
FULL STACK DEVELOPMENT

Notes by Aaplakatta.com

(R20A0516) FULL STACK DEVELOPMENT

COURSE OBJECTIVES:

1. To become knowledgeable about the most recent web development technologies.
2. Idea for creating two tier and three tier architectural web applications.
3. Design and Analyse real time web applications.
4. Constructing suitable client and server side applications.
5. To learn core concept of both front end and back end programming.

UNIT - I

Web Development Basics: Web development Basics - HTML & Web servers Shell - UNIX CLI Version control - Git & Github
HTML, CSS

UNIT - II

Frontend Development: Javascript basics OOPS Aspects of JavaScript Memory usage and Functions in JS AJAX for data exchange with server jQuery Framework jQuery events, UI components etc. JSON data format.

UNIT - III

REACT JS: Introduction to React React Router and Single Page Applications React Forms, Flow Architecture and Introduction to Redux More Redux and Client-Server Communication

UNIT - IV

Java Web Development: JAVA PROGRAMMING BASICS, Model View Controller (MVC) Pattern MVC Architecture using Spring RESTful API using Spring Framework Building an application using Maven

UNIT - V

Databases & Deployment: Relational schemas and normalization Structured Query Language (SQL) Data persistence using Spring JDBC Agile development principles and deploying application in Cloud

TEXT BOOKS:

1. Web Design with HTML, CSS, JavaScript and JQuery Set Book by Jon Duckett ProfessionalJavaScript for Web Developers Book by Nicholas C. Zakas
2. Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to CreatingDynamic Websites by Robin Nixon
3. Full Stack JavaScript: Learn Backbone.js, Node.js and MongoDB. Copyright © 2015 BYAZAT MARDAN

REFERENCE BOOKS:

1. Full-Stack JavaScript Development by Eric Bush.
2. Mastering Full Stack React Web Development Paperback – April 28, 2017 by TomaszDyl , Kamil Przeorski , Maciej Czarnecki

COURSE OUTCOMES:

1. Develop a fully functioning website and deploy on a web server.
2. Gain Knowledge about the front end and back end Tools
3. Find and use code packages based on their documentation to produce working results ina project.
- 4.Create web pages that function using external data.
- 5.Implementation of web application employing efficient database access.

UNIT	TOPIC	PAGE
UNIT - I	Web development Basics - HTML	6
	Web servers Shell - UNIX CLI	90
	Version control - Git &Github HTML	92
	CSS	124
Unit-II	Javascript basics	170
	OOPS Aspects of JavaScript	179
	Memory usage and Functions in JS	183
	AJAX for data exchange with server	187
	jQuery Framework	191
	jQuery events	193
	JSON data format.	194
Unit-III	REACT JS: Introduction to React	195
	React Router and Single Page Applications	198
	React Forms	199
	Introduction to Redux	211
	More Redux	214
	Client-Server Communication	215
UNIT-IV	Java Web Development:	221
	JAVA PROGRAMMING BASICS	225
	Model View Controller (MVC)	237
	MVC Architecture using Spring	242
	RESTful API using Spring Framework	244
	Building an application usingMaven	261
Unit-V	Databases & Deployment	266
	Relational schemas and normalization	268
	Structured Query Language	269

	Data persistence using Spring	274
	JDBC Agile development	276
	principles and deploying application in Cloud	281

What is HTML

HTML is an acronym which stands for **Hyper Text Markup Language** which is used for creating web pages and web applications. Let's see what is meant by Hypertext Markup Language, and Web page.

Hyper Text:HyperText simply means "Text within Text." A text has a link within it, is a hypertext. Whenever you click on a link which brings you to a new webpage, you have clicked on a hypertext. HyperText is a way to link two or more web pages (HTML documents) with each other.

Markup language: A markup language is a computer language that is used to apply layout and formatting conventions to a text document. Markup language makes text more interactive and dynamic. It can turn text into images, tables, links, etc.

Web Page: A web page is a document which is commonly written in HTML and translated by a web browser. A web page can be identified by entering an URL. A Web page can be of the static or dynamic type. **With the help of HTML only, we can create static web pages.**

Hence, HTML is a markup language which is used for creating attractive web pages with the help of styling, and which looks in a nice format on a web browser. An HTML document is made of many HTML tags and each HTML tag contains different content.

Let's see a simple example of HTML.

1. `<!DOCTYPE>`
2. `<html>`
3. `<head>`
4. `<title>Web page title</title>`
5. `</head>`
6. `<body>`
7. `<h1>Write Your First Heading</h1>`
8. `<p>Write Your First Paragraph.</p>`
9. `</body>`
10. `</html>`

Description of HTML Example

<!DOCTYPE>: It defines the document type or it instruct the browser about the version of HTML.

<html >: This tag informs the browser that it is an HTML document. Text between html tag describes the web document. It is a container for all other elements of HTML except `<!DOCTYPE>`

<head>: It should be the first element inside the <html> element, which contains the metadata (information about the document). It must be closed before the body tag opens.

<title>: As its name suggested, it is used to add title of that HTML page which appears at the top of the browser window. It must be placed inside the head tag and should close immediately. (Optional)

<body>: Text between body tag describes the body content of the page that is visible to the end user. This tag contains the main content of the HTML document.

<h1> : Text between <h1> tag describes the first level heading of the webpage.

<p>: Text between <p> tag describes the paragraph of the webpage.

Brief History of HTML

In the late 1980's, a physicist, Tim Berners-Lee who was a contractor at CERN, proposed a system for CERN researchers. In 1989, he wrote a memo proposing an internet based hypertext system.

Tim Berners-Lee is known as the father of HTML. The first available description of HTML was a document called "HTML Tags" proposed by Tim in late 1991. The latest version of HTML is HTML5, which we will learn later in this tutorial.

HTML Versions

Since the time HTML was invented there are lots of HTML versions in market, the brief introduction about the HTML version is given below:

HTML 1.0: The first version of HTML was 1.0, which was the barebones version of HTML language, and it was released in 1991.

HTML 2.0: This was the next version which was released in 1995, and it was standard language version for website design. HTML 2.0 was able to support extra features such as form-based file upload, form elements such as text box, option button, etc.

HTML 3.2: HTML 3.2 version was published by W3C in early 1997. This version was capable of creating tables and providing support for extra options for form elements. It can also support a web page with complex mathematical equations. It became an official standard for any browser till January 1997. Today it is practically supported by most of the browsers.

HTML 4.01: HTML 4.01 version was released on December 1999, and it is a very stable version of HTML language. This version is the current official standard, and it provides added support for stylesheets (CSS) and scripting ability for various multimedia elements.

HTML5 : HTML5 is the newest version of HyperText Markup language. The first draft of this version was announced in January 2008. There are two major organizations one is W3C (World Wide Web Consortium), and another one is WHATWG(Web Hypertext Application Technology Working Group) which are involved in the development of HTML 5 version, and still, it is under development.

Features of HTML

- 1) It is a very **easy and simple language**. It can be easily understood and modified.
- 2) It is very easy to make an **effective presentation** with HTML because it has a lot of formatting tags.
- 3) It is a **markup language**, so it provides a flexible way to design web pages along with the text.
- 4) It facilitates programmers to add a **link** on the web pages (by html anchor tag), so it enhances the interest of browsing of the user.
- 5) It is **platform-independent** because it can be displayed on any platform like Windows, Linux, and Macintosh, etc.
- 6) It facilitates the programmer to add **Graphics, Videos, and Sound** to the web pages which makes it more attractive and interactive.
- 7) HTML is a case-insensitive language, which means we can use tags either in lower-case or upper-case.

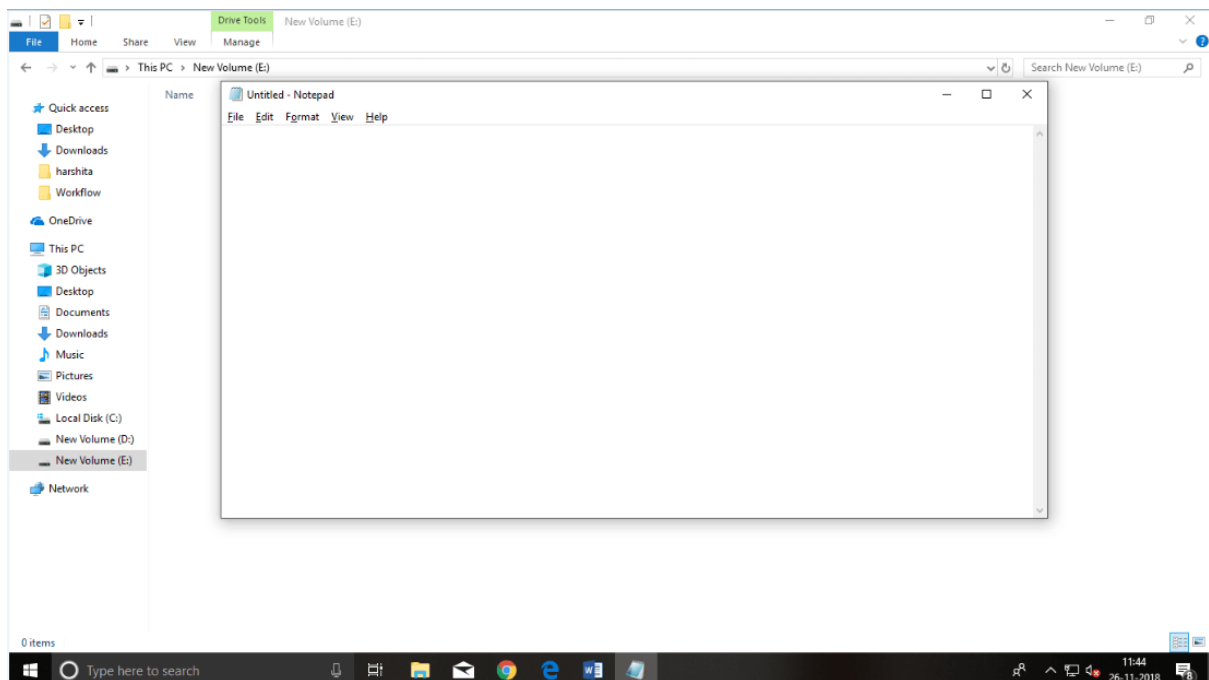
HTML text Editors

- An HTML file is a text file, so to create an HTML file we can use any text editors.
- Text editors are the programs which allow editing in a written text, hence to create a web page we need to write our code in some text editor.
- There are various types of text editors available which you can directly download, but for a beginner, the best text editor is Notepad (Windows) or TextEdit (Mac).
- After learning the basics, you can easily use other professional text editors which are, **Notepad++, Sublime Text, Vim, etc.**
- In our tutorial, we will use Notepad and sublime text editor. Following are some easy ways to create your first web page with Notepad, and sublime text.

A. HTML code with Notepad. (Recommended for Beginners)

Notepad is a simple text editor and suitable for beginners to learn HTML. It is available in all versions of Windows, from where you easily access it.

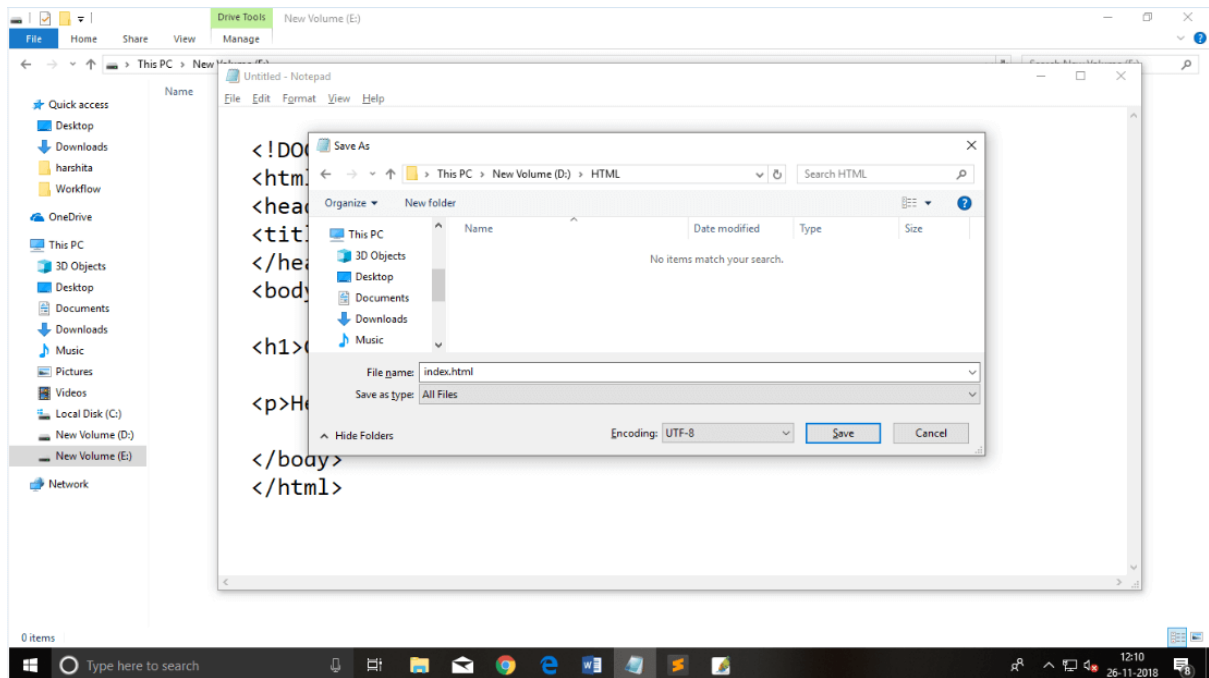
Step 1: Open Notepad (Windows)



Step 2: Write code in HTML

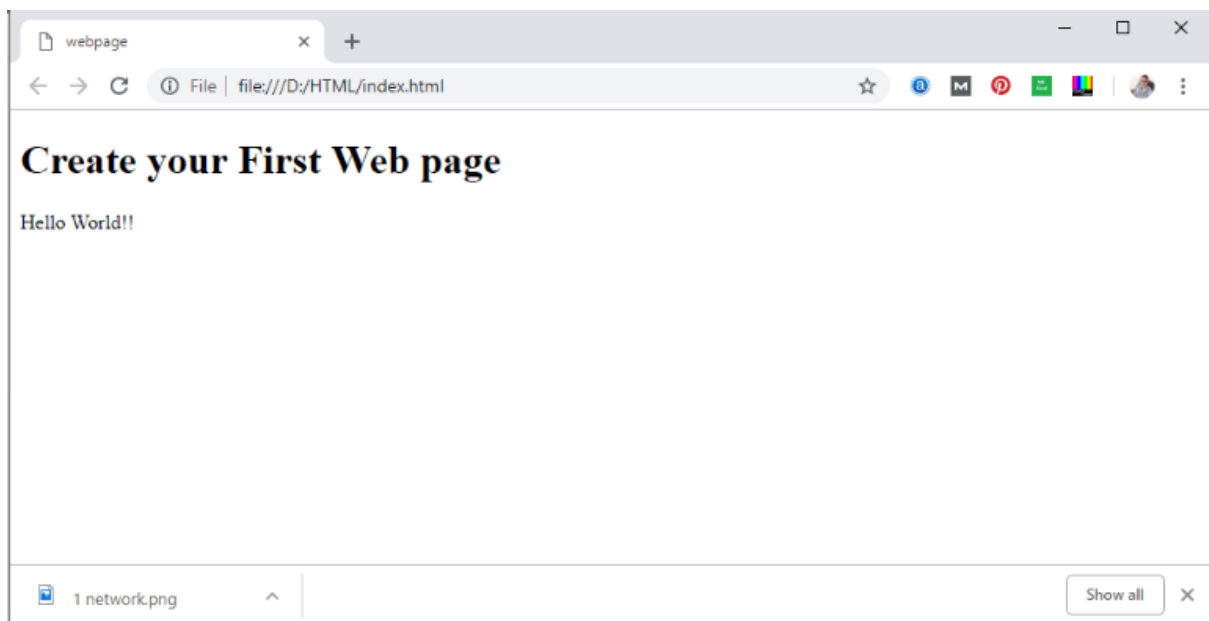
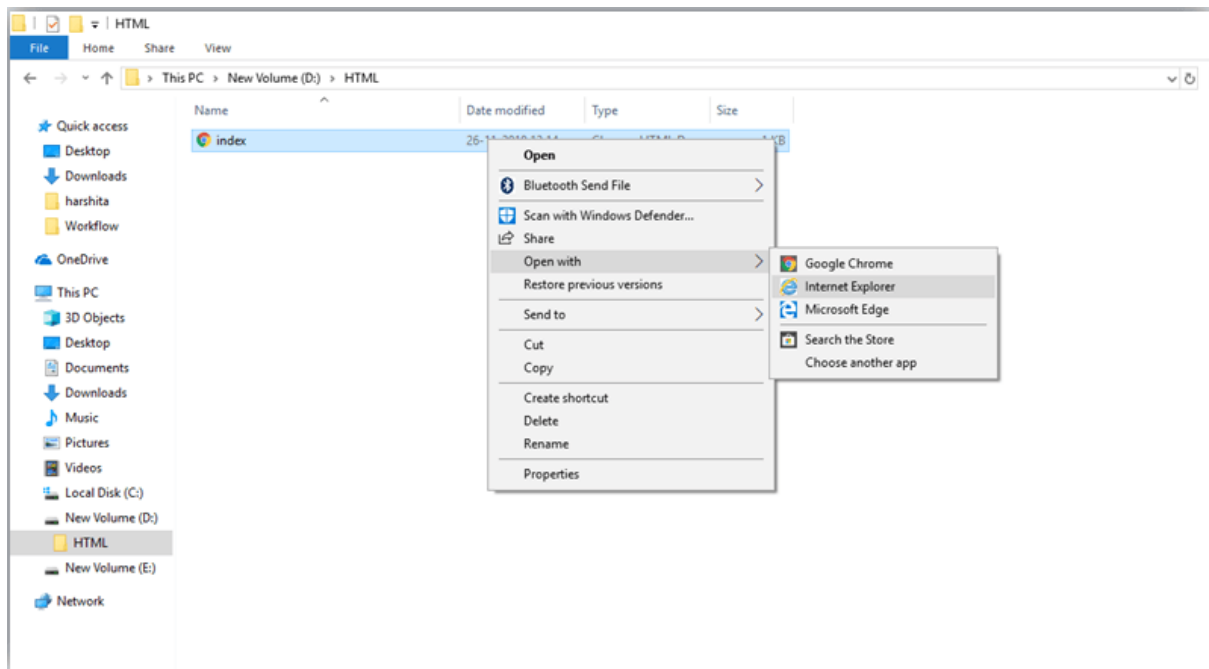


Step 3: Save the HTML file with .htm or .html extension.



Step 4: Open the HTML page in your web browser.

To run the HTML page, you need to open the file location, where you have saved the file and then either double-click on file or click on open with option

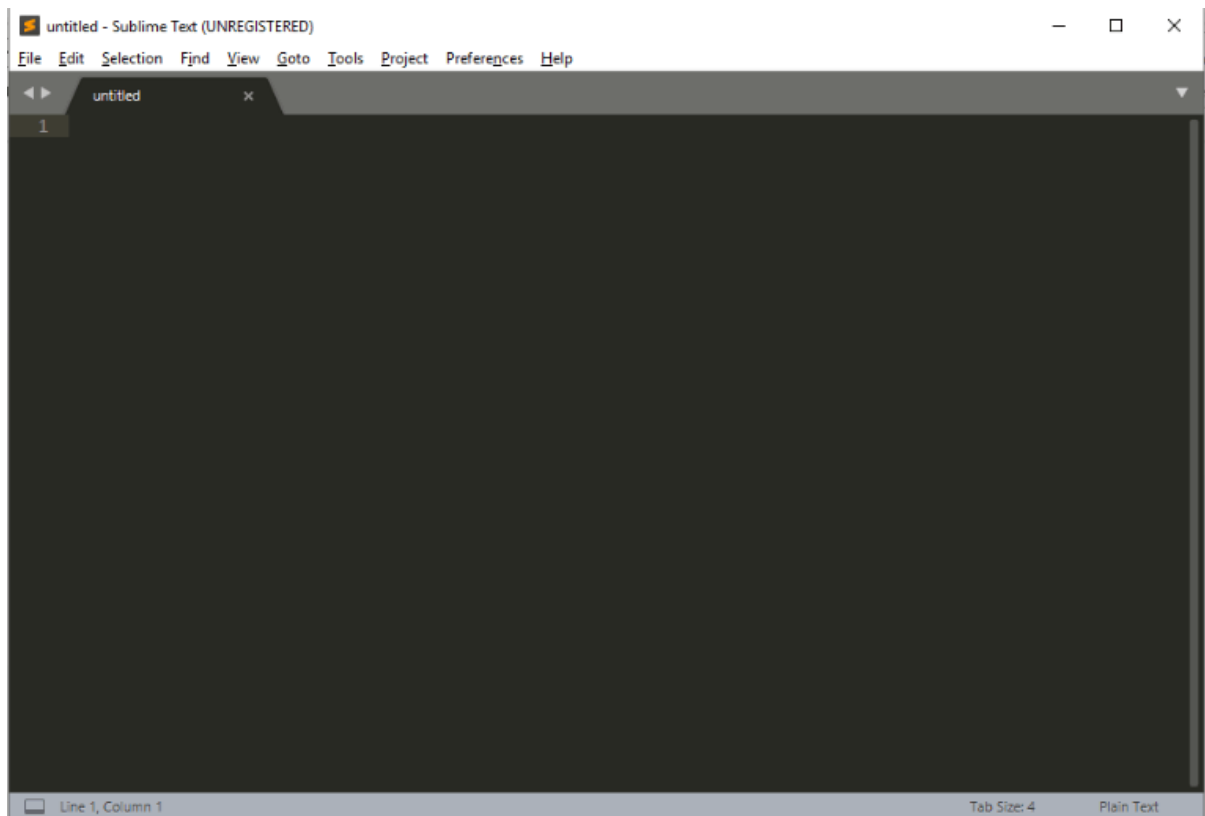


B. HTML code with Sublime Text-editor.(Recommended after learning basics of HTML)

When you will learn the basics of HTML, then you can use some professional text editors, which will help you to write an efficient and fast code. So to use Sublime Text editors, first it needs to download and install from internet. You can easily download it from this <https://www.sublimetext.com/download> link and can install in your PC. When installation of Sublime text editor done then you can follow the simple steps to use it:

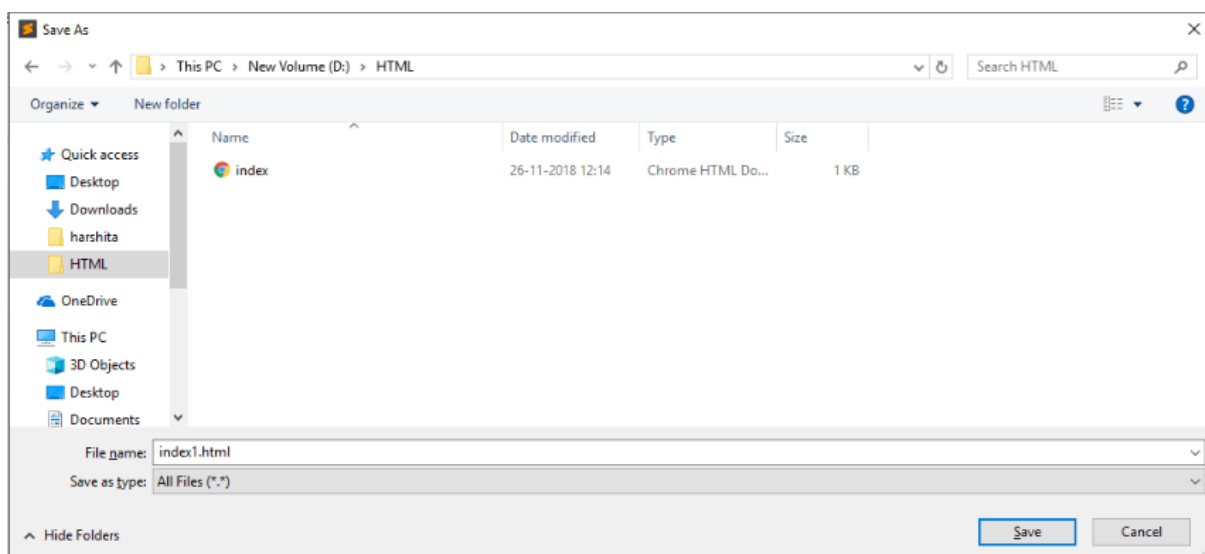
Step 1: Open Sublime Text editor(Windows 8):

To open Sublime Text editor go to **Start screen --> type Sublime Text--> Open** it. To open a new page press **CTRL+N**.

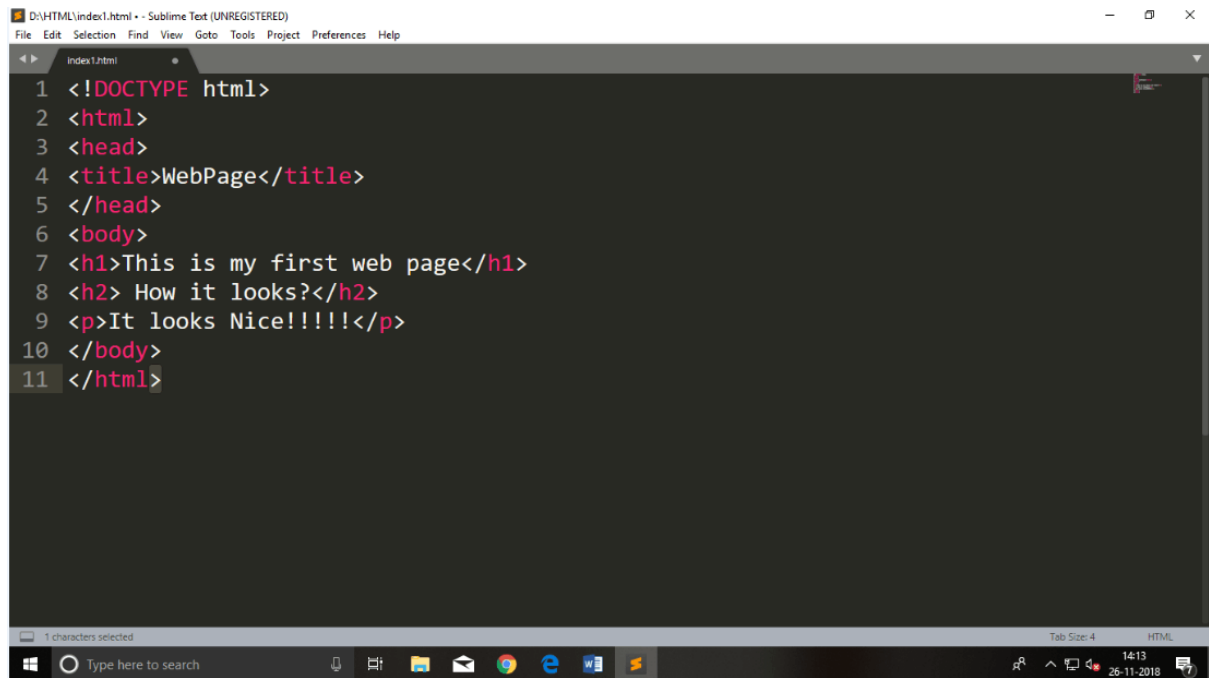


Step 2: Save the page before writing any code.

To save your page in Sublime Text press **Ctrl+S** or go to **File option --> save**, to save a file use extension **.htm** or **.html**. We recommend to save the file first then write the code because after saving the page sublime text editor will give you suggestions to write code.



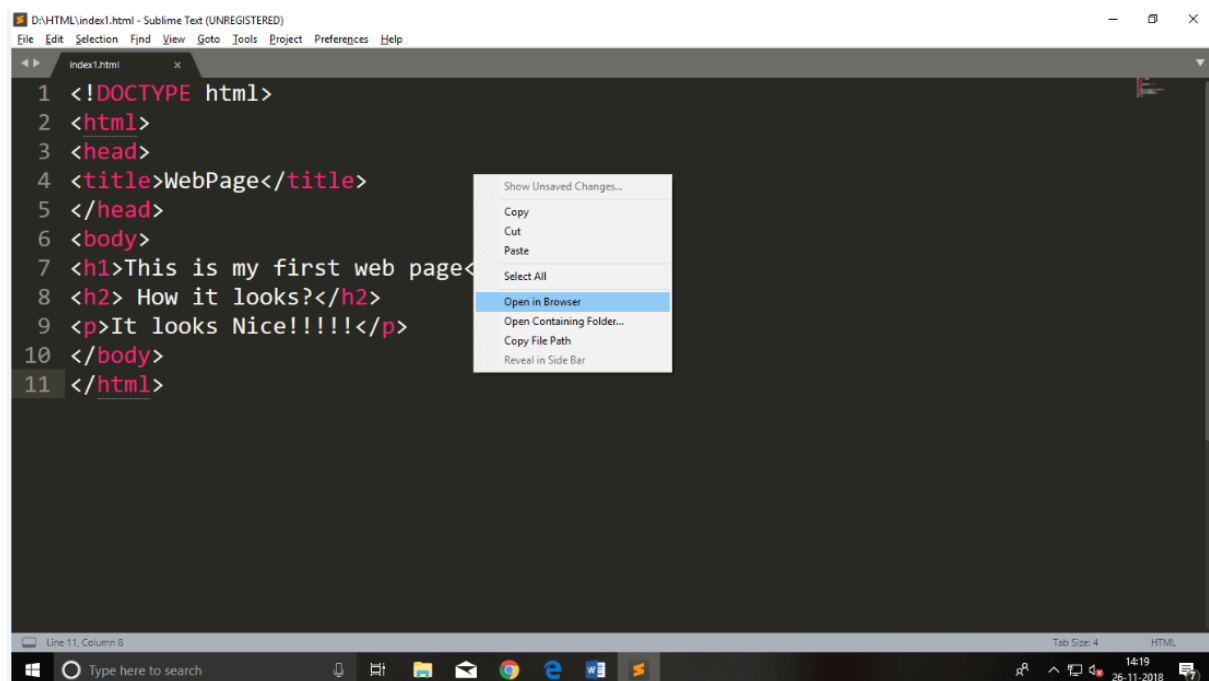
Step 3: Write the code in Sublime Text editor

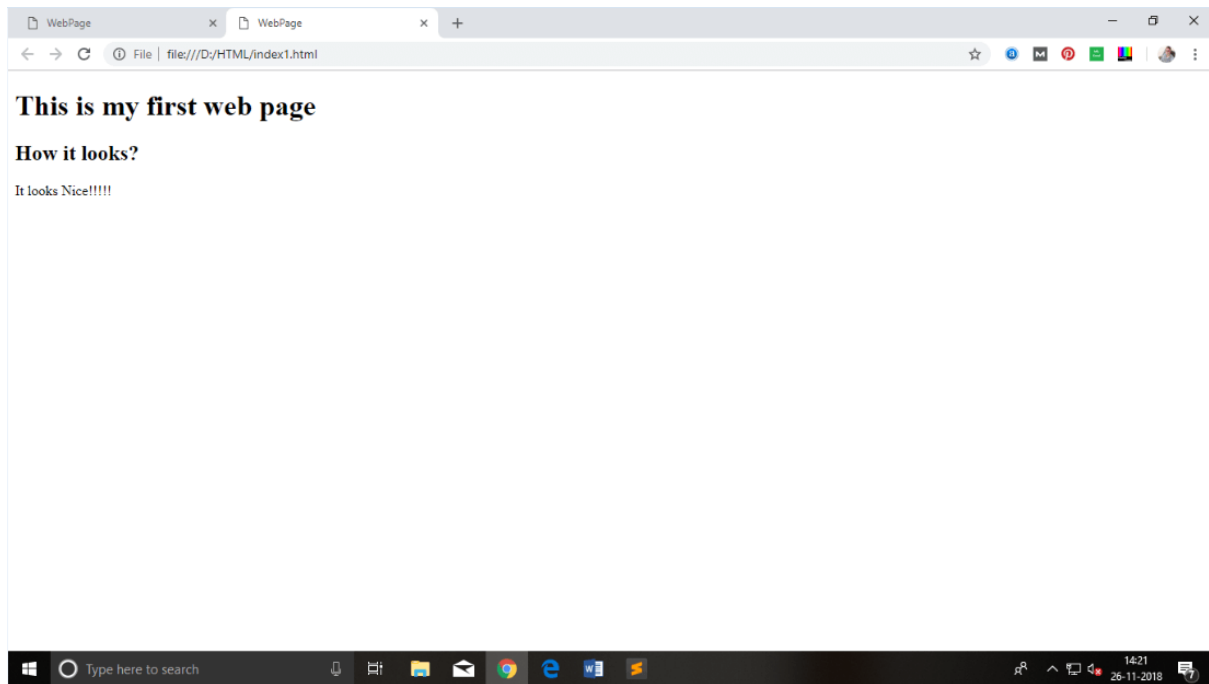


```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>WebPage</title>
5 </head>
6 <body>
7 <h1>This is my first web page</h1>
8 <h2> How it looks?</h2>
9 <p>It looks Nice!!!!</p>
10 </body>
11 </html>
```

Step 4: Open the HTML page in your Browser

To execute or open this page in Web browser just **right click** by mouse on sublime text page and click on **Open in Browser**.





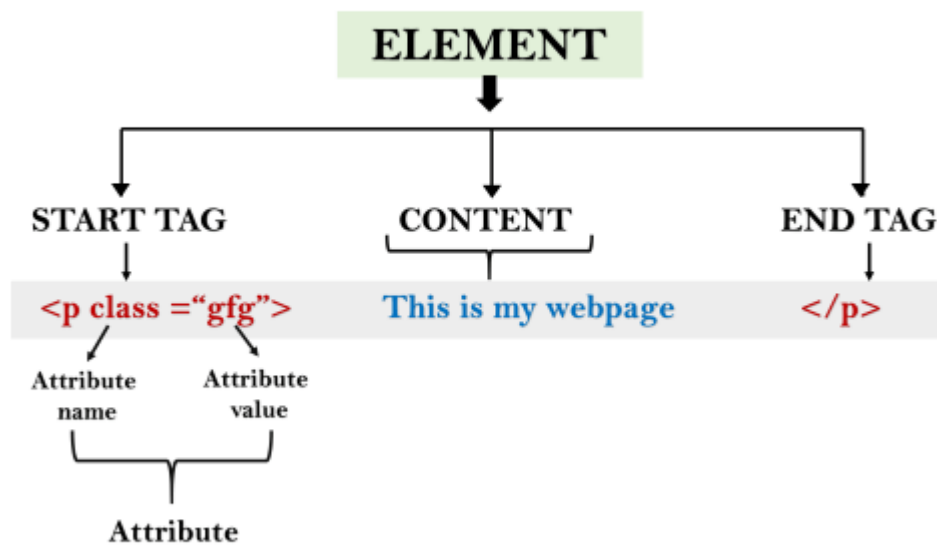
Building blocks of HTML

An HTML document consist of its basic building blocks which are:

- **Tags:** An HTML tag surrounds the content and apply meaning to it. It is written between < and > brackets.
- **Attribute:** An attribute in HTML provides extra information about the element, and it is applied within the start tag. An HTML attribute contains two fields: name & value.

Syntax

1. `<tag name attribute_name= " attr_value"> content </ tag name>`
- **Elements:** An HTML element is an individual component of an HTML file. In an HTML file, everything written within tags are termed as HTML elements.



Example:

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<title>The basic building blocks of HTML</title>`
5. `</head>`
6. `<body>`
7. `<h2>The building blocks</h2>`
8. `<p>This is a paragraph tag</p>`
9. `<p style="color: red">The style is attribute of paragraph tag</p>`
10. `The element contains tag, attribute and content`
11. `</body>`
12. `</html>`

Output:

The building blocks

This is a paragraph tag

HTML Tags

HTML tags are like keywords which defines that how web browser will format and display the content. With the help of tags, a web browser can distinguish between an HTML content and a simple content. HTML tags contain three main parts: opening tag, content and closing tag. But some HTML tags are unclosed tags.

When a web browser reads an HTML document, browser reads it from top to bottom and left to right. HTML tags are used to create HTML documents and render their properties. Each HTML tags have different properties.

An HTML file must have some essential tags so that web browser can differentiate between a simple text and HTML text. You can use as many tags you want as per your code requirement.

- All HTML tags must enclosed within <> these brackets.
- Every tag in HTML perform different tasks.
- If you have used an open tag <tag>, then you must use a close tag </tag> (except some tags)

Syntax

<tag> content </tag>

HTML Tag Examples

Note: HTML Tags are always written in lowercase letters. The basic HTML tags are given below:

<p> Paragraph Tag </p>

<h2> Heading Tag </h2>

Bold Tag

<i>Italic Tag</i>

<u>Underline Tag</u>

Unclosed HTML Tags

Some HTML tags are not closed, for example br and hr.

**
 Tag:** br stands for break line, it breaks the line of the code.

<hr> Tag: hr stands for Horizontal Rule. This tag is used to put a line across the webpage.

HTML Meta Tags

DOCTYPE, title, link, meta and style

HTML Text Tags

<p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, , , <abbr>, <acronym>, <address>, <bdo>, <blockquote>, <cite>, <q>, <code>, <ins>, , <dfn>, <kbd>, <pre>, <samp>, <var> and

HTML Link Tags

<a> and <base>

HTML Image and Object Tags

, <area>, <map>, <param> and <object>

HTML List Tags

, , , <dl>, <dt> and <dd>

HTML Table Tags

table, tr, td, th, tbody, thead, tfoot, col, colgroup and caption

HTML Form Tags

form, input, textarea, select, option, optgroup, button, label, fieldset and legend

HTML Scripting Tags

script and noscript

Note: We will see examples using these tags in later chapters.

HTML Tags List

Following is the complete list of HTML tags with the description which are arranged alphabetically.

HTML Attribute

- HTML attributes are special words which provide additional information about the elements or attributes are the modifier of the HTML element.
- Each element or tag can have attributes, which defines the behaviour of that element.
- Attributes should always be applied with start tag.
- The Attribute should always be applied with its name and value pair.
- The Attributes name and values are case sensitive, and it is recommended by W3C that it should be written in Lowercase only.
- You can add multiple attributes in one HTML element, but need to give space between two attributes.

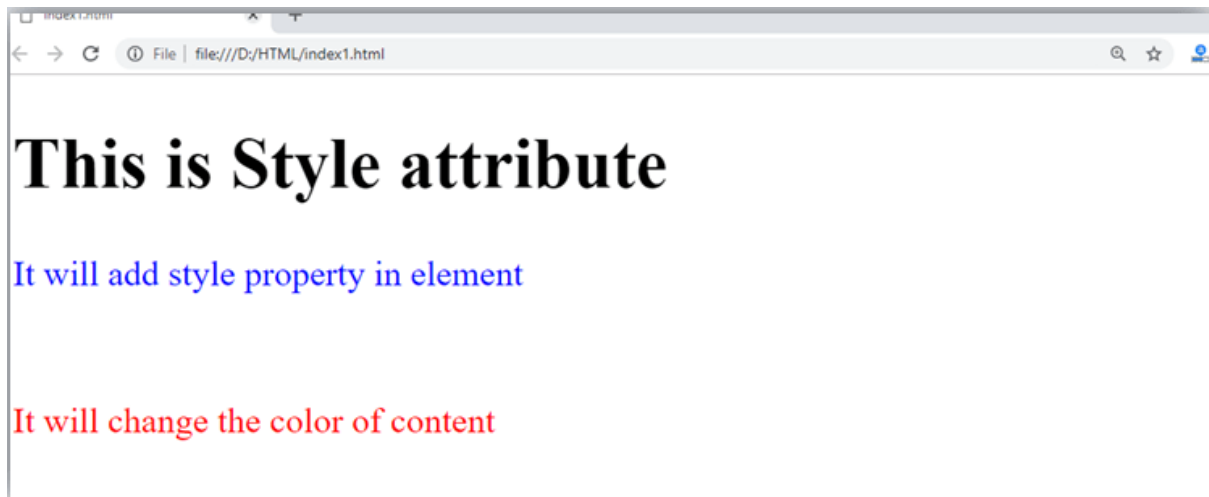
Syntax

1. `<element attribute_name="value">content</element>`

Example

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `</head>`
5. `<body>`
6. `<h1> This is Style attribute</h1>`
7. `<p style="height: 50px; color: blue">It will add style property in element</p>`
8. `<p style="color: red">It will change the color of content</p>`
9. `</body>`
10. `</html>`

Output:



Explanation of above example:

1. `<p style="height: 50px; color: blue">It will add style property in element</p>`

In the above statement, we have used paragraph tags in which we have applied style attribute. This attribute is used for applying CSS property on any HTML element. It provides height to paragraph element of 50px and turns its colour to blue.

1. `<p style="color: red">It will change the color of content</p>`

In the above statement we have again used style attribute in paragraph tag, which turns its colour red.

Note: There are some commonly used attributes given below, and the complete list and explanation of all attributes are given in HTML attributes List.

The title attribute in HTML

Description: The title attribute is used as text tooltip in most of the browsers. It displays its text when the user moves the cursor over a link or any text. You can use it with any text or link to show the description about that link or text. In our example, we are taking this with paragraph tag and heading tag.

Example

With `<h1>` tag:

1. `<h1 title="This is heading tag">Example of title attribute</h1>`

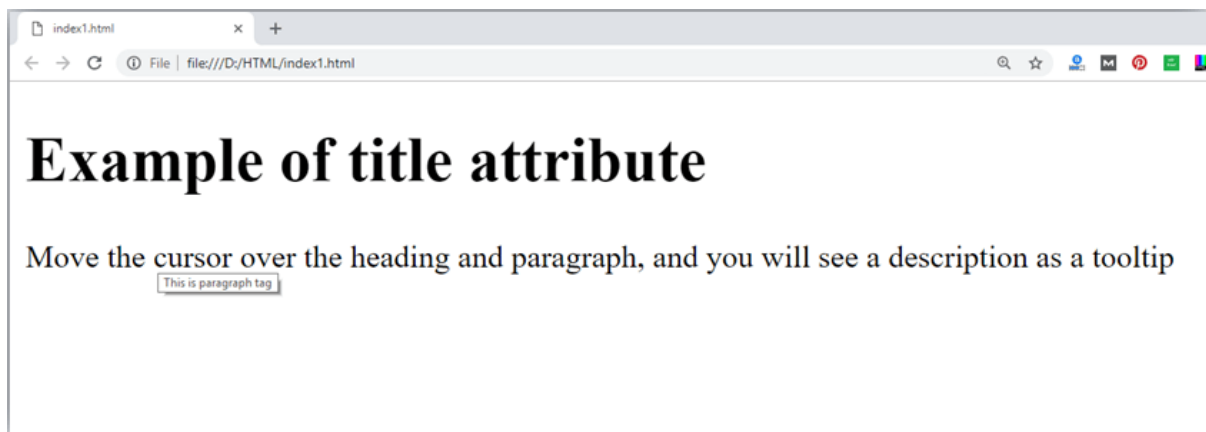
With `<p>` tag:

1. `<p title="This is paragraph tag">Move the cursor over the heading and paragraph, and you will see a description as a tooltip</p>`

Code:

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `</head>`
5. `<body>`
- 6.
7. `<h1 title="This is heading tag">Example of title attribute</h1>`
8. `<p title="This is paragraph tag">Move the cursor over the heading and paragraph, and you will see a description as a tooltip</p>`
- 9.
10. `</body>`
11. `</html>`

Output:



The href attribute in HTML

Description: The href attribute is the main attribute of `<a>` anchor tag. This attribute gives the link address which is specified in that link. **The href attribute provides the hyperlink, and if it is blank, then it will remain in same page.**

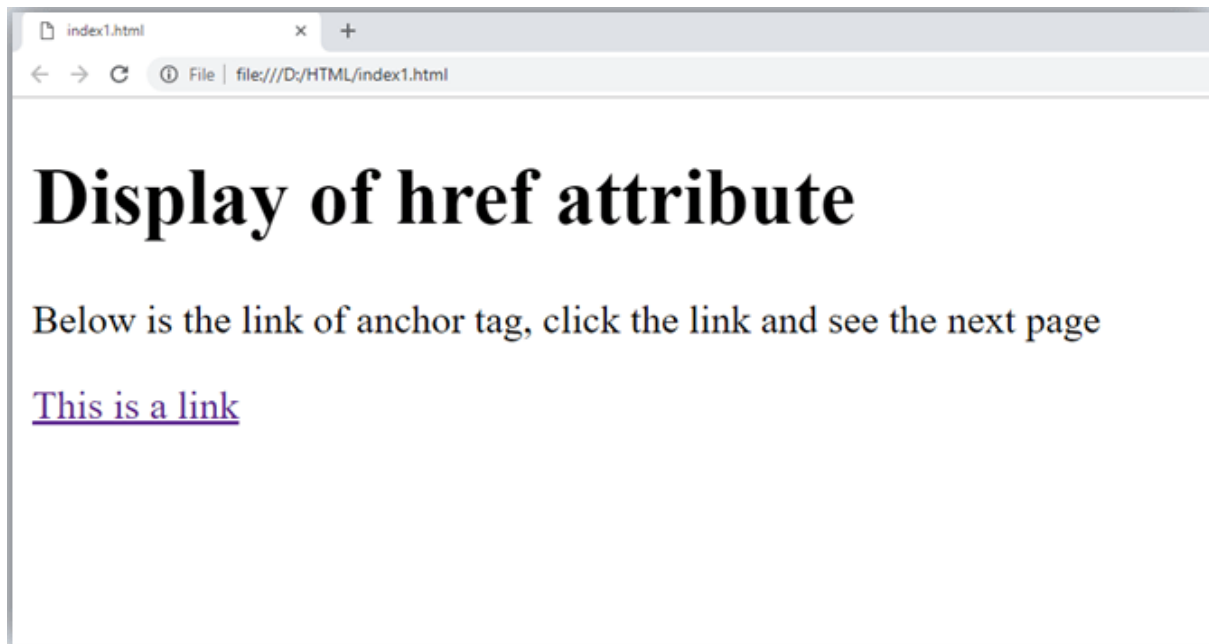
Example

With link address:

1. `This is a link`

Without link address:

1. `This is a link`



The src Attribute

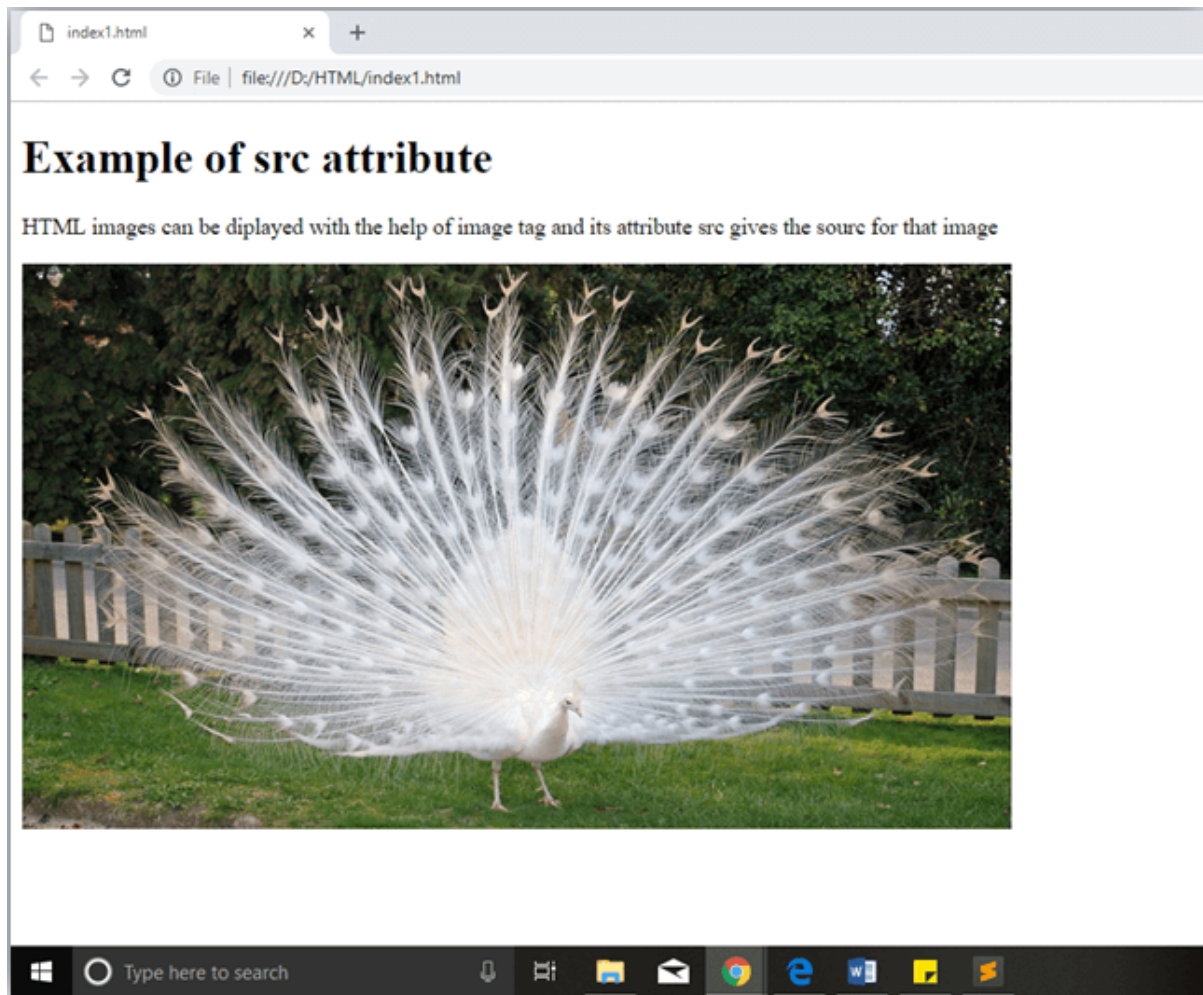
The **src** attribute is one of the important and required attribute of **** element. It is source for the image which is required to display on browser. This attribute can contain image in same directory or another directory. The image name or source should be correct else browser will not display the image.

Example

1. ``

Note: The above example also have height and width attribute, which define the height and width of image on web page.

Output:



Quotes: single quotes or double quotes?

In this chapter you have seen that, we have used attribute with double quotes, but some people might use single quotes in HTML. So use of single quotes with HTML attribute, is also allowed. The following both statements are absolutely fine.

1. `A link to HTML.`
2. `A link to HTML.`

IN HTML5, you can also omit use of quotes around attribute values.

1. `A link to HTML.`

HTML Elements

An HTML file is made of elements. These elements are responsible for creating web pages and define content in that webpage. An element in HTML usually consist of a start tag `<tag name>`, close tag `</tag name>` and content inserted between them. **Technically, an element is a collection of start tag, attributes, end tag, content between them.**

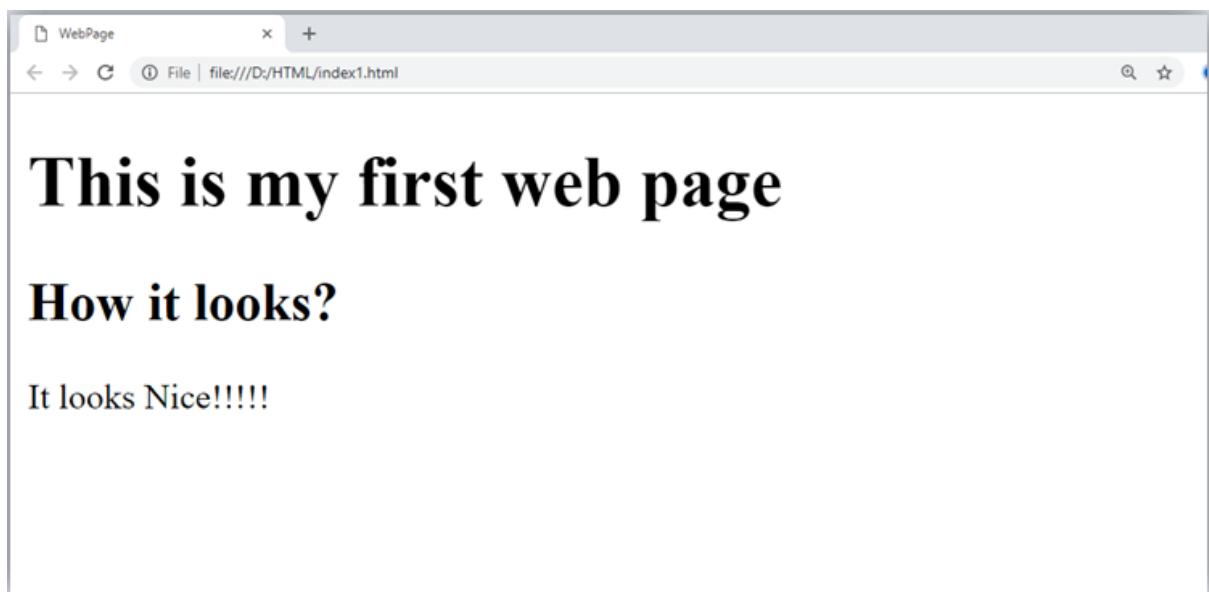
Note: Some elements does not have end tag and content, these elements are termed as empty elements or self-closing element or void elements.

Such as:

1. `<p> Hello world!!! </p>`

Example

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<title>WebPage</title>`
5. `</head>`
6. `<body>`
7. `<h1>This is my first web page</h1>`
8. `<h2> How it looks?</h2>`
9. `<p>It looks Nice!!!!</p>`
10. `</body>`
11. `</html>`



- All the content written between body elements are visible on web page.

Void element: All the elements in HTML do not require to have start tag and end tag, some elements does not have content and end tag such elements are known as Void elements or empty elements. **These elements are also called as unpaired tag.**

Some Void elements are `
` (represents a line break) ,`<hr>`(represents a horizontal line), etc.

