Network: Collection of computers interlinked together is called network. First network name is **ARPANET** (Advanced Research Projects Agency Network). First protocol in IT industry is FTP (File Transfer Protocol).

Internet: Internet stands for international networking.
1990
The Internet is a network of connected computers. No company owns the Internet; it is a cooperative effort governed by a system of standards and rules. The purpose of connecting computers together, of course, is to share information.

Internet is a collection web application,
Web application is group of web pages
Web page is group components (means heading, para, image, button, tables, …)

A Brief History of the Web

The Web was born in a particle physics laboratory (CERN) in Geneva, Switzerland in 1989. There a computer specialist named **Tim Berners-Lee** first proposed a system of information management that used a "hypertext" process to link related documents over a network. He and his partner, **Robert Cailliau**, created a prototype and released it for review. For the first several years, web pages were text-only. It's difficult to believe that in 1992, the world had only about 50 web servers.

Tim Berners-Lee internet (1989-1990)

- ⇨ Html (HyperText Markup Lang)
- ⇨ http (hyper Text Transfer Protocol)
- ⇨ W3C org

The World Wide Web Consortium

World Wide Web Consortium (called W3C) is the organization that oversees the development of web technologies. The group was founded in 1994 by Tim Berners-Lee, the inventor of the Web, at the Massachusetts Institute of Technology (MIT).
Tim Berners-Lee (WWW/HTTP), Cerf & Kahn (TCP/IP), Baran, Davies, Kleinrock & Roberts (packet networking), Bob Metcalfe (Ethernet).

WHAT IS APPLICATION OR SOFTWARE?

Automation of manual business operations by using a programming language.

TYPES OF APPLICATIONS OR SOFTWARES

We can create an application or software in fallowing flavors:

1) **Desktop:** The applications which are installable in local systems are called desktop applications.
2) **Mobile:** The applications which are installable in mobile phones or tablets downloaded from play store for android and apple store for ios.
3) **Web:** The applications which are deployable in any server and can be accessible from any location using browser.

WHAT IS WEB APPLICATION?

Web applications are network enable applications. We can deploy any web applications in servers and we can access them over network using server ip address and application name.

In computing, a **web application** or **web app** is a client–server software **application** which the client (or user interface) runs in a **web browser** and it contains web documents in the form electronic pages(web pages).

A web application typically contains fallowing three layers:

Presentation layer is a user interface (views) which are accessible from any web browser.

Business layer is a server-side program which is nothing but automation of business rules. Client layer will interact with business layer to persist data.

Data layer is database software where we can store client related data. Business layer will interact with data layer.

How the Web Works

1. When you connect to the web, you do so via an Internet Service Provider (ISP). You type a domain name or web address into your browser to visit a site; for example: google.com, oracle.com, microsoft.com.
2. Your computer contacts a network of servers called Domain Name System (DNS)servers. These act like phonebooks; they tell your computer the IP address associated with the requested domain name. An IP address is a number of up to 12 digits separated by periods / full stops. Every device connected to the web has a unique IP address; it is like the phone number for that computer.
3. The unique number that the DNS server returns to your computer allows your browser to contact the web server that hosts the website you requested. A web server is a computer that is constantly connected to the web, and is setup specially to send web pages to users.
4. The web server then sends the page you requested back to your web browser.

What is web browser?

It is client-side lightweight software installed in client machine. It sends http request from client to server; it takes http response from server.
Browser provides navigation among web pages, and browsers executes html, css, JavaScript files and displays output to user.

List of Computer Browsers:
Internet Explorer(1995), Opera(1995), Mozilla Firefox(1998), Safari(2005), Google Chrome(2008) etc…

List of Mobile Browsers:
Mobile Safari (iOS), Android Browser (Android), BlackBerry Browser (RIM), Nokia Browser (Symbian), Opera Mobile and Mini (installed on any device), Internet Explorer Mobile (Windows Phone), Silk (Kindle Fire) etc…

Server

A **server** is a computer or system that provides resources, data, services, or programs to other machines, known as clients, over a network/inet. In theory, whenever computers share resources with client machines, they are considered **servers.**

