written in a key-value pairing format. It is to be noted that the keys must have to be a string, and the value in the key-value pair must have to be anyone among the data types of JSON like: string, number, boolean, array, object, null. Moreover, the key and value are separated by a colon (:) and where there are multiple key-value pairs, all of these are separated by a comma (,)

Creating JSON object

There are 3 ways to create objects in JSON, the type of creation depends on how we create and initialize them. They are as follows:

- **Empty Object**

To create an empty JSON object the syntax is as follows:

Syntax:

```
var objname={};
```

Example:

```
var empobj={};
```

- **New Object**

Another way to create and object in JSON is as follows where you can use new operator along with object method.

Syntax:

```
var objname=new Object();
```

Example:

```
var jsonobj=new Object();
```

- **Object with attribute**

Another way of creating objects is by assigning multiple attributes where the key will be the string, and the value can be either string or a numeric value. Let us take an example where a JSON file will store students' data and their marks.

Example:

var students = {“name” : “Vishal”, “IS” : 35, “IOT” : 42 }

Accessing JSON object value

To access the value of JSON object one can use the square bracket notations as follows for the above mentioned student object of JSON:

ismarks=students[“IS”];

Nesting JSON object

To nest a JSON object within another one, we can use it like follows:

```
var students = {
“name” : “Vishal”,
“course” : “BCA”,
“marks”: {
        “IS” : 45;
        “IOT” : 41;
        “JAVA” : 35;
        }
}
```

Deleting JSON object

It is also possible to delete any properties of an object created within JSON using the delete keyword. The dot (.) or period operator needs to be used to access the nested object elements. The more you access the nested elements, the more you have to add dots. Let suppose you want to delete the “JAVA” from that previous JSON example so that you can use the delete keyword followed by this

delete students.marks.JAVA;

Creating JSON object (with string)

We can use the string as value for the object attribute. It is a double-quoted set of character(s) (Unicode) having backslash escaping. It denotes a single character string value, having a string length of one when you say character. Example of the same are already given in the example above with name, course attributes of JSON object students.

Creating JSON object (with numbers)

We can use the number as value for the object attribute. It provides floating point double precision data format and does not allow octal or hexadecimal number formats. Moreover, it does not assign NaN or infinity to its variables. It can be in 3 categories: Integer, Fraction, Exponent. Example of the same is given in the above JSON object named students with attributes IS, IOT, JAVA.

3.4 JSON Arrays and their examples

An array is a collection of multiple values under one name and is enclosed within square braces as in other programming languages and separated by commas. JSON array represents ordered list of values. JSON array can store multiple values. It can store string, number, boolean or object in JSON array. In JSON array, values must be separated by comma. The [ ] (square bracket) represents JSON array. An array of JSON can be array of strings, numbers, boolean, objects etc. Following are certain examples of JSON arrays:

3.4.1 Arrays of String

An array of string values can be represented as follows for the array days in JSON:

"days": ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"] ;

3.4.2 Arrays of Numbers

An array of number values can be represented as follows for the array in JSON:

"maxtempweek": [23, 24, 22, 22, 23, 24, 21];

3.4.3 Arrays of Boolean

An array of boolean values can be represented as follows for the array in JSON:

"rainydayweek": [true, false, false, true, true, false, true];

3.4.4 Arrays of Objects

JSON array can hold the JSON object as an array elements. Check it in the example below:

```
{"students":[
  {"name":"Manoj", "email":"mjpatel@gmail.com", "rno":57},
  {"name":"Mansi", "email":"mansi124@gmail.com", "rno":45},
  {"name":"Rivan", "email":"rivannaik@gmail.com", "age":15}
]}
```

3.4.5 Multi-dimensional Arrays

JSON multi-dimensional array works in following manner:

- **Creation**

An array of array or saving an array within other one is considered as multi-dimensional JSON array.

```
var webinfo = {
    "name" : "blogger",
    "users_auth" : [
            [ "admins", "7", "7", "7"],
            [ "editors", "4", "4", "1"],
    ]
}
```

- **Iteration**

To iterate through the multi-dimensional array in JSON simple for loop will work.

Code	Output

for (i in webinfo .users_auth)
{
for (j in webinfo.users_auth[i])
{
x = webinfo.users_auth [i][j];
console.log(x);
}
}
	admins
	7
	7
	7
	editors
	4
	4
	1

3.5 JSON Comments

JSON is a data-only format. Comments in the form //, #, or /* */, which are used in popular programming languages, are not allowed in JSON. You can add comments to JSON as custom JSON elements that will hold your comments, but these elements will still be data. To do this, you need to add an element to your JSON file, such as "_comment," which will contain your comment as a value of it. The JSON API endpoint must ignore this particular JSON comment element. In this JSON comment example, we have included two comment elements in the JSON data.

```
{
 "Id": 1007,
 "Customer": "Thomas",
 "Quantity": 5,
 "Price": 100.00,
 "Date":"12-11-21",
  "//first_comment":  "Customer Bill.",
 "//second_comment": "generated with 5 main attributes."
}
```

Douglas Crockford, who popularized the JSON data format, deliberately removed comments from JSON to prevent misuse of the JSON format and keep it as a data-only format. He describes the reason he removed the comments from the JSON as follows:

" I removed comments from JSON because I saw people using them to store parsing directives, which would break compatibility."

Therefore, the only option for adding comments to JSON is a workaround to use custom elements to store comments in a JSON file.

Chapter – 4

AJAX (Asynchronous Javascript And XML)

AJAX is not a new technology, in fact, Ajax is not even really a technology at all. It is getting tremendous industry momentum and several tool kit and frameworks are emerging. AJAX is just a term to describe the process of exchanging data from a web server asynchronously through JavaScript, without refreshing the page. But at the same time, AJAX has browser incompatibility and it is supported by JavaScript, which is hard to maintain and debug.

4.1 Fundamentals of AJAX Technology

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 is just a means of loading data from the server and selectively updating parts of a web page without reloading the whole page. So it is fast.

Basically, what Ajax does is make use of the browser's built-in XMLHttpRequest (XHR) object to send and receive information to and from a web server asynchronously, in the background, without blocking the page or interfering with the user's experience.

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 used for creating interactive web applications. It has become so popular that you hardly find an application that doesn't use Ajax to some extent. The example of some large-scale Ajax-driven online applications are: Gmail, Google Maps, Google Docs, YouTube, Facebook, Flickr, and so many other applications.

Following are certain feature/characteristics of AJAX.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.

- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

AJAX is based on the following open standards −

- Browser-based presentation using HTML and Cascading Style Sheets (CSS).
- Data is stored in XML format and fetched from the server.
- Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
- JavaScript to make everything happen.

AJAX cannot work independently. It is used in combination with other technologies to create interactive webpages. Ajax is based on HTML, CSS, JavaScript, DOM, and XML.

HTML/CSS

HTML/CSS is website markup language for defining web page layout, such as fonts style and colors. CSS allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript.

JavaScript

JavaScript is a web scripting language. JavaScript special object XMLHttpRequest that was designed by Microsoft. XMLHttpRequest provides an easy way to retrieve data from web server without having to do full page refresh. Web page can update just part of the page without interrupting what the users are doing.

Document Object Model

Document Object Model (DOM) method provides a tree structure as a logical view of web page. It is an API for accessing and manipulating structured documents. It represents the structure of XML and HTML documents.

XML or JSON

They are used for carrying data to and from server. XML is a format for retrieve any type of data, not just XML data from the web server. However, you can use other formats such as Plain text, HTML or JSON (JavaScript Object Notation). and it supports protocols HTTP and FTP. XMLHttpRequest is used heavily in AJAX programming. It is glue for the whole AJAX operation. JavaScript object that performs asynchronous interaction with the server. JSON (Javascript Object Notation) is like XML but short and faster than XML.

Advantages:

- Speed is enhanced as there is no need to reload the page again.
- AJAX make asynchronous calls to a web server, this means client browsers avoid waiting for all the data to arrive before starting of rendering.
- Form validation can be done successfully through it.
- Bandwidth utilization – It saves memory when the data is fetched from the same page.
- More interactive.

Disadvantages:

- Ajax is dependent on Javascript. If there is some Javascript problem with the browser or in the OS, Ajax will not support.
- Ajax can be problematic in Search engines as it uses Javascript for most of its parts.
- Source code written in AJAX is easily human readable. There will be some security issues in Ajax.
- Debugging is difficult.
- Problem with browser back button when using AJAX enabled pages.

4.1.1 Difference between Synchronous and Asynchronous Web Application

Let's understand the transmission of data through the synchronous and synchronous manner over the applications.

In **Synchronous Transmission**, data is sent in form of blocks or frames. This transmission is the full duplex type. Between sender and receiver, the synchronization is compulsory. In Synchronous transmission, there is no gap present between data. It is more efficient and more reliable than asynchronous transmission to transfer the large amount of data.