Nested HTML Elements: HTML can be nested, which means an element can contain another element.

Block-level and Inline HTML elements

For the default display and styling purpose in HTML, all the elements are divided into two categories:

- Block-level element
 - Inline element
-

Block-level element:

- These are the elements, which structure main part of web page, by dividing a page into coherent blocks.
- A block-level element always start with new line and takes the full width of web page, from left to right.
- These elements can contain block-level as well as inline elements.

Following are the block-level elements in HTML.

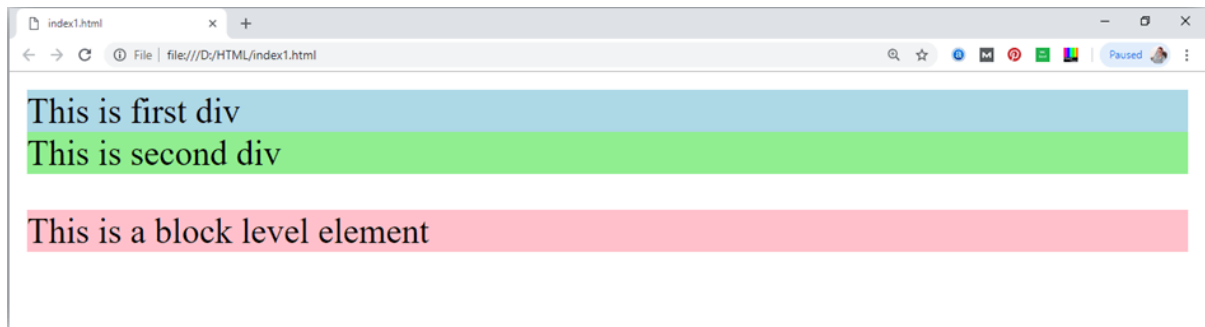
<address>, <article>, <aside>, <blockquote>, <canvas>, <dd>, <div>, <dl>, <dt>, <fieldset>, <figcaption>, <figure>, <footer>, <form>, <h1>-<h6>, <header>, <hr>, , <main>, <nav>, <noscript>, , <output>, <p>, <pre>, <section>, <table>, <tfoot>, and <video>.

Note: All these elements are described in later chapters.

Example:

1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <div style="background-color: lightblue">This is first div</div>
7. <div style="background-color: lightgreen">This is second div</div>
8. <p style="background-color: pink">This is a block level element</p>
9. </body>
10. </html>

Output:



In the above example we have used

tag, which defines a section in a web page, and takes full width of page.

We have used style attribute which is used to styling the HTML content, and the background color are showing that it's a block level element.

Inline elements:

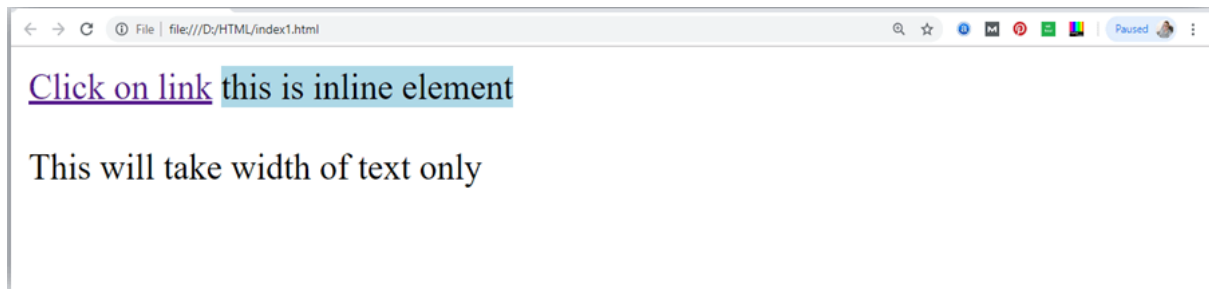
- Inline elements are those elements, which differentiate the part of a given text and provide it a particular function.
- These elements does not start with new line and take width as per requirement.
- The Inline elements are mostly used with other elements.

<a>, <abbr>, <acronym>, , <bdo>, <big>,
, <button>, <cite>, <code>, <dfn>, , <i>, , <input>, <kbd>, <label>, <map>, <object>, <q>, <samp>, <script>, <select>, <small>, , , <sub>, <sup>, <textarea>, <time>, <tt>, <var>.

Example:

1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. Click on link
7. this is inline element
8. <p>This will take width of text only</p>
9. </body>
10. </html>

Output:



Following is the list of the some main elements used in HTML:

HTML Formatting

HTML Formatting is a process of formatting text for better look and feel. HTML provides us ability to format text without using CSS. There are many formatting tags in HTML. These tags are used to make text bold, italicized, or underlined. There are almost 14 options available that how text appears in HTML and XHTML.

In HTML the formatting tags are divided into two categories:

- Physical tag: These tags are used to provide the visual appearance to the text.
- Logical tag: These tags are used to add some logical or semantic value to the text.

NOTE: There are some physical and logical tags which may give same visual appearance, but they will be different in semantics.

Here, we are going to learn 14 HTML formatting tags. Following is the list of HTML formatting text.

Element name	Description
	This is a physical tag, which is used to bold the text written between it.
	This is a logical tag, which tells the browser that the text is important.
<i>	This is a physical tag which is used to make text italic.
	This is a logical tag which is used to display content in italic.
<mark>	This tag is used to highlight text.
<u>	This tag is used to underline text written between it.
<tt>	This tag is used to appear a text in teletype. (not supported in HTML5)
<strike>	This tag is used to draw a strikethrough on a section of text. (Not supported in HTML5)
<sup>	It displays the content slightly above the normal line.

<code><sub></code>	It displays the content slightly below the normal line.
<code></code>	This tag is used to display the deleted content.
<code><ins></code>	This tag displays the content which is added
<code><big></code>	This tag is used to increase the font size by one conventional unit.
<code><small></code>	This tag is used to decrease the font size by one unit from base font size.

1) Bold Text

HTML `` and `` formatting elements

The HTML `` element is a physical tag which display text in bold font, without any logical importance. If you write anything within `.....` element, is shown in bold letters.

See this example:

1. `<p> Write Your First Paragraph in bold text.</p>`

Output:

Write Your First Paragraph in bold text.

The HTML `` tag is a logical tag, which displays the content in bold font and informs the browser about its logical importance. If you write anything between `??????.`, is shown important text.

See this example:

1. `<p>This is an important content, and this is normal content</p>`

Output:

This is an important content, and this is normal content

Example

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<title>formatting elements</title>`
5. `</head>`
6. `<body>`
7. `<h1>Explanation of formatting element</h1>`
8. `<p>This is an important content, and this is normal content</p>`
9. `</body>`
10. `</html>`

2) Italic Text

HTML <i> and formatting elements

The HTML <i> element is physical element, which display the enclosed content in italic font, without any added importance. If you write anything within <i>.....</i> element, is shown in italic letters.

See this example:

1. <p> <i>Write Your First Paragraph in italic text.</i></p>

Output:

Write Your First Paragraph in italic text.

The HTML tag is a logical element, which will display the enclosed content in italic font, with added semantics importance.

See this example:

1. <p>This is an important content, which displayed in italic font.</p>

Output:

This is an important content, which displayed in italic font.

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>formatting elements</title>
5. </head>
6. <body>
7. <h1>Explanation of italic formatting element</h1>
8. <p>This is an important content, which displayed in italic font.</p>
9. </body>
10. </html>

3) HTML Marked formatting

If you want to mark or highlight a text, you should write the content within <mark>.....</mark>.

See this example:

-
1. `<h2> I want to put a <mark> Mark</mark> on your face</h2>`

Output:

I want to put a Mark on your face

4) Underlined Text

If you write anything within `<u>.....</u>` element, is shown in underlined text.

See this example:

1. `<p> <u>Write Your First Paragraph in underlined text.</u></p>`

Output:

Write Your First Paragraph in underlined text.

5) Strike Text

Anything written within `<strike>.....</strike>` element is displayed with strikethrough. It is a thin line which cross the statement.

See this example:

1. `<p> <strike>Write Your First Paragraph with strikethrough</strike>.</p>`

Output:

~~Write Your First Paragraph with strikethrough.~~

6) Monospaced Font

If you want that each letter has the same width then you should write the content within `<tt>.....</tt>` element.

Note: We know that most of the fonts are known as variable-width fonts because different letters have different width. (for example: 'w' is wider than 'i'). Monospaced Font provides similar space among every letter.

See this example:

-
1. `<p>Hello <tt>Write Your First Paragraph in monospaced font.</tt></p>`

Output:

Hello Write Your First Paragraph in monospaced font.

7) Superscript Text

If you put the content within `^{.....}` element, is shown in superscript; means it is displayed half a character's height above the other characters.

See this example:

1. `<p>Hello ^{Write Your First Paragraph in superscript.}</p>`

Output:

Hello Write Your First Paragraph in superscript.

8) Subscript Text

If you put the content within `_{.....}` element, is shown in subscript ; means it is displayed half a character's height below the other characters.

See this example:

1. `<p>Hello _{Write Your First Paragraph in subscript.}</p>`

Output:

Hello Write Your First Paragraph in subscript.

HTML Heading

A HTML heading or HTML h tag can be defined as a title or a subtitle which you want to display on the webpage. When you place the text within the heading tags `<h1>.....</h1>`, it is displayed on the browser in the bold format and size of the text depends on the number of heading.

There are six different HTML headings which are defined with the `<h1>` to `<h6>` tags, from highest level h1 (main heading) to the least level h6 (least important heading).

h1 is the largest heading tag and h6 is the smallest one. So h1 is used for most important heading and h6 is used for least important.

Headings in HTML helps the search engine to understand and index the structure of web page.

Note: The main keyword of the whole content of a webpage should be display by h1 heading tag.

See this example:

1. `<h1>Heading no. 1</h1>`
2. `<h2>Heading no. 2</h2>`
3. `<h3>Heading no. 3</h3>`
4. `<h4>Heading no. 4</h4>`
5. `<h5>Heading no. 5</h5>`
6. `<h6>Heading no. 6</h6>`

Output:

Heading no. 1

Heading no. 2

Heading no. 3

Heading no. 4

Heading no. 5

Heading no. 6

Heading elements (h1....h6) should be used for headings only. They should not be used just to make text bold or big.

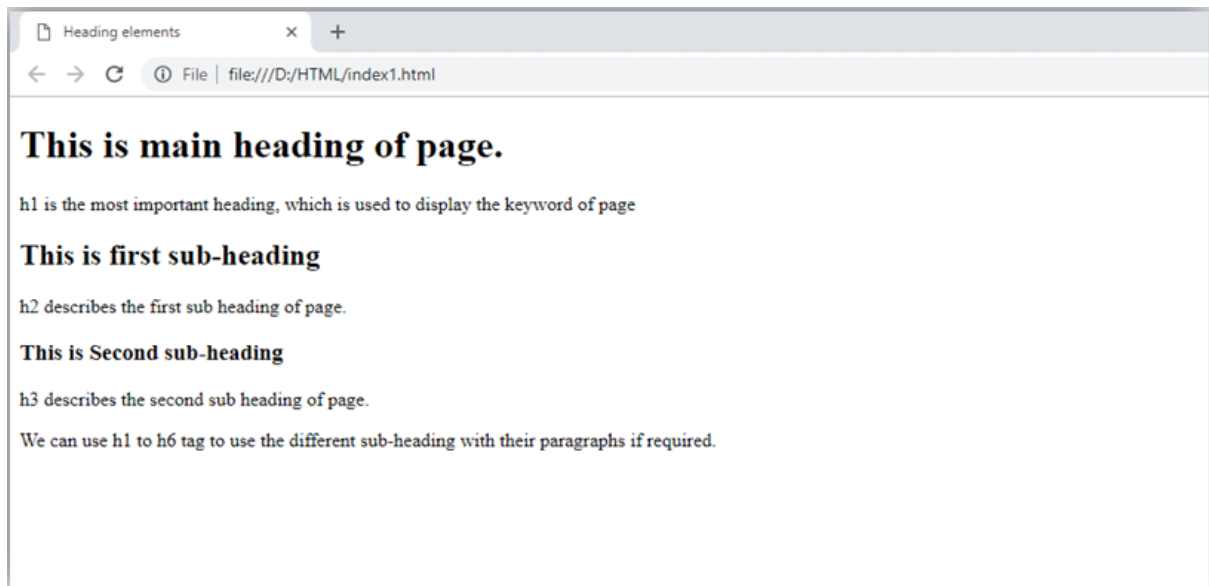
- **HTML headings can also be used with nested elements. Following are different codes to display the way to use heading elements.**

Example:

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<title>Heading elements</title>`
5. `</head>`
6. `<body>`
7. `<h1>This is main heading of page. </h1>`
8. `<p>h1 is the most important heading, which is used to display the keyword of page </p>`
9. `<h2>This is first sub-heading</h2>`
10. `<p>h2 describes the first sub heading of page. </p>`
11. `<h3>This is Second sub-heading</h3>`
12. `<p>h3 describes the second sub heading of page.</p>`
13. `<p>We can use h1 to h6 tag to use the different sub-heading with their paragraphs if`
14. `required.`
15. `</p>`
16. `</body>`

17. `</html>`

Output:



HTML Paragraph

HTML paragraph or HTML p tag is used to define a paragraph in a webpage. Let's take a simple example to see how it work. It is a notable point that a browser itself add an empty line before and after a paragraph. An HTML `<p>` tag indicates starting of new paragraph.

Note: If we are using various `<p>` tags in one HTML file then browser automatically adds a single blank line between the two paragraphs.

See this example:

1. `<p>`This is first paragraph.`</p>`
2. `<p>`This is second paragraph.`</p>`
3. `<p>`This is third paragraph.`</p>`

Output:

This is first paragraph.

This is second paragraph.

This is third paragraph.

Space inside HTML Paragraph

If you put a lot of spaces inside the HTML p tag, browser removes extra spaces and extra line while displaying the page. The browser counts number of spaces and lines as a single one.

1. <p>
2. I am
3. going to provide
4. you a tutorial on HTML
5. and hope that it will
6. be very beneficial for you.
7. </p>
8. <p>
9. Look, I put here a lot
10. of spaces but I know, Browser will ignore it.
11. </p>
12. <p>
13. You cannot determine the display of HTML</p>
14. <p>because resized windows may create different result.
15. </p>

Output:

I am going to provide you a tutorial on HTML and hope that it will be very beneficial for you.

Look, I put here a lot of spaces but I know, Browser will ignore it.

You cannot determine the display of HTML

because resized windows may create different result.

As you can see, all the extra lines and unnecessary spaces are removed by the browser.

How to Use
 and <hr> tag with paragraph?

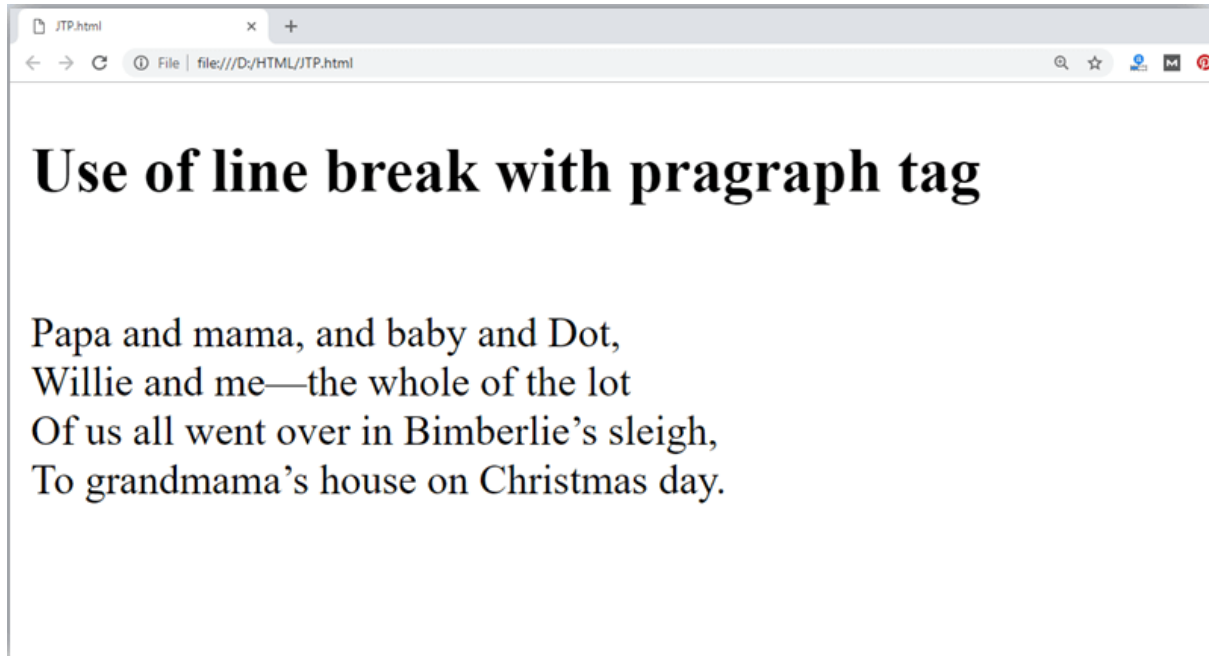
An HTML
 tag is used for line break and it can be used with paragraph elements. Following is the example to show how to use
 with <p> element.

Example:

1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <h2> Use of line break with paragraph tag</h2>
7. <p>
Papa and mama, and baby and Dot,
8.
Willie and me?the whole of the lot
9.
Of us all went over in Bimberlie's sleigh,

10.
To grandmama's house on Christmas day.
11. </p>
12. </body>
13. </html>

Output:

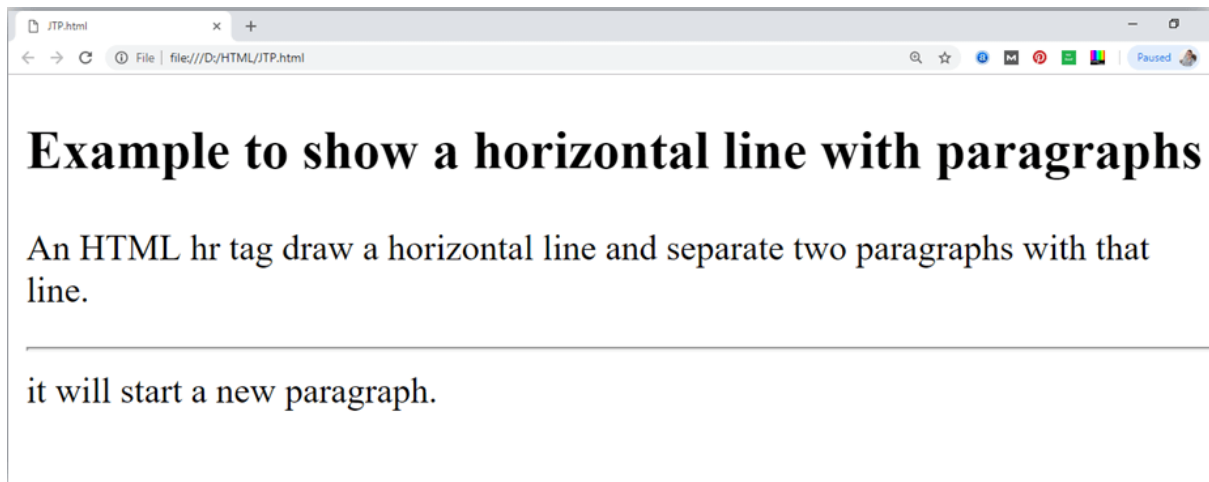


An HTML <hr> tag is used to apply a horizontal line between two statements or two paragraphs. Following is the example which is showing use of <hr> tag with paragraph.

Example:

1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <h2> Example to show a horizontal line with paragraphs</h2>
7. <p> An HTML hr tag draw a horizontal line and separate two paragraphs with that line.<hr> it will start a new paragraph.
8. </p>
9. </body>
10. </html>

Output:



HTML Anchor

The **HTML anchor tag** defines *a hyperlink that links one page to another page*. It can create hyperlink to other web page as well as files, location, or any URL. The "href" attribute is the most important attribute of the HTML a tag, and which links to destination page or URL.

href attribute of HTML anchor tag

The href attribute is used to define the address of the file to be linked. In other words, it points out the destination page.

The syntax of HTML anchor tag is given below.

```
<a href = "....."> Link Text </a>
```

Let's see an example of HTML anchor tag.

1. `Click for Second Page`

Specify a location for Link using target attribute

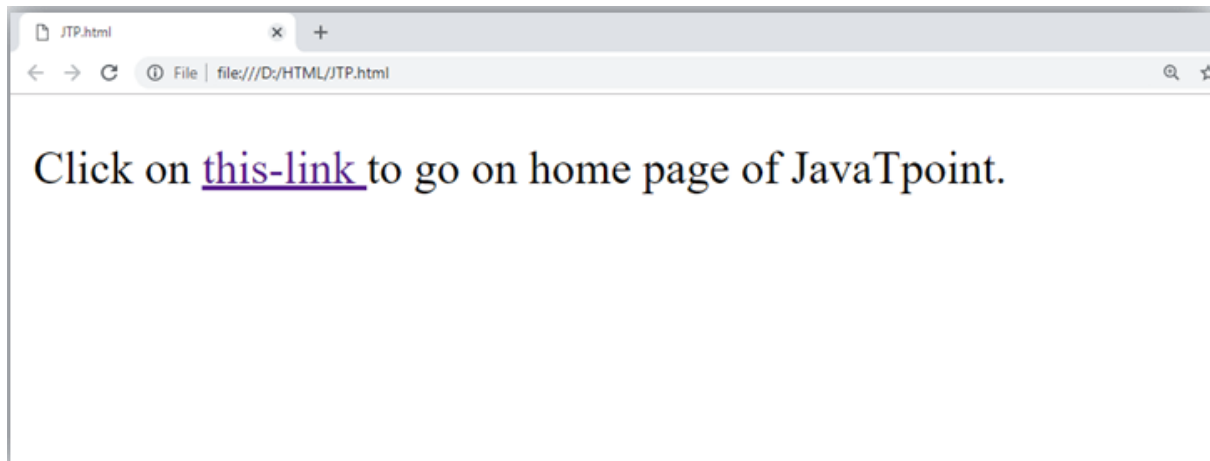
If we want to open that link to another page then we can use target attribute of `<a>` tag. With the help of this link will be open in next page.

Example:

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<title></title>`

5. `</head>`
6. `<body>`
7. `<p>Click on this-link to go on home page of JavaTpoint.</p>`
8. `</body>`
9. `</html>`

Output:



Note:

- The **target** attribute can only use with href attribute in anchor tag.
- If we will not use target attribute then link will open in same page.

Appearance of HTML anchor tag

An **unvisited link** is displayed underlined and blue.

A **visited link** displayed underlined and purple.

An **active link** is underlined and red.

HTML Image

HTML img tag is used to display image on the web page. HTML img tag is an empty tag that contains attributes only, closing tags are not used in HTML image element.

Let's see an example of HTML image.

1. `<h2>HTML Image Example</h2>`
2. ``

Output:



Attributes of HTML img tag

The src and alt are important attributes of HTML img tag. All attributes of HTML image tag are given below.

1) src

It is a necessary attribute that describes the source or path of the image. It instructs the browser where to look for the image on the server.

The location of image may be on the same directory or another server.

2) alt

The alt attribute defines an alternate text for the image, if it can't be displayed. The value of the alt attribute describe the image in words. The alt attribute is considered good for SEO prospective.

3) width

It is an optional attribute which is used to specify the width to display the image. It is not recommended now. You should apply CSS in place of width attribute.

4) height

It h3 the height of the image. The HTML height attribute also supports iframe, image and object elements. It is not recommended now. You should apply CSS in place of height attribute.

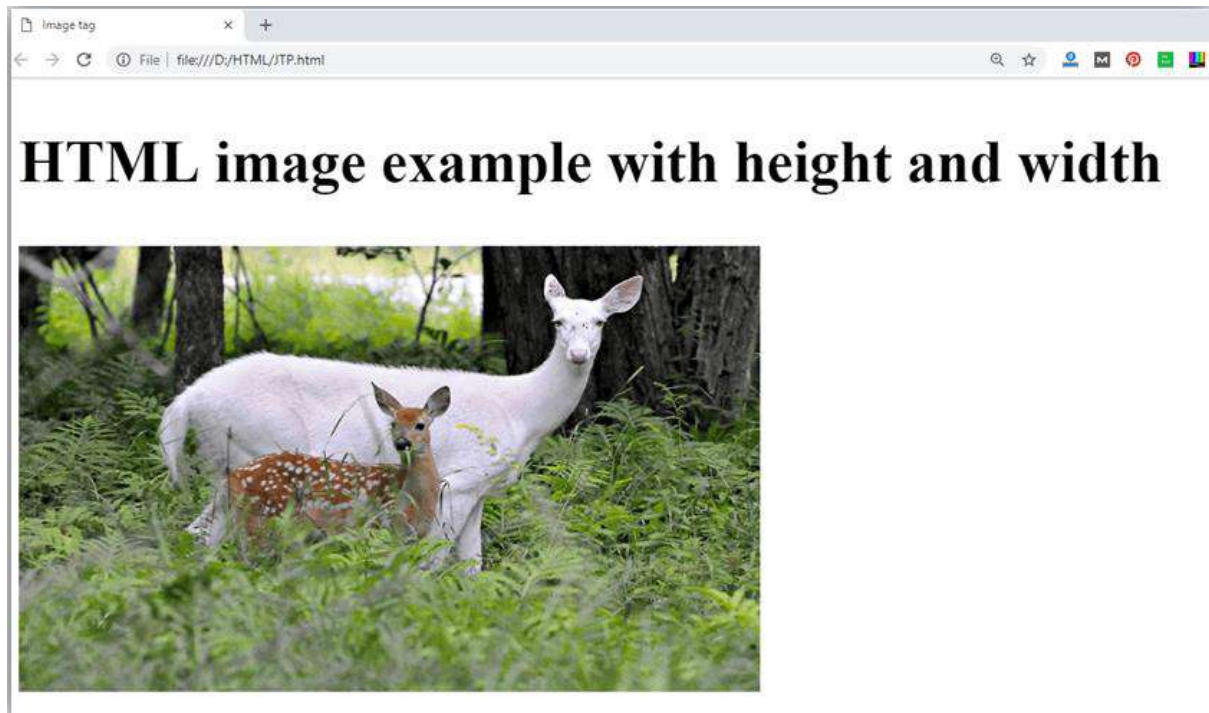
Use of height and width attribute with img tag

You have learnt about how to insert an image in your web page, now if we want to give some height and width to display image according to our requirement, then we can set it with height and width attributes of image.

Example:

1. ``

Output:



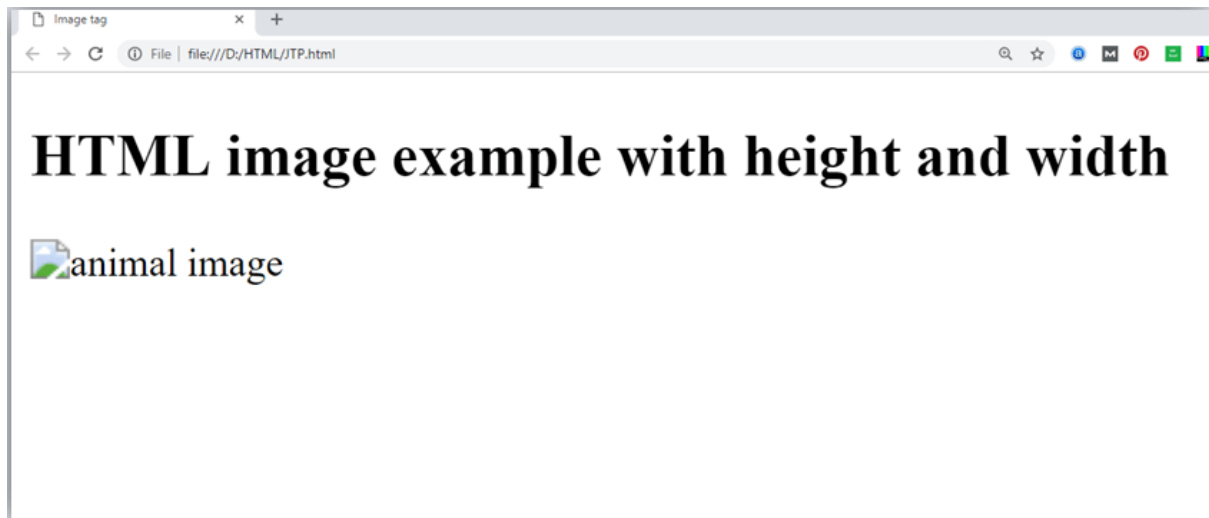
Note: Always try to insert the image with height and width, else it may flicker while displaying on webpage.

Use of alt attribute

We can use alt attribute with `` tag. It will display an alternative text in case if image cannot be displayed on browser. Following is the example for alt attribute:

1. ``

Output:



How to get image from another directory/folder?

To insert an image in your web, that image must be present in your same folder where you have put the HTML file. But if in some case image is available in some other directory then you can access the image like this:

1. ``

In above statement we have put image in local disk E----->images folder----->animal.png.

Note: If src URL will be incorrect or misspell then it will not display your image on web page, so try to put correct URL.

Use tag as a link

We can also link an image with other page or we can use an image as a link. To do this, put tag inside the <a> tag.

Example:

1. ``

Output:



HTML Table

HTML table tag is used to display data in tabular form (row * column). There can be many columns in a row.

We can create a table to display data in tabular form, using `<table>` element, with the help of `<tr>`, `<td>`, and `<th>` elements.

In Each table, table row is defined by `<tr>` tag, table header is defined by `<th>`, and table data is defined by `<td>` tags.

HTML tables are used to manage the layout of the page e.g. header section, navigation bar, body content, footer section etc. But it is recommended to use `div` tag over `table` to manage the layout of the page .

HTML Table Tags

Tag	Description
<code><table></code>	It defines a table.
<code><tr></code>	It defines a row in a table.
<code><th></code>	It defines a header cell in a table.
<code><td></code>	It defines a cell in a table.
<code><caption></code>	It defines the table caption.
<code><colgroup></code>	It specifies a group of one or more columns in a table for formatting.
<code><col></code>	It is used with <code><colgroup></code> element to specify column properties for each column.
<code><tbody></code>	It is used to group the body content in a table.
<code><thead></code>	It is used to group the header content in a table.
<code><tfooter></code>	It is used to group the footer content in a table.

HTML Table Example

Let's see the example of HTML table tag. Its output is shown above.

1. `<table>`
2. `<tr><th>First_Name</th><th>Last_Name</th><th>Marks</th></tr>`
3. `<tr><td>Sonoo</td><td>Jaiswal</td><td>60</td></tr>`
4. `<tr><td>James</td><td>William</td><td>80</td></tr>`
5. `<tr><td>Swati</td><td>Sironi</td><td>82</td></tr>`
6. `<tr><td>Chetna</td><td>Singh</td><td>72</td></tr>`
7. `</table>`

Output:

First_Name	Last_Name	Marks
Sonoo	Jaiswal	60

HTML Lists

HTML Lists are used to specify lists of information. All lists may contain one or more list elements. There are three different types of HTML lists:

1. Ordered List or Numbered List (ol)
2. Unordered List or Bulleted List (ul)
3. Description List or Definition List (dl)

Note: We can create a list inside another list, which will be termed as nested List.

HTML Ordered List or Numbered List

In the ordered HTML lists, all the list items are marked with numbers by default. It is known as numbered list also. The ordered list starts with `` tag and the list items start with `` tag.

1. ``
2. `Aries`
3. `Bingo`
4. `Leo`
5. `Oracle`
6. ``

Output:

1. Aries
2. Bingo
3. Leo
4. Oracle

Click here for full details of HTML ordered list. [HTML Ordered List](#)

HTML Unordered List or Bulleted List

In HTML Unordered list, all the list items are marked with bullets. It is also known as bulleted list also. The Unordered list starts with `` tag and list items start with the `` tag.

1. ``
2. `Aries`
3. `Bingo`
4. `Leo`
5. `Oracle`
6. ``

Output:

- Aries
 - Bingo
 - Leo
 - Oracle
-
-

HTML Description List or Definition List

HTML Description list is also a list style which is supported by HTML and XHTML. It is also known as definition list where entries are listed like a dictionary or encyclopedia.

The definition list is very appropriate when you want to present glossary, list of terms or other name-value list.

The HTML definition list contains following three tags:

1. **`<dl>` tag** defines the start of the list.
 2. **`<dt>` tag** defines a term.
 3. **`<dd>` tag** defines the term definition (description).
-
1. `<dl>`
 2. `<dt>Aries</dt>`
 3. `<dd>-One of the 12 horoscope sign.</dd>`
 4. `<dt>Bingo</dt>`
 5. `<dd>-One of my evening snacks</dd>`
 6. `<dt>Leo</dt>`
 7. `<dd>-It is also an one of the 12 horoscope sign.</dd>`
 8. `<dt>Oracle</dt>`

-
9. `<dd>-It is a multinational technology corporation.</dd>`
 10. `</dl>`

Output:

Aries

-One of the 12 horoscope sign.

Bingo

-One of my evening snacks

Leo

-It is also an one of the 12 horoscope sign.

Oracle

-It is a multinational technology corporation.

Click here for full details of HTML description list. [HTML Description List](#)

HTML Nested List

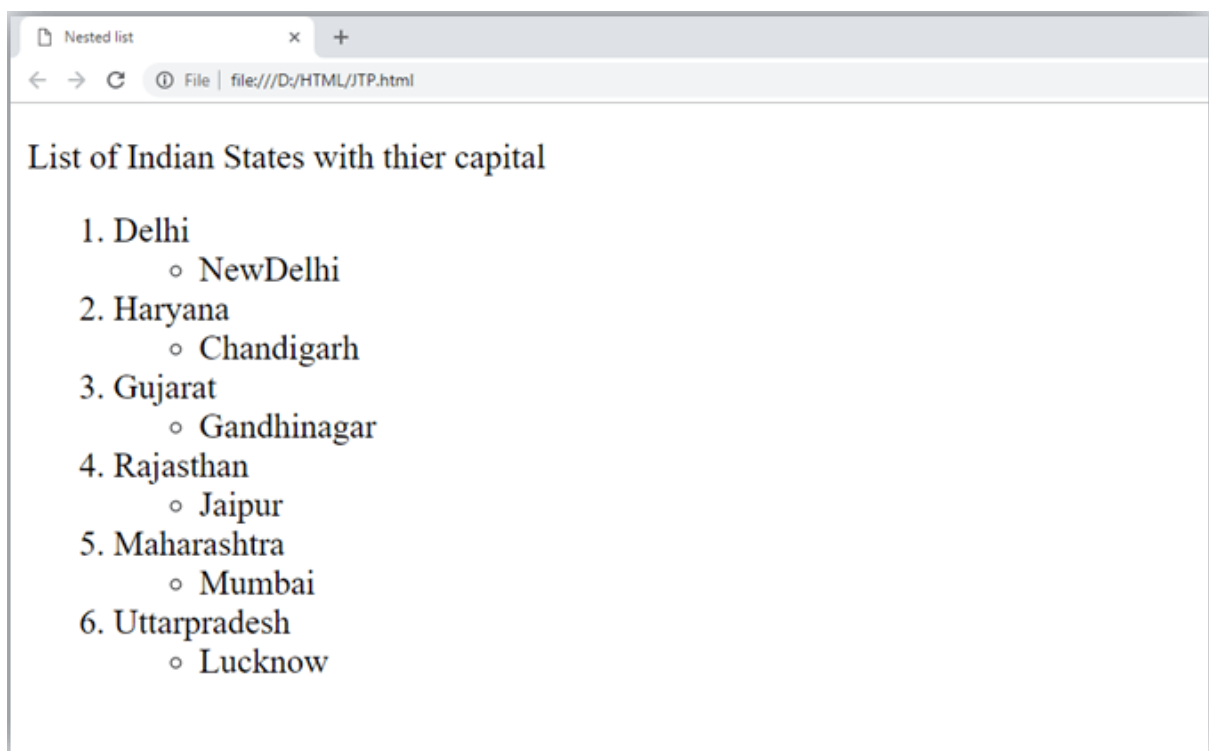
A list within another list is termed as nested list. If you want a bullet list inside a numbered list then such type of list will called as nested list.

Code:

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<title>Nested list</title>`
5. `</head>`
6. `<body>`
7. `<p>List of Indian States with thier capital</p>`
8. ``
9. `Delhi`
10. ``
11. `NewDelhi`
12. ``
13. ``
14. `Haryana`
15. ``
16. `Chandigarh`
17. ``
18. ``
19. `Gujarat`
20. ``

```
21.     <li>Gandhinagar</li>
22.     </ul>
23. </li>
24. <li>Rajasthan
25.     <ul>
26.         <li>Jaipur</li>
27.     </ul>
28. </li>
29. <li>Maharashtra
30.     <ul>
31.         <li>Mumbai</li>
32.     </ul>
33. </li>
34. <li>Uttarpradesh
35.     <ul>
36.         <li>Lucknow</li></ul>
37. </li>
38. </ol>
39. </body>
40. </html>
```

Output:



HTML Ordered List | HTML Numbered List

HTML Ordered List or Numbered List displays elements in numbered format. The HTML `ol` tag is used for ordered list. We can use ordered list to represent items either in numerical order format or alphabetical order format, or any format where an order is emphasized. There can be different types of numbered list:

- Numeric Number (1, 2, 3)
- Capital Roman Number (I II III)
- Small Roman Number (i ii iii)
- Capital Alphabet (A B C)
- Small Alphabet (a b c)

To represent different ordered lists, there are 5 types of attributes in `` tag.

Type	Description
Type "1"	This is the default type. In this type, the list items are numbered with numbers.
Type "I"	In this type, the list items are numbered with upper case roman numbers.
Type "i"	In this type, the list items are numbered with lower case roman numbers.
Type "A"	In this type, the list items are numbered with upper case letters.
Type "a"	In this type, the list items are numbered with lower case letters.

HTML Ordered List Example

Let's see the example of HTML ordered list that displays 4 topics in numbered list. Here we are not defining `type="1"` because it is the default type.

1. ``
2. `HTML`
3. `Java`
4. `JavaScript`
5. `SQL`
6. ``

Output:

1. HTML
 2. Java
 3. JavaScript
 4. SQL
-

`ol type="I"`

Let's see the example to display list in roman number uppercase.

1. `<ol type="I">`

-
2. `HTML`
 3. `Java`
 4. `JavaScript`
 5. `SQL`
 6. ``

Output:

- I. HTML
 - II. Java
 - III. JavaScript
 - IV. SQL
-

ol type="i"

Let's see the example to display list in roman number lowercase.

1. `<ol type="i">`
2. `HTML`
3. `Java`
4. `JavaScript`
5. `SQL`
6. ``

Output:

- i. HTML
 - ii. Java
 - iii. JavaScript
 - iv. SQL
-

ol type="A"

Let's see the example to display list in alphabet uppercase.

1. `<ol type="A">`
2. `HTML`
3. `Java`
4. `JavaScript`
5. `SQL`
6. ``

Output:

- A. HTML
-

-
- B. Java
 - C. JavaScript
 - D. SQL
-

ol type="a"

Let's see the example to display list in alphabet lowercase.

1. `<ol type="a">`
2. `HTML`
3. `Java`
4. `JavaScript`
5. `SQL`
6. ``

Output:

- a. HTML
 - b. Java
 - c. JavaScript
 - d. SQL
-

start attribute

The start attribute is used with ol tag to specify from where to start the list items.

`<ol type="1" start="5">` : It will show numeric values starting with "5".

`<ol type="A" start="5">` : It will show capital alphabets starting with "E".

`<ol type="a" start="5">` : It will show lower case alphabets starting with "e".

`<ol type="I" start="5">` : It will show Roman upper case value starting with "V".

`<ol type="i" start="5">` : It will show Roman lower case value starting with "v".

1. `<ol type="i" start="5">`
2. `HTML`
3. `Java`
4. `JavaScript`
5. `SQL`
6. ``

Output:

-
- v. HTML
 - vi. Java
 - vii. JavaScript
 - viii. SQL

HTML Form

An **HTML form** is *a section of a document* which contains controls such as text fields, password fields, checkboxes, radio buttons, submit button, menus etc.

An HTML form facilitates the user to enter data that is to be sent to the server for processing such as name, email address, password, phone number, etc. .

Why use HTML Form

HTML forms are required if you want to collect some data from of the site visitor.

For example: If a user want to purchase some items on internet, he/she must fill the form such as shipping address and credit/debit card details so that item can be sent to the given address.

HTML Form Syntax

1. <form action="server url" method="get|post">
 2. //input controls e.g. textfield, textarea, radiobutton, button
 3. </form>
-

HTML Form Tags

Let's see the list of HTML 5 form tags.

Tag	Description
<form>	It defines an HTML form to enter inputs by the used side.
<input>	It defines an input control.
<textarea>	It defines a multi-line input control.
<label>	It defines a label for an input element.
<fieldset>	It groups the related element in a form.

-
- | | |
|------------|--|
| <legend> | It defines a caption for a <fieldset> element. |
| <select> | It defines a drop-down list. |
| <optgroup> | It defines a group of related options in a drop-down list. |
| <option> | It defines an option in a drop-down list. |
| <button> | It defines a clickable button. |

HTML 5 Form Tags

Let's see the list of HTML 5 form tags.

Tag	Description
<datalist>	It specifies a list of pre-defined options for input control.
<keygen>	It defines a key-pair generator field for forms.
<output>	It defines the result of a calculation.

HTML <form> element

The HTML <form> element provide a document section to take input from user. It provides various interactive controls for submitting information to web server such as text field, text area, password field, etc.

Note: The <form> element does not itself create a form but it is container to contain all required form elements, such as <input>, <label>, etc.

Syntax:

1. <form>
 2. //Form elements
 3. </form>
-

HTML <input> element

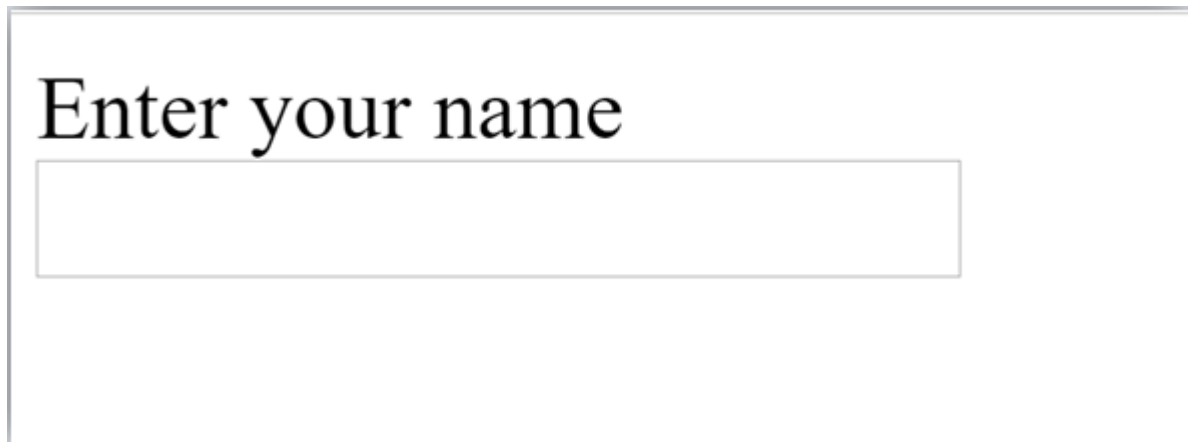
The HTML <input> element is fundamental form element. It is used to create form fields, to take input from user. We can apply different input filed to gather different information form user. Following is the example to show the simple text input.

Example:

1. <body>
2. <form>
3. Enter your name

4. <input type="text" name="username">
5. </form>
6. </body>

Output:



Enter your name

HTML TextField Control

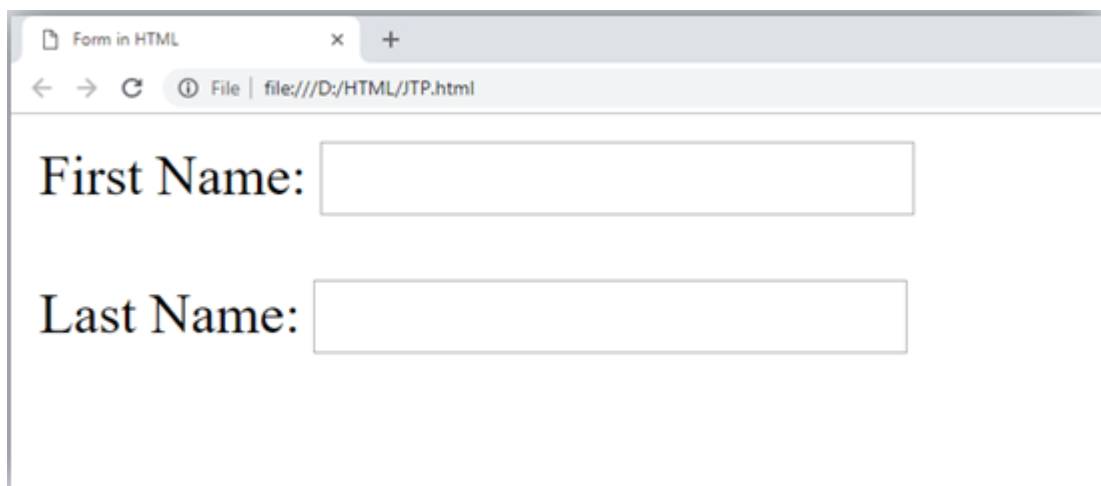
The type="text" attribute of input tag creates textfield control also known as single line textfield control. The name attribute is optional, but it is required for the server side component such as JSP, ASP, PHP etc.

1. <form>
2. First Name: <input type="text" name="firstname"/>

3. Last Name: <input type="text" name="lastname"/>

4. </form>

Output:



First Name:

Last Name:

Note: If you will omit 'name' attribute then the text filed input will not be submitted to server.

HTML <textarea> tag in form

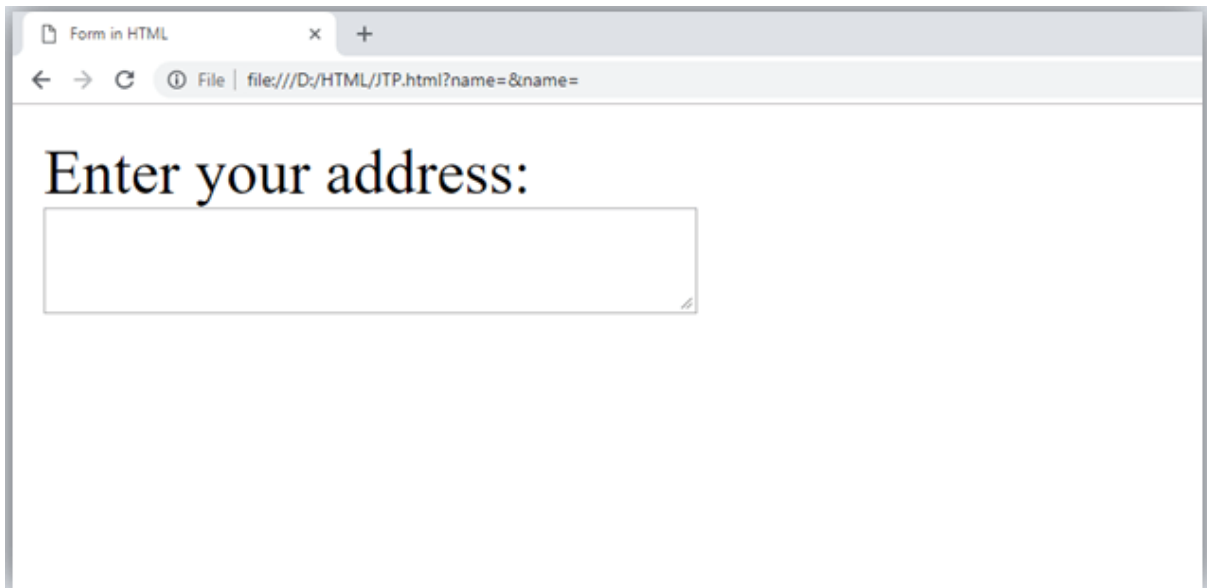
The <textarea> tag in HTML is used to insert multiple-line text in a form. The size of <textarea> can be specify either using "rows" or "cols" attribute or by CSS.

Example:

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>Form in HTML</title>
5. </head>
6. <body>
7. <form>
8. Enter your address:

9. <textarea rows="2" cols="20"></textarea>
10. </form>
11. </body>
12. </html>

Output:



Label Tag in Form

It is considered better to have label in form. As it makes the code parser/browser/user friendly.

If you click on the label tag, it will focus on the text control. To do so, you need to have for attribute in label tag that must be same as id attribute of input tag.

NOTE: It is good to use <label> tag with form, although it is optional but if you will use it, then it will provide a focus when you tap or click on label tag. It is more worthy with touchscreens.

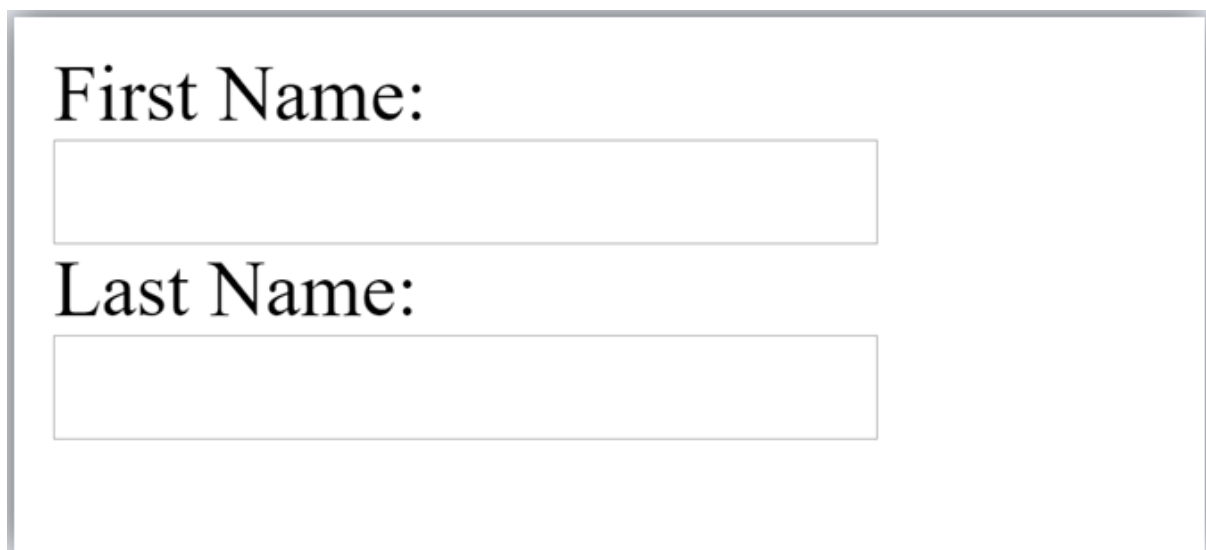
1. <form>
2. <label for="firstname">First Name: </label>

3. <input type="text" id="firstname" name="firstname"/>

4. <label for="lastname">Last Name: </label>
5. <input type="text" id="lastname" name="lastname"/>

6. </form>

Output:

The screenshot shows a web form with two text input fields. The first field is labeled 'First Name:' and the second field is labeled 'Last Name:'. Both labels are in a large, black, serif font. The input fields are simple rectangular boxes with a thin border. The entire form is enclosed in a light gray border.

HTML Password Field Control

The password is not visible to the user in password field control.

1. <form>
2. <label for="password">Password: </label>
3. <input type="password" id="password" name="password"/>

4. </form>

Output:

Password:

HTML 5 Email Field Control

The email field is new in HTML 5. It validates the text for correct email address. You must use @ and . in this field.

1. `<form>`
2. `<label for="email">Email: </label>`
3. `<input type="email" id="email" name="email"/>
`
4. `</form>`

It will display in browser like below:

Email:

Note: If we will not enter the correct email, it will display error like:

Email:



Please include an '@' in the email address.
'example.com' is missing an '@'.

Radio Button Control

The radio button is used to select one option from multiple options. It is used for selection of gender, quiz questions etc.

If you use one name for all the radio buttons, only one radio button can be selected at a time.

Using radio buttons for multiple options, you can only choose a single option at a time.

1. `<form>`
2. `<label for="gender">Gender: </label>`
3. `<input type="radio" id="gender" name="gender" value="male"/>Male`
4. `<input type="radio" id="gender" name="gender" value="female"/>Female
`
5. `</form>`



The screenshot shows a web form with the text 'Gender: Male Female'. There are two radio buttons: one for 'Male' which is unselected, and one for 'Female' which is selected (indicated by a black dot in the center of the circle).

Checkbox Control

The checkbox control is used to check multiple options from given checkboxes.

1. `<form>`
2. `Hobby:
`
3. `<input type="checkbox" id="cricket" name="cricket" value="cricket"/>`
4. `<label for="cricket">Cricket</label>
`
5. `<input type="checkbox" id="football" name="football" value="football"/>`
6. `<label for="football">Football</label>
`
7. `<input type="checkbox" id="hockey" name="hockey" value="hockey"/>`
8. `<label for="hockey">Hockey</label>`
9. `</form>`

Note: These are similar to radio button except it can choose multiple options at a time and radio button can select one button at a time, and its display.

Output:

Hobby:

- ☒ Cricket
- ☒ Football
- ☐ Hockey

Submit button control

HTML `<input type="submit">` are used to add a submit button on web page. When user clicks on submit button, then form get submit to the server.