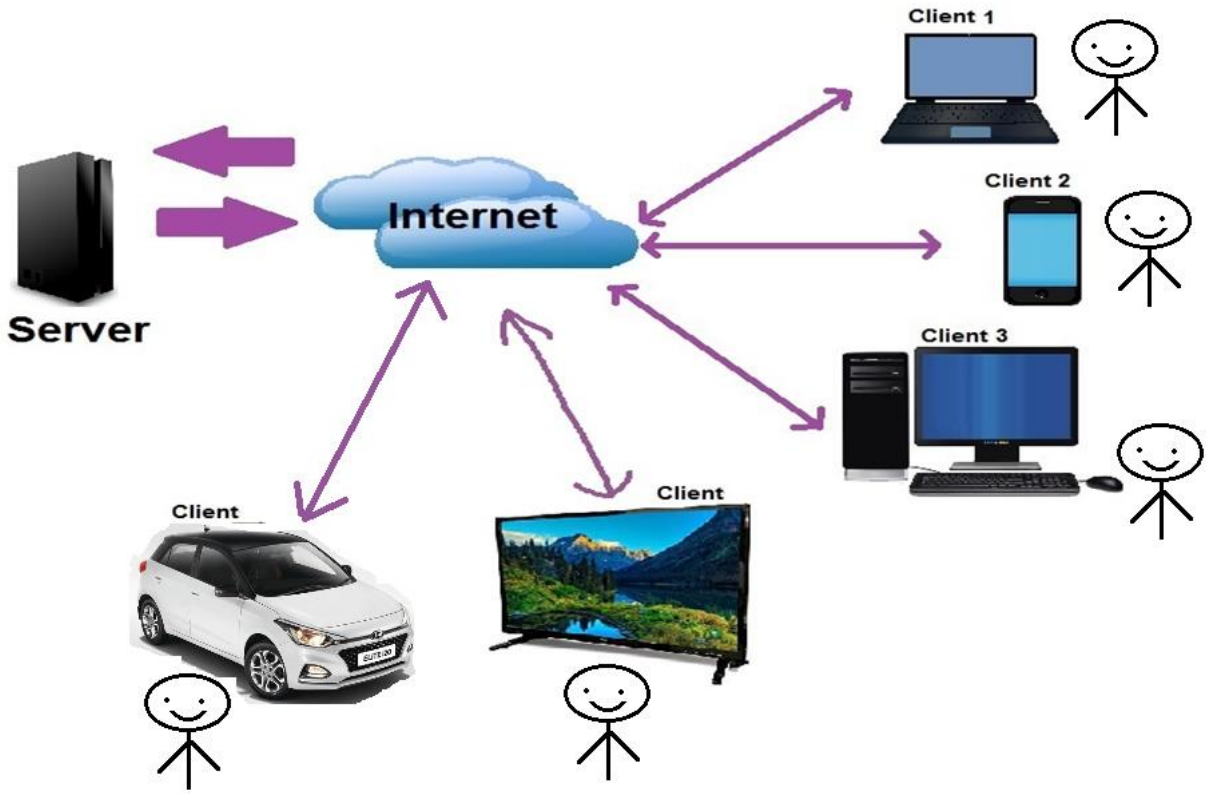
a **server** stores all the data associated with the websites that are hosted by it and shares that info with all computers and mobile devices (like yours) that need to access them.

Client

A client is a electronic device that connects to and uses the resources of a remote computer, or server.
Client maybe a desktop or a laptop or a tablet or a mobile phone or a TV etc.
The device which is used by the user is called as “Client”.

User

The person who is working on/operating client machine is known as User or end-user.



Client:

It is a machine or device (desktop or laptop or tablet or mobile phone or TV etc), which can access the data from server machine.

The device which is used by the user is called as "Client", person who is working on client machine is known as User.

Email: Electronic mail services. It is a free service to communicate with other internet users. Email is invented by **Shabeer Bhatia.** Sabeer Bhatia is an Indian entrepreneur who founded the webmail company Hotmail.com.

SMTP: Simple Mail Transfer Protocol. It takes care of delivering emails from one server to another.

MIME: Multipurpose Internet Mail Extensions. It exchanges different kinds of data.

Blog: It is daily updating website or webpage. Every post displayed in reverse chronological order.

Forum: It is an online discussion website to exchange resources each other.

Http: It is a transfer protocol to exchange hypertext documents in the world wide web.

Http(s): Secured transfer protocol to exchange hypertext documents with the help of SSL(ciphertext).

Ciphertext is encrypted text. Plaintext is what you have before encryption, and **ciphertext** is the encrypted result. The term cipher is sometimes used as a synonym for **ciphertext**, but it more properly **means** the method of encryption rather than the result.

HOW MANY TYPES OF WEB APPLICATIONS WE HAVE?