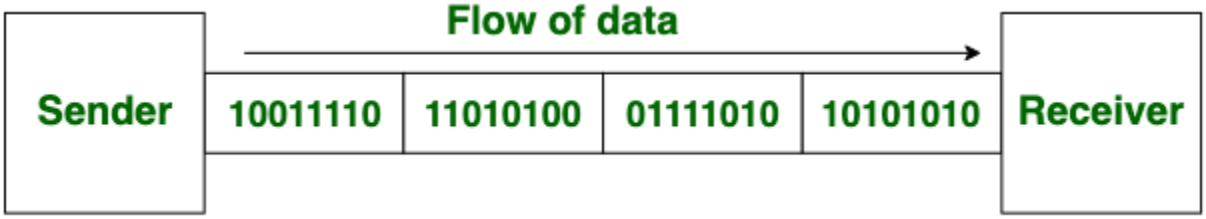


Fig: 4.1 Synchronous Transmission

In **Asynchronous Transmission**, data is sent in form of byte or character. This transmission is the half duplex type transmission. In this transmission start bits and stop bits are added with data. It does not require synchronization.

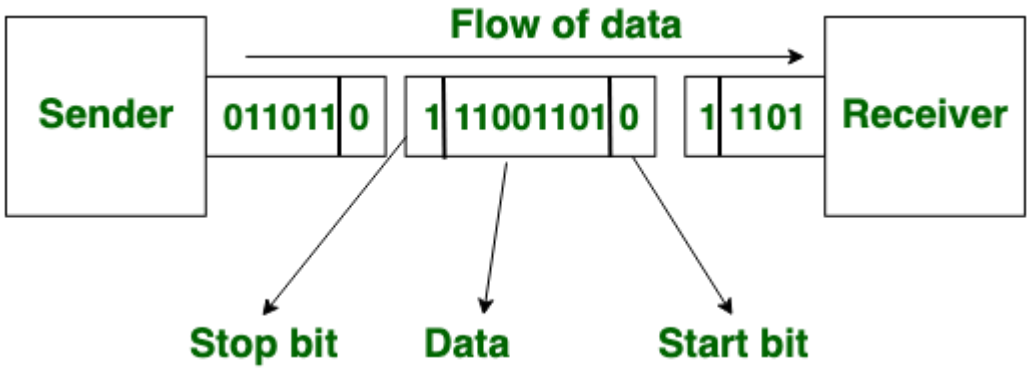


Fig: 4.2 Asynchronous Transmission

Now let's see how basic classic (Synchronous) web application and AJAX (Asynchronous) web application model differs from each other.

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

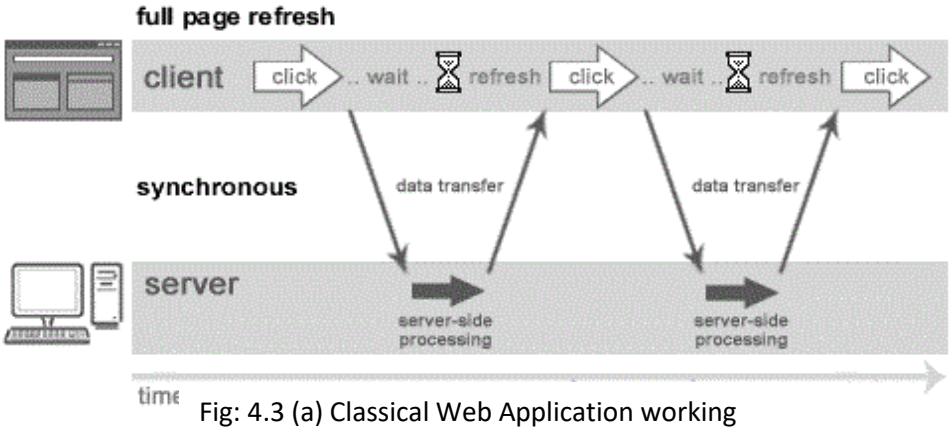


Fig: 4.3 (a) Classical Web Application working synchronous manner

As you can see in the above image, full page is refreshed at request time and user is blocked until request completes. Same can be explained as follows:

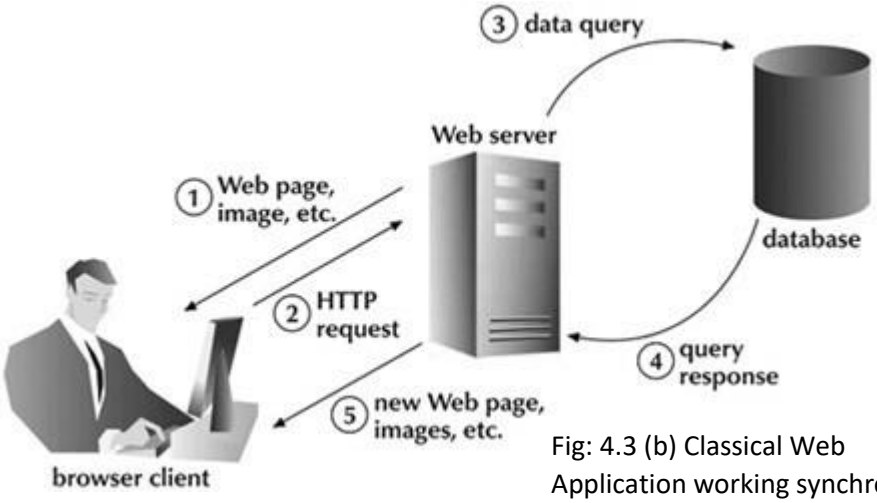


Fig: 4.3 (b) Classical Web Application working synchronous manner

In classic Web-based applications, a user input triggers a number of resource requests. Once the requests have been answered by the server, no further communication takes place until the user's next input. Such communication between client and server is known as **synchronous communication**.

Here is an explanation of classic synchronous communication passing between a browser and a Web server:

1. The user clicks a UI control in a browser-based web application.
2. The browser converts the user's action into one or more HTTP requests and passes them along to the Web-application server.
3. The application server responds to the user's requests by returning the requested data to the user. At this point the application is updated and the synchronous communication loop is complete. A new synchronous communication loop will begin when the user next clicks a UI control in their browser.

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 operation also. In such case, javascript engine of the browser is not blocked.

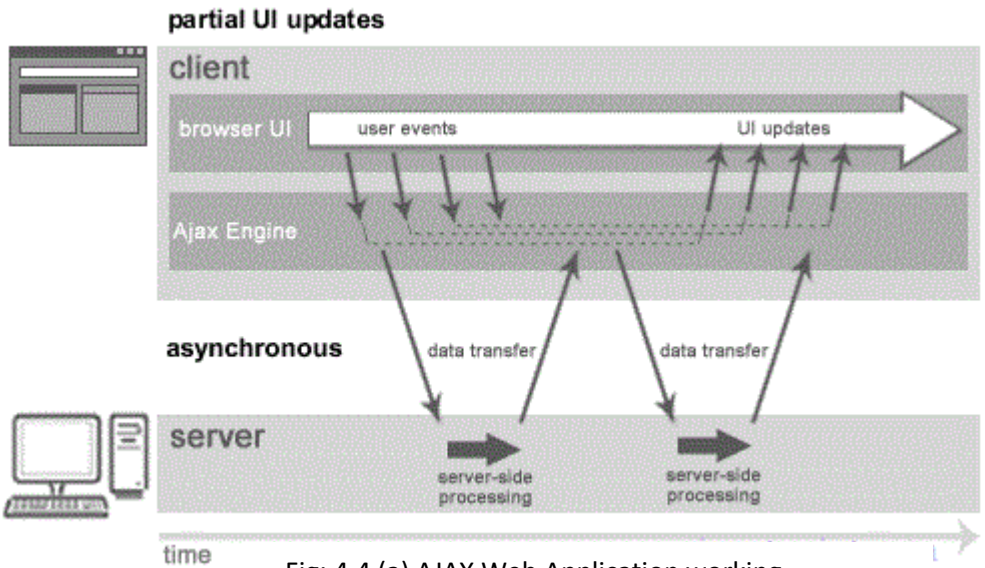


Fig: 4.4 (a) AJAX Web Application working Asynchronous manner

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.

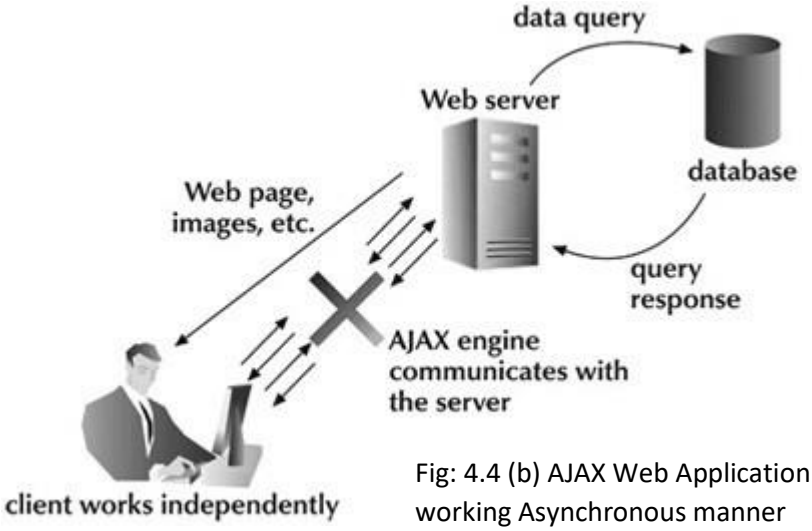


Fig: 4.4 (b) AJAX Web Application working Asynchronous manner

Here is an explanation of AJAX kind of asynchronous communication passing between a browser and a Web server:

1. User sends a request from the webpage and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.
3. Server fetch the data from the database using JSP, PHP, Servlet, ASP.net etc file.
4. Data is retrieved and view the data in webpage using HTML.