Syntax:

1. `<input type="submit" value="submit">`

The type = submit , specifying that it is a submit button

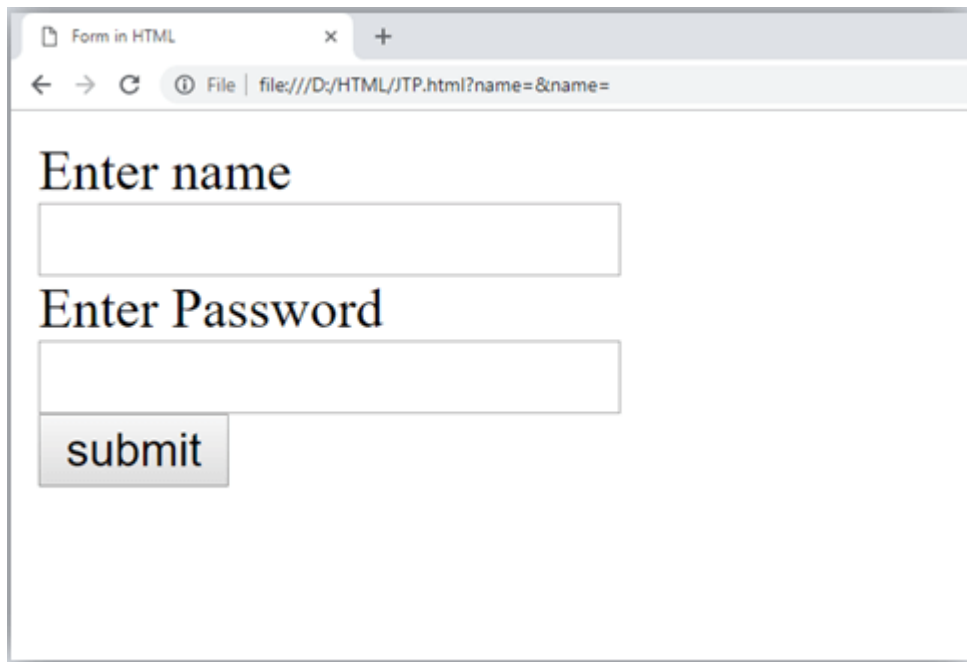
The value attribute can be anything which we write on button on web page.

The name attribute can be omit here.

Example:

1. `<form>`
2. `<label for="name">Enter name</label>
`
3. `<input type="text" id="name" name="name">
`
4. `<label for="pass">Enter Password</label>
`
5. `<input type="Password" id="pass" name="pass">
`
6. `<input type="submit" value="submit">`
7. `</form>`

Output:



HTML <fieldset> element:

The <fieldset> element in HTML is used to group the related information of a form. This element is used with <legend> element which provide caption for the grouped elements.

Example:

1. <form>
2. <fieldset>
3. <legend>User Information:</legend>
4. <label for="name">Enter name</label>

5. <input type="text" id="name" name="name">

6. <label for="pass">Enter Password</label>

7. <input type="Password" id="pass" name="pass">

8. <input type="submit" value="submit">
9. </fieldset>
10. </form>

Output:



HTML Form Example

Following is the example for a simple form of registration.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <title>Form in HTML</title>
5. </head>
6. <body>
7.   <h2>Registration form</h2>
8.   <form>
9.     <fieldset>
10.      <legend>User personal information</legend>
11.      <label>Enter your full name</label><br>
12.      <input type="text" name="name"><br>
13.      <label>Enter your email</label><br>
14.      <input type="email" name="email"><br>
15.      <label>Enter your password</label><br>
16.      <input type="password" name="pass"><br>
17.      <label>confirm your password</label><br>
18.      <input type="password" name="pass"><br>
19.      <br><label>Enter your gender</label><br>
20.      <input type="radio" id="gender" name="gender" value="male"/>Male <br>
21.      <input type="radio" id="gender" name="gender" value="female"/>Female <br>
22.      <input type="radio" id="gender" name="gender" value="others"/>others <br>
23.      <br><label>Enter your Address:<br>
24.      <textarea></textarea><br>
25.      <input type="submit" value="sign-up">
26.    </fieldset>
27.  </form>
28. </body>
29. </html>
```

Output:

Registration form

User personal information

Enter your full name

Enter your email

Enter your password

confirm your password

Enter your gender

☐ Male
☐ Female
☐ others

Enter your Address:

sign-up

HTML Form Input Types

In HTML `<input type=" ">` is an important element of HTML form. The "type" attribute of input element can be various types, which defines information field. Such as `<input type="text" name="name">` gives a text box.

Following is a list of all types of `<input>` element of HTML.

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

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

type=" "	Description
color	Defines an input field with a specific color.
date	Defines an input field for selection of date.
datetime-local	Defines an input field for entering a date without time zone.
email	Defines an input field for entering an email address.
month	Defines a control with month and year, without time zone.
number	Defines an input field to enter a number.
url	Defines a field for entering URL
week	Defines a field to enter the date with week-year, without time zone.
search	Defines a single line text field for entering a search string.
tel	Defines an input field for entering the telephone number.

Following is the description about types of <input> element with examples.

1. `<input type="text">`:

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

Example:

1. `<form>`
2. `<label>Enter first name</label>
`
3. `<input type="text" name="firstname">
`
4. `<label>Enter last name</label>
`
5. `<input type="text" name="lastname">
`
6. `<p>Note:The default maximum cahracter lenght is 20.</p>`
7. `</form>`

Output:

Input "text" type:

The "**text**" field defines a sinlge line input text field.

Enter first name

Enter last name

Note: The default maximum character length is 20.

2. `<input type="password">`:

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

Example:

1. `<form>`
2. `<label>Enter User name</label>
`
3. `<input type="text" name="firstname">
`
4. `<label>Enter Password</label>
`
5. `<input type="Password" name="password">
`
6. `
<input type="submit" value="submit">`
7. `</form>`

Output:

Input "password" type:

The "**password**" field defines a single line input password field to enter the password securely.

Enter User name

Enter Password

3. `<input type="submit">`:

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

Example:

1. `<form action="https://www.javatpoint.com/html-tutorial">`
2. `<label>Enter User name</label>
`
3. `<input type="text" name="firstname">
`
4. `<label>Enter Password</label>
`
5. `<input type="Password" name="password">
`
6. `
<input type="submit" value="submit">`

7. `</form>`

Output:

Input "submit" type:

Enter User name

Enter Password

After clicking on submit button, this will submit the form to server and will redirect the page to **action** value. We will learn about "action" attribute in later chapters

4. `<input type="reset">`:

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

Example:

1. `<form>`
2. `<label>User id: </label>`
3. `<input type="text" name="user-id" value="user">`
4. `<label>Password: </label>`
5. `<input type="password" name="pass" value="pass">

`
6. `<input type="submit" value="login">`
7. `<input type="reset" value="Reset">`
8. `</form>`

Output:

Input "reset" type:

User id: Password:

Try to change the input values of user id and password, then when you click on reset, it will reset input fields with default values.

5. `<input type="radio">`:

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

Example:

1. `<form>`

2. `<p>Kindly Select your favorite color</p>`
3. `<input type="radio" name="color" value="red"> Red
`
4. `<input type="radio" name="color" value="blue"> blue
`
5. `<input type="radio" name="color" value="green">green
`
6. `<input type="radio" name="color" value="pink">pink
`
7. `<input type="submit" value="submit">`
8. `</form>`

Output:

Input "radio" type

Kindly Select your favoritecolor

- ☐ Red
- ☐ blue
- ☐ green
- ☐ pink

6. `<input type="checkbox">`:

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

Note: The "radio" buttons are similar to checkboxes, but there is an important difference between both types: radio buttons allow the user to select only one option at a time, whereas checkbox allows a user to select zero to multiple options at a time.

Example:

1. `<form>`
2. `<label>Enter your Name:</label>`
3. `<input type="text" name="name">`
4. `<p>Kindly Select your favourite sports</p>`
5. `<input type="checkbox" name="sport1" value="cricket">Cricket
`
6. `<input type="checkbox" name="sport2" value="tennis">Tennis
`
7. `<input type="checkbox" name="sport3" value="football">Football
`
8. `<input type="checkbox" name="sport4" value="baseball">Baseball
`
9. `<input type="checkbox" name="sport5" value="badminton">Badminton

`
10. `<input type="submit" value="submit">`
11. `</form>`

Output:

Input "checkbox" type

Registration Form

Enter your Name:

Kindly Select your favorite sports

- ☐ Cricket
- ☐ Tennis
- ☐ Football
- ☐ Baseball
- ☐ Badminton

7. <input type="button">:

The <input> type "button" defines a simple push button, which can be programmed to control a functionally on any event such as, click event.

Note: It mainly works with JavaScript.

Example:

1. <form>
2. <input type="button" value="Click me " onclick="alert('you are learning HTML')">
3. </form>

Output:

Input "button" type.

Click the button to see the result:

Note: In the above example we have used the "alert" of JS, which you will learn in our JS tutorial. It is used to show a pop window.

8. <input type="file">:

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

Example:

1. <form>
2. <label>Select file to upload:</label>

-
3. `<input type="file" name="newfile">`
 4. `<input type="submit" value="submit">`
 5. `</form>`

Output:

Input "file" type.

We can choose any type of file until we do not specify it! The selected file will appear at next to "choose file" option

Select file to upload:

9. `<input type="image">`:

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

Example:

1. `<!DOCTYPE html>`
 2. `<html>`
 3. `<body>`
 4. `<h2>Input "image" type.</h2>`
 5. `<p>We can create an image as submit button</p>`
 6. `<form>`
 7. `<label>User id:</label>
`
 8. `<input type="text" name="name">

`
 9. `<input type="image" alt="Submit" src="login.png" width="100px">`
 10. `</form>`
 - 11.
 12. `</body>`
 13. `</html>`
-

HTML5 newly added `<input>` types element

1. `<input type="color">`:

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

Note: The "color" type only supports color value in hexadecimal format, and the default value is #000000 (black).

Example:

1. `<form>`
2. `Pick your Favorite color:

`

-
3. `<input type="color" name="upclick" value="#a52a2a"> Upclick

`
 4. `<input type="color" name="downclick" value="#f5f5dc"> Downclick`
 5. `</form>`

Output:

Input "color" types:

Pick your Favorite color:

Up-click

Down-click

Note: The default value of "color" type is #000000 (black). It only supports color value in hexadecimal format.

2. `<input type="date">`:

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

Example:

1. `<form>`
2. `Select Start and End Date:

`
3. `<input type="date" name="Startdate"> Start date:

`
4. `<input type="date" name="Enddate"> End date:

`
5. `<input type="submit">`
6. `</form>`

Output:

Input "date" type

Select Start and End Date:

Start date:

End date:

3. `<input type="datetime-local">`:

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

Example:

1. `<form>`
2. `<label>`

-
3. Select the meeting schedule:

 4. Select date & time: <input type="datetime-local" name="meetingdate">

 5. </label>
 6. <input type="submit">
 7. </form>

Output:

Input "datetime-local" type

Select the meeting schedule:

Select date & time:

4. <input type="email">:

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

Example:

1. <form>
2. <label>Enter your Email-address</label>
3. <input type="email" name="email" required>
4. <input type="submit">
5. <p>Note:User can also enter multiple email addresses separat
ing by comma or whitespace as following: </p>
6. <label>Enter multiple Email-addresses</label>
7. <input type="email" name="email" multiple>
8. <input type="submit">
9. </form>

Output:

Input "email" type

Enter your Email-address

Note:User can also enter multiple email addresses separating by comma or whitespace as following:

Enter multiple Email-addresses

5. <input type="month">:

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

Example:

1. <form>
2. <label>Enter your Birth Month-year: </label>
3. <input type="month" name="newMonth">
4. <input type="submit">
5. </form>

Output:

Input "month" type:

Enter your Birth Month-year:

HTML form Attribute

HTML <form> element attributes

In HTML there are various attributes available for <form> element which are given below:

HTML action attribute

The action attribute of <form> element defines the process to be performed on form when form is submitted, or it is a URI to process the form information.

The action attribute value defines the web page where information proceed. It can be .php, .jsp, .asp, etc. or any URL where you want to process your form.

Note: If action attribute value is blank then form will be processed to the same page.

Example:

1. <form action="action.html" method="post">
2. <label>User Name:</label>

3. <input type="text" name="name">

4. <label>User Password</label>

5. <input type="password" name="pass">

6. <input type="submit">
7. </form>

Output:

Demo of action attribute of form element

User Name:

User Password

It will redirect to a new page "action.html" when you click on submit button

HTML method attribute

The method attribute defines the HTTP method which browser used to submit the form. The possible values of method attribute can be:

- **post:** We can use the post value of method attribute when we want to process the sensitive data as it does not display the submitted data in URL.

Example:

1. `<form action="action.html" method="post">`

- **get:** The get value of method attribute is default value while submitting the form. But this is not secure as it displays data in URL after submitting the form.

Example:

1. `<form action="action.html" method="get">`

When submitting the data, it will display the entered data in the form of:

1. `file:///D:/HTML/action.html?name=JavaTPoint&pass=123`

HTML target attribute

The target attribute defines where to open the response after submitting the form. The following are the keywords used with the target attribute.

- **_self:** If we use _self as an attribute value, then the response will display in current page only.

Example:

1. `<form action="action.html" method="get" target="_self">`

- **_blank:** If we use _blank as an attribute it will load the response in a new page.

Example:

1. `<form action="action.html" method="get" target="_blank">`
-

HTML autocomplete attribute

The HTML autocomplete attribute is a newly added attribute of HTML5 which enables an input field to complete automatically. It can have two values "on" and "off" which enables autocomplete either ON or OFF. The default value of autocomplete attribute is "on".

Example:

1. `<form action="action.html" method="get" autocomplete="on">`

Example:

1. `<form action="action.html" method="get" autocomplete="off">`

Note: it can be used with `<form>` element and `<input>` element both.

HTML enctype attribute

The HTML enctype attribute defines the encoding type of form-content while submitting the form to the server. The possible values of enctype can be:

- **application/x-www-form-urlencoded:** It is default encoding type if the enctype attribute is not included in the form. All characters are encoded before submitting the form.

Example:

1. `<form action="action.html" method="post" enctype="application/x-www-form-urlencoded" >`
- **multipart/form-data:** It does not encode any character. It is used when our form contains file-upload controls.

Example:

1. `<form action="action.html" method="post" enctype="multipart/form-data">`
- **text/plain (HTML5):** In this encoding type only space are encoded into + symbol and no any other special character encoded.

Example:

1. `<form action="action.html" method="post" enctype="text/plain" >`

HTML novalidate attribute HTML5

The novalidate attribute is newly added Boolean attribute of HTML5. If we apply this attribute in form then it does not perform any type of validation and submit the form.

Example:

1. `<form action = "action.html" method = "get" novalidate>`

Output:

Fill the form

Enter name:

Enter age:

Enter email:

Try to change the form details with novalidate attribute and without novalidate attribute and see the difference.

HTML <input> element attribute

HTML name attribute

The HTML name attribute defines the name of an input element. The name and value attribute are included in HTTP request when we submit the form.

Note: One should not omit the name attribute as when we submit the form the HTTP request includes both name-value pair and if name is not available it will not process that input field.

Example:

1. `<form action = "action.html" method = "get">`
2. Enter name:
<input type="name" name="uname">

3. Enter age:
<input type="number" name="age">

4. Enter email:
<input type="email">

5. <input type="submit" value="Submit">
6. </form>

Output:

Fill the form

Enter name:

Enter age:

Enter email:

Note: If you will not use name attribute in any input field, then that input field will not be submitted, when submit the form.

Click on submit and see the URL where email is not included in HTTP request as we have not used name attribute in the email input field

HTML value attribute

The HTML value attribute defines the initial value or default value of an input field.

Example:

1. `<form>`
2. `<label>Enter your Name</label>
`
3. `<input type="text" name="uname" value="Enter Name">

`
4. `<label>Enter your Email-address</label>
`
5. `<input type="text" name="uname" value="Enter email">

`
6. `<label>Enter your password</label>
`
7. `<input type="password" name="pass" value="">

`
8. `<input type="submit" value="login">`
9. `</form>`

Output:

Fill the form

Enter your Name

Enter your Email-address

Enter your password

Note: In password input field the value attribute will always be unclear

HTML required attribute HTML5

HTML required is a Boolean attribute which specifies that user must fill that field before submitting the form.

Example:

1. <form>
2. <label>Enter your Email-address</label>

3. <input type="text" name="uname" required>

4. <label>Enter your password</label>

5. <input type="password" name="pass">

6. <input type="submit" value="login">
7. </form>

Output:

Fill the form

Enter your Email-address

Enter your password

If you will try to submit the form without completing email field then it will give an error pop up.

HTML autofocus attribute HTML5

The autofocus is a Boolean attribute which enables a field automatically focused when a webpage loads.

Example:

1. <form>
2. <label>Enter your Email-address</label>

3. <input type="text" name="uname" autofocus>

4. <label>Enter your password</label>

5. <input type="password" name="pass">

6. <input type="submit" value="login">
7. </form>

HTML placeholder attribute HTML5

The placeholder attribute specifies a text within an input field which informs the user about the expected input of that field.

The placeholder attribute can be used with text, password, email, and URL values.

When the user enters the value, the placeholder will be automatically removed.

Example:

1. `<form>`
2. `<label>Enter your name</label>
`
3. `<input type="text" name="uname" placeholder="Your name">

`
4. `<label>Enter your Email address</label>
`
5. `<input type="email" name="email" placeholder="example@gmail.com">

`
6. `<label>Enter your password</label>
`
7. `<input type="password" name="pass" placeholder="your password">

`
8. `<input type="submit" value="login">`
9. `</form>`

Output:

Registration form

Enter your name

Enter your Email address

Enter your password

HTML disabled attribute

The HTML disabled attribute when applied then it disables that input field. The disabled field does not allow the user to interact with that field.

The disabled input field does not receive click events, and these input values will not be sent to the server when submitting the form.

Example:

1. `<input type="text" name="uname" disabled>

`

Output:

Registration form

Enter User name

Enter your Email address

Enter your password

HTML size attribute

The size attribute controls the size of the input field in typed characters.

Example:

1. `<label>Account holder name</label>
`
2. `<input type="text" name="uname" size="40" required>

`
3. `<label>Account number</label>
`
4. `<input type="text" name="an" size="30" required>

`
5. `<label>CVV</label>
`
6. `<input type="text" name="cvv" size="1" required>

`

Output:

Registration form with disbaled attribute

Account holder name

Account number

CVV

HTML form attribute

HTML form attribute allows a user to specify an input filed outside the form but remains the part of the parent form.

Example:

1. User email: `
<input type="email" name="email" form="fcontrol" required>
`
2. `<input type="submit" form="fcontrol">`

Output:

User Name:

User password:

The email field is outside the form but still it will remain part of the form

User email:

HTML style using CSS

Let's suppose we have created our web page using a simple HTML code, and we want something which can present our page in a correct format, and visibly attractive. So to do this, we can style our web page with CSS (Cascading Stylesheet) properties.

CSS is used to apply the style in the web page which is made up of HTML elements. It describes the look of the webpage.

CSS provides various style properties such as background color, padding, margin, border-color, and many more, to style a webpage.

Each property in CSS has a name-value pair, and each property is separated by a semicolon (;).

Note: In this chapter, we have given a small overview of CSS. You will learn everything in depth about CSS in our CSS tutorial.

Example:

1. `<body style="text-align: center;">`
2. `<h2 style="color: red;">Welcome to javaTpoint</h2>`
3. `<p style="color: blue; font-size: 25px; font-style: italic ;">This is a great website to learn technologies in very simple way. </p>`
4. `</body>`

In the above example, we have used a style attribute to provide some styling format to our code.

Output:

Welcome to javaTpoint

This is a great website to learn technologies in very simple way.

Three ways to apply CSS

To use CSS with HTML document, there are three ways:

- **Inline CSS:** Define CSS properties using style attribute in the HTML elements.
 - **Internal or Embedded CSS:** Define CSS using <style> tag in <head> section.
 - **External CSS:** Define all CSS property in a separate .css file, and then include the file with HTML file using tag in section.
-

Inline CSS:

Inline CSS is used to apply CSS in a single element. It can apply style uniquely in each element.

To apply inline CSS, you need to use style attribute within HTML element. We can use as many properties as we want, but each property should be separated by a semicolon (;).

Example:

1. <h3 style="color: red;
2. font-style: italic;
3. text-align: center;
4. font-size: 50px;
5. padding-top: 25px;">Learning HTML using Inline CSS</h3>

Output:

Learning HTML using Inline CSS

Internal CSS:

An Internal stylesheet contains the CSS properties for a webpage in <head> section of HTML document. To use Internal CSS, we can use class and id attributes.

We can use internal CSS to apply a style for a single HTML page.

Example:

1. <!DOCTYPE html>

```
2. <html>
3. <head>
4.     <style>
5.     /*Internal CSS using element name*/
6.     body{ background-color:lavender;
7.         text-align: center;}
8.     h2{font-style: italic;
9.         font-size: 30px;
10.        color: #f08080;}
11.    p{ font-size: 20px;}
12.    /*Internal CSS using class name*/
13.    .blue{color: blue;}
14.    .red{color: red;}
15.    .green{color: green;}
16.    </style>
17. </head>
18. <body>
19. <h2>Learning HTML with internal CSS</h2>
20. <p class="blue">This is a blue color paragraph</p>
21. <p class="red">This is a red color paragraph</p>
22. <p class="green">This is a green color paragraph</p>
23. </body>
24. </html>
```

Note: In the above example, we have used a class attribute which you will learn in the next chapter.

External CSS:

An external CSS contains a separate CSS file which only contains style code using the class name, id name, tag name, etc. We can use this CSS file in any HTML file by including it in HTML file using <link> tag.

If we have multiple HTML pages for an application and which use similar CSS, then we can use external CSS.

There are two files need to create to apply external CSS

- First, create the HTML file
- Create a CSS file and save it using the .css extension (This file only will only contain the styling code.)
- Link the CSS file in your HTML file using tag in header section of HTML document.

Example:

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <link rel="stylesheet" type="text/css" href="style.css">
5. </head>
6. <body>
```

7. <h2>Learning HTML with External CSS</h2>
8. <p class="blue">This is a blue color paragraph</p>
9. <p class="red">This is a red color paragraph</p>
10. <p class="green">This is a green color paragraph</p>
11. </body>
12. </html>

CSS file:

```
body{
background-color:lavender;
text-align: center;
}
h2{
font-style: italic;
size: 30px;
color: #f08080;
}
p{
font-size: 20px;
}
```

```
.blue{
color: blue;
}
.red{
color: red;
}
.green{
color: green;
}
```

Commonly used CSS properties:

Properties-name	Syntax	Description
background-color	background-color:red;	It defines the background color of that element.
color	color: lightgreen;	It defines the color of text of an element
padding	padding: 20px;	It defines the space between content and the border.
margin	margin: 30px; margin-left:	It creates space around an element.
font-family	font-family: cursive;	Font-family defines a font for a particular element.
Font-size	font-size: 50px;	Font-size defines a font size for a particular element.
text-align	text-align: left;	It is used to align the text in a selected position.

HTML Classes

Class Attribute in HTML

The HTML class attribute is used to specify a single or multiple class names for an HTML element. The class name can be used by CSS and JavaScript to do some tasks for HTML elements. You can use this class in CSS with a specific class, write a period (.) character, followed by the name of the class for selecting elements.

A class attribute can be defined within <style> tag or in separate file using the (.) character.

In an HTML document, we can use the same class attribute name with different elements.

Defining an HTML class

To create an HTML class, firstly define style for HTML class using <style> tag within <head> section as following example:

Example:

```
1. <head>
2.   <style>
3.     .headings{
4.       color: lightgreen;
5.       font-family: cursive;
6.       background-color: black; }
7.   </style>
8. </head>
```

We have define style for a class name "headings", and we can use this class name with any of HTML element in which we want to provide such styling. We just need to follow the following syntax to use it.

```
1. <tag class="ghf"> content </tag>
```

Example 1:

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.   <style>
5.     .headings{
6.       color: lightgreen;
7.       font-family: cursive;
8.       background-color: black; }
9.   </style>
10. </head>
11. <body>
12. <h1 class="headings">This is first heading</h1>
```

-
13. <h2 class="headings">This is Second heading</h2>
 14. <h3 class="headings">This is third heading</h3>
 15. <h4 class="headings">This is fourth heading</h4>
 16. </body>
 17. </html>
-

Another Example with different class name

Example:

Let's use a class name "Fruit" with CSS to style all elements.

1. <style>
2. .fruit {
3. background-color: orange;
4. color: white;
5. padding: 10px;
6. }
7. </style>
- 8.
9. <h2 class="fruit">Mango</h2>
10. <p>Mango is king of all fruits.</p>
- 11.
12. <h2 class="fruit">Orange</h2>
13. <p>Oranges are full of Vitamin C.</p>
- 14.
15. <h2 class="fruit">Apple</h2>
16. <p>An apple a day, keeps the Doctor away.</p>

Here you can see that we have used the class name "fruit" with (.) to use all its elements.

Note: You can use class attribute on any HTML element. The class name is case-sensitive.

Class Attribute in JavaScript

You can use JavaScript access elements with a specified class name by using the `getElementsByClassName()` method.

Example:

Let's hide all the elements with class name "fruit" when the user click on the button.

1. <!DOCTYPE html>
2. <html>
3. <body>
- 4.
5. <h2>Class Attribute with JavaScript</h2>

```
6. <p>Click the button, to hide all elements with the class name "fruit", with JavaScript:</p>
7.
8. <button onclick="myFunction()">Hide elements</button>
9.
10.
11. <h2 class="fruit">Mango</h2>
12. <p>Mango is king of all fruits.</p>
13.
14. <h2 class="fruit">Orange</h2>
15. <p>Oranges are full of Vitamin C.</p>
16.
17. <h2 class="fruit">Apple</h2>
18. <p>An apple a day, keeps the Doctor away.</p>
19.
20. <script>
21. function myFunction() {
22.   var x = document.getElementsByClassName("fruit");
23.   for (var i = 0; i < x.length; i++) {
24.     x[i].style.display = "none";
25.   }
26. }
27. </script>
28.
29. </body>
30. </html>
```

Note: You will learn more about JavaScript in our JavaScript tutorial.

Multiple Classes

You can use multiple class names (more than one) with HTML elements. These class names must be separated by a space.

Example:

Let's style elements with class name "fruit" and also with a class name "center".

```
1. <!DOCTYPE html>
2. <html>
3. <style>
4.   .fruit {
5.     background-color: orange;
6.     color: white;
7.     padding: 10px;
8.   }
9.
10.  .center {
11.    text-align: center;
12.  }
13. </style>
```

-
14. <body>
 - 15.
 16. <h2>Multiple Classes</h2>
 17. <p>All three elements have the class name "fruit". In addition, Mango also have the class name "center", which center-aligns the text.</p>
 - 18.
 19. <h2 class="fruit center">Mango</h2>
 20. <h2 class="fruit">Orange</h2>
 21. <h2 class="fruit">Apple</h2>
 - 22.
 23. </body>
 24. </html>

You can see that the first element <h2> belongs to both the "fruit" class and the "center" class.

Same class with Different Tag

You can use the same class name with different tags like <h2> and <p> etc. to share the same style.

Example:

1. <!DOCTYPE html>
2. <html>
3. <style>
4. .fruit {
5. background-color: orange;
6. color: white;
7. padding: 10px;
8. }
9. </style>
10. <body>
11. <h2>Same Class with Different Tag</h2>
12. <h2 class="fruit">Mango</h2>
13. <p class="fruit">Mango is the king of all fruits.</p>
14. </body>
15. </html>

Test it NowHTML Id Attribute

The **id attribute** is used to specify the unique ID for an element of the HTML document. It allocates the unique identifier which is used by the **CSS** and the **JavaScript** for performing certain tasks.

Note: In the Cascading Style sheet (CSS), we can easily select an element with the specific id by using the # symbol followed by id.

Note: JavaScript can access an element with the given ID by using the getElementById() method.

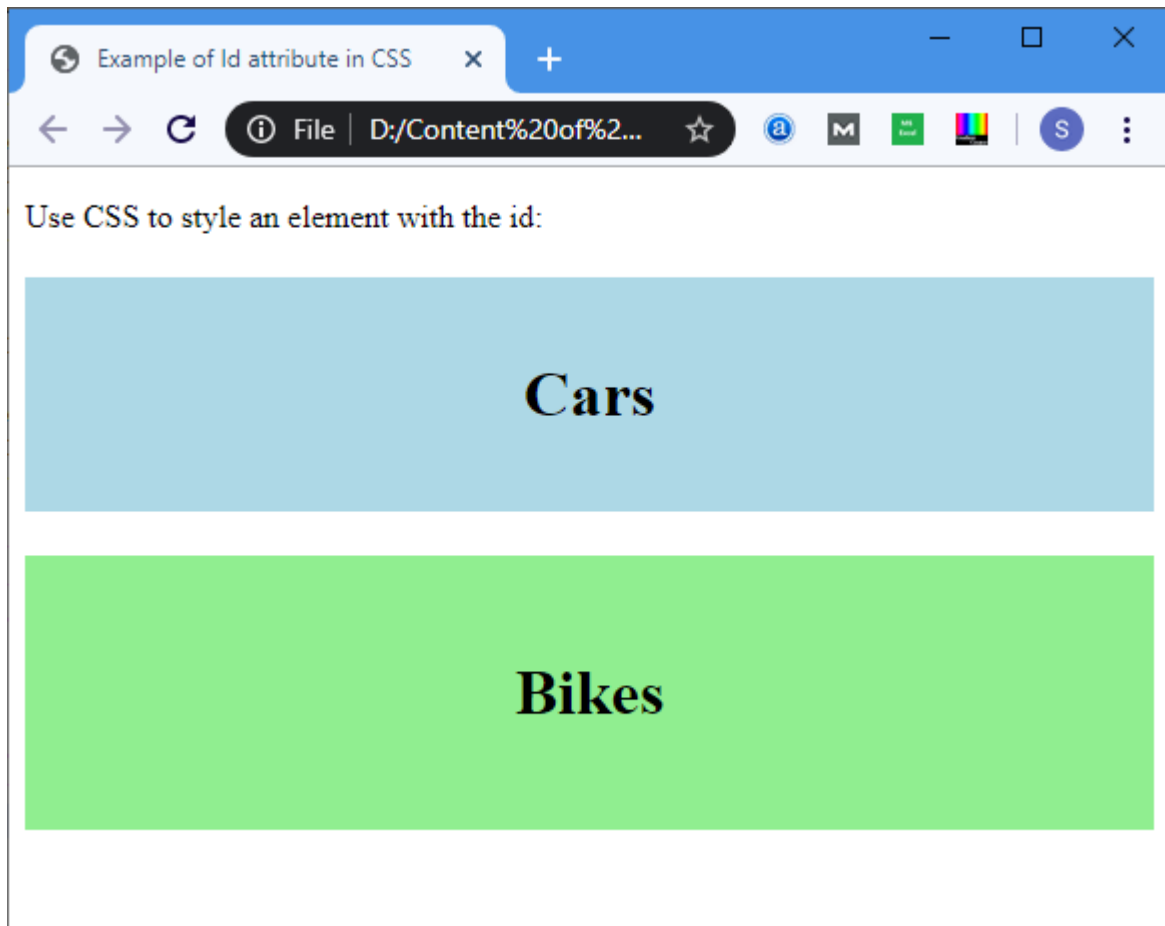
Syntax

1. <tag id="value">

Example 1: The following example describes how to use the id attribute in CSS document:

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>
5. Example of Id attribute in CSS
6. </title>
7. <style>
8. #Cars {
9. padding: 40px;
10. background-color: lightblue;
11. color: black;
12. text-align: center;
13. }
- 14.
15. #Bikes
16. {
17. padding: 50px;
18. background-color: lightGreen;
19. text-align: center;
20. }
21. </style>
22. </head>
23. <body>
24. <p> Use CSS to style an element with the id: </p>
25. <h1 id="Cars"> Cars </h1>
26. <h1 id="Bikes"> Bikes </h1>
27. </body>
28. </html>

Output:



Example 2: The following example describes how to use the ID attribute in JavaScript.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<title> Date Attribute </title>`
5. `<script>`
6. `function viewdate() {`
7. `var x = document.getElementById("dob").value;`
8. `document.getElementById("demo").innerHTML = x;`
9. `</script>`
10. `</head>`
11. `<body>`
12. Employee Name: `<input type="text" placeholder="Your Good name"/>`
13. `
`
14. `
`
15. Date of Joining:
16. `<input type="date" id="dob">`
17. `
`
18. `<button onclick="viewdate()"> Submit`
19. `</button>`
20. `
`
21. `<h2 id="demo"> </h2>`
22. `</body>`
23. `</html>`

Output:

A screenshot of a web browser window. The address bar shows 'Date Attribute' and a file path 'D:/Content%20of%20S...'. The form contains two input fields: 'Employee Name' with the value 'Akki' and 'Date of Joining' with the value '10-12-2016'. Below these fields is a 'Submit' button. The output of the form is displayed as '2016-12-10' in a large, bold, black font.

HTML

List Box

The **list box** is a graphical control element in the HTML document that allows a user to select one or more options from the list of options.

Syntax

To create a list box, use the [HTML element](#) `<select>` which contains two attributes **Name** and **Size**. The **Name** attribute is used to define the name for calling the list box, and **size** attribute is used to specify the numerical value that shows the how many options it contains.

1. `<select Name="Name_of_list_box" Size="Number_of_options">`
2. `<option> List item 1 </option>`
3. `<option> List item 2 </option>`
4. `<option> List item 3 </option>`
5. `<option> List item N </option>`
6. `</select>`

Examples:

Example 1: Consider the below example that creates a simple list box.

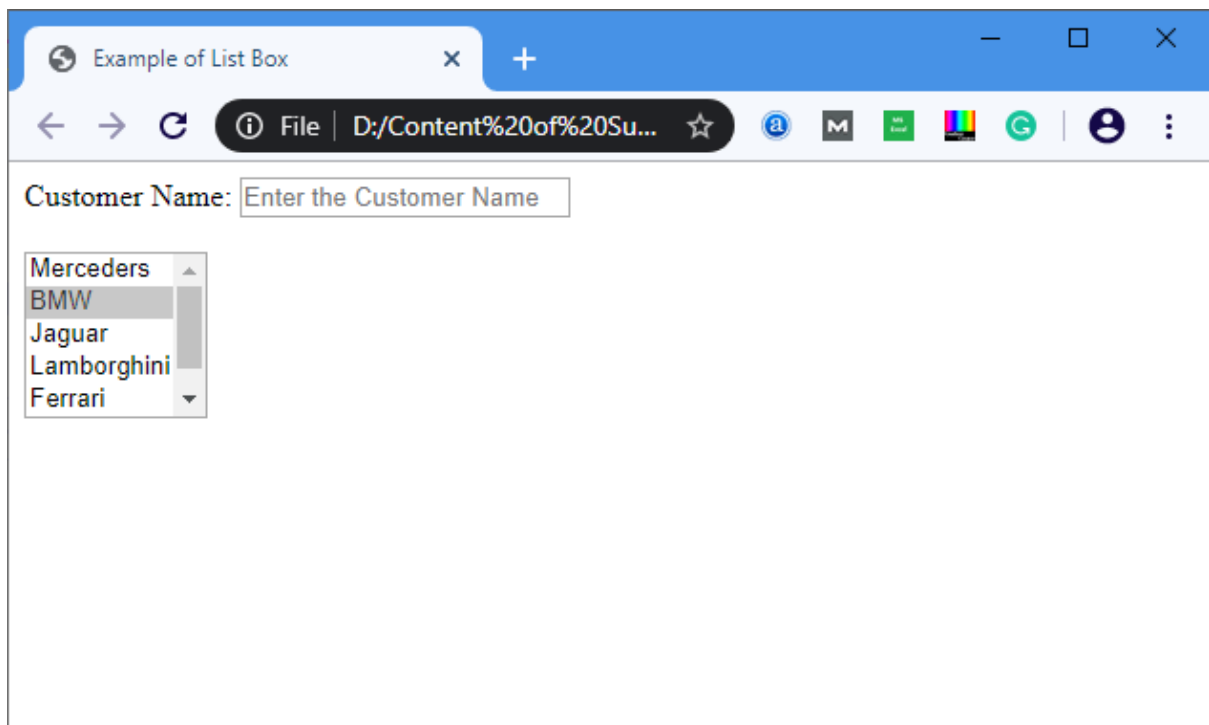
1. `<!DOCTYPE html>`
2. `<html>`

3. <title>
4. Example of List Box
5. </title>
6. <body>
7. Customer Name: <input type="text" Placeholder="Enter the Customer Name"/>
8.

9.

10. <select name="Cars" size="5">
11. <option value="Merceders"> Merceders </option>
12. <option value="BMW"> BMW </option>
13. <option value="Jaguar"> Jaguar </option>
14. <option value="Lamborghini"> Lamborghini </option>
15. <option value="Ferrari"> Ferrari </option>
16. <option value="Ford"> Ford </option>
17. </select>
18. </body>
19. </html>

Output:

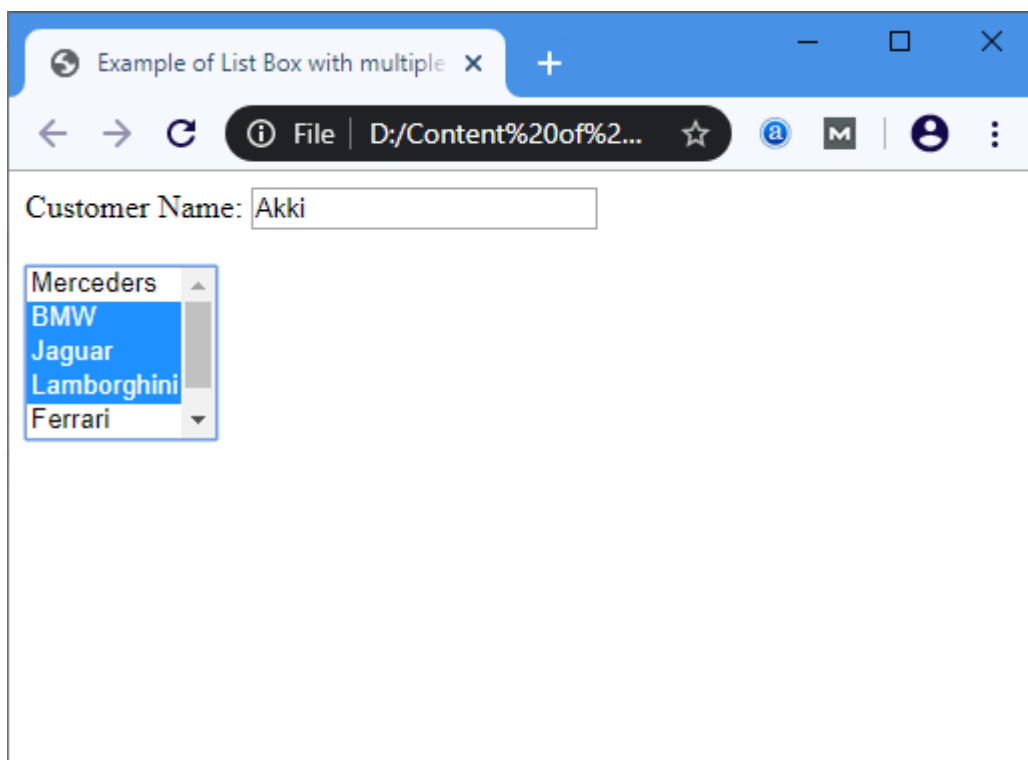


Example 2: Below example uses the **multiple** attribute for selecting the multiple options in a list. We can select multiple options from list box by holding the ctrl key.

1. <!DOCTYPE html>
2. <html>
3. <title>
4. Example of List Box with multiple attribute
5. </title>
6. <body>
7. Customer Name: <input type="text" Placeholder="Enter the Customer Name"/>

```
8. <br>
9. <br>
10. <select name="Cars" size="5" multiple="multiple">
11.   <option value="Merceders"> Merceders </option>
12.   <option value="BMW"> BMW </option>
13.   <option value="Jaguar"> Jaguar </option>
14.   <option value="Lamborghini"> Lamborghini </option>
15.   <option value="Ferrari"> Ferrari </option>
16.   <option value="Ford"> Ford </option>
17. </select>
18. </body>
19. </html>
```

Output:



Unix commands list

This guide has been prepared by me to help myself with the list of frequently used **basic commands in UNIX/LINUX** to be on my finger tip. Thought of sharing it with the others, in case, it might turn out helpful to other readers as well. This is *Unix/Linux basic commands - I*, for 2nd part follow the link given at the end of this article.

Web servers Shell

Unix/Linux file commands guide

This article will serve as a 5 minute guide or tutorial to learn/revisit basic unix or linux commands frequently used while working with files. Unix/Linux command is given along with their *usage or description*.

- **ls** ► use this *command in unix/linux* to see all the directory listing. However, any hidden files will not be listed.
- **ls -al** ► use this *command in unix/linux* to see formatted directory listing along with the hidden files.
- **ls -lt** ► use this *command in unix/linux* to sort the directory listing by their time of modification.
- **pwd** ► use this *command in unix/linux* to show your current working directory.
- **touch fileName** ► use this *command in unix/linux* to create new file with its name as filename.
- **cd** ► use this *command in unix/linux* to move to home directory.
- **cd dirName** ► use this *command in unix/linux* to change current directory to dirName directory.
- **mkdir dirName** ► use this *command in unix/linux* to make or create directory having name as dirName.
- **rm fileName** ► use this *command in unix/linux* to remove or delete file having name as fileName.
- **rm -r dirName** ► use this *command in unix/linux* to remove or delete directory dirName.
- **rm -f filename** ► use this *command in unix/linux* to force remove the file filename.
- **more fileName** ► use this *command in unix/linux* to get the content of file having name as filename
- **head fileName** ► use this *command in unix/linux* to get output of first 10 lines of the file fileName.
- **tail fileName** ► use this *command in unix/linux* to get output of last 10 lines of the file filename.
- **cp fileAfileB** ► use this *command in unix/linux* to copy the content of fileA to fileB.
- **cp -r dirA dirB** ► use this *command in unix/linux* to copy directory dirA to directory dirB and create dirB if not already created.
- **mv fileAfileB** ► use this *command in unix/linux* to rename or move fileA to fileB.
- **cat >file** ► use this *command in unix/linux* to place standard input into the file.

Unix or Linux process management commands guide

This section will serve as a 5 minute guide or tutorial to learn/revisit basic unix or linux commands frequently used while working with process management. Unix/Linux command is given along with their *usage or description*.

- **ps** ► use this command in unix/linux to see currently working processes.
- **top** ► use this command in unix/linux to display all the running processes.

-
- **kill pid** ► use this command in unix/linux to kill the process with given pid.
 - **killallprocessA** ► use this command in unix/linux to kill all the process named as processA
 - **pkill pattern** ► use this command in unix/linux to kill all processes matching the given pattern.
 - **bg** ► use this command in unix/linux to list all the background jobs.
 - **fg** ► use this command in unix/linux to bring the most recent job to foreground.
 - **fg n1** ► use this command in unix/linux to bring job n1 to the foreground.
-

Unix/Linux system info commands guide

This section will serve as a 5 minute guide or tutorial to learn/revisit basic unix or linux commands frequently used while working with system. Unix/Linux command is given along with their *usage or description*.

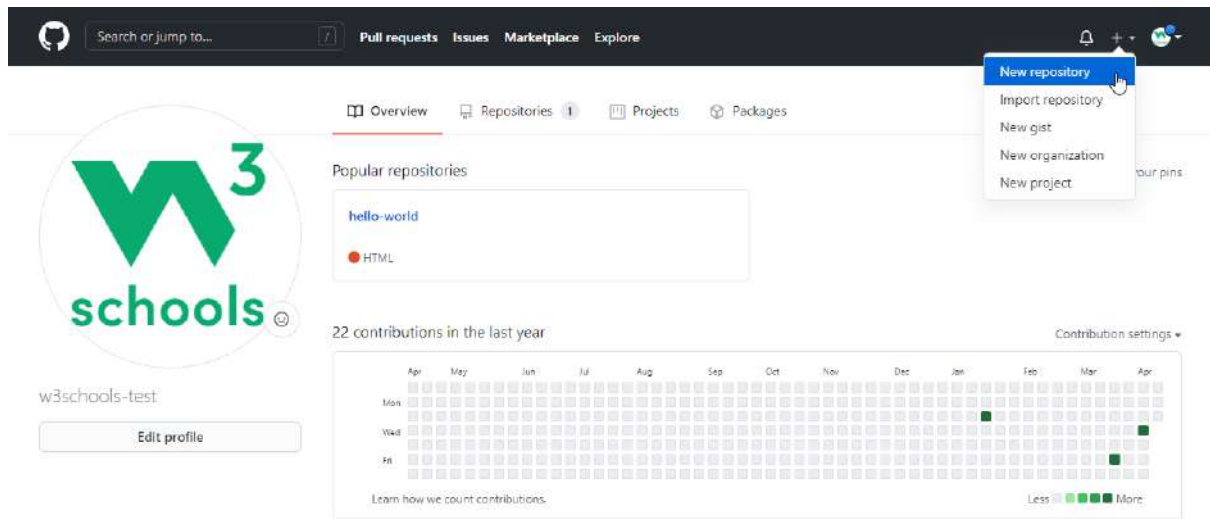
- **cal** ► use this command in unix/linux to show current months calendar.
 - **date** ► use this command in unix/linux to show current date and time.
 - **w** ► use this command in unix/linux to see who all are currently logged in to the system.
 - **whoami** ► use this command in unix/linux to see who you are currently logged in as in the system.
 - **uname -a** ► use this command in unix/linux to see kernel information.
 - **finger user** ► use this command in unix/linux to display information about user.
 - **man command** ► use this command in unix/linux to show the manual for command.
 - **free** ► use this command in unix/linux to show memory and swap usage.
 - **df** ► use this command in unix/linux to see the disk usage.
 - **du** ► use this command in unix/linux to see the directory space usage.
 - **whereis app** ► use this command in unix/linux to show possible location of app.
 - **which app** ► use this command in unix/linux to show which application will be run by default.
-

Version control

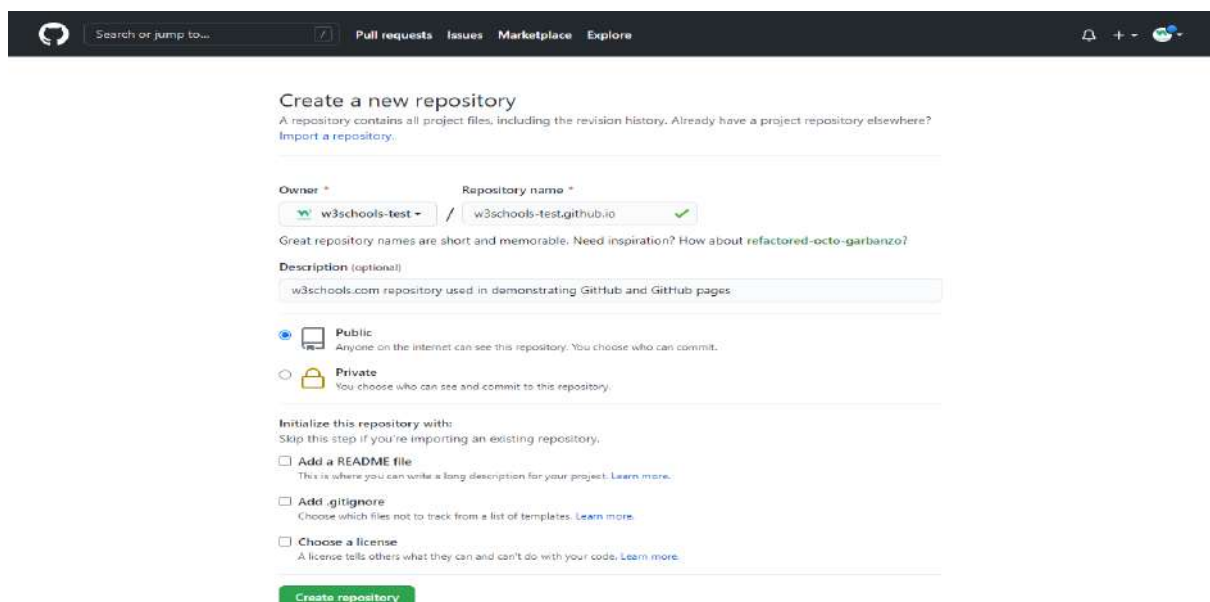
Git GitHub Pages

Create a New Repository

Start by signing in to GitHub. GitHub pages need a special name and setup to work, so we start by creating a new repository:



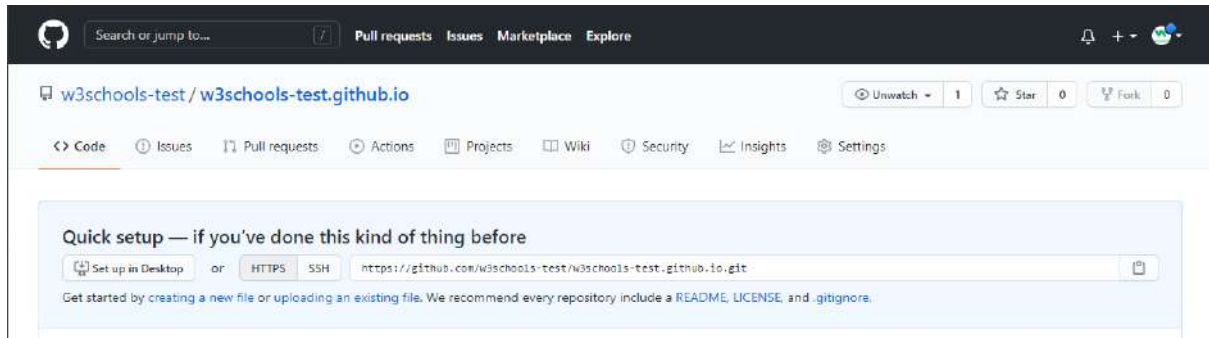
This repository needs a special name to function as a GitHub page. It needs to be your GitHub username, followed by `.github.io`:



Push Local Repository to GitHub Pages

We add this new repository as a remote for our local repository, we are calling it gh-page (for GitHub Pages).

Copy the URL from here:



And add it as a new remote:

Example

git remote add gh-page https://github.com/w3schools-test/w3schools-test.github.io.git

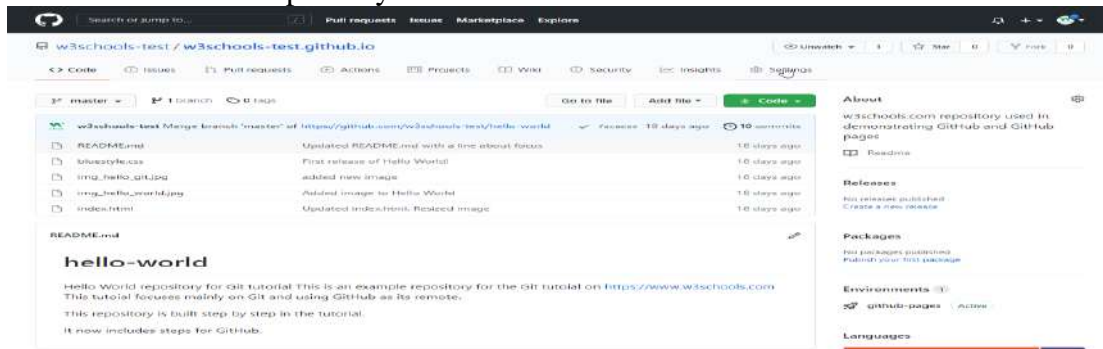
Make sure you are on the masterbranch, then push the masterbranch to the new remote:

Example

```
git push gh-page master
Enumerating objects: 33, done.
Counting objects: 100% (33/33), done.
Delta compression using up to 16 threads
Compressing objects: 100% (33/33), done.
Writing objects: 100% (33/33), 94.79 KiB | 15.80 MiB/s, done.
Total 33 (delta 18), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (18/18), done.
To https://github.com/w3schools-test/w3schools-test.github.io.git
 * [new branch]    master -> master
```

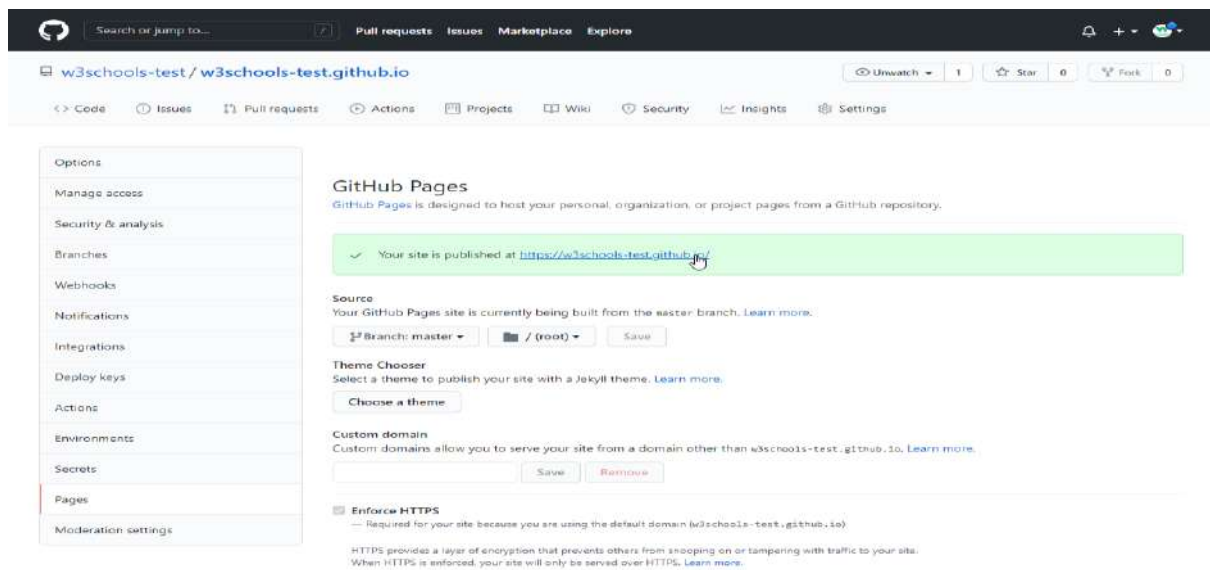
Note: If this is the first time you are connecting to GitHub, you will get some kind of notification to authenticate this connection.

Check that the new repository has received all the files:



Check Out Your Own GitHub Page

That looks good, now click the Settings menu and navigate to the Pages tab:



The GitHub page is created, and you can click the URL to view the result!

Git Tutorial

Learning by Examples

In this tutorial, we will show you Git commands like this:

Example

```
git --version  
git version 2.30.2.windows.1
```

For new users, using the terminal view can seem a bit complicated. Don't worry! We will keep it really simple, and learning this way gives you a good grasp of how Git works.

In the code above, you can see commands (input) and output.

Lines like this are commands we input:

Example

```
git --version
```

Lines like this are the output/response to our commands:

Example

```
git version 2.30.2.windows.1
```

In general, lines with \$ in front of it is input. These are the commands you can copy and run in your terminal.

Git and Remote Repositories

Git and GitHub are different things.

In this tutorial you will understand what Git is and how to use it on the remote repository platforms, like GitHub.

You can choose, and change, which platform to focus on by clicking in the menu on the right:

Git Exercises

Test Yourself With Exercises

Exercise:

Insert the missing part of the command to check which version of Git (if any) is installed.

git

What is Git?

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

It is used for:

- Tracking code changes
- Tracking who made changes
- Coding collaboration

What does Git do?

- Manage projects with **Repositories**
- **Clone** a project to work on a local copy
- Control and track changes with **Staging** and **Committing**
- **Branch** and **Merge** to allow for work on different parts and versions of a project
- **Pull** the latest version of the project to a local copy
- **Push** local updates to the main project

Working with Git

- Initialize Git on a folder, making it a **Repository**
- Git now creates a hidden folder to keep track of changes in that folder

-
- When a file is changed, added or deleted, it is considered **modified**
 - You select the modified files you want to **Stage**
 - The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
 - Git allows you to see the full history of every commit.
 - You can revert back to any previous commit.
 - Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

Why Git?

- Over 70% of developers use Git!
- Developers can work together from anywhere in the world.
- Developers can see the full history of the project.
- Developers can revert to earlier versions of a project.

What is GitHub?

- Git is not the same as GitHub.
- GitHub makes tools that use Git.
- GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.
- In this tutorial, we will focus on using Git with GitHub.

Git Install

You can download Git for free from the following website: <https://www.git-scm.com/>

Using Git with Command Line

To start using Git, we are first going to open up our Command shell.

For Windows, you can use Git bash, which comes included in Git for Windows. For Mac and Linux you can use the built-in terminal.

The first thing we need to do, is to check if Git is properly installed:

Example

```
git --version  
git version 2.30.2.windows.1
```

If Git is installed, it should show something like `git version X.Y`

Configure Git

Now let Git know who you are. This is important for version control systems, as each Git commit uses this information:

Example

```
git config --global user.name "w3schools-test"  
git config --global user.email "test@w3schools.com"
```

Change the user name and e-mail address to your own. You will probably also want to use this when registering to GitHub later on.

Note: Use global to set the username and e-mail for **every repository** on your computer.

If you want to set the username/e-mail for just the current repo, you can remove global

Creating Git Folder

Now, let's create a new folder for our project:

Example

```
mkdirmyproject  
cdmyproject
```

mkdir**makes a new directory.**

cd**changes the current working directory.**

Now that we are in the correct directory. We can start by initializing Git!

Note: If you already have a folder/directory you would like to use for Git:

Navigate to it in command line, or open it in your file explorer, right-click and select "Git Bash here"

Initialize Git

Once you have navigated to the correct folder, you can initialize Git on that folder:

Example

```
gitinit  
Initialized empty Git repository in /Users/user/myproject/.git/
```

You just created your first Git Repository!

Note: Git now knows that it should watch the folder you initiated it on.

Git creates a hidden folder to keep track of changes.

Git Adding New Files

You just created your first local Git repo. But it is empty.

So let's add some files, or create a new file using your favourite text editor. Then save or move it to the folder you just created.

If you want to learn how to create a new file using a text editor, you can visit our HTML tutorial:

[HTML Editors](#)

For this example, I am going to use a simple HTML file like this:

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
</head>
<body>

<h1>Hello world!</h1>
<p>This is the first file in my new Git Repo.</p>

</body>
</html>
```

And save it to our new folder as index.html.