A webpage is an electronic page developed on HTML. It is classified into two types.

Static webpage: A user unable to interact directly with these webpages. Eg: HTML, CSS

Dynamic webpage: End-user can able to interact directly with these webpages. Eg: HTML, CSS & Javascript

Collection of webpages or web documents are called web application(website). These are classified into two types:

STATIC WEB APPS: The applications which can't able to handle business logic are known as static web apps. Static apps will contain only client layer. We can develop static web applications using HTML. To provide look and feel to these static pages

we can use CSS. To handle client layer business logic we ca use Javascript. We can't able to maintain end user interaction(state) using static web apps.

DYNAMIC WEB APPS: The applications which can able to handle business logic are known as dynamic web apps. These type of apps contains at least 2 layers client and business. If we need to store client data then these application contains data layer too. We can develop client layer using HTML, CSS &javascript and business layer using any one of the server programming language like .NET, JAVA/J2EE & PHP etc... We can store end user data using any database like mongo db, MS-SQL, MySql, Oracle etc.

What is HTML?

It is specially designed hypertext for web browsers, with meaningful tags or elements in simple English language.

HTML Versions

From W3C organization there are fallowing versions released.

Version	Specification	Release Date
1.0	**N/A (HTML 1.0)**	**1-Jan-1994**
2.0	**HTML 2.0**	**24-Nov-1995**
3.2	**W3C: HTML 3.2**	**14-Jan-1997**
4.0	**W3C: HTML 4.0**	**24-Apr-1998**
4.1	**W3C: HTML 4.1**	**24-Dec-1999**
5.0	**WHATWG (Adv Markup Language For Mobiles)**	**28-Oct-2014**
5.1	**W3C: HTML 5.1 (Adv Markup Language For Small Electronic Devices)**	**-Nov-2016**
5.2	**W3C: HTML 5.2**	**14-Dec-2017**

HTML Intro

1. HTML was developed by “Tim-Berners-Lee”, released in 1994 and maintained by W3C Org.

2. HTML stands for “Hypertext Markup Language”.

3. Hypertext” means the text that can be transferred from internet server to internet client.

   "Markup Language" means which syntax will be in the form of tags or you simply "markup" a text document with tags that tell a Web browser how to structure it to display.

4. Technically, HTML is not a programming language, but rather a markup language.

5. HTML is used to design "**static web pages**", means HTML is used to create elements (such as headings, paragraphs, icons, menus, logos, images, textboxes, button etc) in the web pages.

   static webpage means, that pages always showing same information.

6. HTML is very easy to understand (no pre-requisites).

7. HTML is “client side tech”. That means the html code executes on the client browser but not in server.

   web tech:

which sw are supporting to design web pages or providing API to dev web coding those sw are called as web tech.

> **client side tech** ex: html/css, js, jquery, BS ...

used for static web pages.

bw rec source code & trans after execution then produced the output.

> **server side tech** ex: servlet, jsp, asp.net, php, cgi, nodejs, cold fusion ...

dynamic web pages.

code trans, execute with in server only, and produced output, this output sent to client machine.

8. HTML is supported by all the browsers such as Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Safari, Opera and other browsers.

9. HTML is used in all real time web sites today; html is the only language available in world for designing Webpages.

1. The file extension either "filename.html" or "filename.htm"

2. HTML is an interpreter-based language. That means the HTML code will be converted into machine language in +. Browser interprets HTML code.

   Translators: converting high level code (human) into machine level code (MP/OS) is called as translation. who performs this operation those called as translators.

   types: >compiler ex: c, cpp,...

   >interpreter ex: html, js, oracle,...

   >assembler

3. for working html no need installs any software, and **browse**r is responsible for executing & producing output of html programs.

4. html is 100% error free programming.

5. HTML is not a case sensitive language that means you can write the html code in either upper case or lower case.

how design & execute html programs

- open any text editor (sw) and type program.

  notepad, editplus, notepad++, textpad, sublime, ms vs, word, atom, coffee, ...

- save that program with any name (.html or .htm) and anywhere in system.

  but filename must be single word (space is not allowed).