Briefly the difference can be given as follows:

Synchronous communication is limited due to the lapses in application updates that are presented to the user at regular intervals. Even if a synchronous application is designed so that it automatically refreshes information from the application server at regular intervals (for example, every 12 seconds), there will still be consistent periods of delay between data refreshes. For many applications, such update delays don't present an issue because the data they manage don't change often. Some application types however, for example stock-trading applications, rely on continuously updated information to provide optimum functionality and usability to their users.

Web 2.0 web-based applications address this issue by relying on asynchronous communication. Asynchronous applications deliver continuously updated application data to users. This is achieved by separating client requests from application updates. Multiple asynchronous communications between client and server may occur simultaneously or in parallel with one another.

While asynchronous communication delivers tremendous value to users, it presents a serious challenge to software-testing tool vendors who have difficulty emulating it with traditional test scripts.

4.1.2 XML HttpRequest technology

XMLHttpRequest object is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side. Mostly all browser platform support XMLHttpRequest object to make HTTP requests.

The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.

Using Ajax XMLHttpRequest object you can make many things easier. So many new things can't possible using HEAD request. This object allows you to making HTTP requests and receive responses from the server in the background, without requiring the user to submit the page to the server (without round trip process).

Using DOM to manipulate received data from the server and make responsive contents are added into live page without user/visual interruptions.

Using this object, you can make very user interactive web application. An object of XMLHttpRequest is used for asynchronous communication between client and server.

It performs following operations:

- Sends data from the client in the background.
- Receives the data from the server.
- Updates the webpage without reloading it.

We will see it in details in next topic.

4.2 XML HttpRequest

Since Ajax requests are usually asynchronous, execution of the script continues as soon as the Ajax request is sent, i.e. the browser will not halt the script execution until the server response comes back.

Sending Request and Receiving Response

For making the AJAX communication possible between the client and server; first of all the object of **XMLHttpRequest** must be initiated. It is possible by creating the variable with new keyword as follows in syntax:

```
var reqvariable = new XMLHttpRequest();
```

By following the above syntax, we create a variable call request1 understanding the concept here as follows:

```
var request1 = new XMLHttpRequest();
```

Now, the next step in sending the request to the server is to instantiating the newly-created request object using the open() method of the XMLHttpRequest object.

The **open()** method typically accepts two parameters— the HTTP request method to use, such as "GET", "POST", etc., and the URL to send the request to, as per the following syntax:

```
open( "method",  "URL" [ , asynchronous_flag [ , "username" [ , "password" ] ] ] )
```

Description of the parameters are as follows:

Parameter	Required?	Description
method	required	Specifies the HTTP method. Valid value GET, POST, HEAD, PUT, DELETE, OPTIONS
URL	required	Specifies URL that may be either relative parameter or absolute full URL.
asynchronous_flag	optional	Specifies whether the request should be handled asynchronously or not. Valid value TRUE, FALSE Default value FALSE TRUE means without waiting for a response, next code processing to execution queue on after the send() method. FALSE means wait for a response after the next code processing.
username	optional	Specifies username of authorize user otherwise set to null.
password	optional	Specifies password of authorize user otherwise set to null.

It migjt be written as follows for the above variable:

```
request1.open(“GET”,”info.txt”); OR request1.open(“POST”,”students.php”);
```

The file mentioned in URL can be of any kind, like .txt or .xml, or server-side scripting files, like .php or .asp, which can perform some actions on the server (e.g. inserting or reading data from database) before sending the response back to the client.

And finally **send** the request to the server using the send() method of the XMLHttpRequest object. For the above variable it might be called as:

```
request1.send(); OR request.send(body);
```

The send() method accepts an optional *body* parameter which allow us to specify the request's body. This is primarily used for HTTP POST requests, since the HTTP GET request doesn't have a request body, just request headers.

4.2.1 Properties: (onReadyStateChange, readyState, responseText, responseXML)

An object of XMLHttpRequest is used for asynchronous communication between client and server. XMLHttpRequest (XHR) is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side.

The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.

It performs following operations:

- Sends data from the client in the background.
- Receives the data from the server.
- Updates the webpage without reloading it.

It has following properties:

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. After you have created the XMLHttpRequest object, but before you have called the open() method. • 1 OPENED open is called but send() is not called. After you have called the open() method, but before you have called send(). • 2 HEADERS_RECEIVED send() is called, and headers and status are available. • 3 LOADING Downloading data; responseText holds the data. After the browser has established a communication with the server, but before the server has completed the response. • 4 DONE The operation is completed fully. After the request has been completed, and the response data has been completely received from the server.
responseText	It returns the response as text

responseXML	Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.
Status	Returns the status as a number 200: "OK" 403: "Forbidden" 404: "Not Found"
statusText	Returns the status as a string (e.g., "Not Found" or "OK")

4.2.2 XMLHttpRequest Methods: (open(), send(), setRequestHeader())

Method	Description
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
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.

Note: * indicated the methods are explained in detail in the topic 4.2 beginning.

4.3 Working of AJAX and its architecture

Working of AJAX

Ajax communicates with the server by using XMLHttpRequest Object. User send request from User Interface and JavaScript call goes to the XMLHttpRequest Object after that XMLHttp request is sent to the XMLHttpRequest Object. At that time server interacts with the database using php, servelet, ASP.net etc. The data is retrieved then the server sends data in the form of XML or Jason data to the XMLHttpRequest Callback function.

Then HTML and CSS displayed the Data on the browser. These all above process we discuss in point by point format for better understanding.

As you can see in the figure below, XMLHttpRequest object plays a important role.

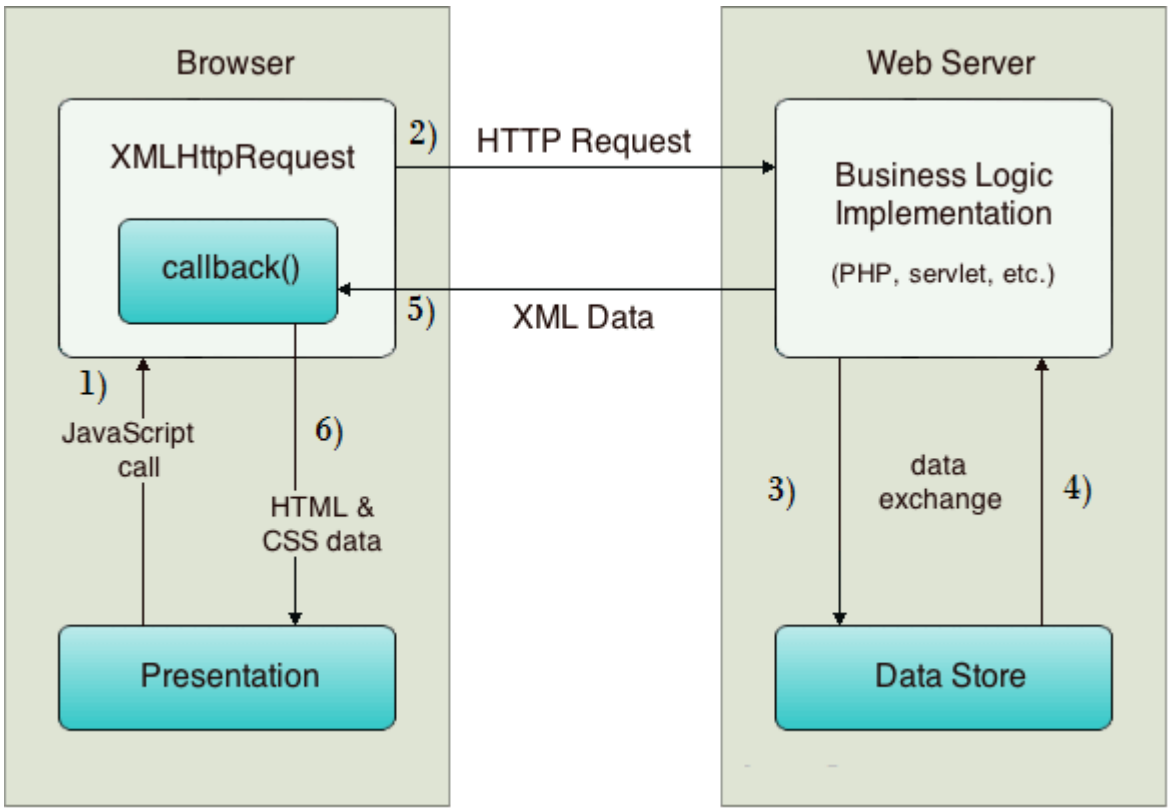


Fig: 4.5 AJAX working example

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.

In Ajax model, there is an Ajax engine involved in between the user and the server, which eliminates the to and fro from the user to the server and back. This Ajax engine is written in JavaScript and is in a hidden frame. It handles the user front by communicating to the user as well as handles the server front by itself. This way, the end user barely faces a waiting period.

Architecture of AJAX

- AJAX Wen application model uses JavaScript and XMLHttpRequest object for asynchronous data exchange. The JavaScript uses the XMLHttpRequest object to exchange data asynchronously over the client and server.

`

- AJAX Web application model resolve the major problem of synchronous request-response model of communication associated with classical Web application model, which keeps the user in waiting state and does not provide the best user experience.