Let's go back to the terminal and list the files in our current working directory:

Example

```
ls
index.html
```

ls will **list** the files in the directory. We can see that index.html is there.

Then we check the Git status and see if it is a part of our repo:

Example

```
git status
On branch master
```

No commits yet

Untracked files:

```
(use "git add ..." to include in what will be committed)
index.html
```

nothing added to commit but untracked files present (use "git add" to track)

Now Git is **aware** of the file, but has not **added** it to our repository!

Files in your Git repository folder can be in one of 2 states:

- Tracked - files that Git knows about and are added to the repository
- Untracked - files that are in your working directory, but not added to the repository

When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

Git Staging Environment

One of the core functions of Git is the concepts of the Staging Environment, and the Commit.

As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

Staged files are files that are ready to be **committed** to the repository you are working on. You will learn more about commit shortly.

For now, we are done working with index.html. So we can add it to the Staging Environment:

Example

```
gitadd index.html
```

The file should be **Staged**. Let's check the status::

Example

```
git status
On branch master
```

```
No commits yet
```

```
Changes to be committed:
  (use "git rm --cached ..." to unstage)
    new file:   index.html
```

Now the file has been added to the Staging Environment.

Git Add More than One File

You can also stage more than one file at a time. Let's add 2 more files to our working folder. Use the text editor again.

A README.md file that describes the repository (recommended for all repositories):

Example

hello-world

Hello World repository for Git tutorial

This is an example repository for the Git tutorial on <https://www.w3schools.com>

This repository is built step by step in the tutorial.

A basic external style sheet (bluestyle.css):

Example

```
body {  
background-color: lightblue;  
}
```

```
h1 {  
color: navy;  
margin-left: 20px;  
}
```

And update index.html to include the stylesheet:

Example

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<linkrel="stylesheet"href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<p>This is the first file in my new Git Repo.</p>  
  
</body>  
</html>
```

Now add all files in the current directory to the Staging Environment:

Example

```
gitadd --all
```

Using --all instead of individual filenames will stage all changes (new, modified, and deleted) files.

Example

```
git status  
On branch master
```

No commits yet

Changes to be committed:

```
(use "git rm --cached ..." to unstage)
new file:   README.md
new file:   bluestyle.css
new file:   index.html
```

Now all 3 files are added to the Staging Environment, and we are ready to do our first commit.

Git Commit

Since we have finished our work, we are ready move from stage to commit for our repo.

Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should **always** include a **message**.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

Example

```
git commit -m "First release of Hello World!"
[master (root-commit) 221ec6e] First release of Hello World!
3 files changed, 26 insertions(+)
create mode 100644 README.md
create mode 100644 bluestyle.css
create mode 100644 index.html
```

The commit command performs a commit, and the -m "*message*" adds a message.

The Staging Environment has been committed to our repo, with the message:
"First release of Hello World!"

Git Commit without Stage

Sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The -a option will automatically stage every changed, already tracked file.

Let's add a small update to index.html:

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
<linkrel="stylesheet"href="bluestyle.css">
</head>
<body>
```

```
<h1>Hello world!</h1>
<p>This is the first file in my new Git Repo.</p>
<p>A new line in our file!</p>

</body>
</html>
```

And check the status of our repository. But this time, we will use the --short option to see the changes in a more compact way:

Example

```
git status --short
M index.html
```

Git Help

If you are having trouble remembering commands or options for commands, you can use Git help.

There are a couple of different ways you can use the help command in command line:

- `git command -help`- See all the available options for the specific command
- `git help --all`- See all possible commands

Let's go over the different commands.

Git -help See Options for a Specific Command

Any time you need some help remembering the specific option for a command, you can use `git command -help`:

Example

```
git commit -help
usage: git commit [] [--] ...
```

```
-q, --quiet      suppress summary after successful commit
-v, --verbose    show diff in commit message template
```

Commit message options

```
-F, --file      read message from file
--author       override author for commit
--date         override date for commit
-m, --message   commit message
-c, --reedit-message
                reuse and edit message from specified commit
-C, --reuse-message
                reuse message from specified commit
--fixup        use autosquash formatted message to fixup specified commit
```

--squash use autosquash formatted message to squash specified commit
--reset-author the commit is authored by me now (used with -C/-c/--amend)
-s, --signoff add a Signed-off-by trailer
-t, --template
 use specified template file
-e, --edit force edit of commit
--cleanup how to strip spaces and #comments from message
--status include status in commit message template
-S, --pgp-sign[=]
 GPG sign commit

Commit contents options

-a, --all commit all changed files
-i, --include add specified files to index for commit
--interactive interactively add files
-p, --patch interactively add changes
-o, --only commit only specified files
-n, --no-verify bypass pre-commit and commit-msg hooks
--dry-run show what would be committed
--short show status concisely
--branch show branch information
--ahead-behind compute full ahead/behind values
--porcelain machine-readable output
--long show status in long format (default)
-z, --null terminate entries with NUL
--amend amend previous commit
--no-post-rewrite bypass post-rewrite hook
-u, --untracked-files[=]
 show untracked files, optional modes: all, normal, no. (Default: all)
--pathspec-from-file
 read pathspec from file
--pathspec-file-nul with --pathspec-from-file, pathspec elements are separated with NUL character

Note: You can also use --help instead of -help to open the relevant Git manual page

Git help --all See All Possible Commands

To list all possible commands, use the help --all command:

Warning: This will display a very long list of commands

Example

```
$ githelp --all
```

See 'git help ' to read about a specific subcommand

Main Porcelain Commands

add Add file contents to the index
am Apply a series of patches from a mailbox
archive Create an archive of files from a named tree
bisect Use binary search to find the commit that introduced a bug
branch List, create, or delete branches
bundle Move objects and refs by archive
checkout Switch branches or restore working tree files

cherry-pick	Apply the changes introduced by some existing commits
citool	Graphical alternative to git-commit
clean	Remove untracked files from the working tree
clone	Clone a repository into a new directory
commit	Record changes to the repository
describe	Give an object a human readable name based on an available ref
diff	Show changes between commits, commit and working tree, etc
fetch	Download objects and refs from another repository
format-patch	Prepare patches for e-mail submission
gc	Cleanup unnecessary files and optimize the local repository
gitk	The Git repository browser
grep	Print lines matching a pattern
gui	A portable graphical interface to Git
init	Create an empty Git repository or reinitialize an existing one
log	Show commit logs
maintenance	Run tasks to optimize Git repository data
merge	Join two or more development histories together
mv	Move or rename a file, a directory, or a symlink
notes	Add or inspect object notes
pull	Fetch from and integrate with another repository or a local branch
push	Update remote refs along with associated objects
range-diff	Compare two commit ranges (e.g. two versions of a branch)
rebase	Reapply commits on top of another base tip
reset	Reset current HEAD to the specified state
restore	Restore working tree files
revert	Revert some existing commits
rm	Remove files from the working tree and from the index
shortlog	Summarize 'git log' output
show	Show various types of objects
sparse-checkout	Initialize and modify the sparse-checkout
stash	Stash the changes in a dirty working directory away
status	Show the working tree status
submodule	Initialize, update or inspect submodules
switch	Switch branches
tag	Create, list, delete or verify a tag object signed with GPG
worktree	Manage multiple working trees

Ancillary Commands / Manipulators

config	Get and set repository or global options
fast-export	Git data exporter
fast-import	Backend for fast Git data importers
filter-branch	Rewrite branches
mergetool	Run merge conflict resolution tools to resolve merge conflicts
pack-refs	Pack heads and tags for efficient repository access
prune	Prune all unreachable objects from the object database
reflog	Manage reflog information
remote	Manage set of tracked repositories
repack	Pack unpacked objects in a repository
replace	Create, list, delete refs to replace objects

Ancillary Commands / Interrogators

annotate	Annotate file lines with commit information
blame	Show what revision and author last modified each line of a file
bugreport	Collect information for user to file a bug report
count-objects	Count unpacked number of objects and their disk consumption
difftool	Show changes using common diff tools
fsck	Verifies the connectivity and validity of the objects in the database
gitweb	Git web interface (web frontend to Git repositories)
help	Display help information about Git
instaweb	Instantly browse your working repository in gitweb

merge-tree	Show three-way merge without touching index
rerere	Reuse recorded resolution of conflicted merges
show-branch	Show branches and their commits
verify-commit	Check the GPG signature of commits
verify-tag	Check the GPG signature of tags
whatchanged	Show logs with difference each commit introduces

Interacting with Others

archimport	Import a GNU Arch repository into Git
cvsexportcommit	Export a single commit to a CVS checkout
cvsimport	Salvage your data out of another SCM people love to hate
cvsserver	A CVS server emulator for Git
imap-send	Send a collection of patches from stdin to an IMAP folder
p4	Import from and submit to Perforce repositories
quiltimport	Applies a quilt patchset onto the current branch
request-pull	Generates a summary of pending changes
send-email	Send a collection of patches as emails
svn	Bidirectional operation between a Subversion repository and Git

Low-level Commands / Manipulators

apply	Apply a patch to files and/or to the index
checkout-index	Copy files from the index to the working tree
commit-graph	Write and verify Git commit-graph files
commit-tree	Create a new commit object
hash-object	Compute object ID and optionally creates a blob from a file
index-pack	Build pack index file for an existing packed archive
merge-file	Run a three-way file merge
merge-index	Run a merge for files needing merging
mktag	Creates a tag object
mktree	Build a tree-object from ls-tree formatted text
multi-pack-index	Write and verify multi-pack-indexes
pack-objects	Create a packed archive of objects
prune-packed	Remove extra objects that are already in pack files
read-tree	Reads tree information into the index
symbolic-ref	Read, modify and delete symbolic refs
unpack-objects	Unpack objects from a packed archive
update-index	Register file contents in the working tree to the index
update-ref	Update the object name stored in a ref safely
write-tree	Create a tree object from the current index

Low-level Commands / Interrogators

cat-file	Provide content or type and size information for repository objects
cherry	Find commits yet to be applied to upstream
diff-files	Compares files in the working tree and the index
diff-index	Compare a tree to the working tree or index
diff-tree	Compares the content and mode of blobs found via two tree objects
for-each-ref	Output information on each ref
for-each-repo	Run a Git command on a list of repositories
get-tar-commit-id	Extract commit ID from an archive created using git-archive
ls-files	Show information about files in the index and the working tree
ls-remote	List references in a remote repository
ls-tree	List the contents of a tree object
merge-base	Find as good common ancestors as possible for a merge
name-rev	Find symbolic names for given revs
pack-redundant	Find redundant pack files
rev-list	Lists commit objects in reverse chronological order
rev-parse	Pick out and massage parameters
show-index	Show packed archive index
show-ref	List references in a local repository
unpack-file	Creates a temporary file with a blob's contents

var	Show a Git logical variable
verify-pack	Validate packed Git archive files

Low-level Commands / Syncing Repositories

daemon	A really simple server for Git repositories
fetch-pack	Receive missing objects from another repository
http-backend	Server side implementation of Git over HTTP
send-pack	Push objects over Git protocol to another repository
update-server-info	Update auxiliary info file to help dumb servers

Low-level Commands / Internal Helpers

check-attr	Display gitattributes information
check-ignore	Debug gitignore / exclude files
check-mailmap	Show canonical names and email addresses of contacts
check-ref-format	Ensures that a reference name is well formed
column	Display data in columns
credential	Retrieve and store user credentials
credential-cache	Helper to temporarily store passwords in memory
credential-store	Helper to store credentials on disk
fmt-merge-msg	Produce a merge commit message
interpret-trailers	Add or parse structured information in commit messages
mailinfo	Extracts patch and authorship from a single e-mail message
mailsplit	Simple UNIX mbox splitter program
merge-one-file	The standard helper program to use with git-merge-index
patch-id	Compute unique ID for a patch
sh-i18n	Git's i18n setup code for shell scripts
sh-setup	Common Git shell script setup code
strip-space	Remove unnecessary whitespace

External commands

askyesno
credential-helper-selector
flow
lfs

Note: If you find yourself stuck in the list view, SHIFT + G to jump the end of the list, then q to exit the view.

Git GitHub Getting Started

Edit Code in GitHub

In addition to being a host for Git content, GitHub has a very good code editor.

Let's try to edit the README.md file in GitHub. Just click the edit button:

Search or jump to... Pull requests Issues Marketplace Explore

w3schools-test/hello-world Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags Go to file Add file Code

w3schools-test merged with hello-world-images after fixing conflicts e06e038 · 2 hours ago 6 commits

README.md	First release of Hello World!	6 hours ago
bluestyle.css	First release of Hello World!	6 hours ago
img_hello_git.jpg	added new image	3 hours ago
img_hello_world.jpg	Added image to Hello World	5 hours ago
index.html	merged with hello-world-images after fixing conflicts	2 hours ago

README.md

hello-world

Hello World repository for Git tutorial This is an example repository for the Git tutorial on <https://www.w3schools.com>

This repository is built step by step in the tutorial.

About No description, website, or topics provided. Readme Releases No releases published. Create a new release Packages No packages published. Publish your first package Languages HTML 81.9% CSS 18.1%

Add some changes to the code, and then commit the changes. For now, we will "Commit directly to the master branch".

Remember to add a description for the commit:

w3schools-test/hello-world Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

hello-world / README.md in master Cancel Changes

Edit file Preview changes Spaces 2 Soft wrap

```
1 # hello-world
2 Hello World repository for Git tutorial
3 This is an example repository for the Git tutorial on https://www.w3schools.com
4
5 This repository is built step by step in the tutorial.
6
7 It now includes steps for GitHub
```

Attach files by dragging & dropping, selecting or pasting them.

Commit changes

Updated README.md with a line about GitHub

Add an optional extended description...

☒ Commit directly to the master branch.

☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes Cancel

That is how you edit code directly in GitHub!

Pulling to Keep up-to-date with Changes

When working as a team on a project, it is important that everyone stays up to date.

Any time you start working on a project, you should get the most recent changes to your local copy.

With Git, you can do that with pull.

pull is a combination of 2 different commands:

- fetch
- merge

Let's take a closer look into how fetch, merge, and pull works.

Git Fetch

fetch gets all the change history of a tracked branch/repo.

So, on your local Git, fetch updates to see what has changed on GitHub:

Example

```
git fetch origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 733 bytes | 3.00 KiB/s, done.
From https://github.com/w3schools-test/hello-world
   e0b6038..d29d69f  master    -> origin/master
```

Now that we have the recent changes, we can check our status:

Example

```
git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)
```

nothing to commit, working tree clean

We are behind the origin/master by 1 commit. That should be the updated README.md, but lets double check by viewing the log:

Example

```
git log origin/master
commit d29d69ffe2ee9e6df6fa0d313bb0592b50f3b853 (origin/master)
```

Author: w3schools-test <77673807+w3schools-test@users.noreply.github.com>
Date: Fri Mar 26 14:59:14 2021 +0100

Updated README.md with a line about GitHub

commit e0b6038b1345e50aca8885d8fd322fc0e5765c3b (HEAD -> master)
Merge: dfa79db 1f1584e
Author: w3schools-test
Date: Fri Mar 26 12:42:56 2021 +0100

merged with hello-world-images after fixing conflicts

...
...

That looks as expected, but we can also verify by showing the differences between our local master and origin/master:

Example

```
gitdiff origin/master
diff --git a/README.md b/README.md
index 23a0122..a980c39 100644
--- a/README.md
+++ b/README.md
@@ -2,6 +2,4 @@
Hello World repository for Git tutorial
This is an example repository for the Git tutoial on https://www.w3schools.com
```

-This repository is built step by step in the tutorial.

-

-It now includes steps for GitHub

+This repository is built step by step in the tutorial.

\ No newline at end of file

That looks precisely as expected! Now we can safely merge.

Git Merge

merge combines the current branch, with a specified branch.

We have confirmed that the updates are as expected, and we can merge our current branch (master) with origin/master:

Example

```
git merge origin/master
Updating e0b6038..d29d69f
Fast-forward
 README.md | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
```

Check our status again to confirm we are up to date:

Example

git status

On branch master

Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

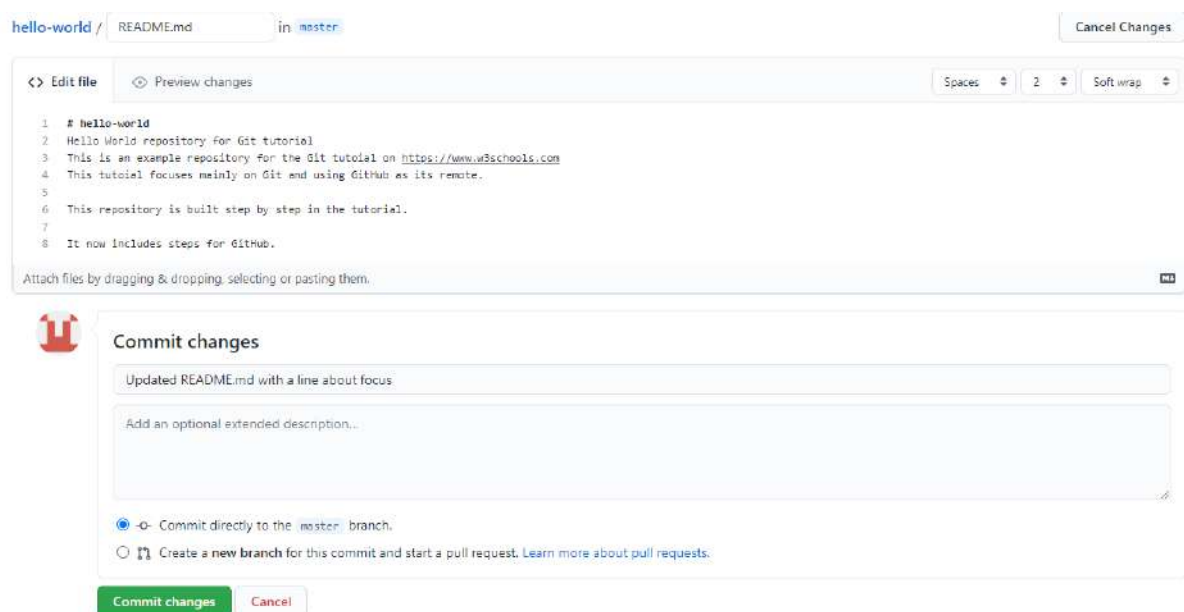
There! Your local git is up to date!

Git Pull

But what if you just want to update your local repository, without going through all those steps?

pull is a combination of fetch and merge. It is used to pull all changes from a remote repository into the branch you are working on.

Make another change to the Readme.md file on GitHub.



Use pull to update our local Git:

Example

git pull origin

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), 794 bytes | 1024 bytes/s, done.

From https://github.com/w3schools-test/hello-world

a7cdd4b..ab6b4ed master -> origin/master

Updating a7cdd4b..ab6b4ed

Fast-forward

README.md | 2 ++
1 file changed, 2 insertions(+)

That is how you keep your local Git up to date from a remote repository. In the next chapter, we will look closer at how push works on GitHub.

Git Push to GitHub

Push Changes to GitHub

Let's try making some changes to our local git and pushing them to GitHub.

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
<link rel="stylesheet" href="bluestyle.css">
</head>
<body>

<h1>Hello world!</h1>
<div><imgsrc="img_hello_world.jpg" alt="Hello World from Space" style="width:100%;max-
width:640px"></div>
<p>This is the first file in my new Git Repo.</p>
<p>This line is here to show how merging works.</p>
<div><imgsrc="img_hello_git.jpg" alt="Hello Git" style="width:100%;max-width:640px"></div>

</body>
</html>
```

Commit the changes:

Example

```
git commit -a -m "Updated index.html. Resized image"
[master e7de78f] Updated index.html. Resized image
1 file changed, 1 insertion(+), 1 deletion(-)
```

And check the status:

Example

```
git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)
```

nothing to commit, working tree clean

Now push our changes to our remote origin:

Example

git push origin

Enumerating objects: 9, done.

Counting objects: 100% (8/8), done.

Delta compression using up to 16 threads

Compressing objects: 100% (5/5), done.

Writing objects: 100% (5/5), 578 bytes | 578.00 KiB/s, done.

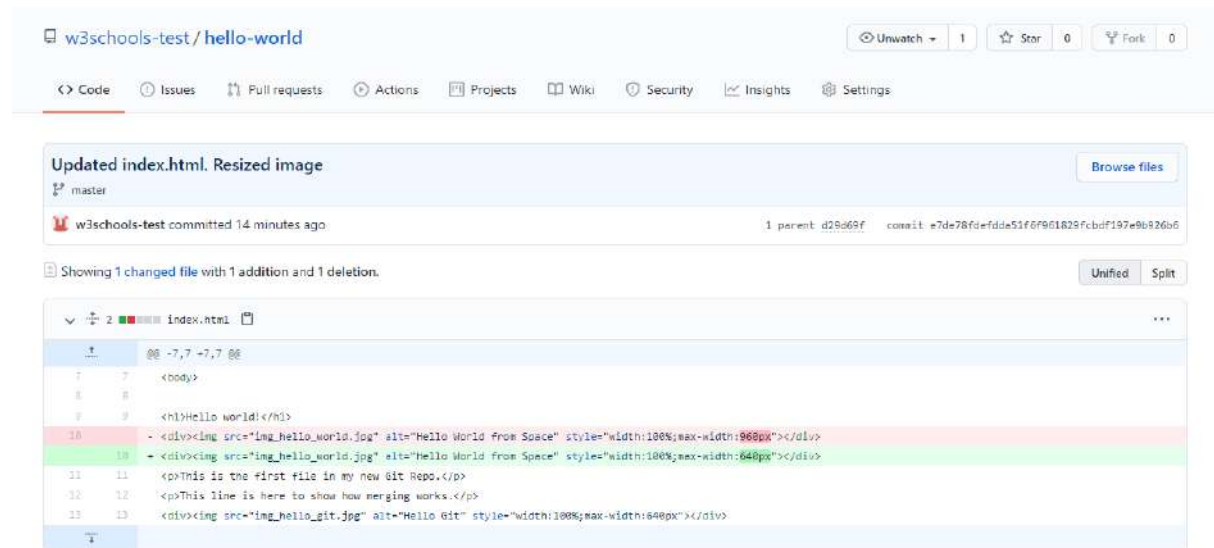
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (3/3), completed with 3 local objects.

To https://github.com/w3schools-test/hello-world.git

5a04b6f..facaee master -> master

Go to GitHub, and confirm that the repository has a new commit:



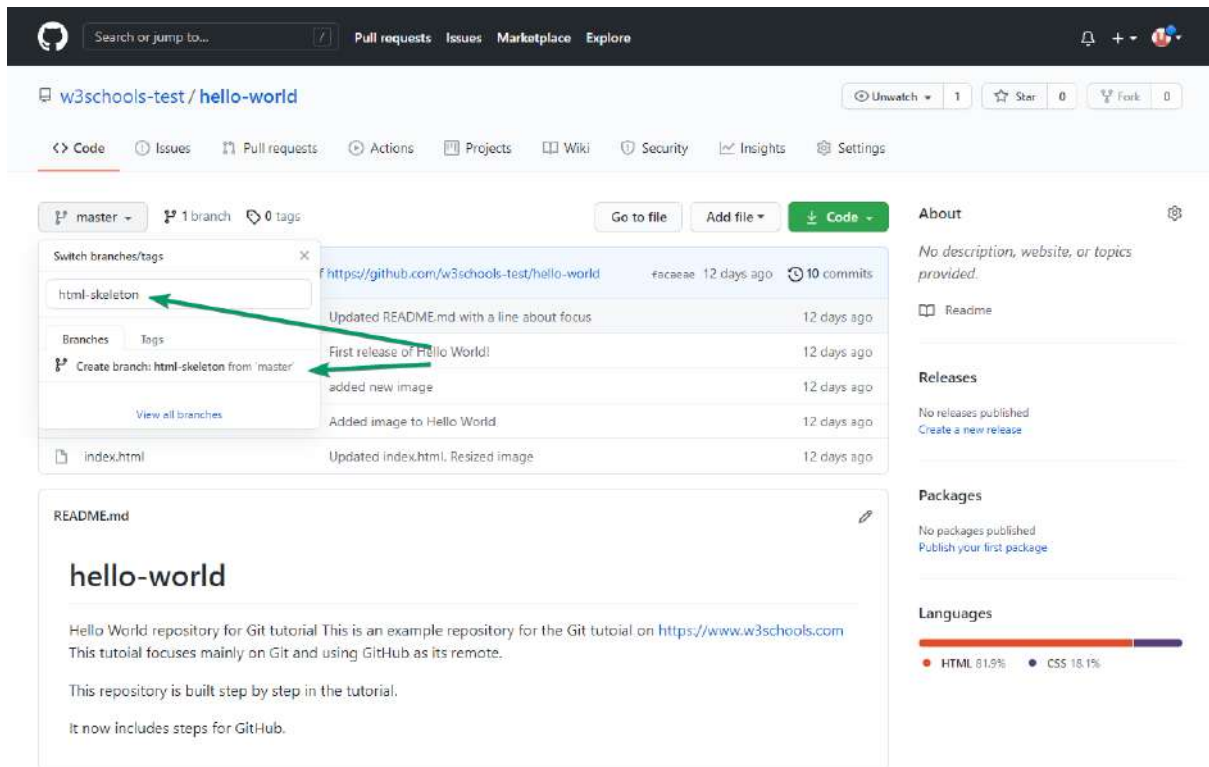
Now, we are going to start working on branches on GitHub.

Git GitHub Branch

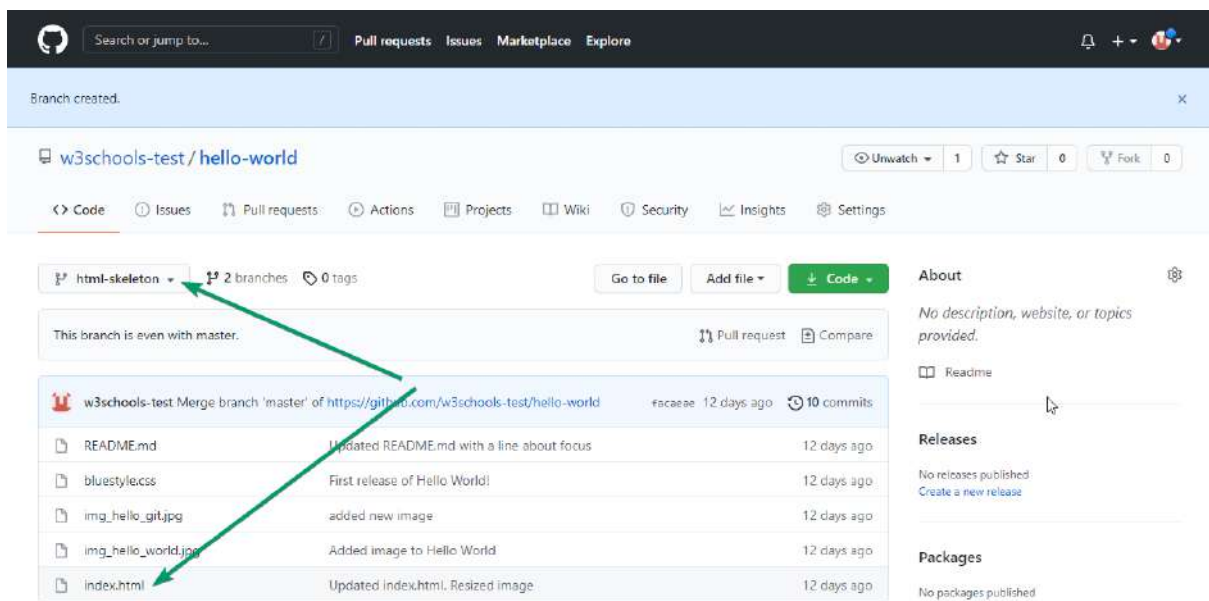
Create a New Branch on GitHub

On GitHub, access your repository and click the "master" branch button.

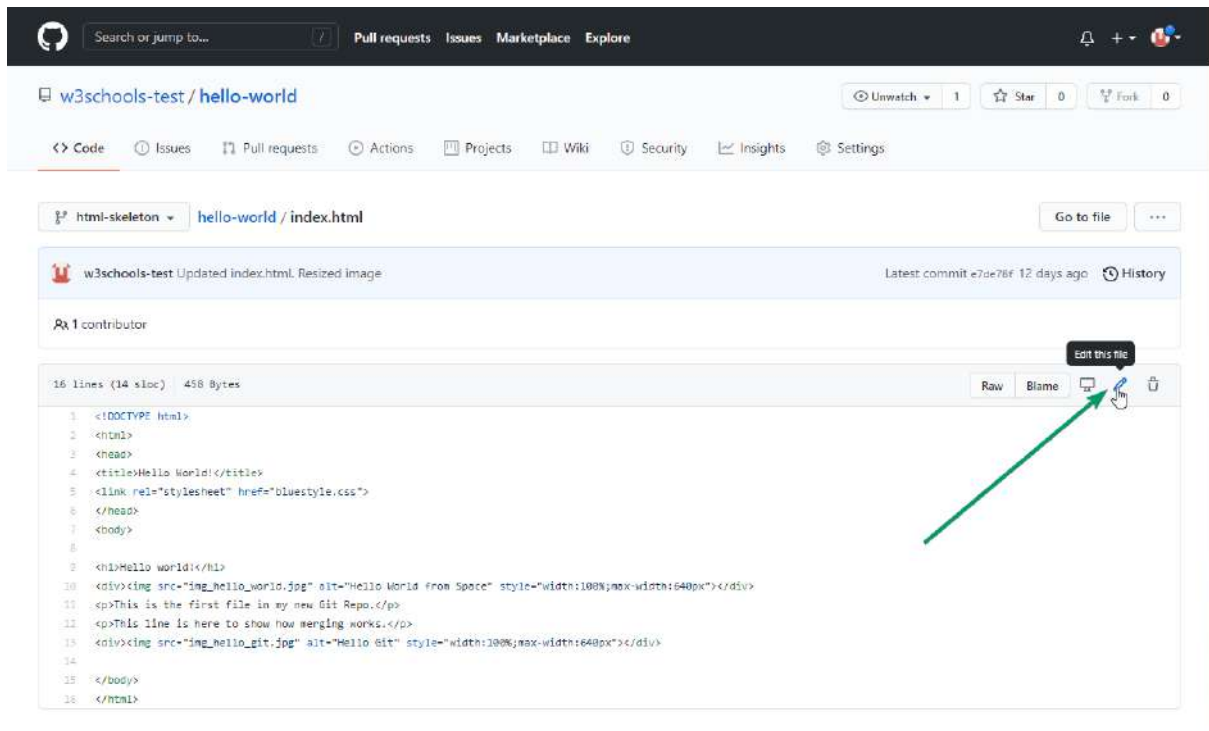
There you can create a new Branch. Type in a descriptive name, and click Create branch:



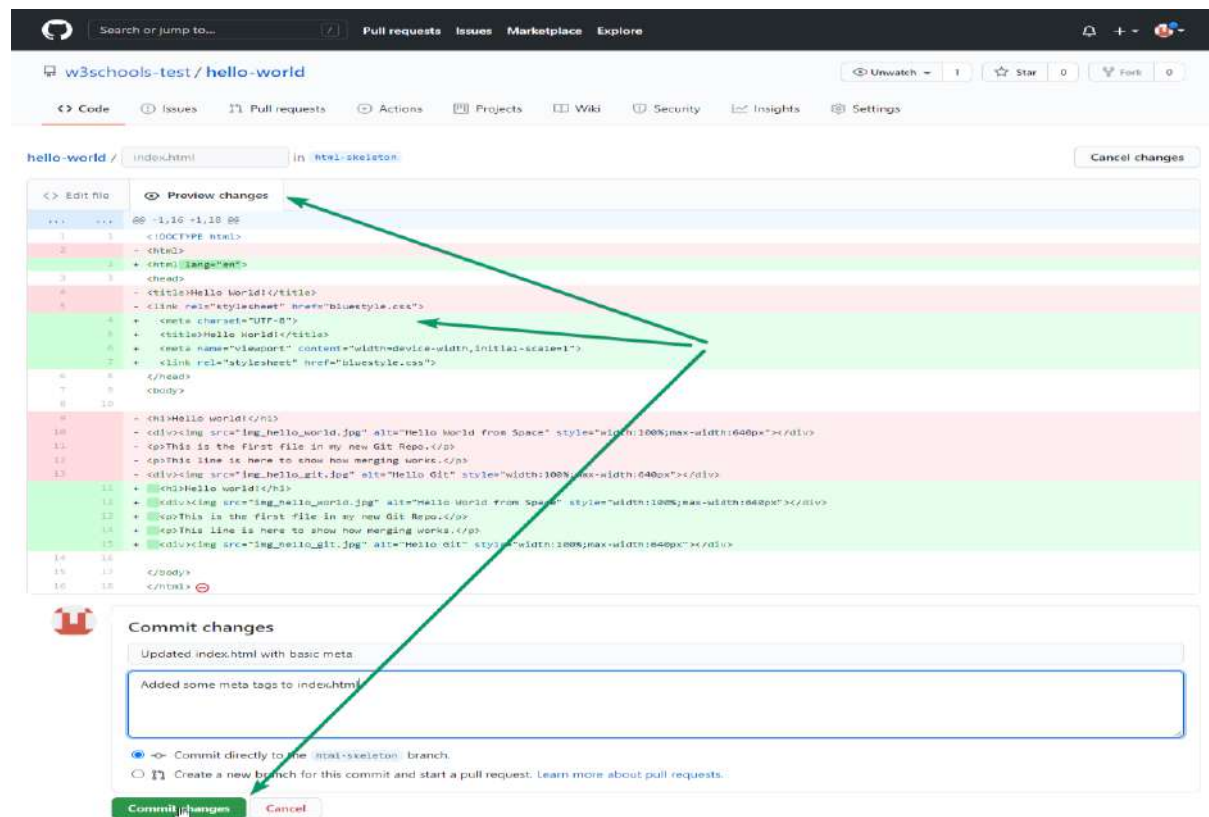
The branch should now be created and active. You can confirm which branch you are working on by looking at the branch button. See that it now says "html-skeleton" instead of "main"?



Start working on an existing file in this branch. Click the "index.html" file and start editing:



After you have finished editing the file, you can click the "Preview changes" tab to see the changes you made highlighted:



If you are happy with the change, add a comment that explains what you did, and click Commit changes.

Git Pull Branch from GitHub

Pulling a Branch from GitHub

Now continue working on our new branch in our local Git.

Letspull from our GitHub repository again so that our code is up-to-date:

Example

```
gitpull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 851 bytes | 9.00 KiB/s, done.
From https://github.com/w3schools-test/hello-world
* [new branch]    html-skeleton -> origin/html-skeleton
Already up to date.
```

Now our main branch is up todate. And we can see that there is a new branch available on GitHub.

Do a quick status check:

Example

```
git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

And confirm which branches we have, and where we are working at the moment:

Example

```
git branch
* master
```

So, we do not have the new branch on our local Git. But we know it is available on GitHub. So we can use the -a option to see all local and remote branches:

Example

```
git branch -a
* master
remotes/origin/html-skeleton
remotes/origin/master
```

Note:branch -r is for remote branches only.

We see that the branch html-skeleton is available remotely, but not on our local git. Lets check it out:

Example

```
git checkout html-skeleton
Switched to a new branch 'html-skeleton'
Branch 'html-skeleton' set up to track remote branch 'html-skeleton' from 'origin'.
```

And check if it is all up to date:

Example

```
gitpull
Already up to date.
```

Which branches do we have now, and where are we working from?

Example

```
git branch
* html-skeleton
master
```

Now, open your favourite editor and confirm that the changes from the GitHub branch carried over.

That is how you pull a GitHub branch to your local Git.

Git Push Branch to GitHub

Push a Branch to GitHub

Let's try to create a new local branch, and push that to GitHub.

Start by creating a branch, like we did earlier:

Example

```
git checkout -b update-readme
Switched to a new branch 'update-readme'
```

And we make some changes to the README.md file. Just add a new line.

So now we check the status of the current branch.

Example

```
git status
On branch update-readme
Changes not staged for commit:
  (use "git add ..." to update what will be committed)
  (use "git restore ..." to discard changes in working directory)
        modified:   README.md
```

no changes added to commit (use "git add" and/or "git commit -a")

We see that README.md is modified but not added to the Staging Environment:

Example

```
gitadd README.md
```

Check the status of the branch:

Example

```
git status
On branch update-readme
Changes to be committed:
  (use "git restore --staged ..." to unstage)
    modified:   README.md
```

We are happy with our changes. So we will commit them to the branch:

Example

```
git commit -m "Updated readme for GitHub Branches"
[update-readme 836e5bf] Updated readme for GitHub Branches
1 file changed, 1 insertion(+)
```

Now push the branch from our local Git repository, to GitHub, where everyone can see the changes:

Example

```
git push origin update-readme
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 366 bytes | 366.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'update-readme' on GitHub by visiting:
remote:   https://github.com/w3schools-test/hello-world/pull/new/update-readme
remote:
To https://github.com/w3schools-test/hello-world.git
* [new branch]   update-readme -> update-readme
```

Go to GitHub, and confirm that the repository has a new branch:

The screenshot displays the GitHub interface for the repository `w3schools-test/hello-world`. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below this, the repository name is shown with options to Unwatch, Star, and Fork. A secondary navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings.

Two yellow banners highlight recent pushes: `html-skeleton` (about 1 hour ago) and `update-readme` (1 minute ago), each with a "Compare & pull request" button. The repository navigation bar shows the `master` branch selected, with 3 branches and 0 tags. Buttons for "Go to file", "Add file", and "Code" are present.

A table lists the repository's files and their commit history:

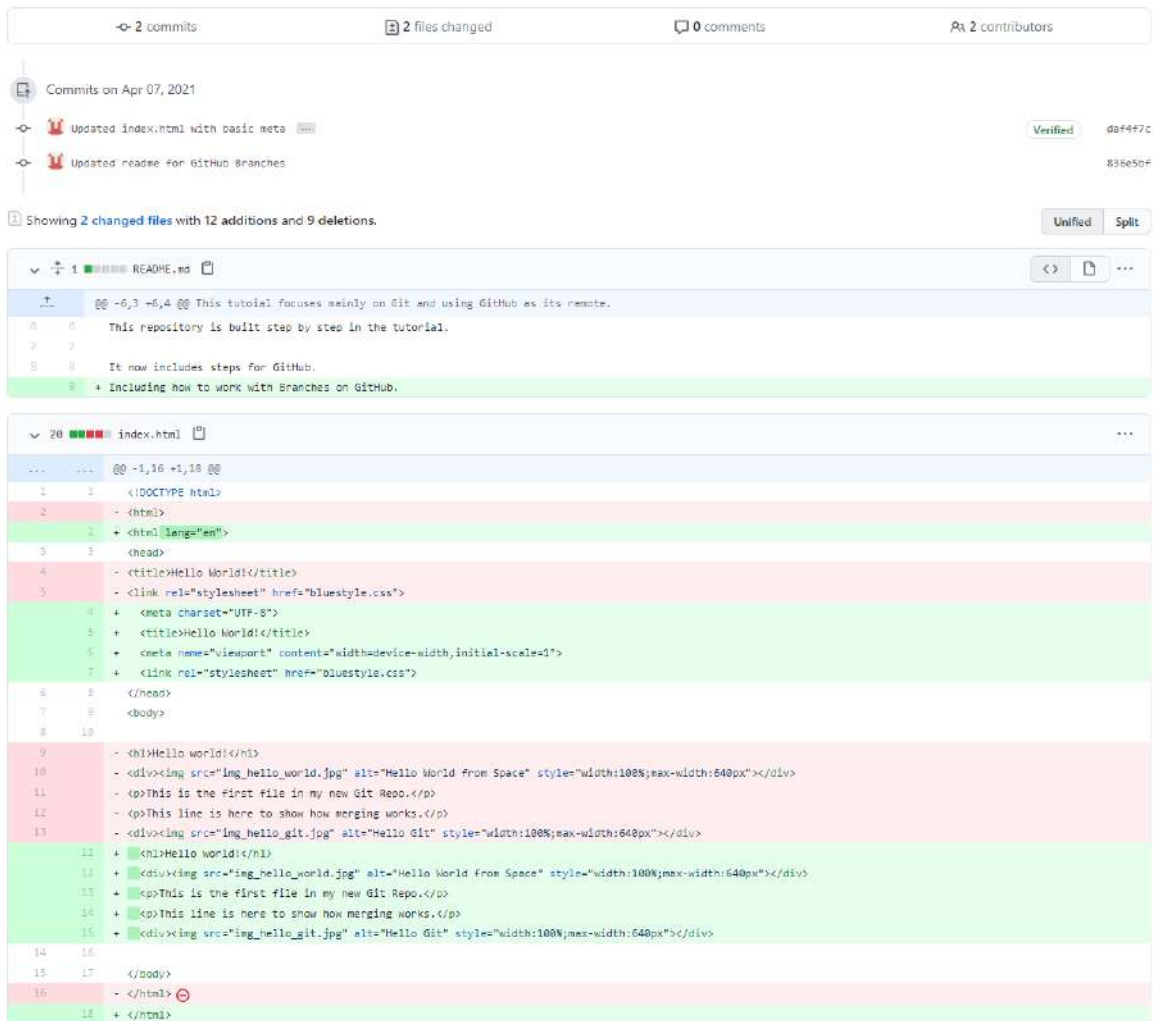
File	Commit Message	Time
README.md	Updated README.md with a line about focus	12 days ago
bluestyle.css	First release of Hello World!	12 days ago
img_hello_git.jpg	added new image	12 days ago
img_hello_world.jpg	Added image to Hello World	12 days ago
index.html	Updated index.html. Resized image	12 days ago

Below the file list, the `README.md` content is displayed, featuring the title "hello-world" and a description of the repository as a Git tutorial example. A green arrow points from the "3 branches" link in the navigation bar to the `master` branch in the commit history table.

On the right side, there are sections for "About" (no description provided), "Releases" (no releases published), "Packages" (no packages published), and "Languages" (HTML 81.9%, CSS 18.1%).

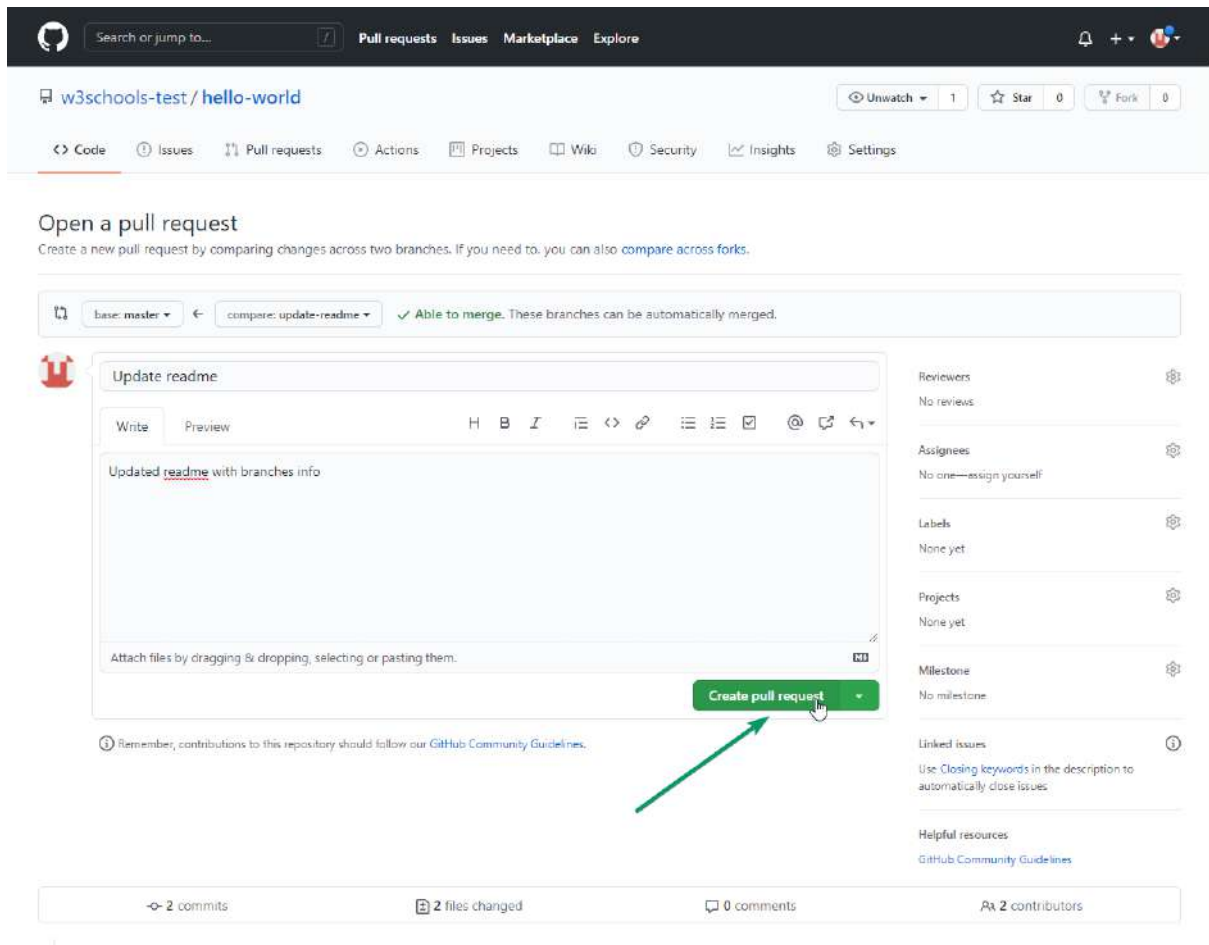
In GitHub, we can now see the changes and merge them into the master branch if we approve it.

If you click the "Compare & pull request", you can go through the changes made and new files added:



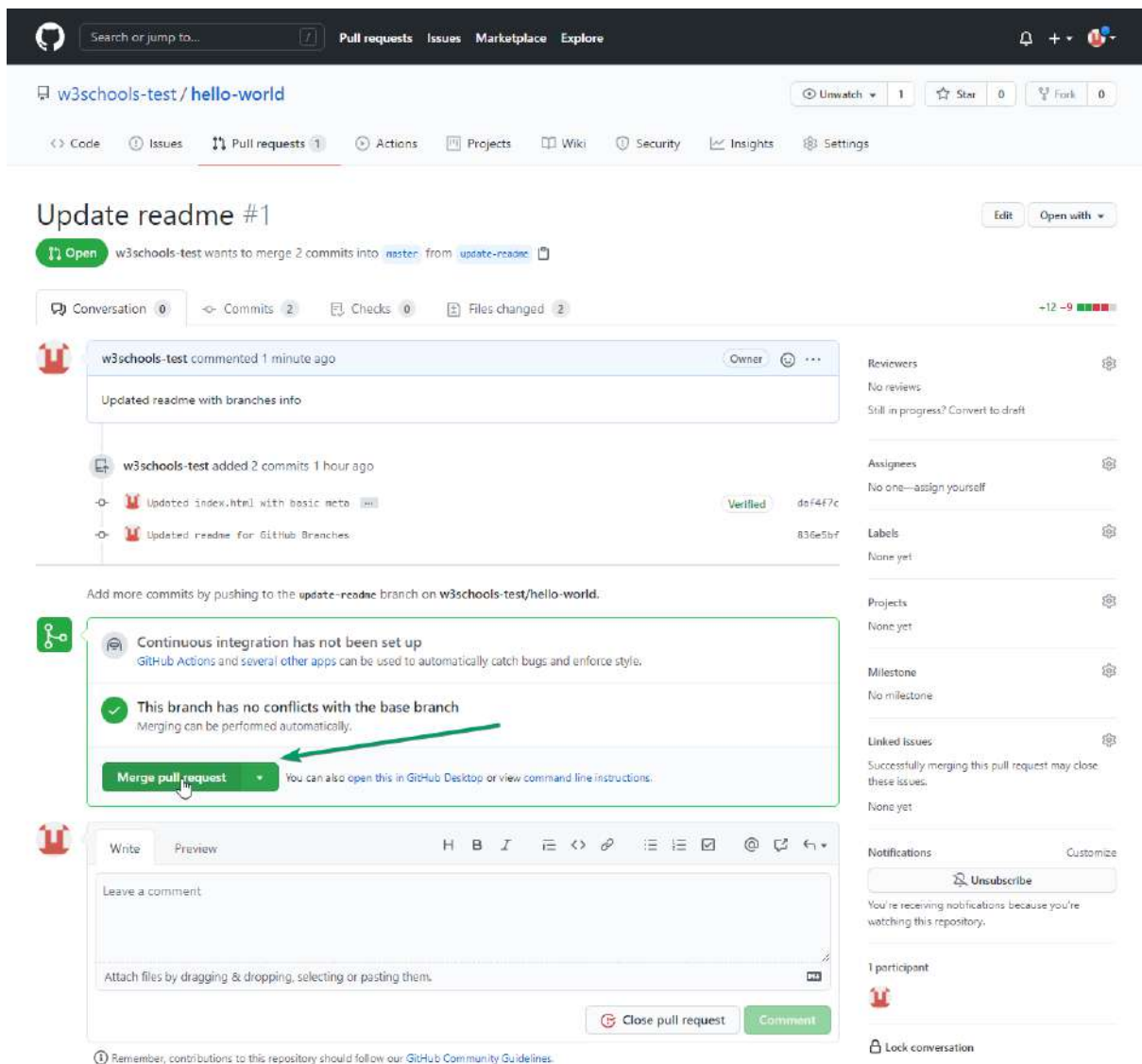
Note: This comparison shows both the changes from update-readme and html-skeleton because we created the new branch FROM html-skeleton.

If the changes look good, you can go forward, creating a pull request:



A pull request is how you propose changes. You can ask some to review your changes or pull your contribution and merge it into their branch.

Since this is your own repository, you can merge your pull request yourself:



The pull request will record the changes, which means you can go through them later to figure out the changes made.

The result should be something like this:

Search or jump to... Pull requests Issues Marketplace Explore

w3schools-test / hello-world

Unwatch 1 Star 0 Fork 0

Code Issues Pull requests 1 Actions Projects Wiki Security Insights Settings

Update readme #1

Merged w3schools-test merged 2 commits into master from update-readme now

Conversation 0 Commits 2 Checks 0 Files changed 2 +12 -9

w3schools-test commented 2 minutes ago

Updated readme with branches info

w3schools-test added 2 commits 1 hour ago

- Updated index.html with basic sets
- Updated readme for GitHub Branches

w3schools-test merged commit 3f4aa5b into master now

Pull request successfully merged and closed

You're all set—the update-readme branch can be safely deleted.

Delete branch

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Comment

Remember, contributions to this repository should follow our GitHub Community Guidelines.

ProTip! Add .patch or .diff to the end of URLs for Git's plaintext views.

Reviewers: No reviews

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

Linked issues: Successfully merging this pull request may close these issues. None yet

Notifications: Unsubscribe

You're receiving notifications because you're watching this repository.

1 participant

Lock conversation

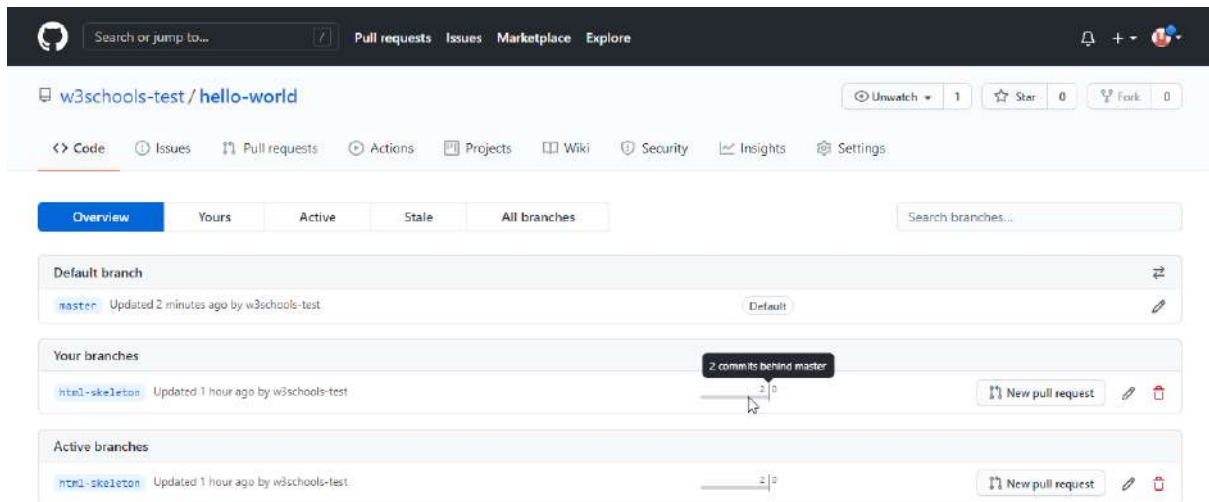
To keep the repo from getting overly complicated, you can delete the now unused branch by clicking "Delete branch".

Pull request successfully merged and closed

You're all set—the update-readme branch can be safely deleted.

Delete branch

After you confirm that the changes from the previous branch were included, delete that as well:



CSS Tutorial

CSS tutorial or CSS 3 tutorial provides basic and advanced concepts of CSS technology. Our CSS tutorial is developed for beginners and professionals. The major points of CSS are given below:

- CSS stands for Cascading Style Sheet.
- CSS is used to design HTML tags.
- CSS is a widely used language on the web.
- HTML, CSS and JavaScript are used for web designing. It helps the web designers to apply style on HTML tags.

CSS Example with CSS Editor

In this tutorial, you will get a lot of CSS examples, you can edit and run these examples with our online CSS editor tool.

1. <!DOCTYPE>
2. <html>
3. <head>
4. <style>
5. h1{
6. color:white;
7. background-color:red;
8. padding:5px;
9. }
10. p{
11. color:blue;
12. }
13. </style>
14. </head>

-
15. <body>
 16. <h1>Write Your First CSS Example</h1>
 17. <p>This is Paragraph.</p>
 18. </body>
 19. </html>

[Test it Now](#)

Output:

Write Your First CSS Example

This is Paragraph.

What is CSS

CSS stands for Cascading Style Sheets. It is a style sheet language which is used to describe the look and formatting of a document written in markup language. It provides an additional feature to HTML. It is generally used with HTML to change the style of web pages and user interfaces. It can also be used with any kind of XML documents including plain XML, SVG and XUL.

CSS is used along with HTML and JavaScript in most websites to create user interfaces for web applications and user interfaces for many mobile applications.

What does CSS do

- You can add new looks to your old HTML documents.
- You can completely change the look of your website with only a few changes in CSS code.

Why use CSS

These are the three major benefits of CSS:

1) Solves a big problem

Before CSS, tags like font, color, background style, element alignments, border and size had to be repeated on every web page. This was a very long process. For example: If you are developing a large website where fonts and color information are added on every single page, it will become a long and expensive process. CSS was created to solve this problem. It was a W3C recommendation.

2) Saves a lot of time

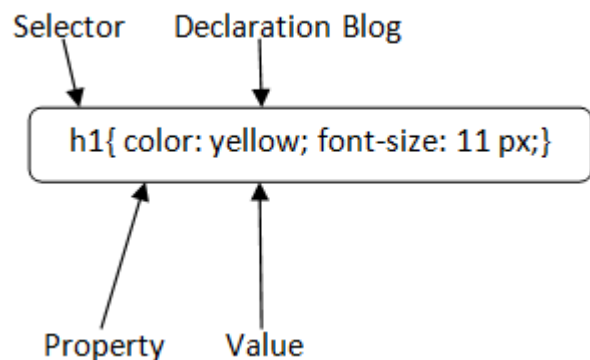
CSS style definitions are saved in external CSS files so it is possible to change the entire website by changing just one file.

3) Provide more attributes

CSS provides more detailed attributes than plain HTML to define the look and feel of the website.

CSS Syntax

A CSS rule set contains a selector and a declaration block.



Selector: Selector indicates the HTML element you want to style. It could be any tag like `<h1>`, `<title>` etc.

Declaration Block: The declaration block can contain one or more declarations separated by a semicolon. For the above example, there are two declarations:

1. `color: yellow;`
2. `font-size: 11 px;`

Each declaration contains a property name and value, separated by a colon.

Property: A Property is a type of attribute of HTML element. It could be color, border etc.

Value: Values are assigned to CSS properties. In the above example, value "yellow" is assigned to color property.

Selector{Property1: value1; Property2: value2;;} CSS Selector

CSS selectors are used *to select the content you want to style*. Selectors are the part of CSS

rule set. CSS selectors select HTML elements according to its id, class, type, attribute etc.

There are several different types of selectors in CSS.

1. CSS Element Selector
2. CSS Id Selector
3. CSS Class Selector
4. CSS Universal Selector
5. CSS Group Selector

1) CSS Element Selector

The element selector selects the HTML element by name.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `p{`
6. `text-align: center;`
7. `color: blue;`
8. `}`
9. `</style>`
10. `</head>`
11. `<body>`
12. `<p>This style will be applied on every paragraph.</p>`
13. `<p id="para1">Me too!</p>`
14. `<p>And me!</p>`
15. `</body>`
16. `</html>`

[Test it Now](#)

Output:

This style will be applied on every paragraph.

Me too!

And me!

2) CSS Id Selector

The id selector selects the id attribute of an HTML element to select a specific element. An id is always unique within the page so it is chosen to select a single, unique element.

It is written with the hash character (#), followed by the id of the element.

Let's take an example with the id "para1".

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. #para1 {
6.     text-align: center;
7.     color: blue;
8. }
9. </style>
10. </head>
11. <body>
12. <p id="para1">Hello Javatpoint.com</p>
13. <p>This paragraph will not be affected.</p>
14. </body>
15. </html>
```

[Test it Now](#)

Output:

Hello Javatpoint.com

This paragraph will not be affected.

3) CSS Class Selector

The class selector selects HTML elements with a specific class attribute. It is used with a period character . (full stop symbol) followed by the class name.

Note: A class name should not be started with a number.

Let's take an example with a class "center".

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. .center {
6.     text-align: center;
7.     color: blue;
8. }
9. </style>
10. </head>
11. <body>
12. <h1 class="center">This heading is blue and center-aligned.</h1>
13. <p class="center">This paragraph is blue and center-aligned.</p>
```

-
14. `</body>`
 15. `</html>`

[Test it Now](#)

Output:

This heading is blue and center-aligned.

This paragraph is blue and center-aligned.

CSS Class Selector for specific element

If you want to specify that only one specific HTML element should be affected then you should use the element name with class selector.

Let's see an example.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `p.center {`
6. `text-align: center;`
7. `color: blue;`
8. `}`
9. `</style>`
10. `</head>`
11. `<body>`
12. `<h1 class="center">This heading is not affected</h1>`
13. `<p class="center">This paragraph is blue and center-aligned.</p>`
14. `</body>`
15. `</html>`

[Test it Now](#)

Output:

This heading is not affected

This paragraph is blue and center-aligned.

4) CSS Universal Selector

The universal selector is used as a wildcard character. It selects all the elements on the pages.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. * {
6.   color: green;
7.   font-size: 20px;
8. }
9. </style>
10. </head>
11. <body>
12. <h2>This is heading</h2>
13. <p>This style will be applied on every paragraph.</p>
14. <p id="para1">Me too!</p>
15. <p>And me!</p>
16. </body>
17. </html>
```

[Test it Now](#)

Output:

This is heading

This style will be applied on every paragraph.

Me too!

And me!

5) CSS Group Selector

The grouping selector is used to select all the elements with the same style definitions.

Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.

Let's see the CSS code without group selector.

```
1. h1 {
2.   text-align: center;
3.   color: blue;
4. }
5. h2 {
6.   text-align: center;
```

```
7.    color: blue;
8.  }
9.  p {
10.    text-align: center;
11.    color: blue;
12.  }
```

As you can see, you need to define CSS properties for all the elements. It can be grouped in following ways:

```
1.  h1,h2,p {
2.    text-align: center;
3.    color: blue;
4.  }
```

Let's see the full example of CSS group selector.

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <style>
5.  h1, h2, p {
6.    text-align: center;
7.    color: blue;
8.  }
9.  </style>
10. </head>
11. <body>
12. <h1>Hello Javatpoint.com</h1>
13. <h2>Hello Javatpoint.com (In smaller font)</h2>
14. <p>This is a paragraph.</p>
15. </body>
16. </html>
```

Output:

Hello Javatpoint.com

Hello Javatpoint.com (In smaller font)

This is a paragraph.

How to add CSS

CSS is added to HTML pages to format the document according to information in the style sheet. There are three ways to insert CSS in HTML documents.

1. Inline CSS
2. Internal CSS

3. External CSS

1) Inline CSS

Inline CSS is used to apply CSS on a single line or element.

For example:

1. `<p style="color:blue">Hello CSS</p>`

For more visit here: [Inline CSS](#)

2) Internal CSS

Internal CSS is used to apply CSS on a single document or page. It can affect all the elements of the page. It is written inside the style tag within head section of html.

For example:

1. `<style>`
2. `p{color:blue}`
3. `</style>`

For more visit here: [Internal CSS](#)

3) External CSS

External CSS is used to apply CSS on multiple pages or all pages. Here, we write all the CSS code in a css file. Its extension must be .css for example style.css.

For example:

1. `p{color:blue}`

You need to link this style.css file to your html pages like this:

1. `<link rel="stylesheet" type="text/css" href="style.css">`

The link tag must be used inside head section of html.

Inline CSS

We can apply CSS in a single element by inline CSS technique.

The inline CSS is also a method to insert style sheets in HTML document. This method mitigates some advantages of style sheets so it is advised to use this method sparingly.

If you want to use inline CSS, you should use the style attribute to the relevant tag.

Syntax:

1. `<htmltag style="cssproperty1:value; cssproperty2:value;"> </htmltag>`

Example:

1. `<h2 style="color:red;margin-left:40px;">Inline CSS is applied on this heading.</h2>`
2. `<p>This paragraph is not affected.</p>`

Output:

Inline CSS is applied on this heading.

This paragraph is not affected.

Disadvantages of Inline CSS

- You cannot use quotations within inline CSS. If you use quotations the browser will interpret this as an end of your style value.
- These styles cannot be reused anywhere else.
- These styles are tough to be edited because they are not stored at a single place.
- It is not possible to style pseudo-codes and pseudo-classes with inline CSS.
- Inline CSS does not provide browser cache advantages.

Internal CSS

The internal style sheet is used to add a unique style for a single document. It is defined in `<head>` section of the HTML page inside the `<style>` tag.

Example:

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `body {`

```
6.    background-color: linen;
7.  }
8.  h1 {
9.    color: red;
10.   margin-left: 80px;
11. }
12. </style>
13. </head>
14. <body>
15. <h1>The internal style sheet is applied on this heading.</h1>
16. <p>This paragraph will not be affected.</p>
17. </body>
18. </html>
```

External CSS

The external style sheet is generally used when you want to make changes on multiple pages. It is ideal for this condition because it facilitates you to change the look of the entire web site by changing just one file.

It uses the <link> tag on every pages and the <link> tag should be put inside the head section.

Example:

```
1. <head>
2. <link rel="stylesheet" type="text/css" href="mystyle.css">
3. </head>
```

The external style sheet may be written in any text editor but must be saved with a .css extension. This file should not contain HTML elements.

Let's take an example of a style sheet file named "mystyle.css".

File: mystyle.css

```
1. body {
2.   background-color: lightblue;
3. }
4. h1 {
5.   color: navy;
6.   margin-left: 20px;
7. }
```

Note: You should not use a space between the property value and the unit. For example: It should be margin-left:20px not margin-left:20 px.

CSS Comments

CSS comments are generally written to explain your code. It is very helpful for the users who reads your code so that they can easily understand the code.

Comments are ignored by browsers.

Comments are single or multiple lines statement and written within `/*.....*/`.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. p {
6.     color: blue;
7.     /* This is a single-line comment */
8.     text-align: center;
9. }
10. /* This is
11. a multi-line
12. comment */
13. </style>
14. </head>
15. <body>
16. <p>Hello Javatpoint.com</p>
17. <p>This statement is styled with CSS.</p>
18. <p>CSS comments are ignored by the browsers and not shown in the output.</p>
19. </body>
20. </html>
```

Output:

Hello Javatpoint.com

This statement is styled with CSS.

CSS comments are ignored by the browsers and not shown in the output.

CSS Background

CSS background property is used to define the background effects on element. There are 5 CSS background properties that affects the HTML elements:

1. background-color
2. background-image
3. background-repeat
4. background-attachment
5. background-position

1) CSS background-color

The background-color property is used to specify the background color of the element.

You can set the background color like this:

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5.   h2,p{
6.     background-color: #b0d4de;
7.   }
8. </style>
9. </head>
10. <body>
11. <h2>My first CSS page.</h2>
12. <p>Hello Javatpoint. This is an example of CSS background-color.</p>
13. </body>
14. </html>
```

Output:

My first CSS page.

Hello Javatpoint. This is an example of CSS background-color.

2) CSS background-image

The background-image property is used to set an image as a background of an element. By default the image covers the entire element. You can set the background image for a page like this.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5.   body {
6.     background-image: url("paper1.gif");
7.     margin-left:100px;
8.   }
9. </style>
10. </head>
11. <body>
12. <h1>Hello Javatpoint.com</h1>
```

13. </body>
14. </html>

Note: The background image should be chosen according to text color. The bad combination of text and background image may be a cause of poor designed and not readable webpage.

3) CSS background-repeat

By default, the background-image property repeats the background image horizontally and vertically. Some images are repeated only horizontally or vertically.

The background looks better if the image repeated horizontally only.

background-repeat: repeat-x;

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. body {
6. background-image: url("gradient_bg.png");
7. background-repeat: repeat-x;
8. }
9. </style>
10. </head>
11. <body>
12. <h1>Hello Javatpoint.com</h1>
13. </body>
14. </html>

background-repeat: repeat-y;

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. body {
6. background-image: url("gradient_bg.png");
7. background-repeat: repeat-y;
8. }
9. </style>
10. </head>
11. <body>
12. <h1>Hello Javatpoint.com</h1>
13. </body>
14. </html>

4) CSS background-attachment

The background-attachment property is used to specify if the background image is fixed or scroll with the rest of the page in browser window. If you set fixed the background image then the image will not move during scrolling in the browser. Let's take an example with fixed background image.

1. background: white url('bbb.gif');
2. background-repeat: no-repeat;
3. background-attachment: fixed;

5) CSS background-position

The background-position property is used to define the initial position of the background image. By default, the background image is placed on the top-left of the webpage.

You can set the following positions:

1. center
 2. top
 3. bottom
 4. left
 5. right
-
1. background: white url('good-morning.jpg');
 2. background-repeat: no-repeat;
 3. background-attachment: fixed;
 4. background-position: center;

CSS Border

The CSS border is a shorthand property used to set the border on an element.

The [CSS](#) border properties are used to specify the style, color and size of the border of an element. The CSS border properties are given below

- border-style
- border-color
- border-width
- border-radius

1) CSS border-style

The Border style property is used to specify the border type which you want to display on the web page.

There are some border style values which are used with border-style property to define a border.

Value	Description
none	It doesn't define any border.
dotted	It is used to define a dotted border.
dashed	It is used to define a dashed border.
solid	It is used to define a solid border.
double	It defines two borders wIth the same border-width value.
groove	It defines a 3d grooved border. effect is generated according to border-color value.
ridge	It defines a 3d ridged border. effect is generated according to border-color value.
inset	It defines a 3d inset border. effect is generated according to border-color value.
outset	It defines a 3d outset border. effect is generated according to border-color value.

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. p.none {border-style: none;}
6. p.dotted {border-style: dotted;}
7. p.dashed {border-style: dashed;}
8. p.solid {border-style: solid;}
9. p.double {border-style: double;}
10. p.groove {border-style: groove;}
11. p.ridge {border-style: ridge;}
12. p.inset {border-style: inset;}
13. p.outset {border-style: outset;}
14. p.hidden {border-style: hidden;}
15. </style>
16. </head>
17. <body>
18. <p class="none">No border.</p>
19. <p class="dotted">A dotted border.</p>
20. <p class="dashed">A dashed border.</p>
21. <p class="solid">A solid border.</p>
22. <p class="double">A double border.</p>
23. <p class="groove">A groove border.</p>
24. <p class="ridge">A ridge border.</p>
25. <p class="inset">An inset border.</p>
26. <p class="outset">An outset border.</p>


```
27. <p class="hidden">A hidden border.</p>
28. </body>
29. </html>
```

Output:

No border.

.....
A dotted border.
.....

A dashed border.

=====

=====

=====

=====

=====

=====

A hidden border.

2) CSS border-width

The border-width property is used to set the border's width. It is set in pixels. You can also use the one of the three pre-defined values, thin, medium or thick to set the width of the border.

Note: The border-width property is not used alone. It is always used with other border properties like "border-style" property to set the border first otherwise it will not work.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. p.one {
6.     border-style: solid;
7.     border-width: 5px;
8. }
9. p.two {
10.    border-style: solid;
11.    border-width: medium;
```

```
12. }
13. p.three {
14.   border-style: solid;
15.   border-width: 1px;
16. }
17. </style>
18. </head>
19. <body>
20. <p class="one">Write your text here.</p>
21. <p class="two">Write your text here.</p>
22. <p class="three">Write your text here.</p>
23. </body>
24. </html>
```

3) CSS border-color

There are three methods to set the color of the border.

- Name: It specifies the color name. For example: "red".
- RGB: It specifies the RGB value of the color. For example: "rgb(255,0,0)".
- Hex: It specifies the hex value of the color. For example: "#ff0000".

There is also a border color named "transparent". If the border color is not set it is inherited from the color property of the element.

Note: The border-color property is not used alone. It is always used with other border properties like "border-style" property to set the border first otherwise it will not work.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5.   p.one {
6.     border-style: solid;
7.     border-color: red;
8.   }
9.   p.two {
10.    border-style: solid;
11.    border-color: #98bf21;
12.  }
13. </style>
14. </head>
15. <body>
16. <p class="one">This is a solid red border</p>
17. <p class="two">This is a solid green border</p>
18. </body>
19. </html>
```

CSS border-collapse property

This CSS property is used to set the border of the table cells and specifies whether the table cells share the separate or common border.

This property has two main values that are **separate** and **collapse**. When it is set to the value **separate**, the distance between the cells can be defined using the **border-spacing** property. When the **border-collapse** is set to the value **collapse**, then the **inset** value of **border-style** property behaves like **groove**, and the **outset** value behaves like **ridge**.

Syntax

1. border-collapse: separate | collapse | initial | inherit;

The values of this CSS property are defined as follows.

Property Values

separate: It is the default value that separates the border of the table cell. Using this value, each cell will display its own border.

collapse: This value is used to collapse the borders into a single border. Using this, two adjacent table cells will share a border. When this value is applied, the **border-spacing** property does not affect.

initial: It sets the property to its default value.

inherit: It inherits the property from its parent element.

Now, let's understand this [CSS](#) property by using some examples. In the first example, we are using the **separate** value of the **border-collapse** property. In the second example, we are using the **collapse** value of the **border-collapse** property.

Example - Using separate value

With this value, we can use the **border-spacing** property to set the distance between the adjacent table cells.

1. <!DOCTYPE html>
2. <html>
- 3.
4. <head>
5. <title> border-collapse property </title>
6. <style>
7. table{
8. border: 2px solid blue;
9. text-align: center;
10. font-size: 20px;
11. width: 80%;
12. height: 50%;
13. }
14. th{

```
15. border: 5px solid red;
16. background-color: yellow;
17. }
18. td{
19. border: 5px solid violet;
20. background-color: cyan;
21. }
22. #t1 {
23. border-collapse: separate;
24. }
25. </style>
26. </head>
27.
28. <body>
29.
30. <h1> The border-collapse Property </h1>
31. <h2> border-collapse: separate; </h2>
32. <table id = "t1">
33. <tr>
34. <th> First_Name </th>
35. <th> Last_Name </th>
36. <th> Subject </th>
37. <th> Marks </th>
38. </tr>
39. <tr>
40. <td> James </td>
41. <td> Gosling </td>
42. <td> Maths </td>
43. <td> 92 </td>
44. </tr>
45. <tr>
46. <td> Alan </td>
47. <td> Rickman </td>
48. <td> Maths </td>
49. <td> 89 </td>
50. </tr>
51. <tr>
52. <td> Sam </td>
53. <td> Mendes </td>
54. <td> Maths </td>
55. <td> 82 </td>
56. </tr>
57. </table>
58. </body>
59.
60. </html>
```

[Test it Now](#)

Output

Example - Using collapse property

The **border-spacing** and [border-radius properties](#) cannot be used with this value.

```
1. <!DOCTYPE html>
2. <html>
3.
4. <head>
5. <title> border-collapse property </title>
6. <style>
7. table{
8. border: 2px solid blue;
9. text-align: center;
10. font-size: 20px;
11. width: 80%;
12. height: 50%;
13. }
14. th{
15. border: 5px solid red;
16. background-color: yellow;
17. }
18. td{
19. border: 5px solid violet;
20. background-color: cyan;
21. }
22. #t1{
23. border-collapse: collapse;
24. }
25. </style>
26. </head>
27.
28. <body>
29.
30. <h1> The border-collapse Property </h1>
31. <h2> border-collapse: collapse; </h2>
32. <table id = "t1">
33. <tr>
34. <th> First_Name </th>
35. <th> Last_Name </th>
36. <th> Subject </th>
37. <th> Marks </th>
38. </tr>
39. <tr>
40. <td> James </td>
41. <td> Gosling </td>
42. <td> Maths </td>
43. <td> 92 </td>
44. </tr>
45. <tr>
46. <td> Alan </td>
```

```
47. <td> Rickman </td>
48. <td> Maths </td>
49. <td> 89 </td>
50. </tr>
51. <tr>
52. <td> Sam </td>
53. <td> Mendes </td>
54. <td> Maths </td>
55. <td> 82 </td>
56. </tr>
57. </table>
58. </body>
59. </html>
```

[Test it Now](#)

Output

CSS border-spacing property

This CSS property is used to set the distance between the borders of the adjacent cells in the table. It applies only when the **border-collapse** property is set to **separate**. There will not be any space between the borders if the [border-collapse](#) is set to **collapse**.

It can be defined as one or two values for determining the vertical and horizontal spacing.

- When only one value is specified, then it sets both horizontal and vertical spacing.
- When we use the two-value syntax, then the first one is used to set the horizontal spacing (i.e., the space between the adjacent columns), and the second value sets the vertical spacing (i.e., the space between the adjacent rows).

Syntax

1. border-spacing: length | initial | inherit;

Property Values

The values of this [CSS](#) property are defined as follows.

length: This value sets the distance between the borders of the adjacent table cells in px, cm, pt, etc. Negative values are not allowed.

initial: It sets the property to its default value.

inherit: It inherits the property from its parent element.

Let's understand this CSS property by using some examples. In the first example, we are using the single value of the **border-spacing** property, and in the second example, we are

using two values of the **border-spacing** property.

Example

Here, we are using the single value of the **border-spacing** property. The **border-collapse** property is set to **separate**, and the value of the **border-spacing** is set to **45px**.

```
1. <!DOCTYPE html>
2. <html>
3.
4. <head>
5. <title> border-spacing property </title>
6. <style>
7. table{
8. border: 2px solid blue;
9. text-align: center;
10. font-size: 20px;
11. background-color: lightgreen;
12. }
13. th{
14. border: 5px solid red;
15. background-color: yellow;
16. }
17. td{
18. border: 5px solid violet;
19. background-color: cyan;
20. }
21. #space{
22. border-collapse: separate;
23. border-spacing: 45px;
24. }
25. </style>
26. </head>
27.
28. <body>
29.
30. <h1> The border-spacing Property </h1>
31. <h2> border-spacing: 45px; </h2>
32. <table id = "space">
33. <tr>
34. <th> First_Name </th>
35. <th> Last_Name </th>
36. <th> Subject </th>
37. <th> Marks </th>
38. </tr>
39. <tr>
40. <td> James </td>
41. <td> Gosling </td>
42. <td> Maths </td>
43. <td> 92 </td>
44. </tr>
45. <tr>
```

```
46. <td> Alan </td>
47. <td> Rickman </td>
48. <td> Maths </td>
49. <td> 89 </td>
50. </tr>
51. <tr>
52. <td> Sam </td>
53. <td> Mendes </td>
54. <td> Maths </td>
55. <td> 82 </td>
56. </tr>
57. </table>
58. </body>
59.
60. </html>
```

Output

Example

Here, we are using two values of the **border-spacing** property. The **border-collapse** property is set to **separate**, and the value of the **border-spacing** is set to **20pt 1em**. The first value, i.e., **20pt** sets the horizontal spacing, and the value **1em** set the vertical spacing.

```
1. <!DOCTYPE html>
2. <html>
3.
4. <head>
5. <title> border-spacing property </title>
6. <style>
7. table{
8.   border: 2px solid blue;
9.   text-align: center;
10.  font-size: 20px;
11.  background-color: lightgreen;
12. }
13. th{
14.   border: 5px solid red;
15.   background-color: yellow;
16. }
17. td{
18.   border: 5px solid violet;
19.   background-color: cyan;
20. }
21. #space{
22.   border-collapse: separate;
23.   border-spacing: 20pt 1em;
24. }
25. </style>
26. </head>
```

```
27.
28. <body>
29.
30. <h1> The border-spacing Property </h1>
31. <h2> border-spacing: 20pt 1em; </h2>
32. <table id = "space">
33. <tr>
34. <th> First_Name </th>
35. <th> Last_Name </th>
36. <th> Subject </th>
37. <th> Marks </th>
38. </tr>
39. <tr>
40. <td> James </td>
41. <td> Gosling </td>
42. <td> Maths </td>
43. <td> 92 </td>
44. </tr>
45. <tr>
46. <td> Alan </td>
47. <td> Rickman </td>
48. <td> Maths </td>
49. <td> 89 </td>
50. </tr>
51. <tr>
52. <td> Sam </td>
53. <td> Mendes </td>
54. <td> Maths </td>
55. <td> 82 </td>
56. </tr>
57. </table>
58. </body>
59.
60. </html>
```

CSS Display

CSS display is the most important property of CSS which is used to control the layout of the element. It specifies how the element is displayed.

Every element has a default display value according to its nature. Every element on the webpage is a rectangular box and the [CSS](#) property defines the behavior of that rectangular box.

CSS Display default properties

default value	inline
inherited	no
animation supporting	no

version	css1
javascript syntax	object.style.display="none"

Syntax

1. display:value;

CSS display values

There are following CSS display values which are commonly used.

1. display: inline;
2. display: inline-block;
3. display: block;
4. display: run-in;
5. display: none;

1) CSS display inline

The inline element takes the required width only. It doesn't force the line break so the flow of text doesn't break in inline example.

The inline elements are:

-
- <a>
-
- etc.

Let's see an example of CSS display inline.

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. p {
6. display: inline;
7. }
8. </style>
9. </head>
10. <body>
11. <p>Hello Javatpoint.com</p>
12. <p>Java Tutorial.</p>
13. <p>SQL Tutorial.</p>
14. <p>HTML Tutorial.</p>
15. <p>CSS Tutorial.</p>

```
16. </body>
17. </html>
```

Output:

Hello Javatpoint.com Java Tutorial. SQL Tutorial. HTML Tutorial. CSS Tutorial.

2) CSS display inline-block

The CSS display inline-block element is very similar to inline element but the difference is that you are able to set the width and height.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. p {
6.   display: inline-block;
7. }
8. </style>
9. </head>
10. <body>
11. <p>Hello Javatpoint.com</p>
12. <p>Java Tutorial.</p>
13. <p>SQL Tutorial.</p>
14. <p>HTML Tutorial.</p>
15. <p>CSS Tutorial.</p>
16. </body>
17. </html>
```

Output:

Hello Javatpoint.com [Java Tutorial](#). [SQL Tutorial](#). [HTML Tutorial](#). CSS Tutorial.

3) CSS display block

The CSS display block element takes as much as horizontal space as they can. Means the block element takes the full available width. They make a line break before and after them.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. p {
6.   display: block;
```

```
7. }
8. </style>
9. </head>
10. <body>
11. <p>Hello Javatpoint.com</p>
12. <p>Java Tutorial.</p>
13. <p>SQL Tutorial.</p>
14. <p>HTML Tutorial.</p>
15. <p>CSS Tutorial.</p>
16. </body>
17. </html>
```

Output:

Hello Javatpoint.com

Java Tutorial.

SQL Tutorial.

HTML Tutorial.

CSS Tutorial.

4) CSS display run-in

This property doesn't work in [Mozilla Firefox](#). These elements don't produce a specific box by themselves.

- If the run-in box contains a block box, it will be same as block.
- If the block box follows the run-in box, the run-in box becomes the first inline box of the block box.
- If the inline box follows the run-in box, the run-in box becomes a block box.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. p {
6.   display: run-in;
7. }
8. </style>
9. </head>
10. <body>
11. <p>Hello Javatpoint.com</p>
12. <p>Java Tutorial.</p>
13. <p>SQL Tutorial.</p>
```

-
14. <p>HTML Tutorial.</p>
 15. <p>CSS Tutorial.</p>
 16. </body>
 17. </html>

Output:

Hello Javatpoint.com

Java Tutorial.

SQL Tutorial.

HTML Tutorial.

CSS Tutorial.

5) CSS display none

The "none" value totally removes the element from the page. It will not take any space.

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. h1.hidden {
6. display: none;
7. }
8. </style>
9. </head>
10. <body>
11. <h1>This heading is visible.</h1>
12. <h1 class="hidden">This is not visible.</h1>
13. <p>You can see that the hidden heading does not contain any space.</p>
14. </body>
15. </html>

Output:

This heading is visible.

You can see that the hidden heading does not contain any space.

Other CSS display values

Property-value	Description
flex	It is used to display an element as an block-level flex container. It is new in css3.
inline-flex	It is used to display an element as an inline-level flex container. It is new in css3.
inline-table	It displays an element as an inline-level table.
list-Item	It makes the element behave like a element.
table	It makes the element behave like a <table> element.
table-caption	It makes the element behave like a <caption> element.
table-column-group	It makes the element behave like a <colgroup> element.
table-header-group	It makes the element behave like a <thead> element.
table-footer-group	It makes the element behave like a <tfoot> element.
table-row-group	It makes the element behave like a <tbody> element.
table-cell	It makes the element behave like a <td> element.
table-row	It makes the element behave like a <tr> element.
table-column	It makes the element behave like a <col> element.

CSS Cursor

It is used to define the type of mouse cursor when the mouse pointer is on the element. It allows us to specify the cursor type, which will be displayed to the user. When a user hovers on the link, then by default, the cursor transforms into the hand from a pointer.

Let's understand the property values of the cursor.

Values	Usage
alias	It is used to display the indication of the cursor of something that is to be created.
auto	It is the default property in which the browser sets the cursor.
all-scroll	It indicates the scrolling.
col-resize	Using it, the cursor will represent that the column can be horizontally resized.
cell	The cursor will represent that a cell or the collection of cells is selected.
context-menu	It indicates the availability of the context menu.
default	It indicates an arrow, which is the default cursor.
copy	It is used to indicate that something is copied.
crosshair	In it, the cursor changes to the crosshair or the plus sign.
e-resize	It represents the east direction and indicates that the edge of the box is to be shifted towards right.
ew-resize	It represents the east/west direction and indicates a bidirectional resize cursor.
n-resize	It represents the north direction that indicates that the edge of the box is to be

	shifted to up.
ne-resize	It represents the north/east direction and indicates that the edge of the box is to be shifted towards up and right.
move	It indicates that something is to be shifted.
help	It is in the form of a question mark or ballon, which represents that help is available.
None	It is used to indicate that no cursor is rendered for the element.
No-drop	It is used to represent that the dragged item cannot be dropped here.
s-resize	It indicates an edge box is to be moved down. It indicates the south direction.
Row-resize	It is used to indicate that the row can be vertically resized.
Se-resize	It represents the south/east direction, which indicates that an edge box is to be moved down and right.
Sw-resize	It represents south/west direction and indicates that an edge of the box is to be shifted towards down and left.
Wait	It represents an hourglass.
<url>	It indicates the source of the cursor image file.
w-resize	It indicates the west direction and represents that the edge of the box is to be shifted left.
Zoom-in	It is used to indicate that something can be zoomed in.
Zoom-out	It is used to indicate that something can be zoomed out.

The illustration of using the above values of cursor property is given below:

Example

```

1. <html>
2.   <head>
3.   </head>
4.   <style>
5.     body{
6.       background-color: lightblue;
7.       color:green;
8.       text-align: center;
9.       font-size: 20px;
10.    }
11.
12. </style>
13.
14. <body>
15.   <p>Move your mouse over the below words for the cursor change.</p>
16.   <div style = "cursor:alias">alias Value</div>
17.   <div style = "cursor:auto">auto Value</div>
18.   <div style = "cursor:all-scroll">all-scroll value</div>
19.   <div style = "cursor:col-resize">col-resize value</div>
20.   <div style = "cursor:crosshair">Crosshair</div>
21.   <div style = "cursor:default">Default value</div>

```

```
22. <div style = "cursor:copy">copy value</div>
23. <div style = "cursor:pointer">Pointer</div>
24. <div style = "cursor:move">Move</div>
25. <div style = "cursor:e-resize">e-resize</div>
26. <div style = "cursor:ew-resize">ew-resize</div>
27. <div style = "cursor:ne-resize">ne-resize</div>
28. <div style = "cursor:nw-resize">nw-resize</div>
29. <div style = "cursor:n-resize">n-resize</div>
30. <div style = "cursor:se-resize">se-resize</div>
31. <div style = "cursor:sw-resize">sw-resize</div>
32. <div style = "cursor:s-resize">s-resize</div>
33. <div style = "cursor:w-resize">w-resize</div>
34. <div style = "cursor:text">text</div>
35. <div style = "cursor:wait">wait</div>
36. <div style = "cursor:help">help</div>
37. <div style = "cursor:progress">Progress</div>
38. <div style = "cursor:no-drop">no-drop</div>
39. <div style = "cursor:not-allowed">not-allowed</div>
40. <div style = "cursor:vertical-text">vertical-text</div>
41. <div style = "cursor:zoom-in">Zoom-in</div>
42. <div style = "cursor:zoom-out">Zoom-out</div>
43. </body>
44. </html>
```

CSS Buttons

In HTML, we use the button tag to create a button, but by using CSS properties, we can style the buttons. Buttons help us to create user interaction and event processing. They are one of the widely used elements of web pages.

During the form submission, to view or to get some information, we generally use buttons.

Let's see the basic styling in buttons.

Basic styling in Buttons

There are multiple properties available that are used to style the button element. Let's discuss them one by one.

background-color

As we have discussed earlier, this property is used for setting the [background color](#) of the button element.

Syntax

1. element {
2. // background-color style

3. }

Example

```
1. <!DOCTYPE html>
2. <html>
3.
4. <head>
5.   <title>
6.     button background Color
7.   </title>
8.
9.   <style>
10.    body{
11.      text-align: center;
12.    }
13.    button {
14.      color:lightgoldenrodyellow;
15.      font-size: 30px;
16.    }
17.    .b1 {
18.      background-color: red;
19.    }
20.    .b2 {
21.      background-color: blue;
22.    }
23.    .b3 {
24.      background-color: violet;
25.    }
26.  </style>
27. </head>
28.
29. <body>
30.  <h1>The background-color property.</h1>
31.  <button class="b1">Red color button</button>
32.  <button class="b2">Blue color button</button>
33.  <button class="b3">Violet color button</button>
34. </body>
35. </html>
```

border

It is used to set the [border](#) of the button. It is the shorthand property for **border-width**, **border-color**, and **border-style**.

Syntax

```
1. element {
2.   // border style
```

3. }

Example

```
1. <!DOCTYPE html>
2. <html>
3.
4. <head>
5.   <title>
6.     button background Color
7.   </title>
8.
9.   <style>
10.    body{
11.      text-align: center;
12.    }
13.    button {
14.      color:lightgoldenrodyellow;
15.      font-size: 30px;
16.    }
17.    .b1 {
18.      background-color: red;
19.      border:none;
20.    }
21.    .b2 {
22.      background-color: blue;
23.      border:5px brown solid;
24.    }
25.    .b3 {
26.      background-color: yellow;
27.      color:black;
28.      border:5px red groove;
29.    }
30.    .b4{
31.      background-color:orange;
32.      border: 5px red dashed;
33.    }
34.    .b5{
35.      background-color: gray;
36.      border: 5px black dotted;
37.    }
38.    .b6{
39.      background-color: lightblue;
40.      border:5px blue double;
41.    }
42.  </style>
43. </head>
44.
45. <body>
46.   <h1>The border property</h1>
```

```
47. <button class="b1">none</button>
48. <button class="b2">solid</button>
49. <button class="b3">groove</button>
50. <button class="b4">dashed</button>
51. <button class="b5">dotted</button>
52. <button class="b6">double</button>
53.
54. </body>
55. </html>
```

border-radius

It is used to make the rounded corners of the button. It sets the border radius of the button.

Syntax

```
1. element {
2.   // border-radius property
3. }
```

Example

```
1. <!DOCTYPE html>
2. <html>
3.
4. <head>
5.   <title>
6.     button background Color
7.   </title>
8.
9.   <style>
10.    body{
11.      text-align: center;
12.    }
13.    button {
14.      color:lightgoldenrodyellow;
15.      font-size: 30px;
16.    }
17.    .b1 {
18.      background-color: red;
19.      border:none;
20.    }
21.    .b2 {
22.      background-color: blue;
23.      border:5px brown solid;
24.      border-radius: 7px;
25.    }
26.    .b3 {
27.      background-color: yellow;
```

```
28.     color:black;
29.     border:5px red groove;
30.     border-radius: 10px;
31. }
32. .b4{
33.     background-color:orange;
34.     border: 5px red dashed;
35.     border-radius: 20px;
36. }
37. .b5{
38.     background-color: gray;
39.     border: 5px black dotted;
40.     border-radius: 30px;
41. }
42. .b6{
43.     background-color: lightblue;
44.     border:5px blue double;
45.     border-radius: 25px;
46. }
47. </style>
48. </head>
49.
50. <body>
51. <h1>The border-radius property</h1>
52. <h2>Below there is the border name and border-radius</h2>
53. <button class="b1">none</button>
54. <button class="b2">solid 7px</button>
55. <button class="b3">groove 10px</button>
56. <button class="b4">dashed 20px</button>
57. <button class="b5">dotted 30px</button>
58. <button class="b6">double 25px</button>
59.
60. </body>
61. </html>
```

box-shadow

As its name implies, it is used to create the shadow of the button box. It is used to add the shadow to the button. We can also create a shadow during the hover on the button.

Syntax

1. box-shadow: [horizontal offset] [vertical offset] [blur radius]
2. [optional spread radius] [color];

Example

1. <!DOCTYPE html>
2. <html>

```
3.
4. <head>
5.   <title>
6.     button background Color
7.   </title>
8.
9.   <style>
10.  body{
11.    text-align: center;
12.  }
13.  button {
14.    color:lightgoldenrodyellow;
15.    font-size: 30px;
16.  }
17.  .b1{
18.    background-color: lightblue;
19.    border:5px red double;
20.    border-radius: 25px;
21.    color:black;
22.    box-shadow : 0 8px 16px 0 black,
23.      0 6px 20px 0 rgba(0, 0, 0, 0.19);
24.  }
25.  .b2{
26.    background-color: lightblue;
27.    border:5px red dotted;
28.    color:black;
29.    border-radius: 25px;
30.  }
31.  .b2:hover{
32.    box-shadow : 0 8px 16px 0 black,
33.      0 6px 20px 0 rgba(0, 0, 0, 0.19);
34.  }
35.  </style>
36.</head>
37.
38.<body>
39.  <button class="b1">Shadow on button</button>
40.  <button class="b2">Box-shadow on hover</button>
41.</body>
42.</html>
```

padding

It is used to set the button padding.

Syntax

1. element {
2. // padding style

3. }

Let's understand it using an illustration.

Example

```
1. <!DOCTYPE html>
2. <html>
3.
4. <head>
5.   <title>
6.     button background Color
7.   </title>
8.
9.   <style>
10.    body{
11.      text-align: center;
12.    }
13.    button {
14.      color:lightgoldenrodyellow;
15.      font-size: 30px;
16.    }
17.    .b1 {
18.      background-color: red;
19.      border:none;
20.      padding: 16px;
21.    }
22.    .b2 {
23.      background-color: blue;
24.      border:5px brown solid;
25.      padding:15px 30px 25px 40px;
26.    }
27.    .b3 {
28.      background-color: yellow;
29.      color:black;
30.      border:5px red groove;
31.      padding-top:30px;
32.    }
33.    .b4{
34.      background-color:orange;
35.      border: 5px red dashed;
36.      padding-bottom:40px;
37.    }
38.    .b5{
39.      background-color: gray;
40.      border: 5px black dotted;
41.      padding-left: 40px;
42.    }
43.    .b6{
44.      background-color: lightblue;
```

```

45.     border:5px blue double;
46.     padding-right: 40px;;
47.     }
48. </style>
49. </head>
50.
51. <body>
52. <h1>The padding property</h1>
53. <button class="b1">none</button>
54. <button class="b2">solid</button>
55. <button class="b3">groove</button>
56. <button class="b4">dashed</button>
57. <button class="b5">dotted</button>
58. <button class="b6">double</button>
59.
60. </body>
61. </html>

```

CSS Line Height

The **CSS line height property** is used to *define the minimal height of line boxes within the element*. It sets the differences between two lines of your content.

It defines the amount of space above and below inline elements. It allows you to set the height of a line independently from the font size.

CSS line-height values

There are some property values which are used with [CSS](#) line-height property.

value	description
normal	This is a default value. it specifies a normal line height.
number	It specifies a number that is multiplied with the current font size to set the line height.
length	It is used to set the line height in px, pt,cm,etc.
%	It specifies the line height in percent of the current font.
initial	It sets this property to its default value.
inherit	It inherits this property from its parent element.

CSS line-height example

```

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. h3.small {

```

```
6.    line-height: 70%;
7.    }
8.    h3.big {
9.        line-height: 200%;
10.   }
11. </style>
12. </head>
13. <body>
14. <h3>
15. This is a heading with a standard line-height.<br>
16. This is a heading with a standard line-height.<br>
17. The default line height in most browsers is about 110% to 120%.<br>
18. </h3>
19. <h3 class="small">
20. This is a heading with a smaller line-height.<br>
21. This is a heading with a smaller line-height.<br>
22. This is a heading with a smaller line-height.<br>
23. This is a heading with a smaller line-height.<br>
24. </h3>
25. <h3 class="big">
26. This is a heading with a bigger line-height.<br>
27. This is a heading with a bigger line-height.<br>
28. This is a heading with a bigger line-height.<br>
29. This is a heading with a bigger line-height.<br>
30. </h3>
31. </body>
32. </html>
```

CSS Margin

CSS Margin property is used to define the space around elements. It is completely transparent and doesn't have any background color. It clears an area around the element.

Top, bottom, left and right margin can be changed independently using separate properties. You can also change all properties at once by using shorthand margin property.

There are following [CSS](#) margin properties:

CSS Margin Properties

Property	Description
margin	This property is used to set all the properties in one declaration.
margin-left	it is used to set left margin of an element.
margin-right	It is used to set right margin of an element.

margin-top It is used to set top margin of an element.

margin-bottom It is used to set bottom margin of an element.

CSS Margin Values

These are some possible values for margin property.

Value	Description
auto	This is used to let the browser calculate a margin.
length	It is used to specify a margin pt, px, cm, etc. its default value is 0px.
%	It is used to define a margin in percent of the width of containing element.
inherit	It is used to inherit margin from parent element.

Note: You can also use negative values to overlap content.

CSS margin Example

You can define different margin for different sides for an element.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. p {
6.   background-color: pink;
7. }
8. p.ex {
9.   margin-top: 50px;
10.  margin-bottom: 50px;
11.  margin-right: 100px;
12.  margin-left: 100px;
13. }
14. </style>
15. </head>
16. <body>
17. <p>This paragraph is not displayed with specified margin. </p>
18. <p class="ex">This paragraph is displayed with specified margin.</p>
19. </body>
20. </html>
```

Output:

This paragraph is not displayed with specified margin.

This paragraph is displayed with specified margin.

Margin: Shorthand Property

CSS shorthand property is used to shorten the code. It specifies all the margin properties in one property.

There are four types to specify the margin property. You can use one of them.

1. `margin: 50px 100px 150px 200px;`
 2. `margin: 50px 100px 150px;`
 3. `margin: 50px 100px;`
 4. `margin 50px;`
-

1) `margin: 50px 100px 150px 200px;`

It identifies that:

top margin value is 50px

right margin value is 100px

bottom margin value is 150px

left margin value is 200px

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `p {`
6. `background-color: pink;`
7. `}`
8. `p.ex {`
9. `margin: 50px 100px 150px 200px;`
10. `}`
11. `</style>`
12. `</head>`
13. `<body>`
14. `<p>This paragraph is not displayed with specified margin. </p>`
15. `<p class="ex">This paragraph is displayed with specified margin.</p>`
16. `</body>`

17. </html>

Output:

This paragraph is not displayed with specified margin.

This paragraph is displayed with specified margin.

2) margin: 50px 100px 150px;

It identifies that:

top margin value is 50px

left and right margin values are 100px

bottom margin value is 150px

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. p {
6.     background-color: pink;
7. }
8. p.ex {
9.     margin: 50px 100px 150px;
10. }
11. </style>
12. </head>
13. <body>
14. <p>This paragraph is not displayed with specified margin. </p>
15. <p class="ex">This paragraph is displayed with specified margin.</p>
16. </body>
17. </html>
```

Output:

This paragraph is not displayed with specified margin.

This paragraph is displayed with specified margin.

3) margin: 50px 100px;

It identifies that:

top and bottom margin values are 50px

left and right margin values are 100px

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5.   p {
6.     background-color: pink;
7.   }
8.   p.ex {
9.     margin: 50px 100px;
10.  }
11. </style>
12. </head>
13. <body>
14. <p>This paragraph is not displayed with specified margin. </p>
15. <p class="ex">This paragraph is displayed with specified margin.</p>
16. </body>
17. </html>
```

Output:

This paragraph is not displayed with specified margin.

This paragraph is displayed with specified margin.

UNIT-II

Frontend Development

Javascript basics

Learn JavaScript Tutorial



JavaScript

Our **JavaScript Tutorial** is designed for beginners and professionals both. JavaScript is used to create client-side dynamic pages.

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

JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

What is JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.

Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

Features of JavaScript

There are following features of JavaScript:

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

History of JavaScript

In 1993, **Mosaic**, the first popular web browser, came into existence. In the **year 1994**, **Netscape** was founded by **Marc Andreessen**. He realized that the web needed to become more dynamic. Thus, a 'glue language' was believed to be provided to HTML to make web designing easy for designers and part-time programmers. Consequently, in 1995, the company recruited **Brendan Eich** intending to implement and embed Scheme programming language to the browser. But, before Brendan could start, the company merged with **Sun Microsystems** for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms. Now, two languages were there: Java and the scripting language. Further, Netscape decided to give a similar name to the scripting language as Java's. It led to 'Javascript'. Finally, in May 1995, Marc Andreessen coined the first code of Javascript named '**Mocha**'. Later, the marketing team replaced the name with '**LiveScript**'. But, due to trademark reasons and certain other reasons, in December 1995, the language was finally renamed to 'JavaScript'. From then, JavaScript came into existence.

Application of JavaScript

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

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

JavaScript Example

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

JavaScript Example

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

Let's create the first JavaScript example.

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

[Test it Now](#)

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

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

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

3 Places to put JavaScript code

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

1) JavaScript Example : code between the body tag

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

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

2) JavaScript Example : code between the head tag

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

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

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

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

External JavaScript file

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

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

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

Let's create an external [JavaScript](#) file that prints Hello Javatpoint in a alert dialog box.

message.js

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

Let's include the JavaScript file into [html](#) page. It calls the [JavaScript function](#) on button click.

index.html

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

Advantages of External JavaScript

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

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

Disadvantages of External JavaScript

There are the following disadvantages of external files:

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

JavaScript Comment

-
1. [JavaScript comments](#)
 2. [Advantage of javaScript comments](#)
 3. [Single-line and Multi-line comments](#)

The **JavaScript comments** are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

1. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.
2. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

Types of JavaScript Comments

There are two types of comments in JavaScript.

1. Single-line Comment
2. Multi-line Comment

JavaScript Single line Comment

It is represented by double forward slashes (//). It can be used before and after the statement.

Let's see the example of single-line comment i.e. added before the statement.

1. `<script>`
2. `// It is single line comment`
3. `document.write("hello javascript");`
4. `</script>`

[Test it Now](#)

Let's see the example of single-line comment i.e. added after the statement.

1. `<script>`
2. `var a=10;`
3. `var b=20;`
4. `var c=a+b;//It adds values of a and b variable`

-
5. `document.write(c);`//prints sum of 10 and 20
 6. `</script>`
-

JavaScript Multi line Comment

It can be used to add single as well as multi line comments. So, it is more convenient.

It is represented by forward slash with asterisk then asterisk with forward slash. For example:

1. `/* your code here */`

It can be used before, after and middle of the statement.

1. `<script>`
2. `/* It is multi line comment.`
3. `It will not be displayed */`
4. `document.write("example of javascript multiline comment");`
5. `</script>`

JavaScript Variable

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

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

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

Correct JavaScript variables

1. `var x = 10;`
 2. `var _value="sonoo";`
-

Incorrect JavaScript variables

1. `var 123=30;`
 2. `var *aa=320;`
-

Example of JavaScript variable

Let's see a simple example of JavaScript variable.

1. `<script>`
2. `var x = 10;`
3. `var y = 20;`
4. `var z=x+y;`
5. `document.write(z);`
6. `</script>`

Output of the above example

30

JavaScript local variable

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

1. `<script>`
2. `function abc(){`
3. `var x=10;//local variable`
4. `}`
5. `</script>`

Or,

1. `<script>`
 2. `If(10<13){`
 3. `var y=20;//JavaScript local variable`
 4. `}`
 5. `</script>`
-

JavaScript global variable

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

1. `<script>`
2. `var data=200;//global variable`
3. `function a(){`
4. `document.writeln(data);`
5. `}`

```
6. function b(){
7. document.writeln(data);
8. }
9. a();//calling JavaScript function
10. b();
11. </script>
```

JavaScript Global Variable

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

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

```
1. <script>
2. var value=50;//global variable
3. function a(){
4. alert(value);
5. }
6. function b(){
7. alert(value);
8. }
9. </script>
```

Declaring JavaScript global variable within function

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

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

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

```
1. function m(){
2. window.value=100;//declaring global variable by window object
3. }
4. function n(){
5. alert(window.value);//accessing global variable from other function
6. }
```

Internals of global variable in JavaScript

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

```
1. var value=50;
```

-
2. `function a(){`
 3. `alert(window.value);`//accessing global variable
 4. `}`

OOPS ASPECTS IN JAVASCRIPT

What Is Object-oriented Programming?

Object-oriented Programming treats data as a crucial element in program development and doesn't allow it to flow freely around the system. It ties data more securely to the function that operates on it and protects it from accidental modification from an outside function. OOP breaks down a problem into several entities called objects and builds data and functions around these objects.

Basic concepts of Object-oriented Programming

Objects

[Objects](#) are the basic run-time bodies in an object-oriented framework. They may represent a place, a person, an account, a table of data, or anything that the program needs to handle. Objects can also represent user-defined data such as vectors, time, and lists.

Consider two objects, “customer” and “account” in a program. The customer object may send a message requesting the bank balance.

Classes

We know that objects hold the data and the functions to manipulate the data. However, the two can be bound together in a user-defined data type with the help of classes. Any number of objects can be created in a class. Each object is associated with the data of type class. A class is therefore a collection of objects of similar types.

For example, consider the class “Fruits”. We can create multiple objects for this class -

```
Fruit Mango;
```

This will create an object mango belonging to the class fruit.

Encapsulation

Encapsulation is the wrapping up/binding of data and function into a single unit called class. Data encapsulation is the most prominent feature of a class wherein the data is not accessible to the outside world, and only those functions wrapped inside the class can access it. These functions serve as the interface between the object's data and the program.

Inheritance

The phenomenon where objects of one class acquire the properties of objects of another class is called Inheritance. It supports the concept of hierarchical classification. Consider the object “car” that falls in the class “Vehicles” and “Light Weight Vehicles”.

In OOP, the concept of inheritance ensures reusability. This means that additional features can be added to an existing class without modifying it. This is made possible by deriving a new class from the existing one.

OOP Concepts in JavaScript

Now that you are familiar with OOP concepts, this section will show you how JavaScript implements them.

Creating Objects in JavaScript

- We can create an object using the string literal in JavaScript.