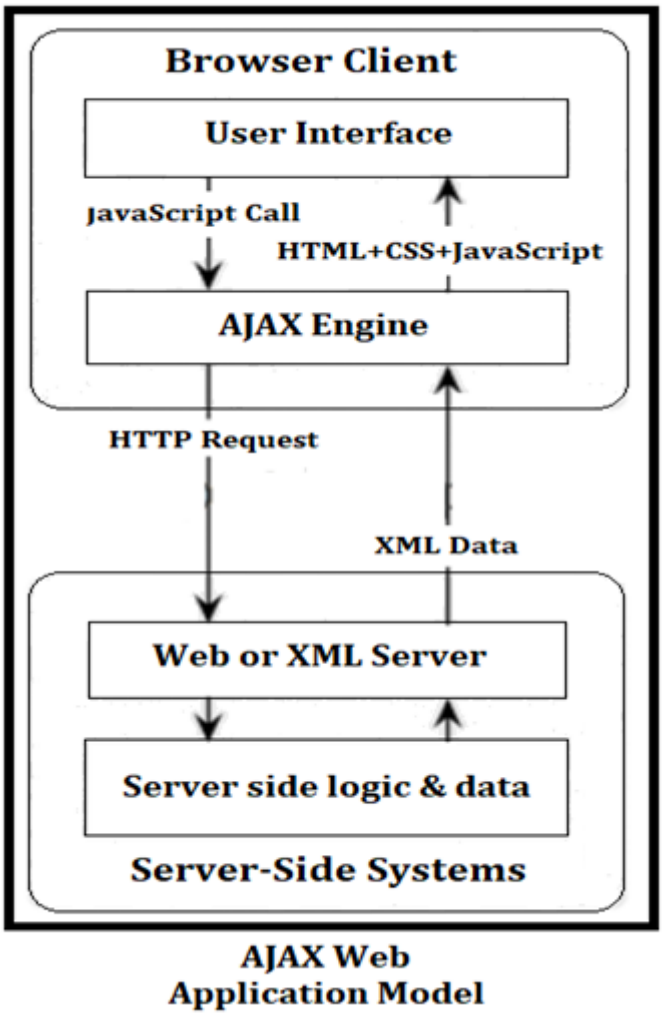


Fig: 4.6 AJAX Application Model

- AJAX, a new approach to Web applications, which is based on several technologies that help in developing applications with better user experience. It uses JavaScript and XML as the main technology for developing interactive Web applications.
- The AJAX application eliminates the start-stop-start-stop nature or the click, wait, and refresh criteria of the client-server interaction by introducing intermediary layer between the user and the Web server.
- Instead of loading the Web page at the beginning of the session, the browser loads the AJAX engine written in JavaScript.
- Every user action that normally would generate an HTTP request takes the form of a JavaScript call to the AJAX Engine.

- The server response comprises data and not the presentation, which implies that the data required by the client is provided by the server as the response, and presentation is implemented on that data with the help of markup language from Ajax engine.
- The JavaScript does not redraw all activities instead only updates the Web page dynamically.
- In JavaScript, it is possible to fill Web forms and click buttons even when the JavaScript has made a request to the Web server and the server is still working on the request in the background. When server completes its processing, code updates just the part of the page that has changed. This way client never has to wait around. That is the power of asynchronous requests.
- AJAX Engine between the client and the application, irrespective of the server, does asynchronous communication. This prevents the user from waiting for the server to complete its processing.
- The AJAX Engine takes care of displaying the UI and the interaction with the server on the user's behalf.

Chapter – 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.

5.1 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.

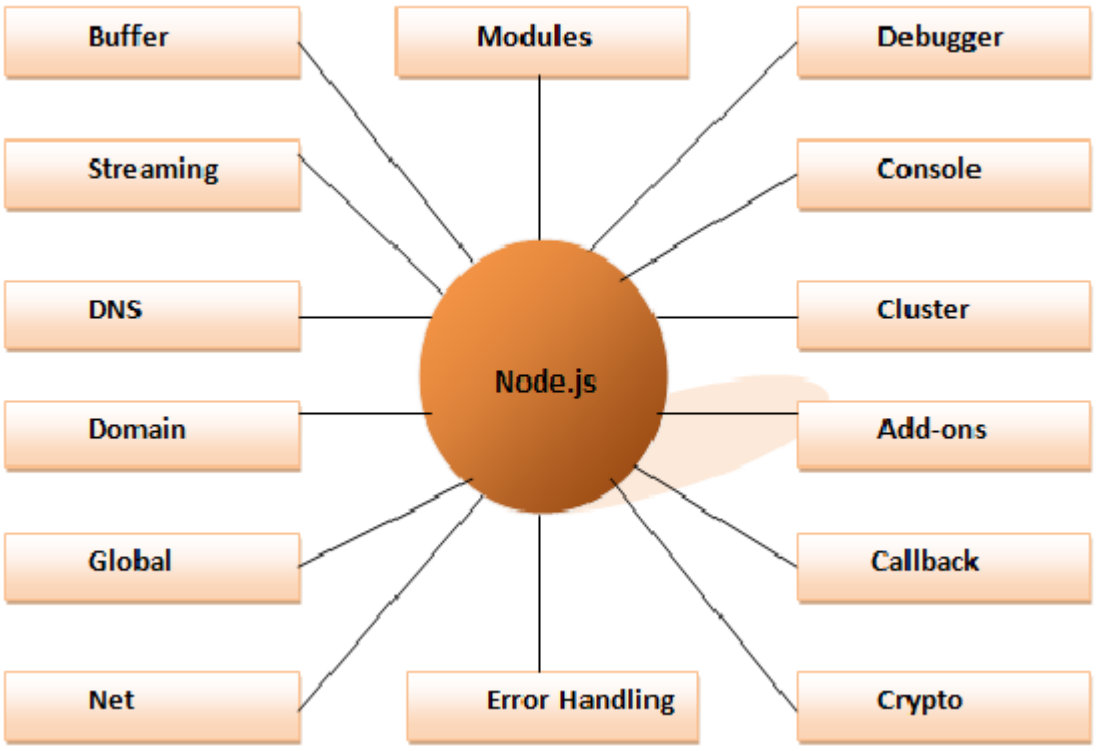


Fig:5.1 Components of Node JS

Features of Node.JS

Following is a list of some important features of Node.js that makes 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 Node.js library are asynchronous i.e. non-blocking. So 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 of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it 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 event mechanism helps the server to 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. These applications simply output the data in chunks.
6. **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
7. **License:** Node.js is released under the MIT license.

5.1.1 Downloading Node.JS

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

1. Text Editor.
2. Node.js Binary installable

You can download the latest version of Node.js installable archive file from https://nodejs.org/en/. Details of installation is mentioned in the next topic.

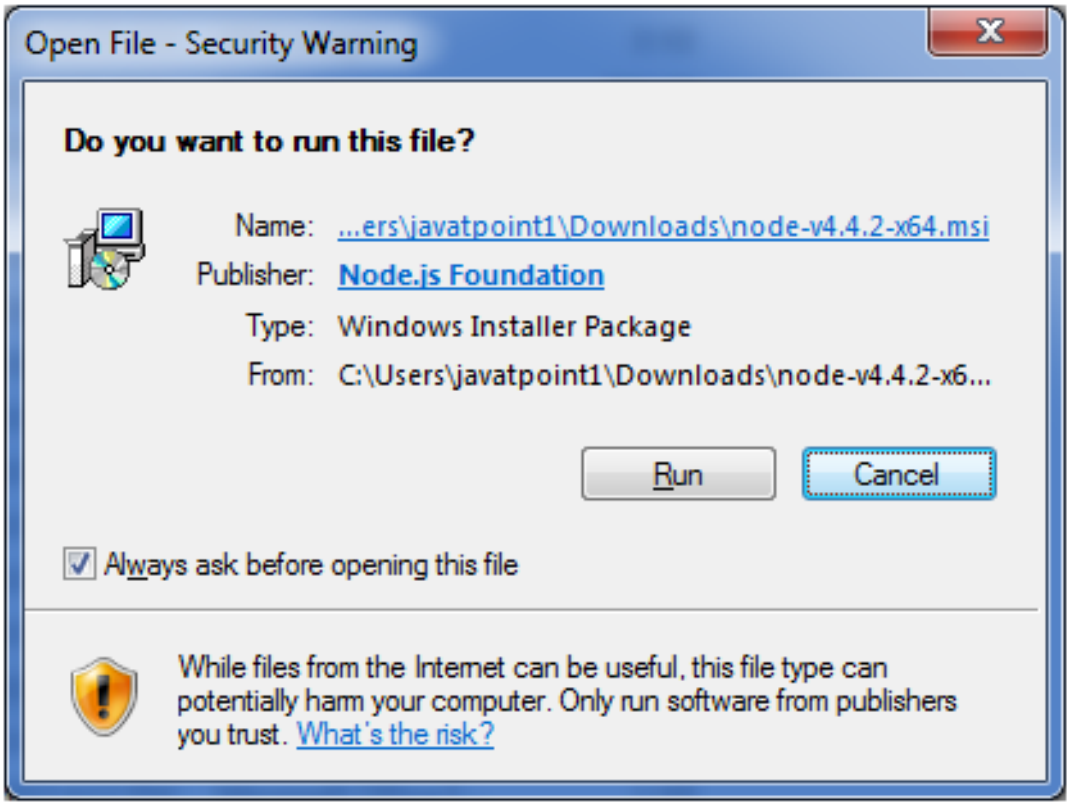
5.2 Setting up Node.JS server (HTTP Server)