```
var student = {  
    name: "pp",  
    age: 21,  
    studies: "Computer Science",  
};  
  
document.getElementById("demo").innerHTML = student.name + " of the age " +  
student.age + " studies " + student.studies;
```

- Creating objects using the new keyword.

```
var student = new Object();  
  
student.name = "pp",  
student.age=21,  
student.studies = "Computer Science";  
  
document.getElementById("demo").innerHTML = student.name + " of the age " +  
student.age + " studies " + student.studies;
```

- Creating an object using the object constructor.

```
function stud(name, age, studies){  
  
    this.name = name;
```

```
this.age = age;

this.studies = studies;

}

var student = stud("Chris", 21, "Computer Science");

document.getElementById("demo").innerHTML = student.name + " of the age " +
student.age + " studies " + student.studies;
```

Class Implementation in JavaScript

JavaScript uses the ES6 standard to define classes. Consider the following example.

```
class Cars {

  constructor(name, maker, price) {

    this.name = name;

    this.maker = maker;

    this.price = price;

  }

  getDetails(){

    return (`The name of the car is ${this.name}.`)

  }

}

let car1 = new Cars('Rolls Royce Ghost', 'Rolls Royce', '$315K');

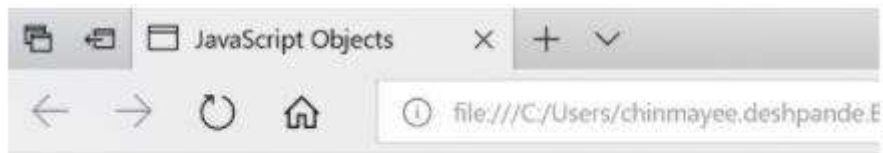
let car2 = new Cars('Mercedes AMG One', 'Mercedes', '$2700K');

console.log(car1.name);

console.log(car2.maker);

console.log(car1.getDetails());
```

The output of the above code is



Creating JavaScript Objects

Chris of the age 21 studies Computer Science

Encapsulation in JavaScript

Encapsulation includes wrapping the property and the function within a single unit. Consider the following example:

```
class Emp_details{  
    constructor(name,id){  
        this.name = name;  
        this.id = id;  
    }  
    add_Address(add){  
        this.add = add;  
    }  
    getDetails(){  
        console.log(`Employee Name: ${this.name}, Address: ${this.add}`);  
    }  
}  
  
let person1 = new Emp_details('Anand',27);  
  
person1.add_Address('Bangalore');  
  
person1.getDetails();
```

Here, the class holds the data variables name and id along with the functions add_Address and getDetails. All are encapsulated within the class Emp_details.

Memory Management in JavaScript

Memory management is an essential task when writing a good and effective program in some programming languages. This article will help you to understand different concepts of memory management in JavaScript. In low-level languages like C and C++, programmers should care about the usage of memory in some manual fashion. On the other hand, Javascript automatically allocates memory when objects are created into the environment and also it cleans the memory when an object is destroyed. JavaScript can manage all of these on its own but this does not imply that the developers do not need to worry about the memory management in JavaScript.

Memory management in any programming language involves three important phases, termed as memory life-cycle –

- Allocating the memory which is required in our program.
- Utilize the allocated memory unit.
- After completion, clear the memory block.

Different Strategies to Allocate Memory in JavaScript

Allocating by value initialization

In JavaScript, we do not need to care about allocating memory for simple variables. We can directly assign values to some variables and it will allocate necessary memory on its own.

Syntax

```
var variable1 = <value>
var variable2 = <value>
```

Example

For simple allocation by values, see the following example.

Source Code

```
<head>
<title>HTML Console</title>
</head>
<body>
<h3> Output Console </h3>
<p> Output:</p>
<div id="output">
</div>
<div id="opError" style="color : #ff0000">
</div>
<script>
var content ="
var error ="
varopDiv=document.querySelector('#output')
varopErrDiv=document.querySelector('#opError')

// actual javascript code
try{
```

```

var number =52;
varst='my_string';
var student ={
    name:'Smith',
    roll:5,
    age:23,
};
vararr=[15,null,'another_string'];
    content += "Allocated memory for number: "+JSON.stringify(number)+'<br>'
    content += "Allocated memory for string: "+JSON.stringify(st)+'<br>'
    content += "Allocated memory for student: "+JSON.stringify(student)+'<br>'
    content += "Allocated memory for array: "+JSON.stringify(arr)+'<br>'
}catch(err){
    error += err
}finally{

// display on output console
opDiv.innerHTML= content
opErrDiv.innerHTML= error
}
</script>
</body>
</html>

```

From the above example, it is clear that numbers and strings are single values, and allocation is also simple. But for objects and arrays, JavaScript can also easily allocate the memory based on their values.

Allocating by Function Call

Like variable value assignment, we can also create some memory blocks by calling some functions. For example, when a function returns a separate object it will automatically assign a new memory block to the system.

Syntax

Memory_reference = <function call which returns any value>

Examples

The following example uses a function that works on an HTML document. So this program will run on a browser or HTML editor.

Source Code

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8"/>
</head>
<body>
<script>
var e =document.createElement('div');
e.innerHTML="<h1> Header from JavaScript </h1>"
document.body.appendChild(e);
</script>
</body>
</html>

```

In this example, the JavaScript code is present inside the <script> tag in HTML. Please notice, in this case, initially, the document does not have any <div> block inside <body>. The JavaScript creates a new component by calling createElement(), and then a new div block is created. This block allocates the memory but only when a function is called. After that, the new component is added as a child of the body tag to use this inside the HTML document.

Using previously Allocated Memory in JavaScript

Using previously allocated memory is just reading or writing values from some variables which are assigned previously. We can update its existing value with some other values. See the following example for a better understanding—

Example

Initially allocating memory for a variable, then reading the value from it. Writing a new value and again reading from it.

Source Code

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Console</title>
</head>
<body>
<h3> Output Console </h3>
<p> Output:</p>
<div id="output">
</div>
<div id="opError" style="color : #ff0000">
</div>
<script>
var content = "
var error = "
opDiv=document.querySelector('#output')
varopErrDiv=document.querySelector('#opError')

// actual javascript code
try{
var a =52;// allocate memory
    content += "Reading value of variable a: "+JSON.stringify(a)+'<br>'
    a =100
    content += "Reading value of variable a: "+JSON.stringify(a)+'<br>'
}
catch(err){
    error += err
}
finally{

// display on output console
opDiv.innerHTML= content
opErrDiv.innerHTML= error
}
</script>
</body>
```

</html>

Deallocating memory blocks in JavaScript

When our purpose is served, we can remove the allocated memory block. In some low-level languages, this is a necessary step, otherwise, it may occupy memory spaces over time and the total system may crash. JavaScript also has native support of Garbage Collector, which cleans unnecessary memory blocks and cleans up the memory. But sometimes the compiler cannot understand whether a block will be used in later cases or not. In such cases, the Garbage Collector does not clean up that memory. To manually remove allocated locations, we can use the 'delete' keyword before the variable name.

Syntax

delete <variable_name>

The variable must be allocated beforehand, otherwise, it will raise an error while trying to delete that variable. Let us see one example to understand this concept clearly.

Example

Source Code

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Console</title>
</head>
<body>
<h3> Output Console </h3>
<p> Output:</p>
<div id="output">
</div>
<div id="opError" style="color : #ff0000">
</div>
<script>
var content = "
var error = "
var opDiv=document.querySelector('#output')
var opErrDiv=document.querySelector('#opError')

// actual javascript code
try{
    a="a simple variable";// allocate memory
    content += "Reading value of variable a: "+JSON.stringify(a)+'<br>'
    delete a
    content += "Reading value of variable a: "+JSON.stringify(a)+'<br>'
}
catch(err){
    error += err
}
finally{

// display on output console
opDiv.innerHTML= content
opErrDiv.innerHTML= error
}
</script>
```

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

Note – The ‘delete’ keyword will only work when the variable is allocated directly (without using the var or let keyword).

Conclusion

Working with any programming language, the programmer should know the overall concept in depth. Memory management is one of the concerning issues, in which developers should properly manage the memory otherwise it will occupy unnecessary memory blocks and create major problems in the environment. JavaScript provides an additional garbage collector tool that automatically cleans the unused memory blocks. However, we can also deallocate memory by using the ‘delete’ keyword just before the variable name

AJAX for data exchange with server jQuery Framework

Short Description of AJAX

Ajax is only a name given to a set of tools that were previously existing.

The main part is XMLHttpRequest, a server-side object usable in JavaScript, that was implemented in Internet Explorer since the 4.0 version.

To get data on the server, XMLHttpRequest provides two methods:

1. open: Creates a connection
2. send: Sends a request to the server

Data furnished by the server will be found in the attributes of the XMLHttpRequest object:

1. responseXml for an XML file, or
2. responseText for a plain text

Take note that a new XMLHttpRequest object has to be created for each new data request.

We have to wait for the data to be available to process it, and in this purpose, the state of availability of data is given by the readyState attribute of XMLHttpRequest.

Attributes of XMLHttpRequest Class

1. *readyState*: The code successively changes value from 0 to 4
- 0: Not initialized

- 1: Connection established
- 2: Request received
- 3: Answer in process
- 4: Finished

2. *status*: 200 is OK

404 if the page is not found

3. *responseText*: Holds loaded data as a string of characters.
4. *responseXml*: Holds an XML loaded file, DOM's method allows to extract data.
5. *onreadystatechange*: Property that takes a function as value that is invoked when the readystatechangeevent is dispatched.

Methods of XMLHttpRequest Class

1. *open(mode, url,boolean)* : *mode*: type of request, GET or POST

url: the location of the file, with a path

boolean: true (asynchronous) / false (synchronous)

optionally, a login and a password may be added to arguments

2. *send("string")*: string: POST data, null for a GET command
3. *abort()* : Cancels the current HTTP request
4. *getAllResponseHeaders()*: Retrieves the values of all the HTTP headers
5. *getResponseHeader(string)*: Retrieves the value of an HTTP header from the response body
string: name of http header
6. *setRequestHeader(name,value)*: Adds a new http header in the request
name: name/identifier of the header
value: value of the header

Using the Code

Here is a simple function 'AjaxRequest' which is implemented to perform the AJAX requests.

JavaScript

Shrink ▲

```
function AjaxRequest(ReadyHandler,URL,Method,Params,QueryString,HttpHeaders) {  
    if (URL == null) { alert("Request URL is Empty"); }  
    else {  
  
        if (window.XMLHttpRequest) { // code for IE7+, Firefox, Chrome, Opera, Safari  
            xmlhttp = new XMLHttpRequest();  
        }  
        else { // code for IE6, IE5  
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");  
        }  
  
        //An anonymous function is assigned to the event indicator.  
        xmlhttp.onreadystatechange = function() {  
  
            //200 status means ok, otherwise some error code is returned, 404 for example
```

```

//The 4 state means for the response is ready and sent by the server.
if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    ResponseText = xmlhttp.responseText; //get text data in the response
    ResponseXML = xmlhttp.responseXML; //get xml data in the response
    ResponseHeaderJSON = xmlhttp.getResponseHeader
        ("CustomHeaderJSON"); // Extract Data in http header
    ResponseHeaders = xmlhttp.getAllResponseHeaders(); //Get a string
        //containing all http headers returned by server

    // Make all the results available in the ReadyHandler via prototyping.
    ReadyHandler.prototype.ResponseText = ResponseText;
    ReadyHandler.prototype.ResponseHeaderJSON = ResponseHeaderJSON;
    ReadyHandler.prototype.ResponseXML = ResponseXML;
    ReadyHandler.prototype.ResponseHeaders = ResponseHeaders;
    // Execute function passed as ReadyHandler
    ReadyHandler();
    }
}

//If querystring is provided Attach it to the url
    if (QueryString != "") {
varQueryStringData = "";
for (QueryStringAttribute in QueryString) {
    QueryStringData = QueryStringAttribute + "=" +
        QueryString[QueryStringAttribute] + "&" + QueryStringData;
    }
QueryStringData = QueryStringData.substring(0,
    QueryStringData.lastIndexOf('&'));
    URL = URL + "?" + escape(QueryStringData); //Here is where the
        //query string is attached to the request url.
    }

//POST or GET URL of the script to execute.true for asynchronous
//(false for synchronous).
xmlhttp.open(Method, URL, true);
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
if (HttpHeaders != "") {
    varHttpHeadersData = "";
    for (HttpHeaderName in HttpHeaders) {
        xmlhttp.setRequestHeader(HttpHeaderName,
            HttpHeaders[HttpHeaderName]); // Here the custom headers are added
    }
}

    //Post data provided then assemble it into single string to be posted to server
    if (Params != "") {
varParamsData = "";
for (ParamName in Params) {
    ParamsData = ParamName + "=" + Params[ParamName] + "&" + ParamsData;
    }
ParamsData = ParamsData.substring(0, ParamsData.lastIndexOf('&'));
    }

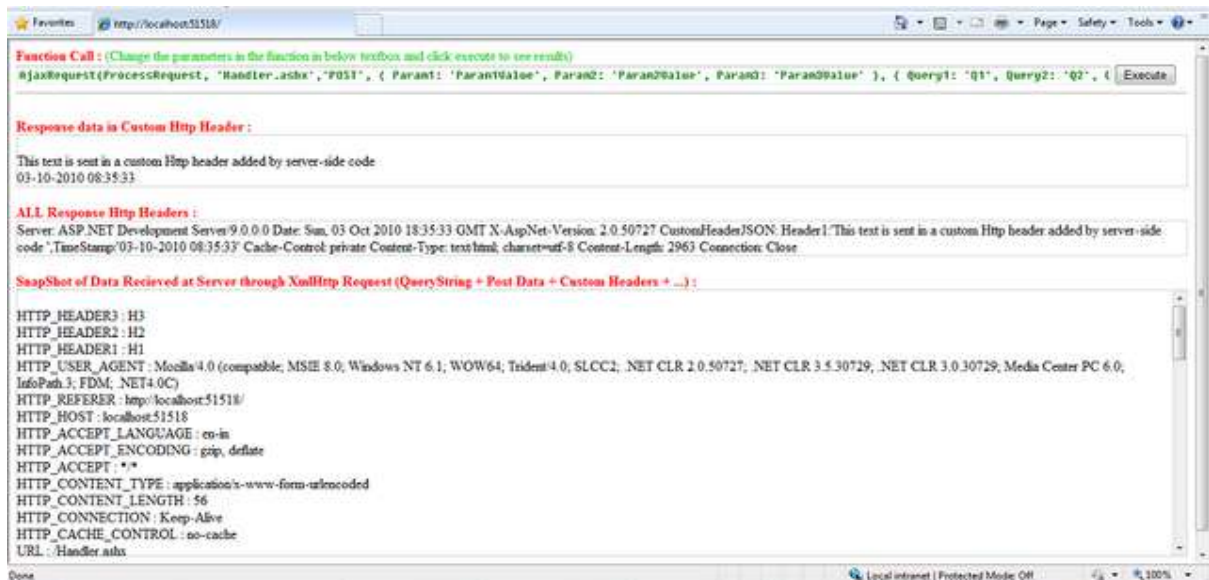
xmlhttp.send(ParamsData); //Send the request with the post data
    }
}

```

[You can find the complete implementation with sufficient comments in the source code.]

It can give a more clear idea of using AJAX in your applications.

In the demo application, you can test the 'AjaxRequest' function by changing the parameters that are passed to it.



Actually all the code that is typed in the text box is executed as JavaScript code on click of 'Execute' button. This is done using the eval() function.

JavaScript

```
FunctionCall = document.getElementById('FunctionCode').value;  
eval(FunctionCall);
```

Function Usage

JavaScript

```
function AjaxRequest(ReadyHandler, URL, Method, Params, QueryString, HttpHeaders)
```

Description

-> *ReadyHandler*: Function to be called after successful completion of the AJAX request

Note: On successful completion of the request, the result of the request will be available in the function passed as *ReadyHandler*.

The result of request will be in 4 variables, namely:

- ResponseText: Text response from server
- ResponseHeaderJSON: Custom HTTP Header String value

This header string may contain a single string value or a you can also use a JSON format for multiple values which then can be parsed in *ReadyHandler* (as shown in the example).

- ResponseHeaders: String containing all Response HTTP Headers
- ResponseXML: XML response from server (XML object available only when the Response contains a proper XML)

->URL: This parameter takes the URL to which the request is to be sent

->Method: Method of request "GET"/"POST"

->Params: POST data to be sent to server. Expects a JSON formatted name value pairs

->QueryString: Data to be sent to the server as QueryString. Expects a JSON formatted name value pairs

->HttpHeaders: Data to be sent as HTTP Headers. Expects JSON formatted name value pairs

Note: While sending the data in headers, you have to take care only ASCII characters where charCode ranging from 32 to 126 are sent or you may get unexpected results. See RFC documentation for HTTP.

The ReadyHandler can contain the code which will dynamically change the contents of the webpage based on the response data.

For example, in the demo application, I have used 'ProcessRequest()' as the Ready handler which sets the response in the respective <Div>.

JavaScript

```
function ProcessRequest() {  
  
    // // Assign the content to the form  
    document.getElementById('ResponseTextDiv').innerHTML = ResponseText;  
  
    document.getElementById('ResponseXMLDiv').innerHTML = ResponseHeaders;  
    eval("var CustomHeaders = { " + ResponseHeaderJSON + "};");  
    var header;  
    var allHeaders = "<br/>";  
    if (CustomHeaders != "") {  
        for (header in CustomHeaders) {  
            allHeaders = allHeaders + CustomHeaders[header] + "<br/>"  
        }  
    }  
    document.getElementById('ResponseHeadersDiv').innerHTML = allHeaders;  
}
```

Example:

JavaScript

```
AjaxRequest(ProcessRequest, 'Handler.ashx', 'POST',  
    { Param1: 'Param1Value', Param2: 'Param2Value', Param3: 'Param3Value' },  
    { Query1: 'Q1', Query2: 'Q2', Query3: 'Q3' },  
    { Header1: 'H1', Header2: 'H2', Header3: 'H3' }  
);
```

For handling the client request, I have implemented a simple **Generic Handler (.ashx)**.

You can access all the data (query string + Post Data + HTTP Headers) that is sent by the client browser in AJAX request.

In the Generic handler, the data is accessible via the `context.Requestobject`.

Though you can access all the data together in `context.Request.Params[]`, you can access the data separately as follows:

- Query String: `context.Request.QueryString[[index/string]]`
- Http Headers: `context.Request.Headers[[index/string]]`

In the example application, what I have done is just echo back the data which is received in the request along with a custom HTTP header added.

JavaScript

```
foreach(string Param in context.Request.Params)
{
    ParamsData = "<br/>" + Param + " : " +
        context.Request.Params[Param].ToString() + ParamsData;
}
context.Response.Write(ParamsData);
```

The above lines capture the data in the request and send it back in the response.

For adding an extra custom HTTP header in response:

C#

```
context.Response.AddHeader("CustomHeaderJSON", CustomHeaderJSON);
```

As you see, the `context.Responseobject` is used to assemble the response which is to be sent back to the browser.

Different methods of *context.Response* can be used to do this.

'*CustomHeaderJSON*' can contain a string , but I have created a JSON format string for supporting multiple values. The values are then parsed at client side using JavaScript.

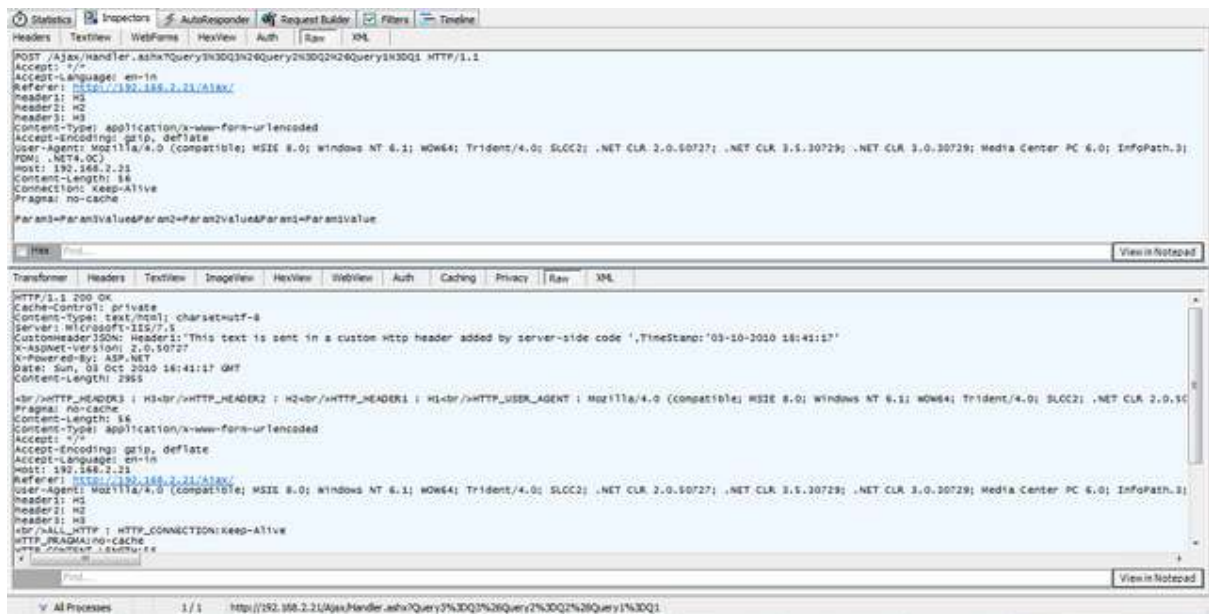
I have just used string concatenate for creating it, but you can also use different JSON parsers/Encoders available at <http://www.json.org/>.

You can also use JSON strings to exchange data through AJAX. It is sometimes better to use JSON than XML. Using JSON results in less bytes transferred than XML.

Points of Interest

This is a basic implementation of AJAX and the function can be tuned and modified according to needs and reconfigurability.

Here is how the request and response looks like [HTTP request in Fiddler]:



jQuery Events

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

These are some examples of events.

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

These events can be categorized on the basis their types:

Mouse Events

- click
- dblclick
- mouseenter
- mouseleave

Keyboard Events

- keyup
- keydown
- keypress

Form Events

- submit
- change
- blur
- focus

Document/Window Events

- load
- unload
- scroll
- resize

Note: A term "fires" is generally used with events. For example: The click event fires in the moment you press a key.

Syntax for event methods

Most of the DOM events have an equivalent jQuery method. To assign a click events to all paragraph on a page, do this:

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

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

UNIT – III

REACT JS

React Introduction

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library responsible only for the view layer of the application. It was created by **Jordan Walke**, who was a software engineer at **Facebook**. It was initially developed and maintained by Facebook and was later used in its products like **WhatsApp&Instagram**. Facebook developed ReactJS in **2011** in its newsfeed section, but it was released to the public in the month of **May 2013**.

Today, most of the websites are built using MVC (model view controller) architecture. In MVC architecture, React is the 'V' which stands for view, whereas the architecture is provided by the Redux or Flux.

A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of HTML code. The components are the heart of all React applications. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses virtual DOM based mechanism to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.

To create React app, we write React components that correspond to various elements. We organize these components inside higher level components which define the application structure. For example, we take a form that consists of many elements like input fields, labels, or buttons. We can write each element of the form as React components, and then we combine it into a higher-level component, i.e., the form component itself. The form components would specify the structure of the form along with elements inside of it.

Why learn ReactJS?

Today, many JavaScript frameworks are available in the market(like angular, node), but still, React came into the market and gained popularity amongst them. The previous frameworks follow the traditional data flow structure, which uses the DOM (Document Object Model). DOM is an object which is created by the browser each time a web page is loaded. It dynamically adds or removes the data at the back end and when any modifications were done, then each time a new DOM is created for the same page. This repeated creation of DOM makes unnecessary memory wastage and reduces the performance of the application.

Therefore, a new technology ReactJS framework invented which remove this drawback. ReactJS allows you to divide your entire application into various components. ReactJS still used the same traditional data flow, but it is not directly operating on the browser's Document Object Model (DOM) immediately; instead, it operates on a virtual DOM. It means rather than manipulating the document in a browser after changes to our data, it resolves changes on a DOM built and run entirely in memory. After the virtual DOM has been updated, React determines what changes made to the actual browser's DOM. The React Virtual DOM exists entirely in memory and is a representation of the web browser's DOM. Due to this, when we write a React component, we did not write directly to the DOM; instead, we are writing virtual components that react will turn into the DOM.

React Router and Single Page Applications

Preparing the React App

Installing the create-react-app Package

If you've ever had the chance to try React, you've probably heard about the **create-react-app** package, which makes it super easy to start with a React development environment.

In this tutorial, we will use this package to initiate our React app.

So, first of all, make sure you have Node.js installed on your computer. It will also install

npm for you.

In your terminal, run `npm install -g create-react-app`. This will globally install **create-react-app** on your computer.

Once it is done, you can verify whether it is there by typing `create-react-app -V`.

Creating the React Project

Now it's time to build our React project. Just run `create-react-app multi-page-app`. You can, of course, replace `multi-page-app` with anything you want.

Now, **create-react-app** will create a folder named **multi-page-app**. Just type `cd multi-page-app` to change directory, and now run `npm start` to initialize a local server.

That's all. You have a React app running on your local server.

Now it's time to clean the default files and prepare our application.

In your `src` folder, delete everything but `App.js` and `index.js`. Then open `index.js` and replace the content with the code below.

```
import React from 'react';
```

```
import ReactDOM from 'react-dom';
```

```
import App from './App';
```

```
ReactDOM.render(<App />, document.getElementById('root'));
```

I basically deleted the `registerServiceWorker` related lines and also the `import './index.css';` line.

Also, replace your `App.js` file with the code below.

```
import React, { Component } from 'react';
```

```
class App extends Component {
```

```
  render() {
```

```
return (  
  
  <div className="App">  
  
    </div>  
  
    );  
  
  }  
  
}  
  
export default App;
```

Now we will install the required modules.

In your terminal, type the following commands to install the **react-router** and **react-transition-group** modules respectively.

```
npm install react-router-dom --save
```

```
npm install react-transition-group@1.x --save
```

After installing the packages, you can check the package.json file inside your main project directory to verify that the modules are included under **dependencies**.

Router Components

There are basically two different router options: **HashRouter** and **BrowserRouter**.

As the name implies, **HashRouter** uses hashes to keep track of your links, and it is suitable for static servers. On the other hand, if you have a dynamic server, it is a better option to use **BrowserRouter**, considering the fact that your URLs will be prettier.

Once you decide which one you should use, just go ahead and add the component to your index.js file.

```
import { HashRouter } from 'react-router-dom'
```

The next thing is to wrap our <App> component with the router component.

So your final index.js file should look like this:

```
import React from 'react';

import ReactDOM from 'react-dom';

import { HashRouter } from 'react-router-dom'

import App from './App';

ReactDOM.render(<HashRouter><App/></HashRouter>, document.getElementById('root'));
```

If you're using a dynamic server and prefer to use **BrowserRouter**, the only difference would be importing the **BrowserRouter** and using it to wrap the <App> component.

By wrapping our <App> component, we are serving the **history** object to our application, and thus other react-router components can communicate with each other.

Inside <App/> Component

Inside our <App> component, we will have two components named <Menu> and <Content>. As the names imply, they will hold the navigation menu and displayed content respectively.

Create a folder named "**components**" in your src directory, and then create the Menu.js and Content.js files.

Menu.js

Let's fill in our Menu.js component.

It will be a stateless functional component since we don't need states and life-cycle hooks.

```
import React from 'react'

const Menu = () =>{

  return(
```

```
<ul>
```

```
<li>Home</li>
```

```
<li>Works</li>
```

```
<li>About</li>
```

```
</ul>
```

```
)
```

```
}
```

```
export default Menu
```

Here we have a `` tag with `` tags, which will be our links.

Now add the following line to your **Menu** component.

```
import { Link } from 'react-router-dom'
```

And then wrap the content of the `` tags with the `<Link>` component.

The `<Link>` component is essentially a **react-router** component acting like an `<a>` tag, but it does not reload your page with a new target link.

Also, if you style your `a` tag in CSS, you will notice that the `<Link>` component gets the same styling.

Note that there is a more advanced version of the `<Link>` component, which is `<NavLink>`. This offers you extra features so that you can style the active links.

Now we need to define where each link will navigate. For this purpose, the `<Link>` component has a `to` prop.

```
import React from 'react'
```

```
import { Link } from 'react-router-dom'

const Menu = () => {

  return(

    <ul>

      <li><Link to="/">Home</Link></li>

      <li><Link to="/works">Works</Link></li>

      <li><Link to="/about">About</Link></li>

    </ul>

  )

}

export default Menu
```

Content.js

Inside our <Content> component, we will define the **Routes** to match the **Links**.

We need the Switch and Route components from **react-router-dom**. So, first of all, import them.

```
import { Switch, Route } from 'react-router-dom'
```

Second of all, import the components that we want to route to. These are the Home, Works and About components for our example. Assuming you have already created those components inside the **components** folder, we also need to import them.

```
import Home from './Home'
```

```
import Works from './Works'
```

```
import About from './About'
```

Those components can be anything. I just defined them as stateless functional components with minimum content. An example template is below. You can use this for all three components, but just don't forget to change the names accordingly.

```
import React from 'react'
```

```
const Home = () =>{
```

```
  return(
```

```
    <div>
```

```
      Home
```

```
    </div>
```

```
  )
```

```
}
```

```
export default Home
```

Switch

We use the `<Switch>` component to group our `<Route>` components. **Switch** looks for all the **Routes** and then returns the first matching one.

Route

Routes are components calling your target component if it matches the path prop.

The final version of our Content.js file looks like this:

```
import React from 'react'

import { Switch, Route } from 'react-router-dom'

import Home from './Home'

import Works from './Works'

import About from './About'

const Content = () =>{

  return(

    <Switch>

      <Route exact path="/" component={ Home }/>

      <Route path="/works" component={ Works }/>

      <Route path="/about" component={ About }/>

    </Switch>

  )

}

export default Content
```

Notice that the extra `exact` prop is required for the **Home** component, which is the main directory. Using `exact` forces the **Route** to match the exact pathname. If it's not used, other pathnames starting with `/` would also be matched by the **Home** component, and for each link, it would only display the **Home** component.

Now when you click the menu links, your app should be switching the content.

Animating the Route Transitions

So far, we have a working router system. Now we will animate the route transitions. In order to achieve this, we will use the **react-transition-group** module.

We will be animating the *mounting* state of each component. When you route different components with the **Route** component inside **Switch**, you are essentially *mounting* and *unmounting* different components accordingly.

We will use **react-transition-group** in each component we want to animate. So you can have a different mounting animation for each component. I will only use one animation for all of them.

As an example, let's use the `<Home>` component.

First, we need to import **CSSTransitionGroup**.

```
import { CSSTransitionGroup } from 'react-transition-group'
```

Then you need to wrap your content with it.

Since we are dealing with the mounting state of the component, we enable `transitionAppear` and set a timeout for it. We also disable `transitionEnter` and `transitionLeave`, since these are only valid once the component is mounted. If you are planning to animate any children of the component, you have to use them.

Lastly, add the specific `transitionName` so that we can refer to it inside the CSS file.

```
import React from 'react'
```

```
import { CSSTransitionGroup } from 'react-transition-group'
```

```
import './styles/homeStyle.css'
```

```
const Home = () =>{
```

```
return(  
  
  <CSSTransitionGroup  
  
    transitionName="homeTransition"  
  
    transitionAppear={true}  
  
    transitionAppearTimeout={500}  
  
    transitionEnter={false}  
  
    transitionLeave={false}>  
  
      <div>  
  
        Home  
  
      </div>  
  
    </CSSTransitionGroup>  
  
  )  
  
}  
  
export default Home
```

We also imported a CSS file, where we define the CSS transitions.

```
.homeTransition-appear{
```

```
opacity: 0;
```

```
}
```

```
.homeTransition-appear.homeTransition-appear-active{
```

```
opacity: 1;
```

```
transition: all .5s ease-in-out;
```

```
}
```

If you refresh the page, you should see the fade-in effect of the **Home** component.

If you apply the same procedure to all the other routed components, you will see their individual animations when you change the content with your **Menu**.

Conclusion

In this tutorial, we covered the **react-router-dom** and **react-transition-group** modules. However, there's more to both modules than we covered in this tutorial. Here is a [working demo](#) of what was covered.

So, to learn more features, always go through the documentation of the modules you are using.

Over the last couple of years, React has grown in popularity. In fact, we have a number of items in the marketplace that are available for purchase, review, implementation, and so on. If you're looking for additional resources around React, don't hesitate to [check them out](#).

React Forms

HTML form elements work a bit differently from other DOM elements in React, because form elements naturally keep some internal state. For example, this form in plain HTML accepts a single name:

```
<form>
<label>
  Name:
<input type="text" name="name" />
```



```
</label>
<input type="submit" value="Submit" />
</form>
```

This form has the default HTML form behavior of browsing to a new page when the user submits the form. If you want this behavior in React, it just works. But in most cases, it's convenient to have a JavaScript function that handles the submission of the form and has access to the data that the user entered into the form. The standard way to achieve this is with a technique called “controlled components”.

Controlled Components

In HTML, form elements such as `<input>`, `<textarea>`, and `<select>` typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with `setState()`.

We can combine the two by making the React state be the “single source of truth”. Then the React component that renders a form also controls what happens in that form on subsequent user input. An input form element whose value is controlled by React in this way is called a “controlled component”.

For example, if we want to make the previous example log the name when it is submitted, we can write the form as a controlled component:

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: '' };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({ value: event.target.value });
  }
  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>Name:</label>
        <input type="text" value={this.state.value} onChange={this.handleChange} />
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

[Try it on CodePen](#)

Since the `value` attribute is set on our form element, the displayed value will always be `this.state.value`, making the React state the source of truth. Since `handleChange` runs on every keystroke to update the React state, the displayed value will update as the user types.

With a controlled component, the input's value is always driven by the React state. While this means you have to type a bit more code, you can now pass the value to other UI elements too, or reset it from other event handlers.

The textarea Tag

In HTML, a `<textarea>` element defines its text by its children:

```
<textarea>
  Hello there, this is some text in a text area
</textarea>
```

In React, a `<textarea>` uses a `value` attribute instead. This way, a form using a `<textarea>` can be written very similarly to a form that uses a single-line input:

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: 'Please write an essay about your favorite DOM element.' };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) { this.setState({ value: event.target.value }); }
  handleSubmit(event) {
    alert('An essay was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Essay:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Notice that `this.state.value` is initialized in the constructor, so that the text area starts off with some text in it.

The select Tag

In HTML, `<select>` creates a drop-down list. For example, this HTML creates a drop-down list of flavors:

```
<select>
  <option value="grapefruit">Grapefruit</option>
  <option value="lime">Lime</option>
  <option selected value="coconut">Coconut</option>
  <option value="mango">Mango</option>
```

```
</select>
```

Note that the Coconut option is initially selected, because of the `selected` attribute. React, instead of using this `selected` attribute, uses a `value` attribute on the root `select` tag. This is more convenient in a controlled component because you only need to update it in one place. For example:

```
class FlavorForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: 'coconut' };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) { this.setState({ value: event.target.value }); }
  handleSubmit(event) {
    alert('Your favorite flavor is: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Pick your favorite flavor:
          <select value={this.state.value} onChange={this.handleChange}>
            <option value="grapefruit">Grapefruit</option>
            <option value="lime">Lime</option>
            <option value="coconut">Coconut</option>
            <option value="mango">Mango</option>
          </select>
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

[Try it on CodePen](#)

Overall, this makes it so that `<input type="text">`, `<textarea>`, and `<select>` all work very similarly - they all accept a `value` attribute that you can use to implement a controlled component.

Note

You can pass an array into the `value` attribute, allowing you to select multiple options in a `select` tag:

```
<select multiple={true} value={['B', 'C']}>
```

The file input Tag

In HTML, an `<input type="file">` lets the user choose one or more files from their device storage to be uploaded to a server or manipulated by JavaScript via the [File API](#).

```
<input type="file" />
```

Because its value is read-only, it is an **uncontrolled** component in React. It is discussed together with other uncontrolled components [later in the documentation](#).

Handling Multiple Inputs

When you need to handle multiple controlled input elements, you can add a name attribute to each element and let the handler function choose what to do based on the value of `event.target.name`.

For example:

```
class Reservation extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      isGoing: true,
      numberOfGuests: 2
    };

    this.handleChange = this.handleChange.bind(this);
  }

  handleChange(event) {
    const target = event.target;
    const value = target.type === 'checkbox' ? target.checked : target.value;
    const name = target.name;
    this.setState({
      [name]: value
    });
  }

  render() {
    return (
      <form>
        <label>
          Is going:
          <input
            name="isGoing"      type="checkbox"
            checked={this.state.isGoing}
            onChange={this.handleChange} />
        </label>
        <br />
        <label>
          Number of guests:
          <input
            name="numberOfGuests" type="number"
            value={this.state.numberOfGuests}
            onChange={this.handleChange} />
        </label>
      </form>
    );
  }
}
```

[Try it on CodePen](#)

Note how we used the ES6 [computed property name](#) syntax to update the state key corresponding to the given input name:

```
this.setState({
  [name]: value});
```

It is equivalent to this ES5 code:

```
var partialState={};
partialState[name]=value;this.setState(partialState);
```

Also, since `setState()` automatically [merges a partial state into the current state](#), we only needed to call it with the changed parts.

Controlled Input Null Value

Specifying the `value` prop on a [controlled component](#) prevents the user from changing the input unless you desire so. If you've specified a value but the input is still editable, you may have accidentally set value to undefined or null.

The following code demonstrates this. (The input is locked at first but becomes editable after a short delay.)

```
ReactDOM.createRoot(mountNode).render(<input value="hi" />);

setTimeout(function(){
  ReactDOM.createRoot(mountNode).render(<input value={null} />);
},1000);
```

Alternatives to Controlled Components

It can sometimes be tedious to use controlled components, because you need to write an event handler for every way your data can change and pipe all of the input state through a React component. This can become particularly annoying when you are converting a preexisting codebase to React, or integrating a React application with a non-React library. In these situations, you might want to check out [uncontrolled components](#), an alternative technique for implementing input forms.

Introduction to Redux

Redux Toolkit

[Redux Toolkit](#) is our official recommended approach for writing Redux logic. It wraps around the Redux core, and contains packages and functions that we think are essential for building a Redux app. Redux Toolkit builds in our suggested best practices, simplifies most Redux tasks, prevents common mistakes, and makes it easier to write Redux applications.

RTK includes utilities that help simplify many common use cases, including [store setup](#), [creating reducers and writing immutable update logic](#), and even [creating entire "slices" of state at once](#).

Whether you're a brand new Redux user setting up your first project, or an experienced user who wants to simplify an existing application, [Redux Toolkit](#) can help you make your Redux code better.

Redux Toolkit is available as a package on NPM for use with a module bundler or in a Node application:

```
# NPM
npm install @reduxjs/toolkit

# Yarn
yarn add @reduxjs/toolkit

Create a React Redux App
```

The recommended way to start new apps with React and Redux is by using the [official Redux+JS template](#) or [Redux+TS template](#) for [Create React App](#), which takes advantage of [Redux Toolkit](#) and React Redux's integration with React components.

```
# Redux + Plain JS template
npx create-react-app my-app --template redux

# Redux + TypeScript template
npx create-react-app my-app --template redux-typescript

Redux Core
```

The Redux core library is available as a package on NPM for use with a module bundler or in a Node application:

```
# NPM
npm install redux

# Yarn
yarn add redux
```

It is also available as a precompiled UMD package that defines a `window.Redux` global variable. The UMD package can be used as a [<script> tag](#) directly.

For more details, see the [Installation](#) page.

Basic Example

The whole global state of your app is stored in an object tree inside a single *store*. The only way to change the state tree is to create an *action*, an object describing what happened, and *dispatch* it to the store. To specify how state gets updated in response to an action, you write pure *reducer* functions that calculate a new state based on the old state and the action.

```
import { createStore } from 'redux'

/**
 * This is a reducer - a function that takes a current state value and an
 * action object describing "what happened", and returns a new state value.
 * A reducer's function signature is: (state, action) => newState
```

```

*
* The Redux state should contain only plain JS objects, arrays, and primitives.
* The root state value is usually an object. It's important that you should
* not mutate the state object, but return a new object if the state changes.
*
* You can use any conditional logic you want in a reducer. In this example,
* we use a switch statement, but it's not required.
*/
function counterReducer(state = { value: 0 }, action) {
  switch (action.type) {
    case 'counter/incremented':
      return { value: state.value + 1 }
    case 'counter/decremented':
      return { value: state.value - 1 }
    default:
      return state
  }
}

// Create a Redux store holding the state of your app.
// Its API is { subscribe, dispatch, getState }.
let store = createStore(counterReducer)

// You can use subscribe() to update the UI in response to state changes.
// Normally you'd use a view binding library (e.g. React Redux) rather than subscribe() directly.
// There may be additional use cases where it's helpful to subscribe as well.

store.subscribe(() => console.log(store.getState()))

// The only way to mutate the internal state is to dispatch an action.
// The actions can be serialized, logged or stored and later replayed.
store.dispatch({ type: 'counter/incremented' })
// {value: 1}
store.dispatch({ type: 'counter/incremented' })
// {value: 2}
store.dispatch({ type: 'counter/decremented' })
// {value: 1}

```

Instead of mutating the state directly, you specify the mutations you want to happen with plain objects called *actions*. Then you write a special function called a *reducer* to decide how every action transforms the entire application's state.

In a typical Redux app, there is just a single store with a single root reducing function. As your app grows, you split the root reducer into smaller reducers independently operating on the different parts of the state tree. This is exactly like how there is just one root component in a React app, but it is composed out of many small components.

This architecture might seem like a lot for a counter app, but the beauty of this pattern is how well it scales to large and complex apps. It also enables very powerful developer tools, because it is possible to trace every mutation to the action that caused it. You can record user sessions and reproduce them just by replaying every action.

Redux Toolkit Example

Redux Toolkit simplifies the process of writing Redux logic and setting up the store. With Redux Toolkit, that same logic looks like:

```
import { createSlice, configureStore } from '@reduxjs/toolkit'

const counterSlice = createSlice({
  name: 'counter',
  initialState: {
    value: 0
  },
  reducers: {
    incremented: state => {
      // Redux Toolkit allows us to write "mutating" logic in reducers. It
      // doesn't actually mutate the state because it uses the Immer library,
      // which detects changes to a "draft state" and produces a brand new
      // immutable state based off those changes
      state.value += 1
    },
    decremented: state => {
      state.value -= 1
    }
  }
})

export const { incremented, decremented } = counterSlice.actions

const store = configureStore({
  reducer: counterSlice.reducer
})

// Can still subscribe to the store
store.subscribe(() => console.log(store.getState()))

// Still pass action objects to `dispatch`, but they're created for us
store.dispatch(incremented())
// {value: 1}
store.dispatch(incremented())
// {value: 2}
store.dispatch(decremented())
// {value: 1}
```

Redux Toolkit allows us to write shorter logic that's easier to read, while still following the same Redux behavior and data flow.

Learn Redux

We have a variety of resources available to help you learn Redux.

Redux Essentials Tutorial

The [Redux Essentials tutorial](#) is a "top-down" tutorial that teaches "how to use Redux the right way", using our latest recommended APIs and best practices. We recommend starting there.

Redux Fundamentals Tutorial

The [Redux Fundamentals tutorial](#) is a "bottom-up" tutorial that teaches "how Redux works" from first principles and without any abstractions, and why standard Redux usage

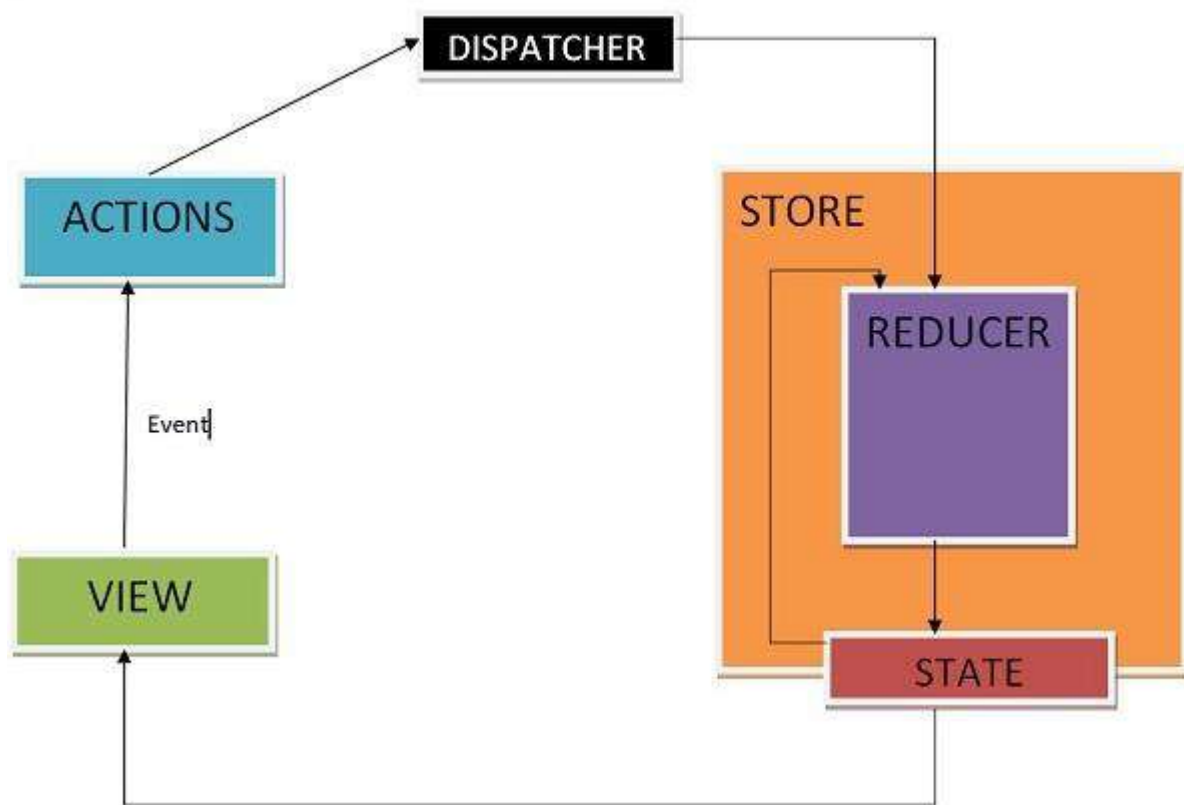
patterns exist.

Redux - Data Flow

Redux - Data Flow

Redux follows the unidirectional data flow. It means that your application data will follow in one-way binding data flow. As the application grows & becomes complex, it is hard to reproduce issues and add new features if you have no control over the state of your application.

Redux reduces the complexity of the code, by enforcing the restriction on how and when state update can happen. This way, managing updated states is easy. We already know about the restrictions as the three principles of Redux. Following diagram will help you understand Redux data flow better –



- An action is dispatched when a user interacts with the application.
- The root reducer function is called with the current state and the dispatched action. The root reducer may divide the task among smaller reducer functions, which ultimately returns a new state.
- The store notifies the view by executing their callback functions.
- The view can retrieve updated state and re-render again.

Client-Server Communication

Let's expand the application so that the notes are stored in the backend. We'll use [json-server](#), familiar from part 2.

The initial state of the database is stored in the file *db.json*, which is placed in the root of the project:

```
{
  "notes": [
    {
      "content": "the app state is in redux store",
      "important": true,
      "id": 1
    },
    {
      "content": "state changes are made with actions",
      "important": false,
      "id": 2
    }
  ]
}
```

We'll install json-server for the project...

```
npm install json-server --save-dev
```

and add the following line to the *scripts* part of the file *package.json*

```
"scripts": {
  "server": "json-server -p3001 --watch db.json",
  // ...
}
```

Now let's launch json-server with the command *npm run server*.

Next, we'll create a method into the file *services/notes.js*, which uses *axios* to fetch data from the backend

```
import axios from 'axios'

const baseUrl = 'http://localhost:3001/notes'

const getAll = async () => {
  const response = await axios.get(baseUrl)
  return response.data
}

export default { getAll }
```

We'll add axios to the project

```
npm install axios
```

We'll change the initialization of the state in *noteReducer*, so that by default there are no notes:

```
const noteSlice = createSlice({
  name: 'notes',
  initialState: [], // ...
})
```

Let's also add a new action *appendNote* for adding a note object:

```
const noteSlice = createSlice({
  name: 'notes',
  initialState: [],
  reducers: {
    createNote(state, action) {
      const content = action.payload

      state.push({
        content,
        important: false,
        id: generateId(),
      })
    },
    toggleImportanceOf(state, action) {
      const id = action.payload

      const noteToChange = state.find(n => n.id === id)

      const changedNote = {
        ...noteToChange,
        important: !noteToChange.important
      }

      return state.map(note =>
        note.id !== id ? note : changedNote
      )
    },
    appendNote(state, action) {
      state.push(action.payload)
    },
  },
})

export const { createNote, toggleImportanceOf, appendNote } = noteSlice.actions
export default noteSlice.reducer
```

A quick way to initialize the notes state based on the data received from the server is to fetch the notes in the *index.js* file and dispatch an action using the *appendNote* action creator for each individual note object:

```
// ...
import noteService from './services/notes'
import noteReducer, { appendNote } from './reducers/noteReducer'
const store = configureStore({
  reducer: {
    notes: noteReducer,
    filter: filterReducer,
  }
})

noteService.getAll().then(notes => notes.forEach(note => {
  store.dispatch(appendNote(note))
}))
// ...
```

Dispatching multiple actions seems a bit impractical. Let's add an action creator *setNotes* which can be used to directly replace the notes array. We'll get the action creator from the

createSlice function by implementing the *setNotes* action:

```
// ...

const noteSlice = createSlice({
  name: 'notes',
  initialState: [],
  reducers: {
    createNote(state, action) {
      const content = action.payload

      state.push({
        content,
        important: false,
        id: generateId(),
      })
    },
    toggleImportanceOf(state, action) {
      const id = action.payload

      const noteToChange = state.find(n => n.id === id)

      const changedNote = {
        ...noteToChange,
        important: !noteToChange.important
      }

      return state.map(note =>
        note.id !== id ? note : changedNote
      )
    },
    appendNote(state, action) {
      state.push(action.payload)
    },
    setNotes(state, action) { return action.payload } },
})

export const { createNote, toggleImportanceOf, appendNote, setNotes } = noteSlice.actions
export default noteSlice.reducer
```

Now, the code in the *index.js* file looks a lot better:

```
// ...
import noteService from './services/notes'
import noteReducer, { setNotes } from './reducers/noteReducer'
const store = configureStore({
  reducer: {
    notes: noteReducer,
    filter: filterReducer,
  }
})

noteService.getAll().then(notes =>
  store.dispatch(setNotes(notes)))
```

NB: why didn't we use *await* in place of promises and event handlers (registered to *then*-methods)?

Await only works inside *async* functions, and the code in *index.js* is not inside a function, so due to the simple nature of the operation, we'll abstain from using *async* this time.

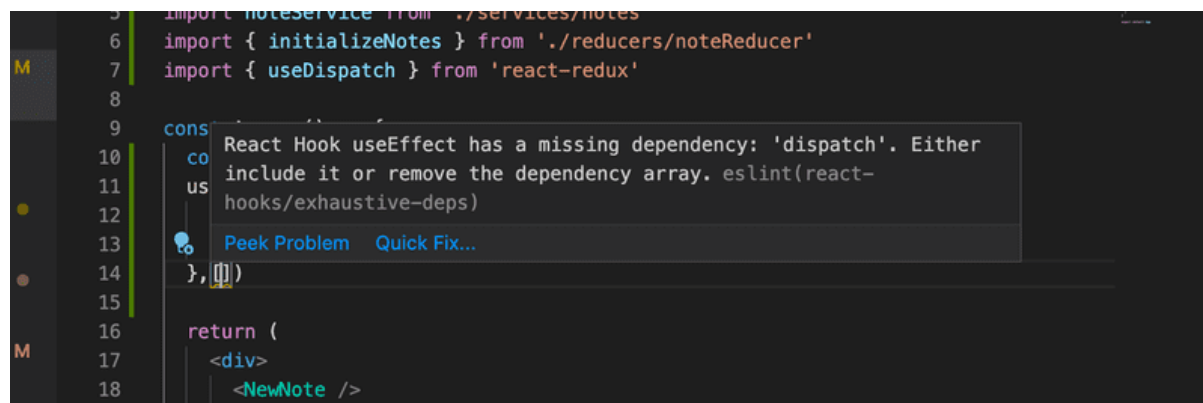
We do, however, decide to move the initialization of the notes into the *App* component, and, as usual, when fetching data from a server, we'll use the *effect hook*.

```
import { useEffect } from 'react'
import NewNote from './components/NewNote'
import Notes from './components/Notes'
import VisibilityFilter from './components/VisibilityFilter'
import noteService from './services/notes'
import { setNotes } from './reducers/noteReducer'
import { useDispatch } from 'react-redux'

const App = () => {
  const dispatch = useDispatch()
  useEffect(() => {
    noteService
      .getAll().then((notes) =>
        dispatch(setNotes(notes)))
    }, [])
  return (
    <div>
      <NewNote />
      <VisibilityFilter />
      <Notes />
    </div>
  )
}

export default App
```

Using the *useEffect* hook causes an eslint warning:



We can get rid of it by doing the following:

```
const App = () => {
  const dispatch = useDispatch()
  useEffect(() => {
    noteService
      .getAll().then((notes) => dispatch(setNotes(notes)))
    }, [dispatch])
  // ...
}
```

Now the variable *dispatch* we define in the *App* component, which practically is the dispatch function of the redux store, has been added to the array *useEffect* receives as a parameter. **If** the value of the *dispatch* variable would change during runtime, the effect would be executed again. This however cannot happen in our application, so the warning is unnecessary.

Another way to get rid of the warning would be to disable ESLint on that line:

```
const App = () => {
  const dispatch = useDispatch()
  useEffect(() => {
    noteService
    .getAll().then(notes => dispatch(setNotes(notes)))
    }, []) // eslint-disable-line react-hooks/exhaustive-deps
    // ...
  }
```

Generally disabling ESLint when it throws a warning is not a good idea. Even though the ESLint rule in question has caused some [arguments](#), we will use the first solution.

More about the need to define the hooks dependencies in [the react documentation](#).

We can do the same thing when it comes to creating a new note. Let's expand the code communicating with the server as follows:

```
const baseUrl = 'http://localhost:3001/notes'

const getAll = async () => {
  const response = await axios.get(baseUrl)
  return response.data
}

const createNew = async (content) => { const object = { content, important: false } const response = await
  axios.post(baseUrl, object) return response.data }
export default {
  getAll,
  createNew,
}
```

The method *addNote* of the component *NewNote* changes slightly:

```
import { useDispatch } from 'react-redux'
import { createNote } from '../reducers/noteReducer'
import noteService from '../services/notes'
const NewNote = (props) => {
  const dispatch = useDispatch()

  const addNote = async (event) => { event.preventDefault()
    const content = event.target.note.value
    event.target.note.value = ""
    const newNote = await noteService.createNew(content) dispatch(createNote(newNote)) }

  return (
    <form onSubmit={addNote}>
      <input name="note" />
      <button type="submit">add</button>
    </form>
  )
}

export default NewNote
```

Because the backend generates ids for the notes, we'll change the action creator *createNote*

accordingly:

```
createNote(state, action) {  
  state.push(action.payload)  
}
```

Changing the importance of notes could be implemented using the same principle, by making an asynchronous method call to the server and then dispatching an appropriate action.

UNIT – IV

Java Web Development

Web development is known as website development or web application development. The web development creates, maintains, and updates web development applications using a browser. This web development requires web designing, backend programming, and database management. The development process requires software technology.

Web development creates web applications using servers. We can use a web server or machine server like a CPU. The Web server or virtual server requires web application using technology. Web development requires server-side programming language or technology. Mostly Java, PHP, and other server-side languages require for web development.

Java web development creates a server-side website and web application. The majority of Java web apps do not execute on the server directly. A web container on the server hosts Java web applications.

For Java web applications, the container acts as a runtime environment. What the Java Virtual Machine is for locally running Java applications, the container is for Java web applications. JVM is used to run the container itself.

Java distinguishes between two types of containers: web and Java EE. Additional functionality, such as server load distribution, can be supported by a container. A web container supports Java servlets and JSP (JavaServer Pages). In Java technology, Tomcat is a common web container.

A web container is usually a minimal need for web frameworks. GWT, Struts, JavaServer Faces, and the Spring framework are common Java web frameworks. Servlets are at the heart of most modern Java web frameworks.

Functions of Java Web Development

Java web development creates applications and websites using static and dynamic resources. The static resource refers to HTML pages with images, and a dynamic resource refers to classes, jars, Servlet, and JSP. Java web development uses several packages, files, and online links. Java web development requires web archive files known as a WAR files.

Java web development works on three main factors. These development factors show below.

- Front-end web development using Java technology.
- Backend web development using Java server technology.
- Database management using Java database driver.

The above three factors create, update, remove, display and operate data or information.

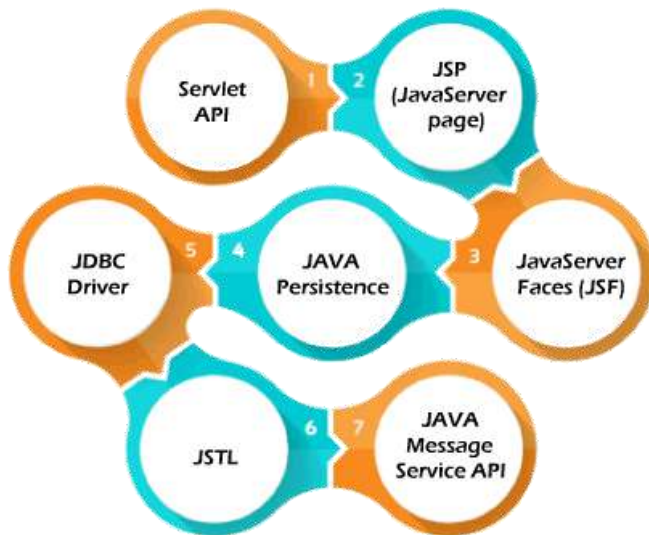
Front-end web development: The front-end technology interacts with the user and Java interface. It helps to insert and submit data. Java web development uses JavaServer Pages or JSP for the front-end form or table.

Backend web development: The backend technology maintains and updates data of the database. Java uses Servlet, spring, and other advanced technology.

Database management handles or fetches data from the database using the Java database driver. The Java technology uses JDBC, Hibernate to handle the database.

Types of the Java Web Technologies

- Servlet API
- JSP (JavaServer page)
- JDBC Driver
- JAVA Persistence
- JavaServer Faces (JSF)
- JSTL
- JAVA Message Service API



Java Web Technologies

Servlet API (JAVA Web application programming interface)

Servlet, filter, filter chain, servlet config, and other interfaces are available in the javax. Servlet package. The capabilities of servers that host apps are increased by using Servlet.

The request-response model is used in web development applications written with Java servlets. From initialization to garbage collection, a servlet has a life cycle.

Servlets are useful for various tasks, including collecting data via web page forms, presenting data from a database or any other third-party source, etc.

Servlets are Java programs that run on a web application and send client requests to databases or servers. After talking with the database, the servlets help process the client's request and provide results.

JSP (JavaServer Page Web application programming technology)

Developers employ JavaServer Pages or JSP technology to quickly produce platform- and server-independent online content. Normally, the developer works on separate Common Gateway Interface files to embed dynamic elements in HTML pages. Java JSP technology can be used, as it has access to the whole Java API family.

The JSP technology pieces code to control web information and moves dynamically. A JSP page comprises static data written in HTML, WML, XML, and other markup languages. Special JSP tags simplify Java code into HTML pages, making web development user-friendly.

The JSP technology allows embedding bits of servlet code in a text-based document. JSP is a popular Java EE technology that allows programmers to create complex dynamic web pages quickly.

JDBC Driver or Java Database Connectivity

JDBC Driver is a connector between database and Java web application. Java database

connectivity helps to update and modify data using queries. The jdbc driver is an essential part of Java web development. This driver helps to send data to the database and retrieve data from the database.

Within a Java program, the JDBC driver allows to perform the following tasks:

- Make a data source connection
- To the data source, send queries and update statements
- Displays require data from a database.
- Organize application information.

JDBC is a set of methods and queries for accessing databases written in Java. Clients can use web applications using JDBC drivers to update any information in the database.

JDBC drivers connect to databases in four ways: JDBC-ODBC Bridge Driver, Network Protocol Driver, Native Driver, and Thin Driver.

Persistence API for Java

For web development, the Java Persistence API employs object-relational mapping. This mapping connects a database to an object-oriented model. Java Persistence makes it simple to manage relational data in Java web applications. The Java Persistence API aids in database data management. This API sends data to a database and retrieves data from it regularly.

Large amounts of code, proprietary frameworks, and other files are not required. JPA gives a straightforward technique of database communication. A database is an object-relational approach for interacting with Java web development. JPA is a set of lightweight classes and methods for interacting with databases.

Technology of the JavaServer Faces

JavaServer Faces is called a JSF Technology. This technology provides a framework for developing web-based interfaces. JSF provides a simple model for components in various scripting or markup languages.

The data sources and server-side event handlers are coupled to the User Interface widgets. JSF aids in the creation and maintenance of web applications by minimizing the time and effort required.

- Construct Java web development pages.
- Drop components on a web page by adding component tags to a web page.
- Connect Java web development page components to server-side data.
- Connect component-generated events to application code running on the server.
- Extend the life of server requests by storing and restoring the application state.

Standard Tag Library for JavaServer Pages (JSTL)

The JavaServer Pages Standard Tag Library or JSTL abstracts common functionality of JSP-based applications. We use a single standard set of tags to incorporate tags from various vendors into web applications. This standardization enables the establishment of Java

applications on any JSP container. It supports JSTL and increases the tags to optimize during implementation.

JSTL includes iterator and conditional tags for controlling flow. These tags work for manipulating XML documents and tags for internationalization. This JSTL is also used for SQL database access and tags for frequently used functions.

API for Java Message Service

Messaging is a way for software components or apps to communicate with one another. A messaging system is a type of peer-to-peer network. In other words, a messaging client can communicate with and be communicated with by any other client.

Each client establishes a connection with a messaging agent, facilitating the creation, transmission, receipt, and reading of messages.

The Java Message Service (JMS) API provides a strong tool for resolving enterprise computing problems by integrating Java technology and enterprise messaging.

Enterprise messaging enables the secure and flexible sharing of business data. The JMS API extends this by providing a uniform API and provider framework that facilitates the building of portable message-based Java applications.

Special Features of the Java web development

- Java is a mature, versatile, and powerful programming language.
- Additionally, it is popular, which means that tools and assistance for Java web development are readily available.
- Java's platform freedom is one of its strongest characteristics. Java code can be executed on any platform, including a Mac or a Windows computer. On any operating system, we can run a Java web application.
- Java is also capable of running mobile applications on smartphones and tablets.
- Java web development does not require additional effort to design and run web apps across several platforms.
- Java also includes an enormous standard library. This library readily works with common tasks such as input and output, networking, and graphic user interfaces. It provides tools to help web application developers.

Conclusion

Java programming language is easy to handle and programmer's first choice for web development. Java web development has basic rules apart from operating data. This technology does not need an extra operation or advanced programming.

Java web development creates multiple web applications using a single type of code on multiple pages. If we know the working procedure, then JAVA technology develops any application.

JAVA PROGRAMMING BASICS

What is Java?

Java is a high-level, general-purpose, object-oriented, and secure programming language developed by James Gosling at Sun Microsystems, Inc. in 1991. It is formally known as OAK. In 1995, Sun Microsystem changed the name to Java. In 2009, Sun Microsystem takeover by Oracle Corporation.

Editions of Java

Each edition of Java has different capabilities. There are three editions of Java:

- **Java Standard Editions (JSE):** It is used to create programs for a desktop computer.
- **Java Enterprise Edition (JEE):** It is used to create large programs that run on the server and manages heavy traffic and complex transactions.
- **Java Micro Edition (JME):** It is used to develop applications for small devices such as set-top boxes, phone, and appliances.

Types of Java Applications

There are four types of Java applications that can be created using Java programming:

- **Standalone Applications:** Java standalone applications uses GUI components such as AWT, Swing, and JavaFX. These components contain buttons, list, menu, scroll panel, etc. It is also known as desktop alienations.
- **Enterprise Applications:** An application which is distributed in nature is called enterprise applications.
- **Web Applications:** An applications that run on the server is called web applications. We use JSP, Servlet, Spring, and Hibernate technologies for creating web applications.
- **Mobile Applications:** Java ME is a cross-platform to develop mobile applications which run across smartphones. Java is a platform for App Development in Android.

Java Platform

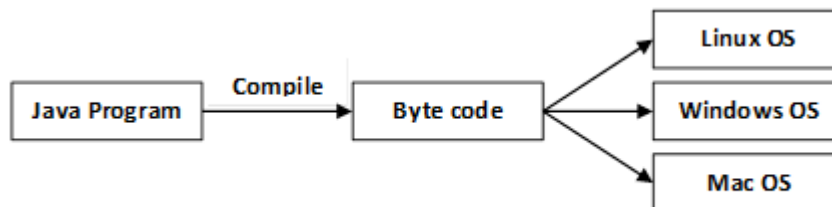
Java Platform is a collection of programs. It helps to develop and run a program written in the Java programming language. Java Platform includes an execution engine, a compiler and set of libraries. Java is a platform-independent language.

Features of Java

- **Simple:** Java is a simple language because its syntax is simple, clean, and easy to understand. Complex and ambiguous concepts of C++ are either eliminated or re-implemented in Java. For example, pointer and operator overloading are not used in Java.
- **Object-Oriented:** In Java, everything is in the form of the object. It means it has some data and behavior. A program must have at least one class and object.
- **Robust:** Java makes an effort to check error at run time and compile time. It uses a

strong memory management system called garbage collector. Exception handling and garbage collection features make it strong.

- **Secure:** Java is a secure programming language because it has no explicit pointer and programs runs in the virtual machine. Java contains a security manager that defines the access of Java classes.
- **Platform-Independent:** Java provides a guarantee that code writes once and run anywhere. This byte code is platform-independent and can be run on any machine.



- **Portable:** Java Byte code can be carried to any platform. No implementation-dependent features. Everything related to storage is predefined, for example, the size of primitive data types.
- **High Performance:** Java is an interpreted language. Java enables high performance with the use of the Just-In-Time compiler.
- **Distributed:** Java also has networking facilities. It is designed for the distributed environment of the internet because it supports TCP/IP protocol. It can run over the internet. EJB and RMI are used to create a distributed system.
- **Multi-threaded:** Java also supports multi-threading. It means to handle more than one job a time.

OOPs (Object Oriented Programming System)

Object-oriented programming is a way of solving a complex problem by breaking them into a small sub-problem. An object is a real-world entity. It is easier to develop a program by using an object. In OOPs, we create programs using class and object in a structured manner.

Class: A class is a template or blueprint or prototype that defines data members and methods of an object. An object is the instance of the class. We can define a class by using the class keyword.

Object: An object is a real-world entity that can be identified distinctly. For example, a desk, a circle can be considered as objects. An object has a unique behavior, identity, and state. Data fields with their current values represent the state of an object (also known as its properties or attributes).

Abstraction: An abstraction is a method of hiding irrelevant information from the user. For example, the driver only knows how to drive a car; there is no need to know how does the car run. We can make a class abstract by using the keyword abstract. In Java, we use abstract class and interface to achieve abstraction.

Encapsulation: An encapsulation is the process of binding data and functions into a single unit. A class is an example of encapsulation. In Java, Java bean is a fully encapsulated class.

Inheritance: Inheritance is the mechanism in which one class acquire all the features of another class. We can achieve inheritance by using the extends keyword. It facilitates the reusability of the code.

Polymorphism: The polymorphism is the ability to appear in many forms. In other words, single action in different ways. For example, a boy in the classroom behaves like a student, in house behaves like a son. There are two types of polymorphism: run time polymorphism and compile-time polymorphism.

Java Variables

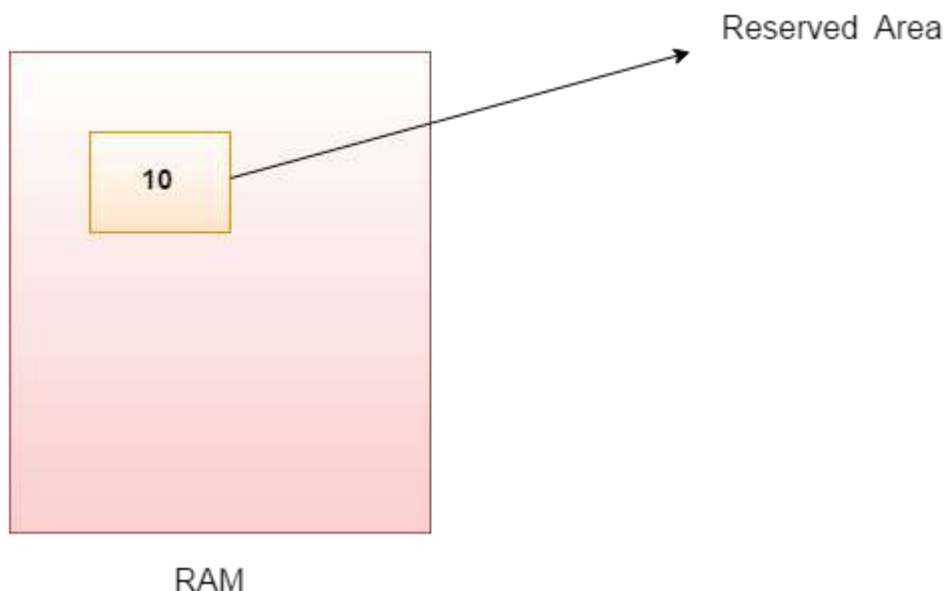
A variable is a container which holds the value while the [Java program](#) is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of [data types in Java](#): primitive and non-primitive.

Variable

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.



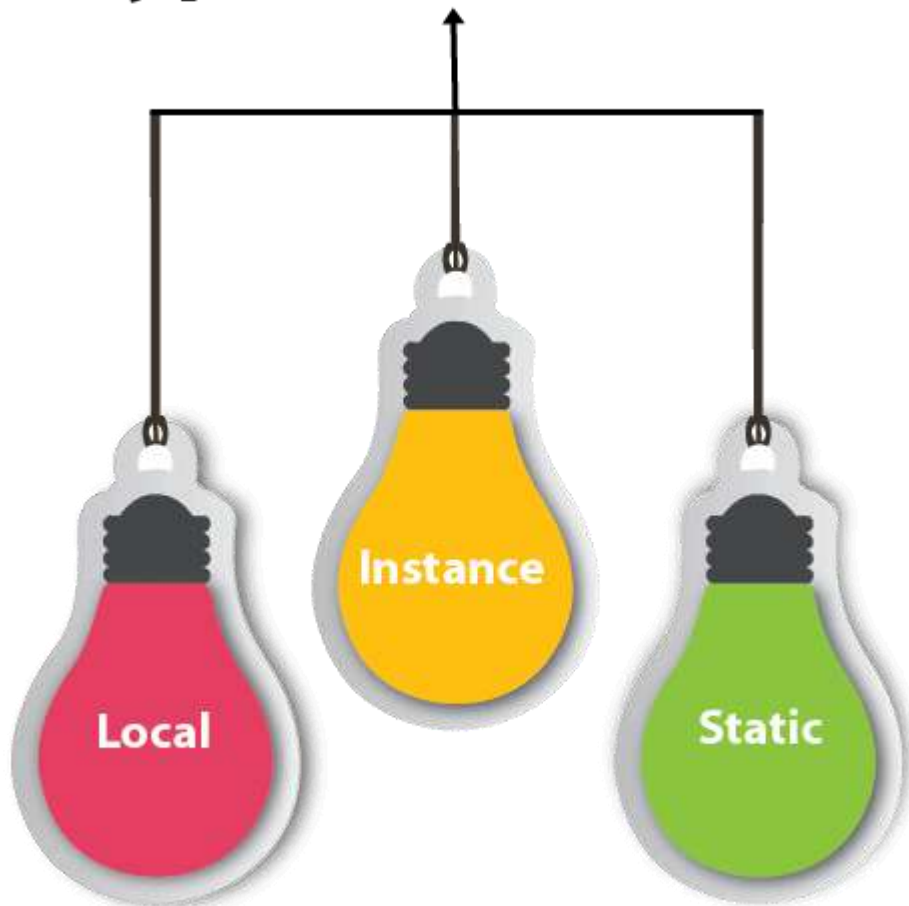
1. `int data=50;`//Here data is variable

Types of Variables

There are three types of variables in [Java](#):

- local variable
- instance variable
- static variable

Types of Variables



1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](#).

It is called an instance variable because its value is instance-specific and is not shared among

instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
1. public class A
2. {
3.     static int m=100;//static variable
4.     void method()
5.     {
6.         int n=90;//local variable
7.     }
8.     public static void main(String args[])
9.     {
10.        int data=50;//instance variable
11.    }
12. }//end of class
```

Java Variable Example: Add Two Numbers

```
1. public class Simple{
2.     public static void main(String[] args){
3.         int a=10;
4.         int b=10;
5.         int c=a+b;
6.         System.out.println(c);
7.     }
8. }
```

Output:

20

Java Variable Example: Widening

```
1. public class Simple{
2.     public static void main(String[] args){
3.         int a=10;
4.         float f=a;
5.         System.out.println(a);
6.         System.out.println(f);
7.     }}
```

Output:

10

10.0

Java Variable Example: Narrowing (Typecasting)

```
1. public class Simple{
2.     public static void main(String[] args){
3.         float f=10.5f;
4.         //int a=f;//Compile time error
5.         int a=(int)f;
6.         System.out.println(f);
7.         System.out.println(a);
8.     }}
```

Output:

10.5
10

Java Variable Example: Overflow

```
1. class Simple{
2.     public static void main(String[] args){
3.         //Overflow
4.         int a=130;
5.         byte b=(byte)a;
6.         System.out.println(a);
7.         System.out.println(b);
8.     }}
```

Output:

130
-126

Java Variable Example: Adding Lower Type

```
1. class Simple{
2.     public static void main(String[] args){
3.         byte a=10;
4.         byte b=10;
5.         //byte c=a+b;//Compile Time Error: because a+b=20 will be int
6.         byte c=(byte)(a+b);
7.         System.out.println(c);
8.     }}
```

Output:

20

Java OOPs Concepts

1. [Object-Oriented Programming](#)
2. [Advantage of OOPs over Procedure-oriented programming language](#)

3. [Difference between Object-oriented and Object-based programming language.](#)

In this page, we will learn about the basics of OOPs. Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

Simula is considered the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as a truly object-oriented programming language.

Smalltalk is considered the first truly object-oriented programming language.

The popular object-oriented languages are [Java](#), [C#](#), [PHP](#), [Python](#), [C++](#), etc.

The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.

OOPs (Object-Oriented Programming System)

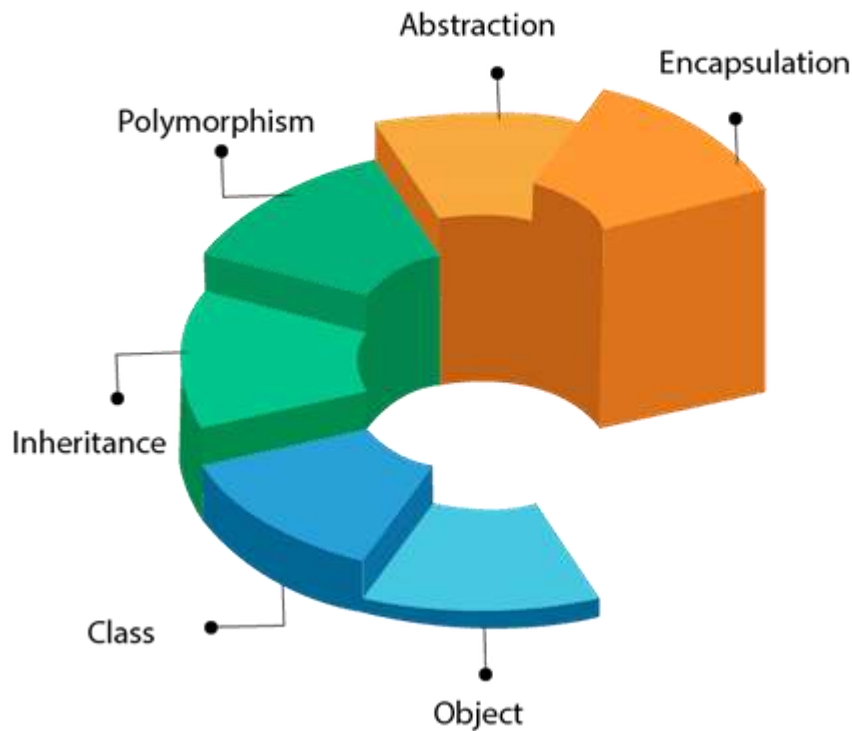
Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- [Object](#)
- Class
- [Inheritance](#)
- [Polymorphism](#)
- [Abstraction](#)
- [Encapsulation](#)

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

- Coupling
- Cohesion
- Association
- Aggregation
- Composition

OOPs (Object-Oriented Programming System)



Object



Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes

up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.



Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.



Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

- One to One
- One to Many
- Many to One, and
- Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be unidirectional or bidirectional.

Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java. Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

Advantage of OOPs over Procedure-oriented programming language

- 1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.
- 2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

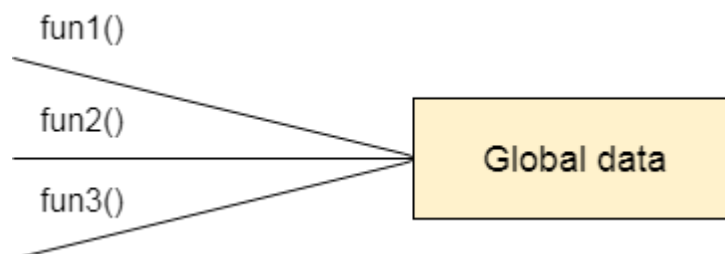


Figure: Data Representation in Procedure-Oriented Programming

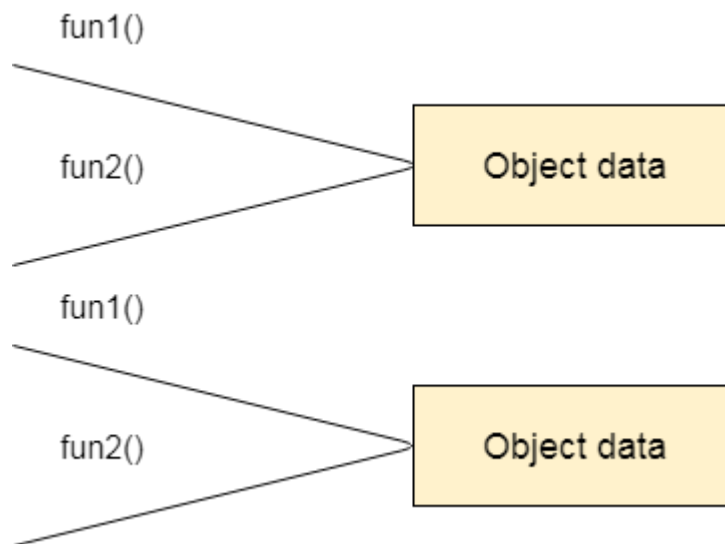


Figure: Data Representation in Object-Oriented Programming

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

What is the difference between an object-oriented programming language and object-based programming language?

Object-based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object-based programming languages.

MVC Architecture in Java

The Model-View-Controller (MVC) is a well-known [design pattern](#) in the web development field. It is way to organize our code. It specifies that a program or application shall consist of data model, presentation information and control information. The MVC pattern needs all these components to be separated as different objects.

In this section, we will discuss the MVC Architecture in Java, alongwith its advantages and disadvantages and examples to understand the implementation of MVC in Java.

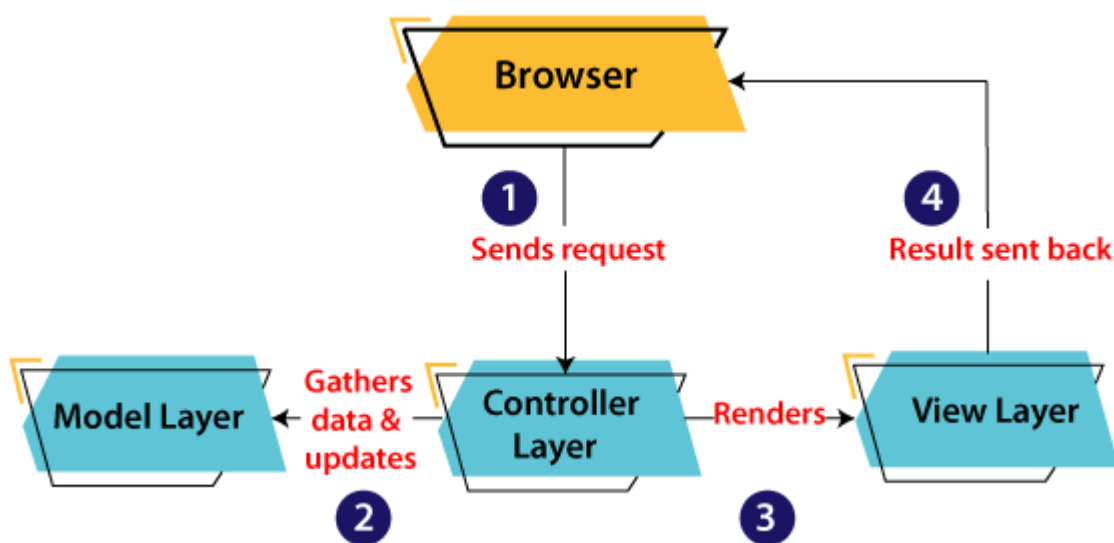
What is MVC architecture in Java?

The model designs based on the MVC architecture follow MVC design pattern. The application logic is separated from the user interface while designing the software using model designs.

The MVC pattern architecture consists of three layers:

- **Model:** It represents the business layer of application. It is an object to carry the data that can also contain the logic to update controller if data is changed.
- **View:** It represents the presentation layer of application. It is used to visualize the data that the model contains.
- **Controller:** It works on both the model and view. It is used to manage the flow of application, i.e. data flow in the model object and to update the view whenever data is changed.

In Java Programming, the Model contains the simple [Java classes](#), the View used to display the data and the Controller contains the [servlets](#). Due to this separation the user requests are processed as follows:



1. A client (browser) sends a request to the controller on the server side, for a page.
2. The controller then calls the model. It gathers the requested data.
3. Then the controller transfers the data retrieved to the view layer.
4. Now the result is sent back to the browser (client) by the view.

Advantages of MVC Architecture

The advantages of MVC architecture are as follows:

- MVC has the feature of scalability that in turn helps the growth of application.
- The components are easy to maintain because there is less dependency.
- A model can be reused by multiple views that provides reusability of code.
- The developers can work with the three layers (Model, View, and Controller) simultaneously.
- Using MVC, the application becomes more understandable.
- Using MVC, each layer is maintained separately therefore we do not require to deal with massive code.
- The extending and testing of application is easier.

Implementation of MVC using Java

To implement MVC pattern in Java, we are required to create the following three classes.

- **Employee Class**, will act as model layer
- **EmployeeView Class**, will act as a view layer
- **EmployeeContoller Class**, will act a controller layer

MVC Architecture Layers

Model Layer

The Model in the MVC design pattern acts as a data layer for the application. It represents the business logic for application and also the state of application. The model object fetch and store the model state in the database. Using the model layer, rules are applied to the data that represents the concepts of application.

Let's consider the following code snippet that creates a which is also the first step to implement MVC pattern.

Employee.java

```
1. // class that represents model
2. public class Employee {
3.
4.     // declaring the variables
5.     private String EmployeeName;
6.     private String EmployeeId;
7.     private String EmployeeDepartment;
8.
9.     // defining getter and setter methods
10.    public String getId() {
11.        return EmployeeId;
12.    }
13.
14.    public void setId(String id) {
15.        this.EmployeeId = id;
16.    }
17.
18.    public String getName() {
19.        return EmployeeName;
20.    }
21.
22.    public void setName(String name) {
23.        this.EmployeeName = name;
24.    }
25.
26.    public String getDepartment() {
27.        return EmployeeDepartment;
28.    }
29.
30.    public void setDepartment(String Department) {
31.        this.EmployeeDepartment = Department;
32.    }
33.
```

```
34. }
```

The above code simply consists of getter and setter methods to the Employee class.

View Layer

As the name depicts, view represents the visualization of data received from the model. The view layer consists of output of application or user interface. It sends the requested data to the client, that is fetched from model layer by controller.

Let's take an example where we create a view using the EmployeeView class.

EmployeeView.java

```
1. // class which represents the view
2. public class EmployeeView {
3.
4.     // method to display the Employee details
5.     public void printEmployeeDetails (String EmployeeName, String EmployeeId, String
        EmployeeDepartment){
6.         System.out.println("Employee Details: ");
7.         System.out.println("Name: " + EmployeeName);
8.         System.out.println("Employee ID: " + EmployeeId);
9.         System.out.println("Employee Department: " + EmployeeDepartment);
10.    }
11. }
```

Controller Layer

The controller layer gets the user requests from the view layer and processes them, with the necessary validations. It acts as an interface between Model and View. The requests are then sent to model for data processing. Once they are processed, the data is sent back to the controller and then displayed on the view.

Let's consider the following code snippet that creates the controller using the EmployeeController class.

EmployeeController.java

```
1. // class which represent the controller
2. public class EmployeeController {
3.
4.     // declaring the variables model and view
5.     private Employee model;
6.     private EmployeeView view;
7.
8.     // constructor to initialize
9.     public EmployeeController(Employee model, EmployeeView view) {
10.         this.model = model;
11.         this.view = view;
12.     }
```

```
13.
14. // getter and setter methods
15. public void setEmployeeName(String name){
16.     model.setName(name);
17. }
18.
19. public String getEmployeeName(){
20.     return model.getName();
21. }
22.
23. public void setEmployeeId(String id){
24.     model.setId(id);
25. }
26.
27. public String getEmployeeId(){
28.     return model.getId();
29. }
30.
31. public void setEmployeeDepartment(String Department){
32.     model.setDepartment(Department);
33. }
34.
35. public String getEmployeeDepartment(){
36.     return model.getDepartment();
37. }
38.
39. // method to update view
40. public void updateView() {
41.     view.printEmployeeDetails(model.getName(), model.getId(), model.getDepart
ment());
42. }
43. }
```

Main Class Java file

The following example displays the main file to implement the MVC architecture. Here, we are using the MVCMain class.

MVCMain.java

```
1. // main class
2. public class MVCMain {
3.     public static void main(String[] args) {
4.
5.         // fetching the employee record based on the employee_id from the database
6.         Employee model = retrieveEmployeeFromDatabase();
7.
8.         // creating a view to write Employee details on console
9.         EmployeeView view = new EmployeeView();
10.
11.         EmployeeController controller = new EmployeeController(model, view);
```

```
12.
13.     controller.updateView();
14.
15.     //updating the model data
16.     controller.setEmployeeName("Nirnay");
17.     System.out.println("\n Employee Details after updating: ");
18.
19.     controller.updateView();
20. }
21.
22. private static Employee retrieveEmployeeFromDatabase(){
23.     Employee Employee = new Employee();
24.     Employee.setName("Anu");
25.     Employee.setId("11");
26.     Employee.setDepartment("Salesforce");
27.     return Employee;
28. }
29. }
```

The **MVCMain** class fetches the employee data from the method where we have entered the values. Then it pushes those values in the model. After that, it initializes the view (EmployeeView.java). When view is initialized, the Controller (EmployeeController.java) is invoked and bind it to Employee class and EmployeeView class. At last the updateView() method (method of controller) update the employee details to be printed to the console.

Output:

```
Employee Details:
Name: Anu
Employee ID: 11
Employee Department: Salesforce
```

```
Employee Details after updating:
Name: Nirnay
Employee ID: 11
Employee Department: Salesforce
```

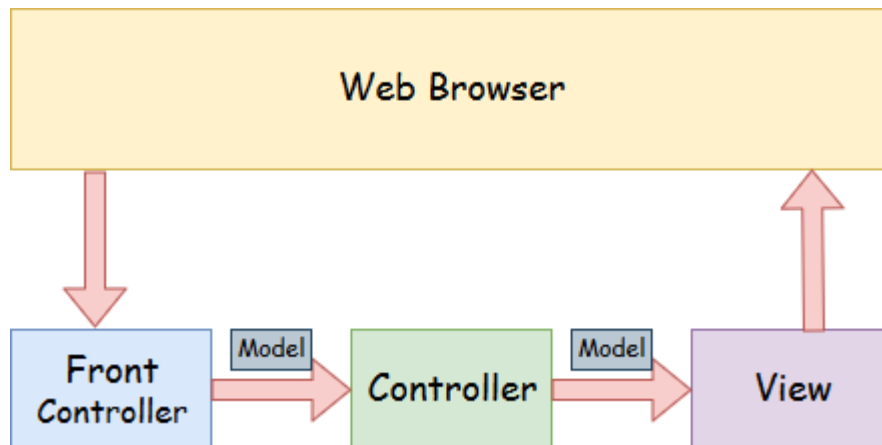
In this way, we have learned about MVC Architecture, significance of each layer and its implementation in Java.

Spring MVC Tutorial

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

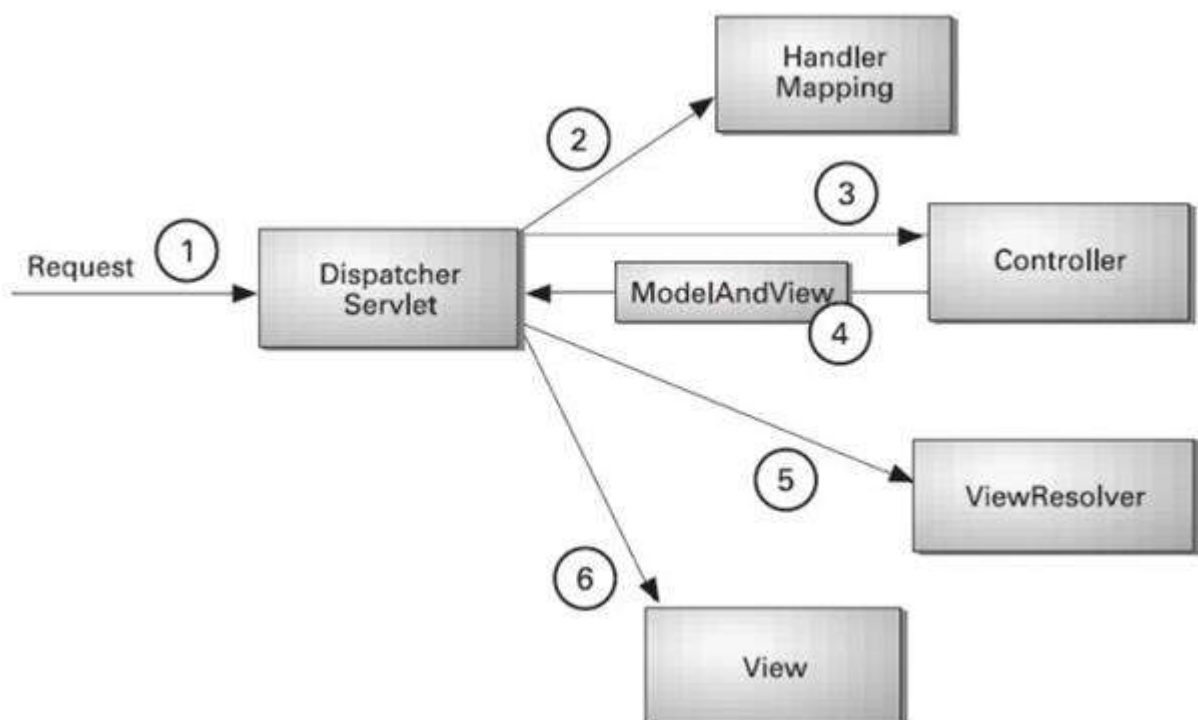
A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

Spring Web Model-View-Controller



- **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
- **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
- **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
- **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

Understanding the flow of Spring Web MVC



-
- As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
 - The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
 - The controller returns an object of ModelAndView.
 - The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.
-

Advantages of Spring MVC Framework

Let's see some of the advantages of Spring MVC Framework:-

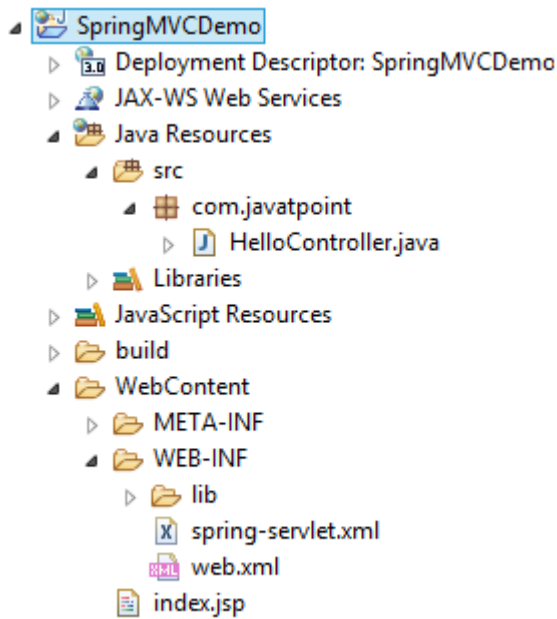
- **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
 - **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
 - **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
 - **Rapid development** - The Spring MVC facilitates fast and parallel development.
 - **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
 - **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
 - **Flexible Mapping** - It provides the specific annotations that easily redirect the page.
-

Spring Web MVC Framework Example

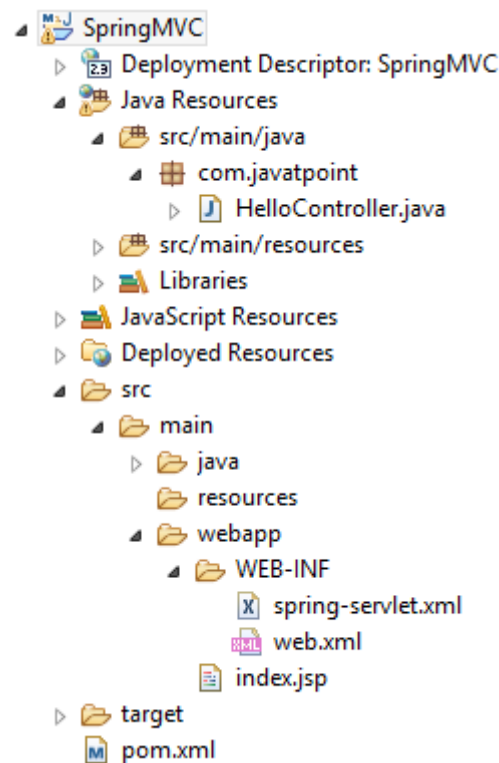
Let's see the simple example of a Spring Web MVC framework. The steps are as follows:

- Load the spring jar files or add dependencies in the case of Maven
 - Create the controller class
 - Provide the entry of controller in the web.xml file
 - Define the bean in the separate XML file
 - Display the message in the JSP page
 - Start the server and deploy the project
-

Directory Structure of Spring MVC



Directory Structure of Spring MVC using Maven



Required Jar files or Maven Dependency

To run this example, you need to load:

- Spring Core jar files

- Spring Web jar files
- JSP + JSTL jar files (If you are using any another view technology then load the corresponding jar files).

Download Link: [Download all the jar files for spring including JSP and JSTL.](#)

If you are using Maven, you don't need to add jar files. Now, you need to add maven dependency to the pom.xml file.

1. Provide project information and configuration in the pom.xml file.

pom.xml

1. `<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
2. `xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">`
3. `<modelVersion>4.0.0</modelVersion>`
4. `<groupId>com.javatpoint</groupId>`
5. `<artifactId>SpringMVC</artifactId>`
6. `<packaging>war</packaging>`
7. `<version>0.0.1-SNAPSHOT</version>`
8. `<name>SpringMVC Maven Webapp</name>`
9. `<url>http://maven.apache.org</url>`
10. `<dependencies>`
11. `<dependency>`
12. `<groupId>junit</groupId>`
13. `<artifactId>junit</artifactId>`
14. `<version>3.8.1</version>`
15. `<scope>test</scope>`
16. `</dependency>`
- 17.
18. `<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->`
19. `<dependency>`
20. `<groupId>org.springframework</groupId>`
21. `<artifactId>spring-webmvc</artifactId>`
22. `<version>5.1.1.RELEASE</version>`
23. `</dependency>`
- 24.
25. `<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->`
26. `<dependency>`
27. `<groupId>javax.servlet</groupId>`
28. `<artifactId>servlet-api</artifactId>`
29. `<version>3.0-alpha-1</version>`
30. `</dependency>`
- 31.
32. `</dependencies>`
33. `<build>`
34. `<finalName>SpringMVC</finalName>`
35. `</build>`
36. `</project>`

2. Create the controller class

To create the controller class, we are using two annotations @Controller and @RequestMapping.

The @Controller annotation marks this class as Controller.

The @RequestMapping annotation is used to map the class with the specified URL name.

HelloController.java

```
1. package com.javatpoint;
2. import org.springframework.stereotype.Controller;
3. import org.springframework.web.bind.annotation.RequestMapping;
4. @Controller
5. public class HelloController {
6.     @RequestMapping("/")
7.     public String display()
8.     {
9.         return "index";
10.    }
11. }
```

3. Provide the entry of controller in the web.xml file

In this xml file, we are specifying the servlet class DispatcherServlet that acts as the front controller in Spring Web MVC. All the incoming request for the html file will be forwarded to the DispatcherServlet.

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.co
   m/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
   app_3_0.xsd" id="WebApp_ID" version="3.0">
3.     <display-name>SpringMVC</display-name>
4.     <servlet>
5.         <servlet-name>spring</servlet-name>
6.         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
7.         <load-on-startup>1</load-on-startup>
8.     </servlet>
9.     <servlet-mapping>
10.         <servlet-name>spring</servlet-name>
11.         <url-pattern>/</url-pattern>
12.     </servlet-mapping>
13. </web-app>
```

4. Define the bean in the xml file

This is the important configuration file where we need to specify the View components.

The context:component-scan element defines the base-package where DispatcherServlet will search the controller class.

This xml file should be located inside the WEB-INF directory.

spring-servlet.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:context="http://www.springframework.org/schema/context"
5.     xmlns:mvc="http://www.springframework.org/schema/mvc"
6.     xsi:schemaLocation="
7.         http://www.springframework.org/schema/beans
8.         http://www.springframework.org/schema/beans/spring-beans.xsd
9.         http://www.springframework.org/schema/context
10.        http://www.springframework.org/schema/context/spring-context.xsd
11.        http://www.springframework.org/schema/mvc
12.        http://www.springframework.org/schema/mvc/spring-mvc.xsd">
13.
14.    <!-- Provide support for component scanning -->
15.    <context:component-scan base-package="com.javatpoint" />
16.
17.    <!--Provide support for conversion, formatting and validation -->
18.    <mvc:annotation-driven/>
19.
20. </beans>
```

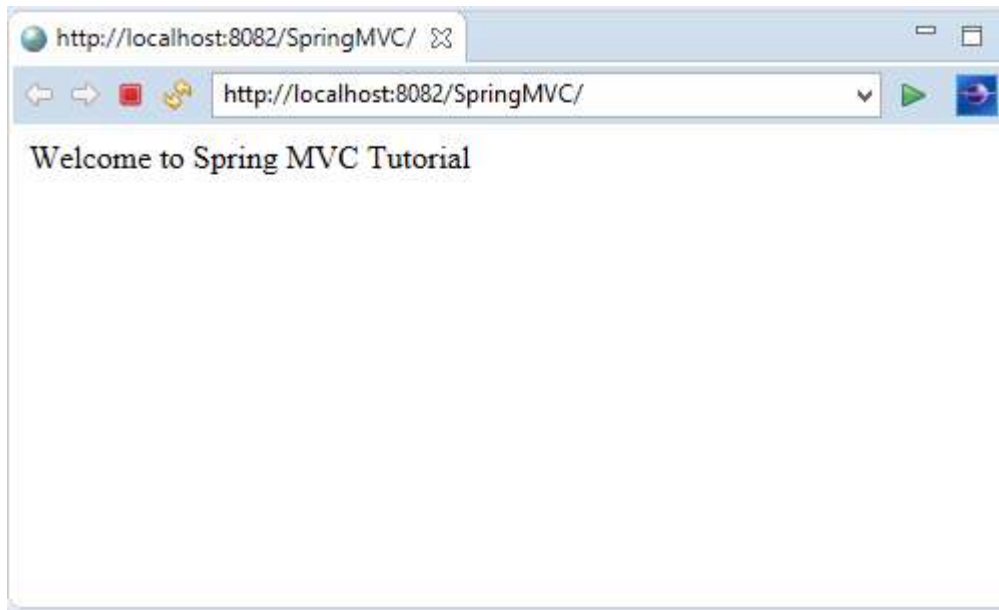
5. Display the message in the JSP page

This is the simple JSP page, displaying the message returned by the Controller.

index.jsp

```
1. <html>
2. <body>
3. <p>Welcome to Spring MVC Tutorial</p>
4. </body>
5. </html>
```

Output:



RESTful API using Spring Framework

Building REST services with Spring

REST has quickly become the de-facto standard for building web services on the web because they're easy to build and easy to consume.

There's a much larger discussion to be had about how REST fits in the world of microservices, but — for this tutorial — let's just look at building RESTful services.

Why REST? REST embraces the precepts of the web, including its architecture, benefits, and everything else. This is no surprise given its author, Roy Fielding, was involved in probably a dozen specs which govern how the web operates.

What benefits? The web and its core protocol, HTTP, provide a stack of features:

- Suitable actions (GET, POST, PUT, DELETE, ...)
- Caching
- Redirection and forwarding
- Security (encryption and authentication)

These are all critical factors on building resilient services. But that is not all. The web is built out of lots of tiny specs, hence it's been able to evolve easily, without getting bogged down in "standards wars".

Developers are able to draw upon 3rd party toolkits that implement these diverse specs and

instantly have both client and server technology at their fingertips.

By building on top of HTTP, REST APIs provide the means to build:

- Backwards compatible APIs
- Evolvable APIs
- Scaleable services
- Securable services
- A spectrum of stateless to stateful services

What's important to realize is that REST, however ubiquitous, is not a standard, *per se*, but an approach, a style, a set of *constraints* on your architecture that can help you build web-scale systems. In this tutorial we will use the Spring portfolio to build a RESTful service while leveraging the stackless features of REST.

Getting Started

As we work through this tutorial, we'll use [Spring Boot](#). Go to [Spring Initializr](#) and add the following dependencies to a project:

- Web
- JPA
- H2

Change the Name to "Payroll" and then choose "Generate Project". A .zip will download. Unzip it. Inside you'll find a simple, Maven-based project including a pom.xml build file (NOTE: You *can* use Gradle. The examples in this tutorial will be Maven-based.)

Spring Boot can work with any IDE. You can use Eclipse, IntelliJ IDEA, Netbeans, etc. [The Spring Tool Suite](#) is an open-source, Eclipse-based IDE distribution that provides a superset of the Java EE distribution of Eclipse. It includes features that make working with Spring applications even easier. It is, by no means, required. But consider it if you want that extra **oomph** for your keystrokes. Here's a video demonstrating how to get started with STS and Spring Boot. This is a general introduction to familiarize you with the tools.

The Story so Far...

Let's start off with the simplest thing we can construct. In fact, to make it as simple as possible, we can even leave out the concepts of REST. (Later on, we'll add REST to understand the difference.)

Big picture: We're going to create a simple payroll service that manages the employees of a company. We'll store employee objects in a (H2 in-memory) database, and access them (via something called JPA). Then we'll wrap that with something that will allow access over the internet (called the Spring MVC layer).

The following code defines an Employee in our system.

nonrest/src/main/java/payroll/Employee.java

package payroll;

import java.util.Objects;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

@Entity

class Employee{

private @Id @GeneratedValue Long id;

private String name;

private String role;

Employee(){}

Employee(String name,String role){

this.name = name;

this.role= role;

}

public Long getId(){

return this.id;

}

public String getName(){

return this.name;

}

public String getRole(){

return this.role;

}

public void setId(Long id){

this.id = id;

}

public void setName(String name){

this.name = name;

}

public void setRole(String role){

this.role= role;

}

@Override

public boolean equals(Object o){

if(this== o)

return true;

if(!(o instanceof Employee))

return false;

Employee employee=(Employee) o;

return Objects.equals(this.id, employee.id)&&Objects.equals(this.name, employee.name)

&&Objects.equals(this.role,employee.role);

}

```

@Override
public int hashCode(){
return Objects.hash(this.id,this.name,this.role);
}

@Override
public String toString(){
return "Employee{"+"id="+this.id+", name='"+this.name+"'", role='"+this.role+"'+'}';
}
}

```

Despite being small, this Java class contains much:

- @Entity is a JPA annotation to make this object ready for storage in a JPA-based data store.
- id, name, and role are attributes of our Employee [domain object](#). id is marked with more JPA annotations to indicate it's the primary key and automatically populated by the JPA provider.
- a custom constructor is created when we need to create a new instance, but don't yet have an id.

With this domain object definition, we can now turn to [Spring Data JPA](#) to handle the tedious database interactions.