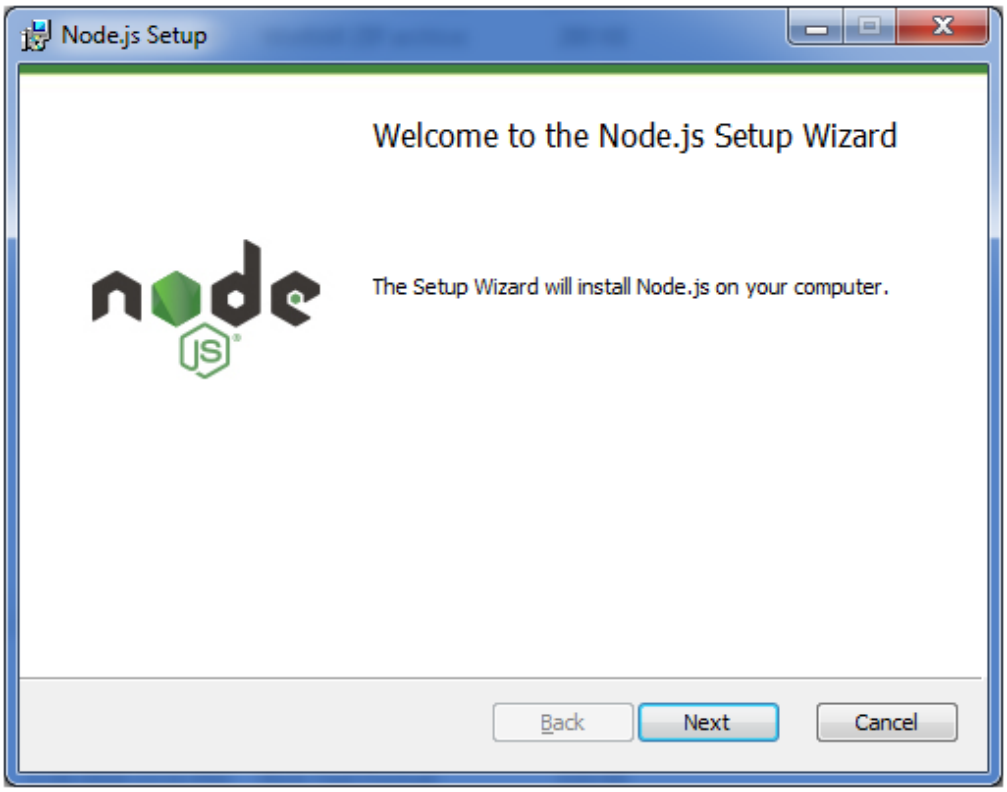
5.2.1 Installing On Windows

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

Download the installer for windows by clicking on 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.

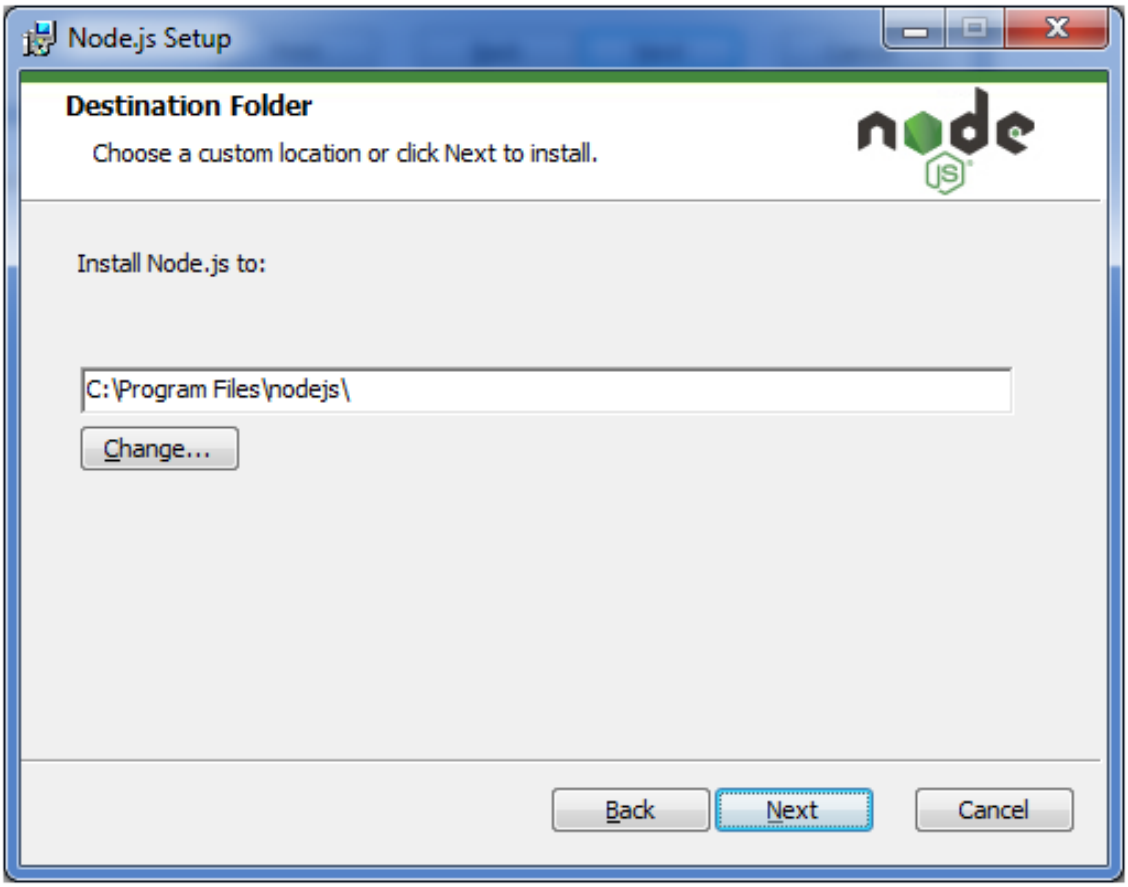
After you download the MSI, double-click on it to start the installation as shown below.

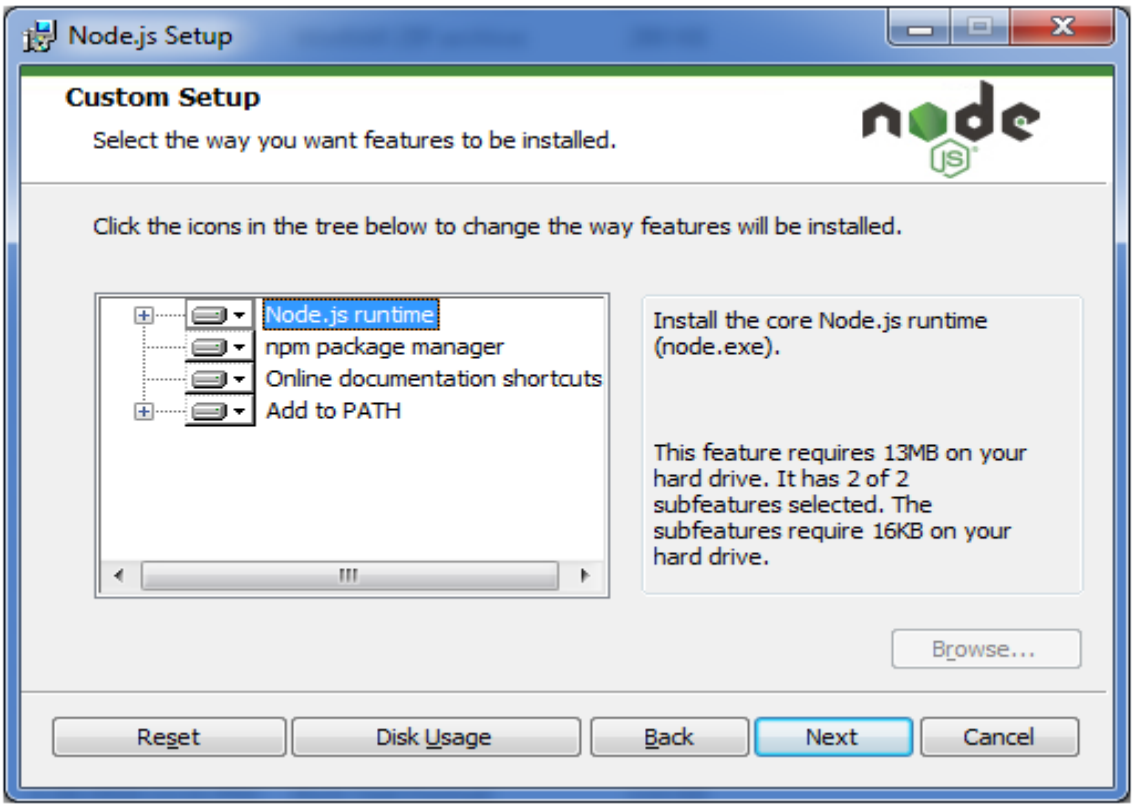




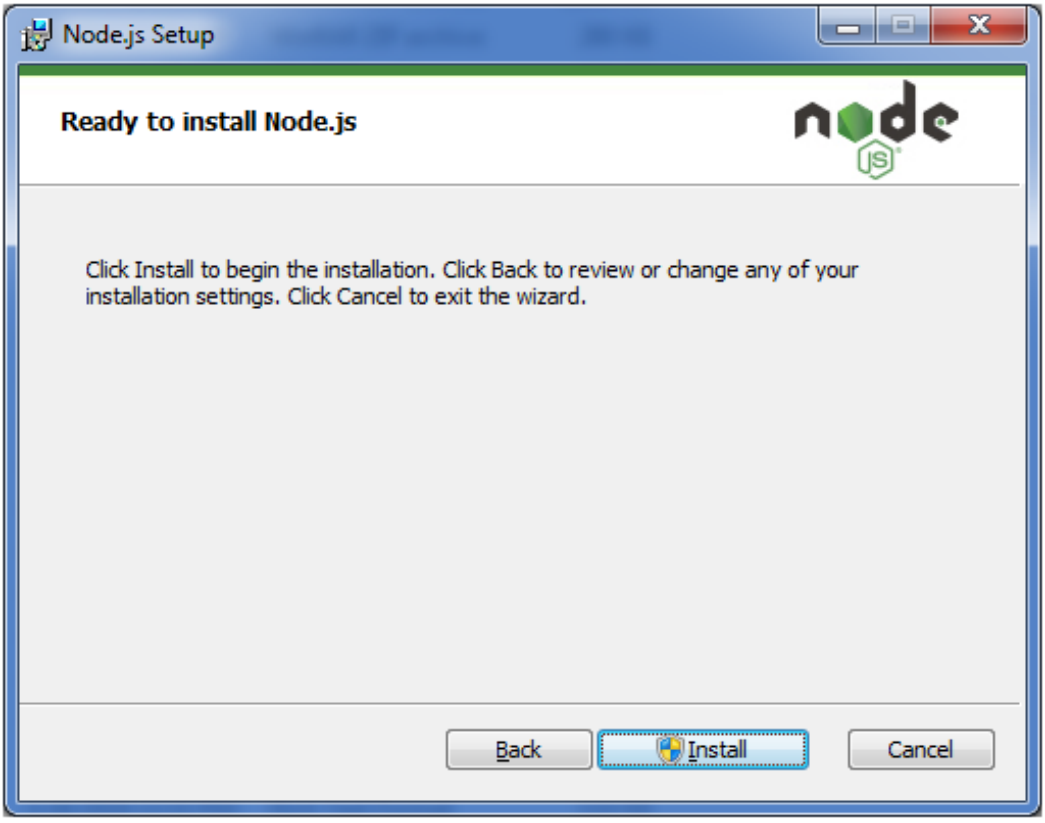
Accept the terms of license agreement.

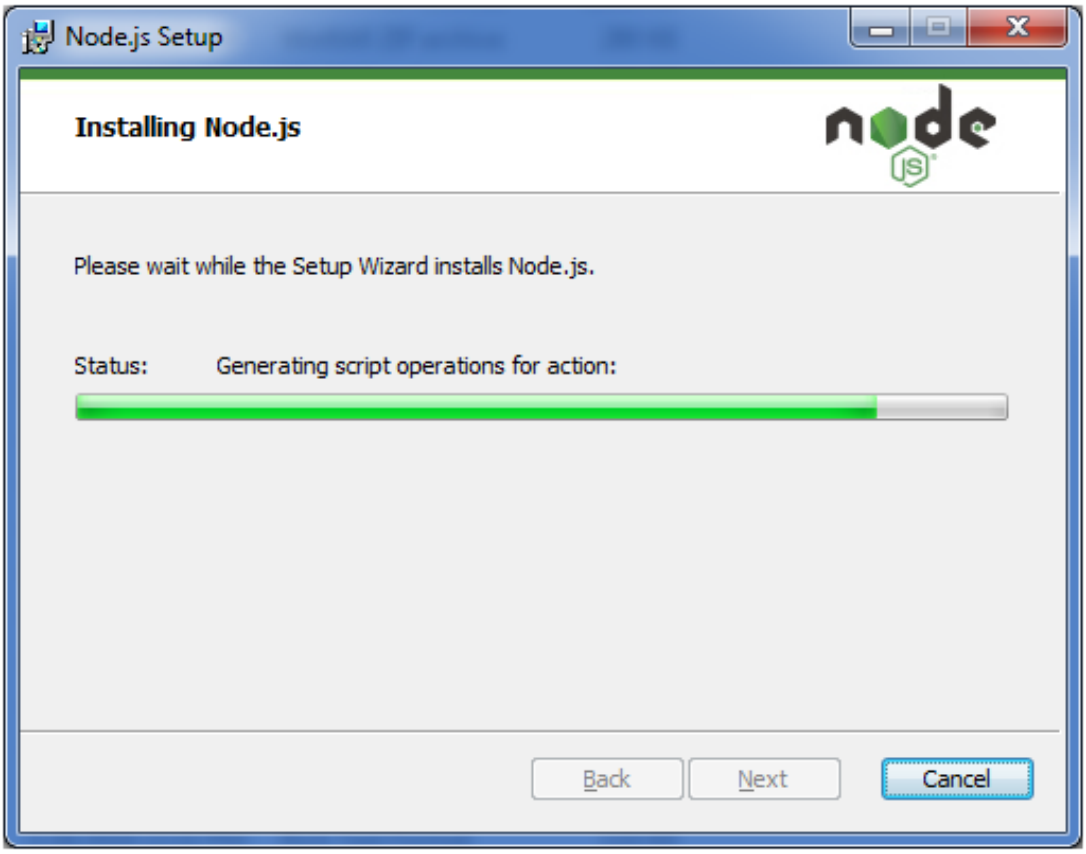
Choose the location where you want to install.

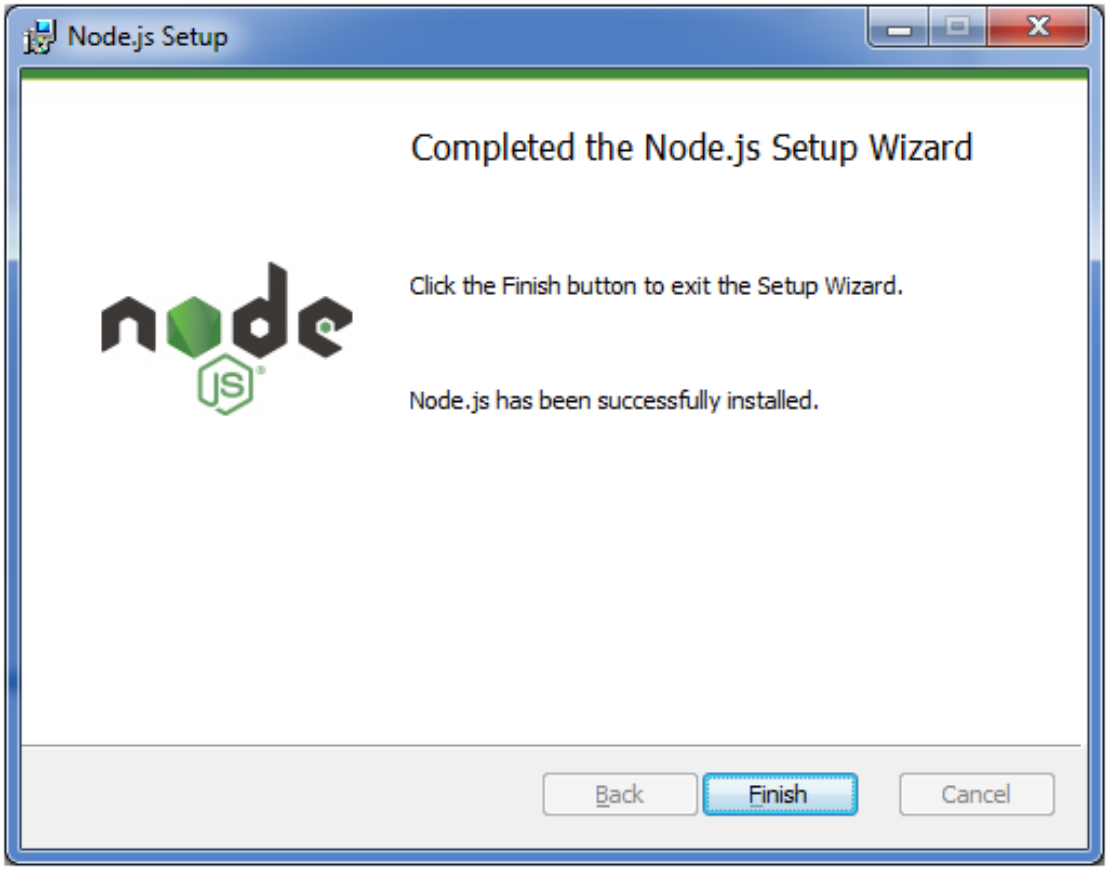




Ready to install:

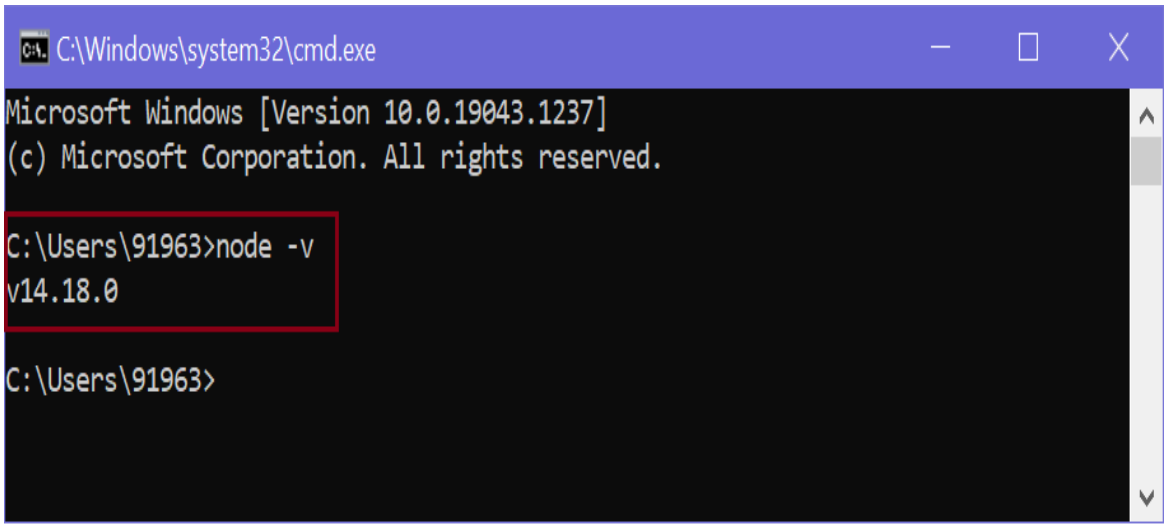






Verify Installation

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 then it will display the version of the Node.js installed on your machine, as shown below.



5.2.2 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 which will listen to client's request similar to Apache HTTP Server.
3. **Read request and return response:** Server created in the second step will read HTTP request made by client which can be a browser or console and return the response.

We will see them in details below:

5.2.2.1 Required modules, Create Server (http.createserver())

Require Modules: 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 too.

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 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 –**block1**

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

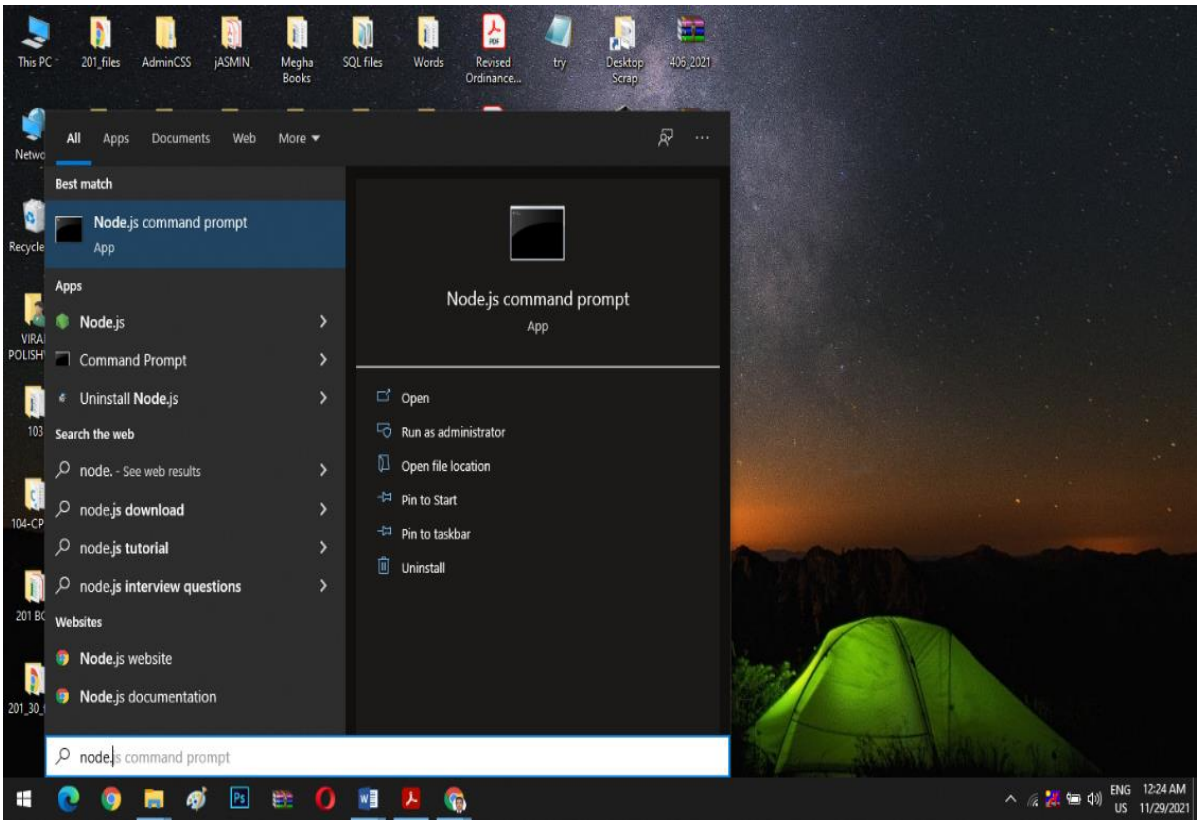
In the above example, require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. **http.createServer()**.

Create server: In the second component, you have to use created http instance and call http.createServer() method to create server instance and then bind it at port 8081 using listen method associated with server instance. Pass it a function with request and response parameters and write the sample implementation to return "Hello World". Check the example below: **block2**

```
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/');
```

5.2.2.2 Request and Response

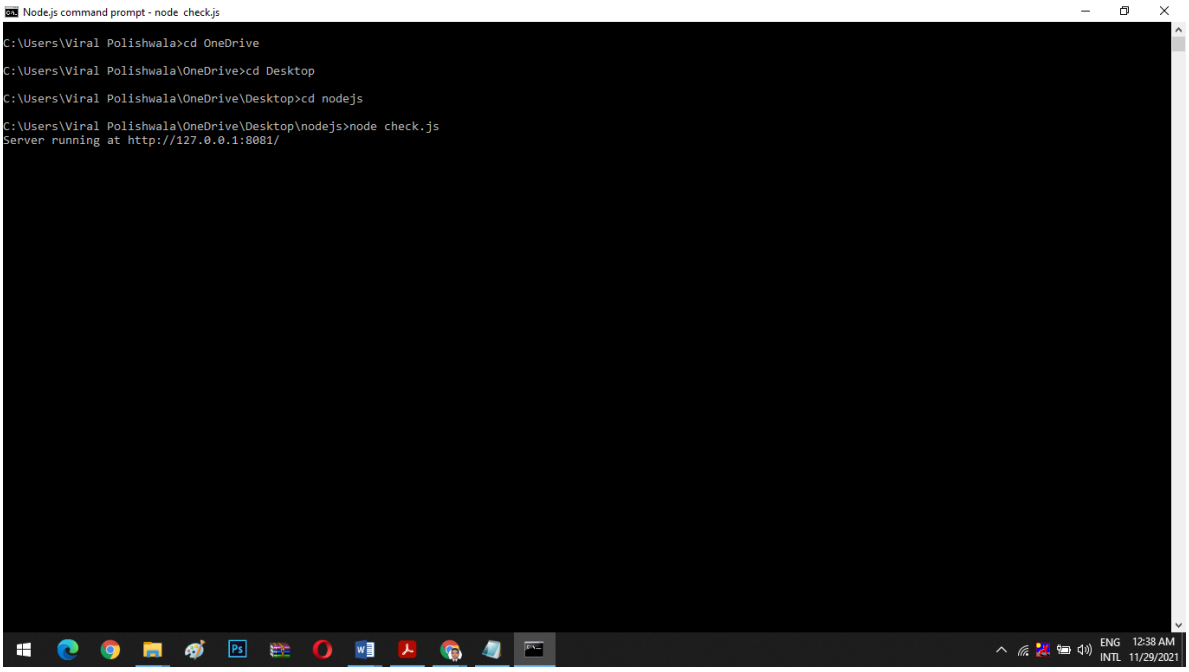
Now combine the block 1 and block 2 and save the file as **check.js**; while executing the above it will just create an HTTP server which listens to the port mentioned as 8081 on the local machine.