Spring Data JPA repositories are interfaces with methods supporting creating, reading, updating, and deleting records against a back end data store. Some repositories also support data paging, and sorting, where appropriate. Spring Data synthesizes implementations based on conventions found in the naming of the methods in the interface.

There are multiple repository implementations besides JPA. You can use Spring Data MongoDB, Spring Data GemFire, Spring Data Cassandra, etc. For this tutorial, we'll stick with JPA.

Spring makes accessing data easy. By simply declaring the following EmployeeRepository interface we automatically will be able to

- Create new Employees
- Update existing ones
- Delete Employees
- Find Employees (one, all, or search by simple or complex properties)

nonrest/src/main/java/payroll/EmployeeRepository.java

```

package payroll;

import org.springframework.data.jpa.repository.JpaRepository;

interface EmployeeRepository extends JpaRepository<Employee, Long>{

}

```

To get all this free functionality, all we had to do was declare an interface which extends

Spring Data JPA's `JpaRepository`, specifying the domain type as `Employee` and the id type as `Long`.

Spring Data's [repository solution](#) makes it possible to sidestep data store specifics and instead solve a majority of problems using domain-specific terminology.

Believe it or not, this is enough to launch an application! A Spring Boot application is, at a minimum, a public static void main entry-point and the `@SpringBootApplication` annotation. This tells Spring Boot to help out, wherever possible.

```
nonrest/src/main/java/payroll/PayrollApplication.java
```

```
package payroll;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class PayrollApplication {
```

```
    public static void main(String... args) {
        SpringApplication.run(PayrollApplication.class, args);
    }
}
```

`@SpringBootApplication` is a meta-annotation that pulls in **component scanning**, **autoconfiguration**, and **property support**. We won't dive into the details of Spring Boot in this tutorial, but in essence, it will fire up a servlet container and serve up our service.

Nevertheless, an application with no data isn't very interesting, so let's preload it. The following class will get loaded automatically by Spring:

```
nonrest/src/main/java/payroll/LoadDatabase.java
```

```
package payroll;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
class LoadDatabase {
```

```
    private static final Logger log = LoggerFactory.getLogger(LoadDatabase.class);
```

```
    @Bean
    CommandLineRunner initDatabase(EmployeeRepository repository) {
```

```
        return args -> {
            log.info("Preloading " + repository.save(new Employee("Bilbo Baggins", "burglar")));
            log.info("Preloading " + repository.save(new Employee("Frodo Baggins", "thief")));
        };
    }
}
```

What happens when it gets loaded?

- Spring Boot will run ALL CommandLineRunner beans once the application context is loaded.
- This runner will request a copy of the EmployeeRepository you just created.
- Using it, it will create two entities and store them.

Right-click and **Run** PayRollApplication, and this is what you get:

Fragment of console output showing preloading of data

```
...
2018-08-09 11:36:26.169 INFO 74611 --- [main] payroll.LoadDatabase : Preloading Employee(id=1,
name=Bilbo Baggins, role=burglar)
2018-08-09 11:36:26.174 INFO 74611 --- [main] payroll.LoadDatabase : Preloading Employee(id=2,
name=Frodo Baggins, role=thief)
...
```

This isn't the **whole** log, but just the key bits of preloading data. (Indeed, check out the whole console. It's glorious.)

HTTP is the Platform

To wrap your repository with a web layer, you must turn to Spring MVC. Thanks to Spring Boot, there is little in infrastructure to code. Instead, we can focus on actions:

```
nonrest/src/main/java/payroll/EmployeeController.java

package payroll;

import java.util.List;

import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
class EmployeeController {

    private final EmployeeRepository repository;

    EmployeeController(EmployeeRepository repository) {
        this.repository = repository;
    }

    // Aggregate root
    // tag::get-aggregate-root[]
    @GetMapping("/employees")
    List<Employee> all() {
        return repository.findAll();
    }
}
```



```

}
// end::get-aggregate-root[]

@PostMapping("/employees")
Employee newEmployee(@RequestBody Employee newEmployee){
    return repository.save(newEmployee);
}

// Single item

@GetMapping("/employees/{id}")
Employee one(@PathVariable Long id){

    return repository.findById(id)
        .orElseThrow(() -> new EmployeeNotFoundException(id));
}

@PutMapping("/employees/{id}")
Employee replaceEmployee(@RequestBody Employee newEmployee, @PathVariable Long id){

    return repository.findById(id)
        .map(employee -> {
            employee.setName(newEmployee.getName());
            employee.setRole(newEmployee.getRole());
            return repository.save(employee);
        })
        .orElseGet(() -> {
            newEmployee.setId(id);
            return repository.save(newEmployee);
        });
}

@DeleteMapping("/employees/{id}")
void deleteEmployee(@PathVariable Long id){
    repository.deleteById(id);
}
}

```

- @RestController indicates that the data returned by each method will be written straight into the response body instead of rendering a template.
- An EmployeeRepository is injected by constructor into the controller.
- We have routes for each operation (@GetMapping, @PostMapping, @PutMapping and @DeleteMapping, corresponding to HTTP GET, POST, PUT, and DELETE calls). (NOTE: It's useful to read each method and understand what they do.)
- EmployeeNotFoundException is an exception used to indicate when an employee is looked up but not found.

nonrest/src/main/java/payroll/EmployeeNotFoundException.java

package payroll;

class EmployeeNotFoundException extends RuntimeException{

```

    EmployeeNotFoundException(Long id){
        super("Could not find employee "+ id);
    }
}

```

When an `EmployeeNotFoundException` is thrown, this extra tidbit of Spring MVC configuration is used to render an **HTTP 404**:

```
nonrest/src/main/java/payroll/EmployeeNotFoundAdvice.java
```

```
package payroll;
```

```
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
```

```
@ControllerAdvice
```

```
class EmployeeNotFoundAdvice{
```

```
    @ResponseBody
```

```
    @ExceptionHandler(EmployeeNotFoundException.class)
```

```
    @ResponseStatus(HttpStatus.NOT_FOUND)
```

```
    String employeeNotFoundHandler(EmployeeNotFoundException ex){
```

```
        return ex.getMessage();
```

```
    }
```

```
}
```

- `@ResponseBody` signals that this advice is rendered straight into the response body.
- `@ExceptionHandler` configures the advice to only respond if an `EmployeeNotFoundException` is thrown.
- `@ResponseStatus` says to issue an `HttpStatus.NOT_FOUND`, i.e. an **HTTP 404**.
- The body of the advice generates the content. In this case, it gives the message of the exception.

To launch the application, either right-click the public static void main in `PayRollApplication` and select **Run** from your IDE, or:

Spring Initializr uses maven wrapper so type this:

```
$ ./mvnw clean spring-boot:run
```

Alternatively using your installed maven version type this:

```
$ mvn clean spring-boot:run
```

When the app starts, we can immediately interrogate it.

```
$ curl -v localhost:8080/employees
```

This will yield:

```
* Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> GET /employees HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.54.0
```

```
> Accept: */*
>
< HTTP/1.1 200
< Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked
< Date: Thu, 09 Aug 2018 17:58:00 GMT
<
* Connection #0 to host localhost left intact
[{"id":1,"name":"Bilbo Baggins","role":"burglar"}, {"id":2,"name":"Frodo Baggins","role":"thief"}]
```

Here you can see the pre-loaded data, in a compacted format.

If you try and query a user that doesn't exist...

```
$ curl -v localhost:8080/employees/99
```

You get...

```
* Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> GET /employees/99 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 404
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 26
< Date: Thu, 09 Aug 2018 18:00:56 GMT
<
* Connection #0 to host localhost left intact
Could not find employee 99
```

This message nicely shows an **HTTP 404** error with the custom message **Could not find employee 99**.

It's not hard to show the currently coded interactions...

If you are using Windows Command Prompt to issue cURL commands, chances are the below command won't work properly. You must either pick a terminal that support single quoted arguments, or use double quotes and then escape the ones inside the JSON.

To create a new Employee record we use the following command in a terminal—the \$ at the beginning signifies that what follows it is a terminal command:

```
$ curl -X POST localhost:8080/employees -H 'Content-type:application/json' -d '{"name": "Samwise Gamgee", "role": "gardener"}'
```

Then it stores newly created employee and sends it back to us:

```
{"id":3,"name":"Samwise Gamgee","role":"gardener"}
```

You can update the user. Let's change his role.

```
$ curl -X PUT localhost:8080/employees/3 -H 'Content-type:application/json' -d '{"name": "Samwise Gamgee", "role": "ring bearer"}'
```

And we can see the change reflected in the output.

```
{"id":3,"name":"Samwise Gamgee","role":"ring bearer"}
```

The way you construct your service can have significant impacts. In this situation, we said **update**, but **replace** is a better description. For example, if the name was NOT provided, it would instead get nulled out.

Finally, you can delete users like this:

```
$ curl -X DELETE localhost:8080/employees/3
```

```
# Now if we look again, it's gone
$ curl localhost:8080/employees/3
Could not find employee 3
```

This is all well and good, but do we have a RESTful service yet? (If you didn't catch the hint, the answer is no.)

What's missing?

What makes something RESTful?

So far, you have a web-based service that handles the core operations involving employee data. But that's not enough to make things "RESTful".

- Pretty URLs like /employees/3 aren't REST.
- Merely using GET, POST, etc. isn't REST.
- Having all the CRUD operations laid out isn't REST.

In fact, what we have built so far is better described as **RPC (Remote Procedure Call)**. That's because there is no way to know how to interact with this service. If you published this today, you'd also have to write a document or host a developer's portal somewhere with all the details.

This statement of Roy Fielding's may further lend a clue to the difference between **REST** and **RPC**:

I am getting frustrated by the number of people calling any HTTP-based interface a REST API. Today's example is the SocialSite REST API. That is RPC. It screams RPC. There is so much coupling on display that it should be given an X rating.

What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is

there some broken manual somewhere that needs to be fixed?

— Roy Fielding

<https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>

The side effect of NOT including hypermedia in our representations is that clients **MUST** hard code URIs to navigate the API. This leads to the same brittle nature that predated the rise of e-commerce on the web. It's a signal that our JSON output needs a little help.

Introducing [Spring HATEOAS](#), a Spring project aimed at helping you write hypermedia-driven outputs. To upgrade your service to being RESTful, add this to your build:

Adding Spring HATEOAS to dependencies section of pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>
```

This tiny library will give us the constructs to define a RESTful service and then render it in an acceptable format for client consumption.

A critical ingredient to any RESTful service is adding [links](#) to relevant operations. To make your controller more RESTful, add links like this:

Getting a single item resource

```
@GetMapping("/employees/{id}")
EntityModel<Employee>one(@PathVariableLong id){

    Employee employee=repository.findById(id)//
    .orElseThrow(()->new EmployeeNotFoundException(id));

    return EntityModel.of(employee,//
        linkTo(methodOn(EmployeeController.class).one(id)).withSelfRel(),
        linkTo(methodOn(EmployeeController.class).all()).withRel("employees"));
}
```

This tutorial is based on Spring MVC and uses the static helper methods from `WebMvcLinkBuilder` to build these links. If you are using Spring WebFlux in your project, you must instead use `WebFluxLinkBuilder`.

This is very similar to what we had before, but a few things have changed:

- The return type of the method has changed from `Employee` to `EntityModel<Employee>`. `EntityModel<T>` is a generic container from Spring HATEOAS that includes not only the data but a collection of links.
- `linkTo(methodOn(EmployeeController.class).one(id)).withSelfRel()` asks that Spring HATEOAS build a link to the `EmployeeController` 's `one()` method, and flag it as a [self](#) link.
- `linkTo(methodOn(EmployeeController.class).all()).withRel("employees")` asks Spring HATEOAS to build a link to the aggregate root, `all()`, and call it "employees".

What do we mean by "build a link"? One of Spring HATEOAS's core types is `Link`. It includes a **URI** and a **rel** (relation). Links are what empower the web. Before the World Wide

Web, other document systems would render information or links, but it was the linking of documents WITH this kind of relationship metadata that stitched the web together.

Roy Fielding encourages building APIs with the same techniques that made the web successful, and links are one of them.

If you restart the application and query the employee record of *Bilbo*, you'll get a slightly different response than earlier:

Curling prettier

When your curl output gets more complex it can become hard to read. Use this or [other tips](#) to prettify the json returned by curl:

```
# The indicated part pipes the output to json_pp and asks it to make your JSON pretty. (Or use whatever tool you like!)
#                               v-----v
curl -v localhost:8080/employees/1 | json_pp
RESTful representation of a single employee
```

```
{
  "id":1,
  "name":"Bilbo Baggins",
  "role":"burglar",
  "_links":{"
    "self":{"
      "href":"http://localhost:8080/employees/1"
    },
    "employees":{"
      "href":"http://localhost:8080/employees"
    }
  }
}
```

This decompressed output shows not only the data elements you saw earlier (id, name and role), but also a `_links` entry containing two URIs. This entire document is formatted using [HAL](#).

HAL is a lightweight [mediatype](#) that allows encoding not just data but also hypermedia controls, alerting consumers to other parts of the API they can navigate toward. In this case, there is a "self" link (kind of like a this statement in code) along with a link back to the [aggregate root](#).

To make the aggregate root ALSO more RESTful, you want to include top level links while ALSO including any RESTful components within.

So we turn this

Getting an aggregate root

```
@GetMapping("/employees")
List<Employee>all(){
    return repository.findAll();
}
```

into this

Getting an aggregate root **resource**

```
@GetMapping("/employees")
CollectionModel<EntityModel<Employee>>all(){

    List<EntityModel<Employee>> employees =repository.findAll().stream()
    .map(employee ->EntityModel.of(employee,
        linkTo(methodOn(EmployeeController.class).one(employee.getId())).withSelfRel(),
        linkTo(methodOn(EmployeeController.class).all()).withRel("employees")))
    .collect(Collectors.toList());

    returnCollectionModel.of(employees,linkTo(methodOn(EmployeeController.class).all()).withSelfRel());
}
```

Wow! That method, which used to just be repository.findAll(), is all grown up! Not to worry. Let's unpack it.

CollectionModel<> is another Spring HATEOAS container; it's aimed at en

Building an application usingMaven

[Maven](#) is one of the open-source [Java build tools](#) developed by Apache Software Foundation. It can compile, test, and package a java program into .jar or .war format.

Maven makes use of the pom.xml file to build java projects.

Project Object Model (POM) is an XML file that contains the java project details, configurations, and settings required for maven to build the project.

The **pom.xml** file is present in the root of the java project directory. Primarily it contains the project dependencies.

For example, when a developer wants to implement a [PostgreSQL](#) database connectivity functionality, he will make use of the [PostgreSQL JDBC Driver](#) dependency from the maven repository by adding it to the pom.xml file.

So when you build the code with maven, it reads the pom.xml file and downloads all the dependencies from the maven repository. Dependencies could be third-party libraries from the public Maven Repository or common libraries hosted within an organization's private maven repository. You can compare it with Python pip, Nodejs npm, or Ruby gems

Commonly organizations use Sonatype nexus as a [private hosted maven repository](#).

By default, maven uses the public repository but if you have in-house private maven repositories, you configure custom maven repository URLs in settings.xml maven configuration present in the maven installation directory. for example, /opt/apache-maven-3.8.6/conf/settings.xml

Maven Prerequisites

For maven to work you need the following installed on your system

1. Java JDK
2. Maven

To install and configure JDK and maven, follow the [maven installation guide](#).

Build Java Application Using Maven

For this example, we will be using the open-source **java spring boot application** named pet-clinic.

First, clone the application to your development machine or server.

```
git clone https://github.com/spring-projects/spring-petclinic.git
```

The code base has the following important folders and files. It is common in real-time project code as well.

1. **/src folder:** This folder contains the source code based on the java spring framework.
2. **/src/tests folder:** This folder contains the unit tests & integration tests of the code under the tests folder.
3. **pom.xml file:** It contains all the dependencies required for the pet-clinic applications. As it is an open-source application, all the dependencies are from the public maven repository.

To build the project, cd into the project root directory. In my case its spring-petclinic. It should contain the pom.xml file

```
cd spring-petclinic
```

From a CI perspective, we just have to **build, test, and package** the project to create a deployable artifact(jar file)


So commonly in the CI process, we build and package the java projects using the following maven command. It compiles the code, tests it, package it as a jar file in the target folder, and will also install(copy) the jar package in the local .m2 repository.

```
mvn clean install
```

After executing the above command, you will see a folder named **target** in the root directory. Inside the target directory, you will see the packaged jar file as shown below. We call it a deployable artifact.


```
vagrant@ubuntu-focal: ~/spring-petclinic/target$ tree -L 1
.
├── checkstyle-cachefile
├── checkstyle-checker.xml
├── checkstyle-header.txt
├── checkstyle-result.xml
├── checkstyle-suppressions.xml
├── classes
├── generated-sources
├── generated-test-sources
├── jacoco.exec
├── maven-archiver
├── maven-status
├── site
├── spring-petclinic-2.7.3.jar
├── spring-petclinic-2.7.3.jar.original
├── surefire-reports
└── test-classes

8 directories, 8 files
vagrant@ubuntu-focal: ~/spring-petclinic/target$
```



Even time you run **mvn clean install**, it deletes target directory and packages from the local **.m2** repository and replaces it with the latest build files and packages.

If you want to skip the test during build, you can add the **-Dmaven.test.skip=true** parameter as shown below.

```
mvn clean install -Dmaven.test.skip=true
```

Now that you have understood how to build a java project using maven, let's look into the maven lifecycle. Few commands we don't have to use in the CI pipelines. However, it is good to know about the maven lifecycle commands and you can use them depending on your CI pipeline requirement.

Maven Lifecycle Explained

Let's take a look at each maven lifecycle phase in order. Each phase executes all the phases before it. For example, if you execute the third phase, one, two, and three get executed.

1. Maven Validate (mvn validate)

mvn validate validates the maven project. It downloads all the required dependencies to the

local **.m2** repository.

2. Maven Compile (mvn compile)

mvn compile compiles the java project. It runs validate first and then compiles the code.

3. Maven Test (mvn test)

mvn test command runs the unit test that is part of the code. You can test classes individually, methods individually, or add patterns to run tests on all methods that match the pattern.

4. Maven Package (mvn package)

mvn package commands compile the code, test it and finally package it in the required format (jar or war)

5. Maven Verify (mvn verify)

mvn verify command runs all the phases explained before in order and runs checks on integration tests and [checkstyles](#) if they are defined in the project.

6. Maven Install (mvn install)

mvn install command installs the packaged code in the local maven repository.

7. Maven Deploy (mvn deploy)

mvn deploy command, deploys the package to the remote maven repository. When you run deploy, it first runs validate, compile, test, package, verify, install, and then finally deploys the package to the remote maven repository.

Possible Maven Build Errors

java.lang.IllegalStateException: Unable to load cache item

If maven doesn't support the Java version, you will get the above error.

To rectify it, install the latest maven version that supports the installed Java version.

```
vagrant@ubuntu-focal:~/spring-petclinic$ mvn dependency:resolve
[ERROR] Error executing Maven.
[ERROR] java.lang.IllegalStateException: Unable to load cache item
[ERROR] Caused by: Unable to load cache item
[ERROR] Caused by: Could not initialize class com.google.inject.internal.
vagrant@ubuntu-focal:~/spring-petclinic$
```

If you try to execute the maven command from the location where there is no **pom.xml** file,

you will get the following error.

The goal you specified requires a project to execute but there is no POM in this directory

To rectify this, execute the maven command from the folder that has the pom.xml file.

Maven Build FAQs

Does mvn package run tests?

Yes. By default, the mvn package command runs the test. However, you can add the flag -Dmaven.test.skip to skip the tests.

What does Maven test do?

mvn test runs all the unit tests for the java project.

Conclusion

As a [Devops engineer](#), it is very important to understand the java build process if you are working on deploying java projects.

UNIT – V

Databases & Deployment

Functional dependency defines the relationship of two or more attributes, typically between the primary key and non-key attributes of another table. It is also defined by the relation of one attribute to another attribute in DBMS.

$\text{empId} \rightarrow \{ \text{empName}, \text{skill}, \text{dependent}, \text{contract}, \text{project} \}$, \rightarrow Here, empId can determine or defines the values of fields empName, dependent, contract and employee project

Username Tables:

$\text{userName} \rightarrow \text{dateCreate}$ here if we can know the userName like we have email account if we know the email Id of user then there is possibility to find the date when account was created.

Multivalued Dependency:

Multivalency Dependency occurs in such a condition or time when two or more attributes in table are independent to each other but, both of them depend upon the third attributes.

Employee Table:

The attributes like empName, skill, dependent, contract, project all are independent of each other means not depends on one another but depends upon empId example empName can determine skill, or any other employee attribute because there can be or even more than one employee with same name or constraints.

$\text{empId} \twoheadrightarrow \text{skill}$ $\text{empId} \twoheadrightarrow \text{contract}$ $\text{empId} \twoheadrightarrow \text{project}$ $\text{empId} \twoheadrightarrow \text{dependent}$

These all of the columns is the multivalued dependency on the empId

Username Table:

We only have two attributes here, but there are no multiple attributes that are independent of each other but rely solely on the third variable.

$\text{userName} \rightarrow \text{dateCreate}$ here dateCreate is an attribute that depends or relate upon

the userName only dateCreate when there is not sufficient to find anything. b)

Minimal key is the minimum no of attributes which can find out other attributes of a table

i.e., a primary key or the candidate key. **In the Context of Employee Table:**

empId \rightarrow { empName, skill, dependent, contract, project }

In the Context of Username Table:

userName \rightarrow dateCreate

In the Context of Subject Table:

Consider Subject table which has sub_Code, subName sub_Code \rightarrow subName

In the Context of Enrollment Table:

Considering the enrollment table which has the attributes like: enrollment Id, Name of employee, field in which employee enrolled and date

c)

We have the following Employee and Username Tables:

In Context of Employee Table:

Employee table is not in normalize or the normal forms. Because the Attributes in it like: Skill, project, contract and dependent attributes might have one or more values. According to the 1NF principle every field must contain the atomic values if they don't have the atomic value. There is need to decompose the table since the table should have the 1 value in each field.

In the Context of Username Table:

It is normalizing one Since it has two fields[UserName and dateCreate] in which both have atomic values or data , is fully functional dependent, no transition dependency etc.

d)

Normalization, Decomposition process will be done.

Normalization is a process or technique of organizing or collecting the data in database. It is mainly done for two purposes: Eliminating the redundancy or even the useless data

In 1st NF:

Every field must contain the single atomic value and the attribute like: skill, project, contract and dependent attribute has one or more than the decompose table so that the each and every field has atomic value which will increase the number of tuples in the table name "employee".

In the 2nd NF:

Each table should be at 1st NF.

- There should not be any functional dependency. So, in this case, after it is in 1st NF table is in 2nd NF Since the empId can find out all the attributes of the employees.

In 3rd NF:

- Table should be at 2nd NF Form.
- There should not be any transitive dependency in the table in which the non-primitive attribute can find another non-primitive attribute i.e., empName, skill, dependent, project is the non-primitive attribute and they cannot find the each other but the main prime attribute can or able to find all of them.

In BCNF:

- Table should be at 3rd NF.
- The LHS Side of attribute should have the candidate key or the super key. • So, In this case $\text{empId} \rightarrow \{\text{empName, skill, dependent, contract, project}\}$, The attribute empId is a primary key and can find out all other attributes.

In the 4th NF:

Table should be at BCNF Form.

There should not be any multivalued Dependency.

So, in current Employee Table context, Employee might contain the multivalued dependency I.e.: skills, projects [0 or more], dependency [0 or more] and contract [1 or more]. So, there is lots of multivalued attributes or dependency on the empId which might increase the no of entries in the table which might increase the no of entries in the table after making it to 1stNF.

In the case of making the Employee table in 4th NF, it will decompose the employee table into following tables: $\text{empId} \rightarrow \{\text{empName, skill, dependent, contract, project}\}$

EmployeeSkills

empId, empName,

skill

EmployeeDependency empId, empName, dependent **EmployeeContract** empId, empName,

contract

EmployeeProject empId, empName,

project

There is no need to change the Username Table since it is already on 4th NF.

Structured Query Language

SQL Tutorial

SQL tutorial provides basic and advanced concepts of SQL. Our SQL tutorial is designed for both beginners and professionals.

SQL (Structured Query Language) is used to perform operations on the records stored in the database, such as updating records, inserting records, deleting records, creating and modifying database tables, views, etc.

SQL is not a database system, but it is a query language.

Suppose you want to perform the queries of SQL language on the stored data in the database. You are required to install any database management system in your systems, for example, [Oracle](#), [MySQL](#), [MongoDB](#), [PostgreSQL](#), [SQL Server](#), [DB2](#), etc.

What is SQL?

SQL is a short-form of the structured query language, and it is pronounced as S-Q-L or sometimes as See-Quell.

This database language is mainly designed for maintaining the data in relational database management systems. It is a special tool used by data professionals for handling structured data (data which is stored in the form of tables). It is also designed for stream processing in RDSMS.

You can easily create and manipulate the database, access and modify the table rows and columns, etc. This query language became the standard of ANSI in the year of 1986 and ISO in the year of 1987.

If you want to get a job in the field of data science, then it is the most important query language to learn. Big enterprises like Facebook, Instagram, and LinkedIn, use SQL for storing the data in the back-end.

Why SQL?

Nowadays, SQL is widely used in data science and analytics. Following are the reasons which explain why it is widely used:

- The basic use of SQL for data professionals and SQL users is to insert, update, and delete the data from the relational database.
- SQL allows the data professionals and users to retrieve the data from the relational database

management systems.

- It also helps them to describe the structured data.
- It allows SQL users to create, drop, and manipulate the database and its tables.
- It also helps in creating the view, stored procedure, and functions in the relational database.
- It allows you to define the data and modify that stored data in the relational database.
- It also allows SQL users to set the permissions or constraints on table columns, views, and stored procedures.

History of SQL

"A Relational Model of Data for Large Shared Data Banks" was a paper which was published by the great computer scientist "E.F. Codd" in 1970.

The IBM researchers Raymond Boyce and Donald Chamberlin originally developed the SEQUEL (Structured English Query Language) after learning from the paper given by E.F. Codd. They both developed the SQL at the San Jose Research laboratory of IBM Corporation in 1970.

At the end of the 1970s, relational software Inc. developed their own first SQL using the concepts of E.F. Codd, Raymond Boyce, and Donald Chamberlin. This SQL was totally based on RDBMS. Relational Software Inc., which is now known as Oracle Corporation, introduced the Oracle V2 in June 1979, which is the first implementation of SQL language. This Oracle V2 version operates on VAX computers.

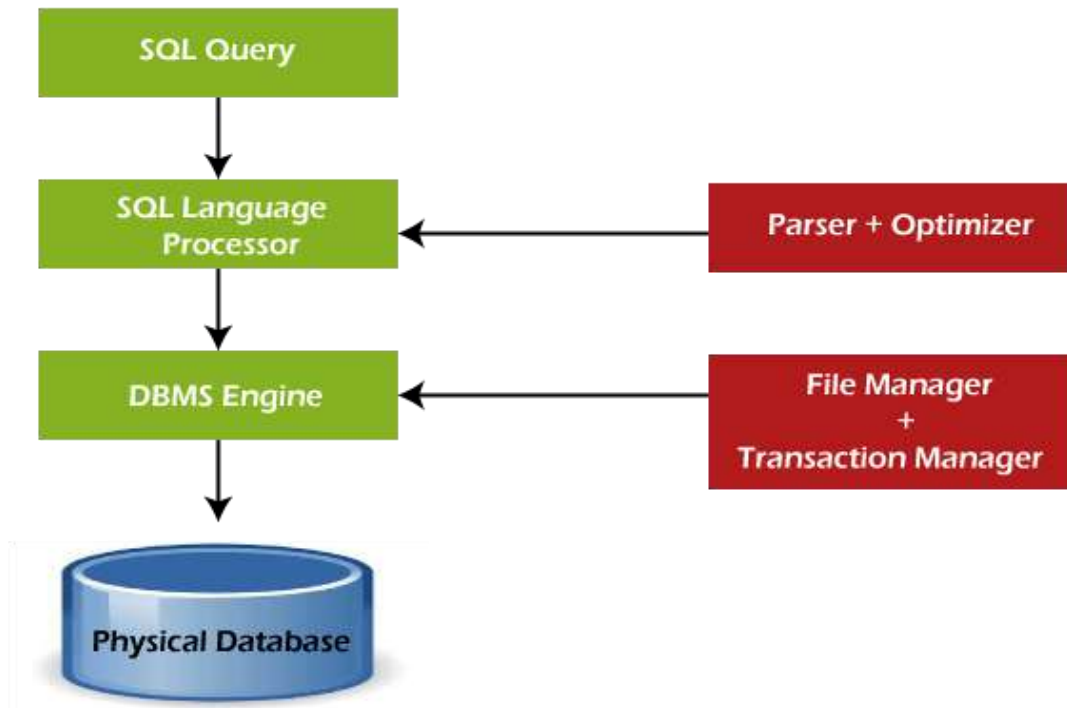
Process of SQL

When we are executing the command of SQL on any Relational database management system, then the system automatically finds the best routine to carry out our request, and the SQL engine determines how to interpret that particular command.

Structured Query Language contains the following four components in its process:

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

A classic query engine allows data professionals and users to maintain non-SQL queries. The architecture of SQL is shown in the following diagram:



Some SQL Commands

The SQL commands help in creating and managing the database. The most common SQL commands which are highly used are mentioned below:

1. CREATE command
2. UPDATE command
3. DELETE command
4. SELECT command
5. DROP command
6. INSERT command

CREATE Command

This command helps in creating the new database, new table, table view, and other objects of the database.

UPDATE Command

This command helps in updating or changing the stored data in the database.

DELETE Command

This command helps in removing or erasing the saved records from the database tables. It erases single or multiple tuples from the tables of the database.

SELECT Command

This command helps in accessing the single or multiple rows from one or multiple tables of the database. We can also use this command with the WHERE clause.

DROP Command

This command helps in deleting the entire table, table view, and other objects from the database.

INSERT Command

This command helps in inserting the data or records into the database tables. We can easily insert the records in single as well as multiple rows of the table.

SQL vs No-SQL



The following table describes the [differences between the SQL and NoSQL](#), which are necessary to understand:

SQL	No-SQL
1. SQL is a relational database management system.	1. While No-SQL is a non-relational or distributed database management system.
2. The query language used in this database system is a structured query language.	2. The query language used in the No-SQL database systems is a non-declarative query language.
3. The schema of SQL databases is predefined, fixed, and static.	3. The schema of No-SQL databases is a dynamic schema for unstructured data.
4. These databases are vertically scalable.	4. These databases are horizontally scalable.
5. The database type of SQL is in the form of tables, i.e., in the form of rows and columns.	5. The database type of No-SQL is in the form of documents, key-value, and graphs.
6. It follows the ACID model.	6. It follows the BASE model.

-
- | | |
|--|---|
| 7. Complex queries are easily managed in the SQL database. | 7. NoSQL databases cannot handle complex queries. |
| 8. This database is not the best choice for storing hierarchical data. | 8. While No-SQL database is a perfect option for storing hierarchical data. |
| 9. All SQL databases require object-relational mapping. | 9. Many No-SQL databases do not require object-relational mapping. |
| 10. Gauges, CircleCI, Hootsuite, etc., are the top enterprises that are using this query language. | 10. Airbnb, Uber, and Kickstarter are the top enterprises that are using this query language. |
| 11. SQLite, Ms-SQL, Oracle, PostgreSQL, and MySQL are examples of SQL database systems. | 11. Redis, MongoDB, Hbase, BigTable, CouchDB, and Cassandra are examples of NoSQL database systems. |

Advantages of SQL

SQL provides various advantages which make it more popular in the field of data science. It is a perfect query language which allows data professionals and users to communicate with the database. Following are the best advantages or benefits of Structured Query Language:

1. No programming needed

SQL does not require a large number of coding lines for managing the database systems. We can easily access and maintain the database by using simple SQL syntactical rules. These simple rules make the SQL user-friendly.

2. High-Speed Query Processing

A large amount of data is accessed quickly and efficiently from the database by using SQL queries. Insertion, deletion, and updation operations on data are also performed in less time.

3. Standardized Language

SQL follows the long-established standards of ISO and ANSI, which offer a uniform platform across the globe to all its users.

4. Portability

The structured query language can be easily used in desktop computers, laptops, tablets, and even smartphones. It can also be used with other applications according to the user's requirements.

5. Interactive language

We can easily learn and understand the SQL language. We can also use this language for communicating with the database because it is a simple query language. This language is also

used for receiving the answers to complex queries in a few seconds.

6. More than one Data View

The SQL language also helps in making the multiple views of the database structure for the different database users.

Disadvantages of SQL

With the advantages of SQL, it also has some disadvantages, which are as follows:

1. Cost

The operation cost of some SQL versions is high. That's why some programmers cannot use the Structured Query Language.

2. Interface is Complex

Another big disadvantage is that the interface of Structured query language is difficult, which makes it difficult for SQL users to use and manage it.

3. Partial Database control

The business rules are hidden. So, the data professionals and users who are using this query language cannot have full database control.

Data persistence using Spring

I'm used to using Spring Roo to generate my entities and having it handle injecting the entityManager as well as the persist and other methods via AspectJ classes. Now I'm trying to use Spring Boot to do something simple that will write things to the database ...

```
@Entity
@Table(name = "account")
public class Account {

    transient EntityManager entityManager;

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "username", nullable = false, unique = true)
    private String username;

    @Column(name = "password", nullable = false)
    private String password;

    ... getters and setters

    @Transactional
    public void persist() {
```

```

if (this.entityManager == null) this.entityManager = entityManager();
this.entityManager.persist(this);
}

```

```

@Transactional
public Account merge() {
if (this.entityManager == null) this.entityManager = entityManager();
Accountmerged=this.entityManager.merge(this);
this.entityManager.flush();
return merged;
}

```

When I'm calling persist or merge, entityManager is obviously null.

I've also tried adding implements CrudRepository<Account, Long> to the Account class to see it'll give me that functionality via a Default Implementation, but what I'm getting is simply empty classes that needs to be filled in.

I've had a look at the Spring Boot docs, they cover it very briefly omitting just enough detail to so that it's not obvious what I'm missing.

I have an Application class that bootstraps the application:

```

@Configuration
@ComponentScan
@EnableAutoConfiguration
public class Application {

    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class, args);
    }

}

```

My properties file looks like this:

```

spring.application.name: Test Application

spring.datasource.url: jdbc:mysql://localhost/test
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update

```

This database is automatically being created thanks to the ddl-auto=update property

What is the correct way to persist entities in Spring Boot + JPA and if what I've done is correct so far, how do I "autowire" or auto-create the entityManager?

JDBC Agile development principles

What are the Agile Principles?

There are 12 [agile](#) principles outlined in [The Agile Manifesto](#) in addition to the 4 agile values. These 12 principles for agile software development help establish the tenets of the agile mindset. They are not a set of rules for practicing agile, but a handful of principles to help instill agile thinking.

Below we will review each of the 12 agile principles and describe how they may be practiced.

Agile Principle 1

“Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.”

The best ways to ensure you make customers happy while continuously delivering valuable software are to ship early, iterate frequently, and listen to your market continually.

Unlike traditional approaches to product development, which have notoriously long development cycles, agile principles encourage minimizing the time between ideation and launch. The idea is to get a working product in the hands of customers as soon as possible. Doing this successfully means product managers are able to quickly get a [minimum viable product \(MVP\)](#) out and into the world and use it to get feedback from real customers. This feedback is then fed back into the product development process and used to inform future releases.

[Download the Product Development Roadmap Checklist →](#)

How it looks in practice:

- Product teams use minimum viable products and rapid experimentation to test hypothesis and validate ideas.
- Frequent releases help fuel a continuous feedback cycle between customer and product.
- Shipped and done are not the same thing. Instead of releasing a “finished” product, iterations continue to make incremental improvements to product based on customer and market feedback.

Agile Principle 2

“Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.”

In the world around us, change is the only constant. Agile principles and values support responding to these changes rather than moving forward in spite of them. Previous approaches to product development were often change adverse; detailed, well-documented

plans were made before development began and were set in stone regardless of new findings. Agile principles support observing changing markets, customer needs, and competitive threats and changing course when necessary.

How it looks in practice:

- Product teams are guided by high-level strategic goals and perhaps even [themes](#) below those goals. The product department's success is measured against progress toward those strategic goals rather than by delivery of a predefined feature set.
- Product constantly has its ear to the ground monitoring the market, customer feedback, and other factors which could influence product direction. When actionable insight is uncovered, plans are adjusted to better serve customer and business needs.
- Product strategy and tactical plans are reviewed, adjusted, and shared on a regular cadence to reflect changes and new findings. As such, product needs to manage the expectations of executive stakeholders appropriately and ensure they understand the *why* behind changes.

Agile Principle 3

“Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.”

Agile philosophy favors breaking a product's development into smaller components and “shipping” those components frequently. Using an agile approach, therefore—and building in more frequent mini-releases of your product—can speed the product's overall development.

This agile approach, with short-term development cycles of smaller portions of the product, results in less time spent drafting and poring over the large amounts of documentation that characterizes Waterfall product development. More importantly, this frequent-release approach creates more opportunities for you and your teams to validate your product ideas and strategies from the qualified constituencies who see each new release.

How it looks in practice:

- Agile development cycles, often called “sprints” or “iterations” break down product initiatives into smaller chunks that can be completed in a set timeframe. Often this timeframe is between 2 and 4 weeks which truly is a sprint if you consider the marathon-like development cycles waterfall teams often follow.
- Another popular alternative to agile sprints is continuous deployment. This method of shipping software frequently works less in terms of predetermined time boxes and more in terms of simply deciding what to do and doing it.

Agile Principle 4

“Business people and developers must work together daily throughout the project.”

Communication is a critical component of any project or team's success, and agile principles essentially mandate that it's a daily event. It takes a village to raise a child they say, and that applies to product as well.

A successful product requires insight from the business and technical sides of an organization which can only happen if these two teams work together consistently. Regular communication between business people and developers helps improve alignment across the organization by building trust and transparency.

How it looks in practice:

- Cross-functional agile product development teams include product people. This means that product is represented on the development team and bridges the gap between technical and business aspects of the product.
- Daily update meetings, or standups, are one technique many agile shops use to put this principle in practice and keep everyone connected.

Agile Principle 5

“Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.”

A key part of the agile philosophy is empowering individuals and teams through trust and autonomy. The agile team needs to be carefully built to include the right people and skill sets to get the job done, and responsibilities need to be clearly defined before the beginning of a project. Once the work has begun, however, there’s no place in agile for micromanagement or hand holding.

How it looks in practice:

- Product must clearly ensure engineering understands strategy and requirements before development starts. This means not only sharing user stories with the cross-functional team but also the bigger picture outlined in the product roadmap.
- Product is not responsible for explaining “how” something should be built. They need to share what and why, but it’s the delivery team’s job to determine the how. Furthermore, during sprints product does not micromanage outcome, instead they make themselves available to answer questions and provide support as needed.

Get Strategic Project Alignment →

Agile Principle 6

“The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.”

With so many distributed or [remote development teams](#) these days, this principle gets a bit of critique. But at the root of it, effective communication with developers means getting these conversations out of Slack and email and favoring more human interaction (even if done by video conference calls). The overall objective behind this principle is to encourage product people and developers to truly communicate in real time about the product, requirements, and the high-level strategy driving those things.

How it looks in practice:

- Daily standup meetings
- Collaborative [backlog grooming sessions](#)
- Sprint planning meetings
- Frequent demos
- Pair-programming

Agile Principle 7

“Working software is the primary measure of progress.”

Proponents of the agile philosophy are quick to remind us that we’re in the business of building software, and that’s where our time should be spent. Perfect, detailed documentation is secondary to working software. This mentality pushes to get products to the market quickly rather than let documentation or an “it’s not done until it’s perfect” mentality become a bottleneck. The ultimate measure for success is a working product that customers love.

How it looks in practice:

- Designing and releasing “Minimum Viable Features” rather than fully-developed feature sets means thinking first and foremost about the smallest things we can ship to start getting customer feedback and validate as we continue to build software.
- A fail fast mentality means moving forward even in times of uncertainty and testing ideas rapidly.
- Ship software often: a useful product now is better than a perfect one later.

Agile Principle 8

“Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.”

Keeping up with a demanding, rapid release schedule can be taxing on a team. Especially if expectations are set too high. Agile principles encourage us to be mindful of this and set realistic, clear expectations. The idea is to keep morale high and improve work-life balance to prevent burnout and turnover among members of cross functional teams.

How it looks in practice:

- Before every sprint, careful consideration of the amount of work that can be committed to is made. Development teams don’t over promise on what they can and cannot deliver. Effort estimations are a common practice in setting output expectations for development teams.
- Everyone agrees on what will get done during a sprint. Once a sprint has begun, no additional tasks are to be added except in rare cases.
- Product managers should act as gatekeepers to reduce the noise from other stakeholders and to avoid squeezing in additional unplanned work during an ongoing sprint.
- Product people should do their part in promoting a sense of psychological safety across the cross-functional team that encourages open communication and freely flowing feedback.

Agile Principle 9

“Continuous attention to technical excellence and good design enhances agility.”

While the agile philosophy encourages shorter cycles and frequent releases, it also puts emphasis on the importance of keeping things neat and tidy so they don’t cause problems in the future. Product managers often forget about this aspect of development because they mostly don’t spend their days wading through their products’ codebases, but it is still of the utmost importance to them.

How it looks in practice:

- The team needs to be cognizant of [technical debt](#) and the technical debt implications of any new features or initiatives added to the backlog. Developers and product need to work together to understand if and when technical debt is acceptable.
- On a regular basis, product will need to allocate development resources to refactoring efforts. Refactoring cannot be an afterthought, it needs to be an ongoing consideration.

Agile Principle 10

“Simplicity—the art of maximizing the amount of work not done—is essential.”

You’ve probably heard of the 80/20 rule—the concept that you can usually get 80% of your intended results with just 20% of the work. Agile principles encourage thinking this way; doing the things that can have the most impact. In a product management context this means having a laser sharp focus on organizational objectives and making some cutthroat [prioritization decisions](#). Agile principles discourage building merely for the sake of building by emphasizing the importance of being strategic and building with purpose.

How it looks in practice:

- Product managers need to make very focused product decisions and closely align product strategy with organizational goals while being extremely picky about what user stories and features actually make the cut. Using prioritization techniques to prioritize initiatives by effort and predicted impact is one way product teams can apply this agile principle to product development.
- The short sprints that agile is characterized by present many opportunities for rapid testing and experimentation which can help reduce uncertainty around whether initiatives will truly have the predicted impact. Using experiments to validate ideas before building them up to spec is a great way to weed out bad ideas and identify good ones.

Agile Principle 11

“The best architectures, requirements, and designs emerge from self-organizing teams.”

In traditional software development methodologies, you’ll often see pyramid shaped teams where management makes key decisions for contributors. Agile principles suggest the use of self-organizing teams which work with a more “flat” management style where decisions are made as a group rather than by a singular manager or management team. The concept ties into agile’s value of teams and interactions over processes and tools, and the intent behind the concept is to empower teams to work together as they need to.

How it looks in practice:

- Self-organizing teams are autonomous groups within the organization who take control and responsibility over their respective projects and have ownership of those areas. Different organizations practice this principle differently. Spotify, for example uses “product squads” to practice this.

Learn more about managing complex requirements in an agile world in the webinar below.

deploying application in Cloud

About Deploying Oracle Agile PLM on Cloud

If your organization wants to develop, deploy, and/or update parts of an Agile Product Lifecycle Management (PLM) application in a faster, more agile way, instead of investing in building on-premise implementations, then deploy Agile PLM on Oracle Cloud Infrastructure.

By using Agile PLM on Oracle Cloud, replication from on-premise to cloud and cloud-to-cloud platforms can easily be established and managed. You can also gain the benefits of faster infrastructure updates, easier scaling up (and down), lower capital expenditure, and fewer personnel dedicated to basic infrastructure maintenance.

Key Workload Requirements

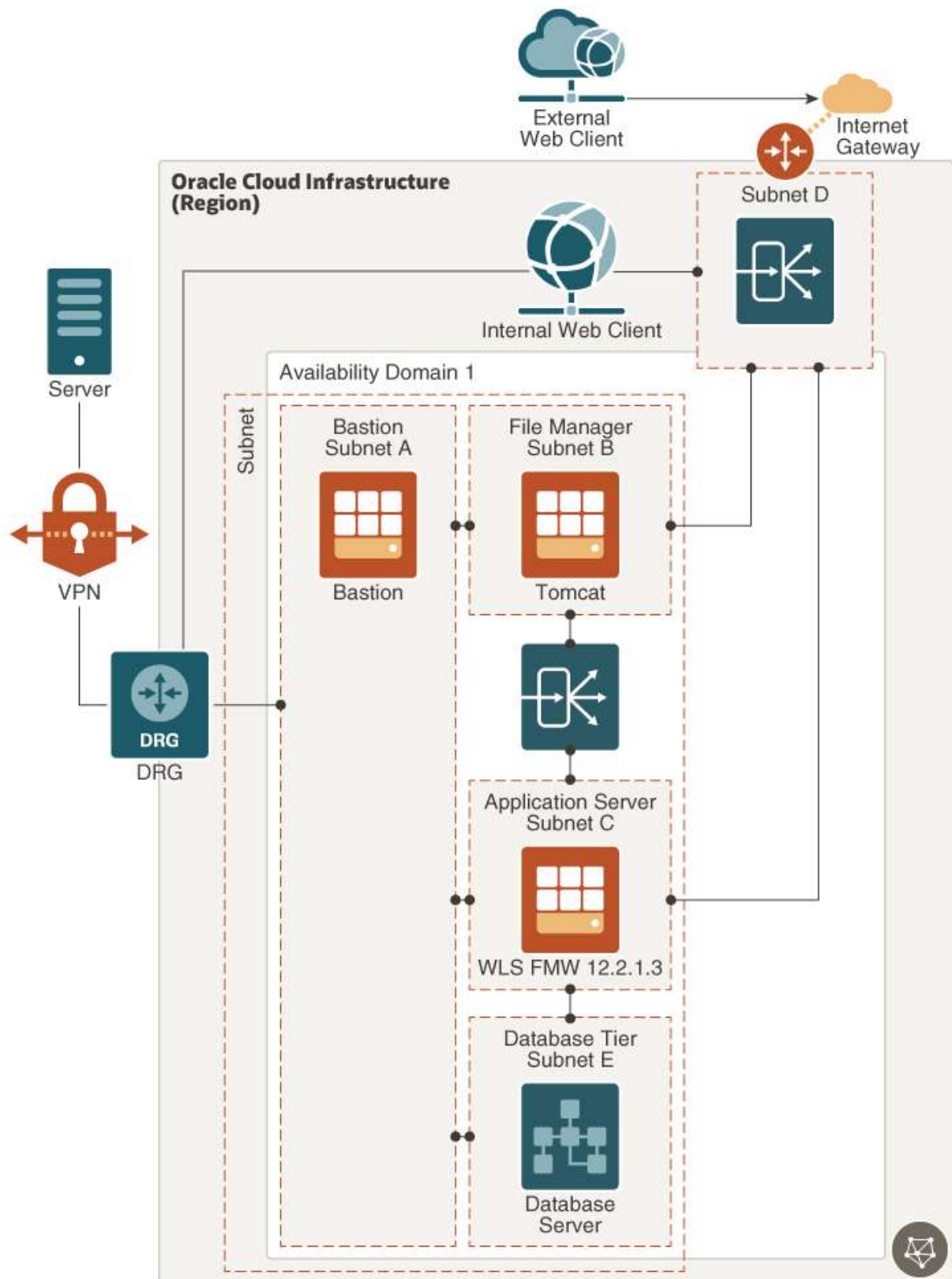
The architectures that Oracle provides help you address these requirements:

- Designing for high availability and disaster recovery
- Deploying a secure architecture.
- Matching your high-performance and highly isolated network model.
- Deploying your application and database environments into the cloud.
- Maintaining visibility over costs and usage.
- Monitoring infrastructure health and performance.

Architecture for Deploying Agile PLM on Cloud

You can deploy Agile PLM in a single availability domain while ensuring high availability. Use this architecture when you want to ensure that your application is available even when an application instance goes down. The other available application instances in the availability domain continue to process the requests.

Oracle Agile PLM can be deployed on cloud in a multi-tiered architecture. The architecture consists of a virtual cloud network (VCN) with the bastion host, load balancer tier, application tier, and database tier. The tiers are placed in separate subnets of the VCN in a single availability domain.



[Description of the illustration agile_plm_reference_architecture_high_availability.png](#)

The Agile PLM application server can be set up in a standalone or clustered configuration. In the image shown, a standalone server is considered, which has only one Oracle WebLogic Server instance. All client servers and users connect to the application server either directly

or indirectly. To permit traffic to the web server from the internet, you can create load balancers in the public subnet. You can access Oracle Cloud instances in the private subnet from your data centers by connecting through the dynamic routing gateway (DRG). The DRG is the gateway that connects your on premise network to your cloud network and you can enable communication between the two using VPN. You'll also have to update the route table to enable traffic to and from the DRG.

The load balancer receives requests from users, and then routes these requests to the application tier. You can allow for redundancy (and scalability) by configuring multiple instances of the WebLogic server for the core application, Tomcat for File Manager, and RAC for database. You can augment redundancy through the use of fault domains so that you can continue accessing the application even if an instance goes down. All instances are active and receive traffic from the load balancer.

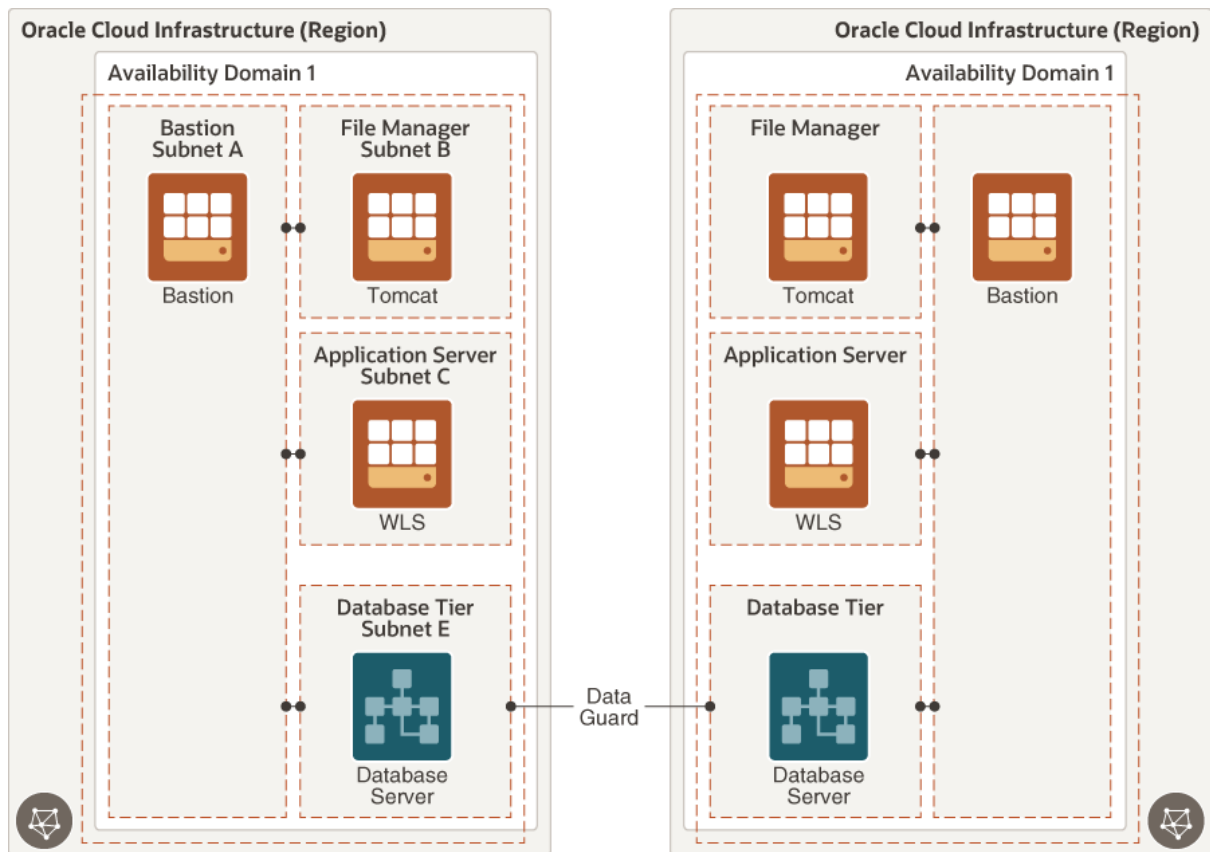
There's a private Load Balancer between File Manager and Application Server to distribute traffic to your application instances within a VCN. This service provides a primary and a standby instance of the load balancer to ensure that if the primary load balancer becomes unavailable, the standby load balancer forwards the requests. The load balancer ensures that requests are routed to the healthy application instances. If there's a problem with an application instance, then the load balancer removes that instance and starts routing requests to the remaining healthy application instances.

The database server stores all product content and system settings and is placed in the private subnet. This database is accessed only by the application server. For performance and high availability requirements, Oracle recommends that you use two-node Oracle Real Application Clusters (Oracle RAC) database systems in Oracle Cloud Infrastructure.

Architecture of Agile PLM Disaster Recovery

Oracle Cloud provides Agile PLM implementations that ensure you can build disaster recovery (DR) into your deployment in unforeseen events that would require you to failover and still keep Agile PLM up and running.

The following image illustrates the reference architecture for deploying Agile PLM in multiple regions with high availability and disaster recovery.



[Description of the illustration agile_plm_reference_architecture_high_availability_and_dr.png](#)

Oracle Data Guard protects your database tier by replicating data across availability domains.