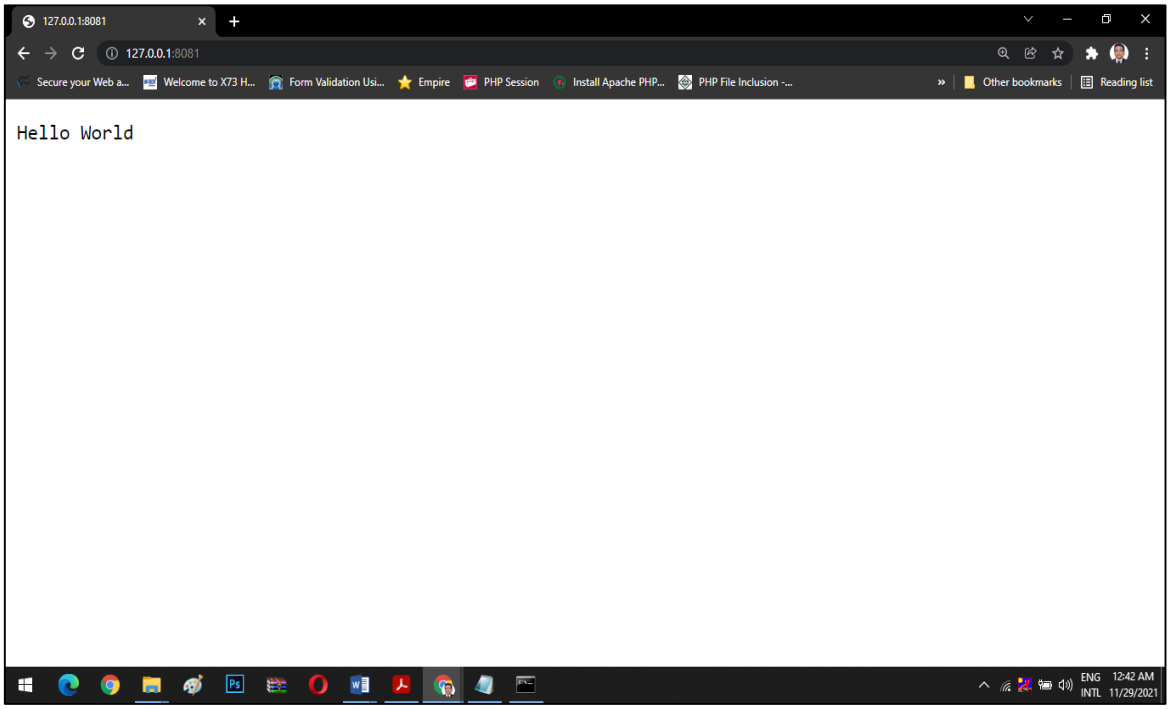


For starting the server, we need to start the command prompt as follows:

Click the start menu and type node.js command prompt. Once the command prompt is open just move 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 the 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 request which are sent over https server are responded from the port 8081.

5.3 Built-In Modules

In Node.js, Modules are the blocks of encapsulated code that communicates with an external application on the basis of their related functionality. Modules can be a single file or a collection of multiples files/folders. The reason programmers are heavily reliant on modules is because of their re-usability 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 comes with Node.js installation. Node Js Core Modules comes 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.

5.3.1 require () function

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

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

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

```
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);
```

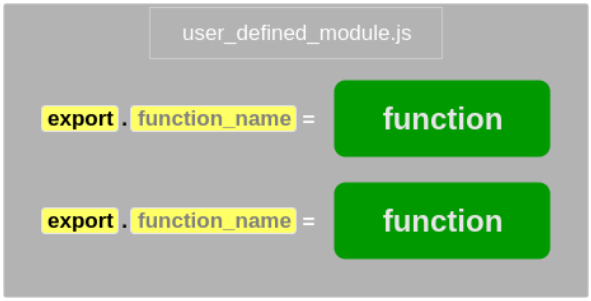
In the above example, the require() function returns an object because the Http module returns its functionality as an object. The function http.createServer() method will be executed when someone tries to access the computer on port 3000. The res.writeHead() method is the status code where 200 means 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. var os = require("os")
Path Module	Provides utilities for handling and transforming file paths. var path = require("path")
Net Module	Provides both servers and clients as streams. Acts as a network wrapper. var net = require("net")
DNS Module	Provides functions to do actual DNS lookup as well as to use underlying operating system name resolution functionalities. var dns = require("dns")

Domain Module	Provides ways to handle multiple different I/O operations as a single group. var domain = require("domain")

5.3.2 User Defined Modules: Create and include

Modules are the collection of JavaScript codes in a separate logical file that can be used in external applications on the basis of their related functionality. Modules are popular as they are easy to use and are reusable. Sometimes it is required that, 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 which 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 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 using which we can access this function in other programs.

As mentioned above, exports is an object. So it exposes whatever you assigned 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 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. 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 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: 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 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: app.js

```
var msg = require('./Log.js');
msg.log('Hello World');
```

Run and see the output in 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 = {
  firstName: 'James',
  lastName: 'Bond'
}
```

app.js

```
var person = require('./data.js');
console.log(person.firstName + ' ' + person.lastName);
```

Run the above example and see the result, as shown below.

```
C:\> node app.js
James Bond
```

5.3.3 HTTP module

To make HTTP requests in Node.js, there is a built-in module HTTP in Node.js to transfer data over the 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 a HTTP server with the help of **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
```

Above code will start the webserver 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();
```

Above example will display the status code of the request made on server.

The HTTP module provides some properties and methods, and some 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 description:

```
<img width="599" height="706" src="">> 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',
<img width="599" height="658" src="">  '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 the HTTP client requests by default. It is used to control connections persistence, and reuse for HTTP clients. Moreover, it is a significant constituent in Node.js HTTP networking.

http methods are described in the next topic.

5.4 Node.JS as 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 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, like an HTML page or JSON from an API. To access web pages of any web application, you need a web server. The web server will handle all the http requests for the web application e.g 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. *Front-end code* is concerned with how the content is presented, such as the color of a navigation bar and the text styling. *Back-end 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 back-end code.

Node.js allows developers to use JavaScript to write back-end code, even though traditionally it was used in the browser to write front-end code. Having both the 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 back-end code.

There are a variety of modules such as the “http” and “request” module, which helps in processing server related requests in the webserver 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 application but it is recommended to use Node.js web server.

5.4.1 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 the code below to understand the concept; filename:helloserver.js

```
var http=require('http')                                        1

var server=http.createServer((function(request,response)        2

{

        response.writeHead(200,                                 3

        {"Content-Type" : "text/plain"});

        response.end("Hello World\n");                          4

}));

server.listen(8081);                                            5
```

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 it 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 a 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 which 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 no 8081. You can specify any available port over here.

Run the above web server by writing node helloserver.js command in command prompt or terminal window and it will display message of node.js is running of port no. 8081.

The http.createServer() method includes request and response parameters which is supplied by Node.js. 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 a current HTTP request.

```
var http = require('http'); // Import Node.js core module

var server = http.createServer(function (req, res) {   //create web server
   if (req.url == '/') { //check the URL of the current request

      // set response header
      res.writeHead(200, { 'Content-Type': 'text/html' });

      // set response content
      res.write('<html><body><p>This is home Page.</p></body></html>');
      res.end();

   }
   else if (req.url == "/student") {

      res.writeHead(200, { 'Content-Type': 'text/html' });
      res.write('<html><body><p>This is student Page.</p></body></html>');
      res.end();
```

```
  }
  else if (req.url == "/admin")
    {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is admin Page.</p></body></html>');
    res.end();
    }
  else
    res.end('Invalid Request!');
});
server.listen(5000); //6 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')
```

5.4.2 Reading Query String, Split Query String

In Node.js, functionality to aid in the accessing of 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 is an example script using this handy function and an explanation on 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 (app.js is name of the file) on the terminal and then go to your browser and type http://localhost:8080/app.js?foo=bad&baz=foo on 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 things from the inside-out. First off, req.url will look like /app.js?foo=bad&baz=foo. This is the part that is 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 have many key value pairs one of which is the query object. Some other handy information returned by the method include host, pathname, 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 querystring

Another way to access query string parameters is parsing them using the querystring builtin Node.js module.

This method, however, must be passed just a querystring portion of a url. Passing it the whole url, like you did in the url.parse example, won't parse the querystrings.

```
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' }
```

5.5 File System Module

This module, as the name suggests, enables developers to work with the file system on its computers. The require () method is used to include the File System module. Some of the most common uses of this module is to create, read, update, delete, rename or read files. For example, the function fs.readFile() is used to read files on the computer.

5.5.1 Read files (readFile())

The Node.js file system module (fs module) allows you to work with the files. The fs.readFile() is used to read from file.

There are two ways to read 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);
```

5.5.2 Create Files (appendFile(), open (), writeFile())

There are a bunch of methods used for creating new files are: fs.appendFile (), fs.open (), fs.writeFile ().

The append method is used to, as the name suggests, append the given file. Also, if one particular file is appended but does not exist then a file of the same name will be created. The syntax of fs.appendFile () would look something like -

```
fs.appendFile('newfile.txt', 'Hello Konfinity!', function (err) {
if (err) throw err;
console.log('Done!');
});
```

The next method to discuss is the fs.open() method. This method takes a "flag" as the second argument. If the flag is "w", the specified file is opened for writing and if the called file does not exist then an empty file is created. The syntax of the fs.open() method will look like

```
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 then the fs.writeFile() method is used. Also, 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!');
});
```

Apart from these methods included in the file system module of Nodejs, there are a couple of other methods included too. Let's look at some of them and understand the concept in detail.

5.5.3 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 mentioned in the code. The syntax for the method fs.appendFile() would look like –

```
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 that goes by the name "newfile1.txt".

Another method is the fs.writeFile() one which replaces the file and content mentioned in the code. The syntax of the method is:

```
fs.writeFile('mynewfile3.txt', 'This is my text', function (err) {
if (err) throw err;
console.log('Replaced!');
});
```

The code above replaces the content of the file "newfile3.txt": There are methods to delete files too.

5.5.4 Delete Files (unlink())

The fs.unlink() method is used to delete a particular file with the File System module. The syntax of this method will look like –

```
fs.unlink('newfile2.txt', function (err) {
if (err) throw err;
console.log ('File deleted!');
});
```

The code basically deletes the file "newfile2.txt":

5.5.5 Rename Files (rename())

In Nodejs, 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 will look like -

```
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 is prefixed with a variable fs. In these examples above, we took the variable to be 'fs', while executing your code, you would have to explicitly mention it in your code. The code will look something like - var fs = require('fs');