> execution:

1st Approch: goto file location, then dbl click on file

2nd Approch: goto file location, then right click on file and click on open then select browser

3rd Approch: open any browser, then goto address bar and type filename with address.

d:\siva\test.html e:\test.html

Tag:

- A tag is a keyword, enclosed within "<" and ">" in HTML language.
- It is special kind of text placed between left angular brace and right angular brace(<tag_name>).
- Tag is predefined program, program is instructions / command to browser.
- Tag is used to display some specific output in the web page.
- browser was not identified the tag; it shows blank page or it prints as text.
- tags also represented as elements.
- tag has some attributes(properties), those are used to change look & feel (components or output).

types of tags:

in html we have **two** types tags, those are:

> **paired tags**

contains open tag and closing tag.

opening tag specifies starting point of operation/output, closing tag specifies ending point of operation/output.

Syn: **<tagname>** something **</tagname>**

ex: <html> ... </html>
<head> ... </head>
<body> … </body>
<script> ... </script>
<style> ... </style>
 …

note: paired tags also called as body-full tags

>**unpaired tags**

contains only open tag.

VOID => ITS not RETURNING ANY VALUE

Syn: <tagname> or <tagname/>

ex:
 <img/> <input/>
<hr>
<link>

note: Unpaired tags also called as body-less tags

Structure of HTML

as per **W3C** we have to follow the following structure to design web pages (but it's not comp).

```
<!DOCTYPE html>
<html>   web page/document designing starts here
    <head>
        -> non-content sec(non-result)
        -> settings/internal info about page
        -> 1st executed sec
            Ex: title tag, link tag, style tag, script tag, meta tag
    </head>
    <body>
        -> content sec(result)
        -> it contains page designing
        -> 2nd executed sec
        Ex: form, h1, h2, h3, h4, h5, h6, p, div, table, img, a, button, audio,
        video, iframe, etc…
    </body>
</html>   web page/document designing ends here
```

generally, html page contains three parts, those are:

- > versioning section
- > head section
- > body section

versioning section

this is providing information to browser which version we are using in webpage/program. So, browser is interpreting code and producing output as per given specification.

Syn:

```
<!DOCTYPE  html  version-url>
```

HTML4.0:

```
<!DOCTYPE html public "-//W3C//DTD HTML 4.0//EN"
"http://www.w3c.org/TR/html4/strict.dtd">
```

XHTML:

```
<!DOCTYPE html public "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3c.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Html+xml => xhtml

HTML5:

<!DOCTYPE html> **current version**

strcit.dtd file (document type definition), it contains definitions of tags, specifications.

doctype is not case-sen, so we type in any case.

ex: DOCTYPE => valid (recommended)

doctype => valid

DocType => valid

html tag

the <html> tag represents starting and ending of html program. Html tag contains two child/sub tags those are head tag and body tag.

head tag

head tag represents non-content section (means not output) of the web page.

this information doesn't appear on webpage/in browser (it's called as non-content), but it's used internally by the browser.

this tag is used to set icons, title, to provide some meta data (info about web app), css settings, java scripting etc...

head tag contains some child/sub tags, those are

Syn:

```
<head>
   <link>
   <title> </title>
   <meta>
   <style> </style>
   <script> </script>
   ...
</head>
```

body tag

body tag represents content information (means output) of the web page.

this information appears on webpage/in browser (it's called as content).

this tag is used to design UI or to display output.

body tag contains so many child/sub tags.

some of tags:

```
<body>
        <form>
        <h1>
        <h2>
        <p>
        <div>
        <input>
        <a>
        <audio>
        <video>
        <iframe> etc...
</body>
```

html is collection of tags(elements) and attributes.

comment lines

comment lines are to provide some description about of our program.

Syn:

<!-- comments -- >

comments are not executed by browser.

title tag

this tag used to set the title for a webpage, means every webpage they have individual title.

its paired tag.

<title> is the sub tag of <head> tag.

Web site => 10 web pages => Title 10 times

Syn:

<title> text **</title>**

Note: one web page/one title

Link tag

Link tag used to set the icon/logo for a webpage.

Un-paired tag.

<link> is the sub tag of <head> tag.

Syn: <link rel="icon" href="filename"/>

Relative

Hyper reference => .jpg .bmp **.png** .gif .tif **.ico**

Preferable image size:

18px X 18px
20X20px
30X30px
40X40px

heading tags

these tags are used to print data/text in heading format.
html providing 6 heading tags, those are h1, h2, h3, h4, h5, h6.
These 6 tags are used to display headings in different sizes.
six tags are paired tags and block level elements.
Syn:

<h1> text </h1>
<h2> text </h2>
<h3> text </h3>
<h4> text </h4>
<h5> text </h5>
<h6> text </h6>

Note: inside body section we can repeated any tag and no.of times.

p tag

> p stands for paragraph.
> this tag is used to display/print more lines of text (paragraph)
> its paired tag and block level.
> browser display an empty line(gap) between paragraphs
Syn:

** text or info **

Note:

> browser/html doesn't accept more than one space (space bar & tab key), means while designing of program we given more space but browser prints only one space.
> browser/html doesn't accepts enter key (line breaking), means while designing of program we use enter key but browser prints data without breaking line.

br tag

- br stands for break line (enter key)
- its non-paired

Syn: **
 or
**

Html operators

Special characters or html entities

Syn: &operator;

nbsp

nbsp stands for non-remove blank space
nbsp produce one space.
html operator/ html entity
Syn:
** **
© ™ ® € £ ¥

< > ¼ ½ ¾ etc…

Label tag

>label tag used for displaying prompting text.
>its paired tag, inline tag
Syn: <label> text </label>

Span tag

>span tag used for small textual data, like as error message, mandatory specification.
> in continuity of text, if we want to highlight couple of word or **letters**, we use span tag
>its paired tag, inline tag
Syn: ** text **

pre tag

> pre stands for pre-formatting (alignment)
> pre tag is used to print data/text, how we typed in same format
> pre is paired tag, block level
Syn:

<pre> text </pre>

formatting tags

** **	**<u>**	**<i> **	**<strike>**	**<sub>**	**<sup>**
Apple	apple	*4568*	~~apple~~	H_20	a^2

All are paired tags
Inline tags

b tag or **strong**

> b stands for bold
> b tag used to print text in bold format
> both are paired tags & inline tags
Syn:

 text
 text

u tag

> u stands for underline
> u tag used to print text with underline (draws a line base of text)
> u is paired tag
Syn:

<u> text </u>

strikeout tag

> strikeout tag used to print text with line (draws a line middle of text)
> strikeout is paired tag
Syn:

<strike> text </strike>

superscript tag

>this tag used to display text top of upper line
> superscript is paired tag

Syn:

^{text}

subscript tag

>this tag used to display text bottom of baseline
> subscript is paired tag
Syn:

_{text}

i or em tag

> i stand for italic (inclined)
> i tag used to print text with little banding
> i is paired
Syn:

 text
 text

All 6 tags are paired tags & inline tags

Attributes

> attribute is a special feature/setting/property of a tag.
> attributes are used to change the default look/feel of data(elements).
> every tag they have attributes

Syn:

<tagname attribute="parameter" attribute='parameter' ...> Html
Width="100px" height=200px

Note:

- parameter means the value of attribute.
- Parameter should enclosed within " " or ' ' or without quotes.
- Every attribute must be separated by a space

<tagname style="attribute:value; attribute:value; ..."> C S S

Note:

- quotes are comp
- Every attribute must be separated by a ;

C S S change look/feel of html elements

Css provide only styles but not tags
Style is group of attribute/properties

Different ways to implement css:
1st Approch (inline):
Html tag along with css properties
<tag Style="attribute:value; attribute:value; …">

2nd Approch (internal):
Html tags and css styles are designed in the same program, but not in same line.
Internal css should be implements in Style tag, style tag must be sub tag head tag.

```
<style>
   Selector {
        attribute:value;
        attribute:value;
        …
   }
   Selector{
        attribute:value;
        attribute:value;
        …
   }
   …….
</style>
```

3rd Approch (external)
Css styles are designed in sperate file and should be save with ".css", and html code saved with ".html"
Use link tag for mapping css file to html file
Syn: <link rel="stylesheet" href="filename.css"/>

types:
as per html4 we have 3types of attributes, those are
>**global attributes** (core)
Ex: id, name, class, style, align, width, height, title etc…

>**specific attributes** (personal)
Ex: rel, href, src, colspan, rowspan, action, alt etc…
>**event attributes** (dynamic)
Ex: onclick, onload, onfocus, onblur, onchange, onsubmit, onkeypres etc…
>**optional attribute** <== html5
Ex: lang, type etc…

global attributes:

these attributes are common for most of tags (99% of tags)
those attributes are:
class, id, name, style, align etc...
ex:

```
<h1 class="" id="" name="a" style="" ...>
<p id="" class="" name="b" style="" ...>
<pre class="" id="" name="c" style="" ...>
```

specific attributes:

these attributes are specific to some tags/elements only (not common).
those attributes are:
src, href, rel, target, colspan, rowspan, alt, placeholder, poster, loop, etc...
ex:

```
<a href="url" ... >
<img src="" ...>
<audio controls>
```

event attributes:

these attributes are used to some logical operations.
logical operations we can perform by using JavaScript, these also called dynamic attribute.
attributes are:
onclick, oninput, onfocus, onexit, onload, onchange, onblur, ...
ex:

```
<button onclick="js code/fun1">
<body onload="js code">
```

optional attributes:

these attributes are not comp to specify/to use.
these type attributes are supported since html5.0.
those attributes are:
lang, type, method ...
ex:
<style type="text/css" ...>
<script type="text/javascirpt" ...>
<link type="image/jpg" ...>
<form method="get" ...>

categories of attributes:

html attributes <tag attribute="value">
css attributes <tag style="css-attribute:value; css-attribute:value;...">

note:

- css attributes we can't use in place of html attributes.
- html attributes we can't in place of css attributes.
- “Id” attribute we are using while writing javascript code.
- “Name” attribute we are using while writing server-side code. (servlet, jsp, asp, php, noedjs)

images

> "img" tag is used to display images on webpage.
> in one web pages we can display any no.of images and any type of images.
> it is strongly recommended to place all images in side root folder or create sub folder with name images in root folder
> its un-paired tag, and its inline element
Syn:

.jfif .svg .jpg .bmp .gif .tif .png .webp

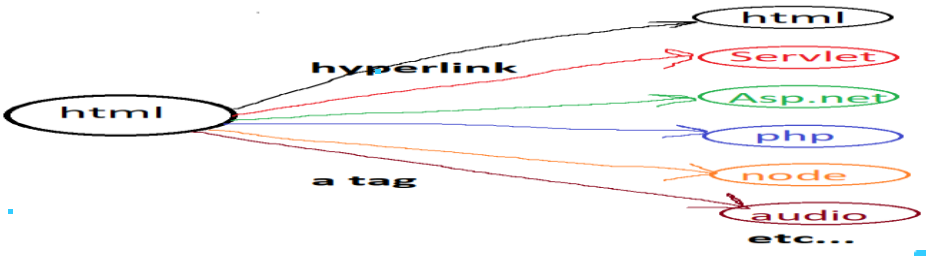
attributes:
src => to specify which img you want to display
width => width of image (pixel)
height => height of image (pixel)
title => it is used to specify tool tip. (whenever mouse pointer comes on top of image)
alt => alternative text, if image not loaded in webpage/not display, we want to display text message to user it called as alt

+
global attributes

opacity: 0.5;
filter: blur(5px);
brightness(125%)
contrast(135%)
grayscale(100%)
invert(100%)
hue-rotate(180deg)
saturate(8)
sepia(100%)
drop-shadow(8px 8px 10px green)

hyper links

> a stand for "anchor"
> "a" tag is used to create hyperlink, hyperlinks are used to move from one webpage to another webpage.
> whenever user click on the hyperlink, it moves to the specified page.
> destination page sometime within same application or other application.



>web application basically contains links to other pages, so it's very commonly used tag.
> by default, every browser provides built-in style for each hyperlink,
i.e blue color+hand symbol+under line.
we can customize these styles by using CSS.
> its paired tag, and inline element

Syn:

<a attributes> Display Text
<a attributes> <img>

attributes:

href : hyper reference, used to specify the address of webpage or web site,
i.e whenever user clicks on this link, which page you want to open

url may be html page, server-side file, image, audio file, video, pdf file, documents etc...

href="url"
"https://www.abc.com/login.aspx"
"" self-calling
"." ⍰ home page of web site/home dir of web application
"#id" ⍰ it creates book marks (moving within same page)

target : where you want open destination page
_blank ==> opens the link in a window/tab
_self ==> opens the link in current working tab/window (its default)
_parent ==> opens the link in parent frame
_top ==> opens the link in full body of window
framename ==> opens the link in specified frame

html colors

html supports 3types of patterns, those are
> named colors
> RGB colors
> Hexadecimal colors

named colors:
>it supports to write direct color name
>we have some limited colors
ex: white, black, red, green etc...
> color names are not case-sen

RGB colors:
>RGB model specifies that the composition of 3 basic colors (Red, Green, Blue)
>RGB produces 16millions colors.

Syn: **rgb**(red, green, blue)
red => 0 - 255
green => 0 - 255
blue => 0 – 255
ex: **rgb**(10, 45, 201) 401%255 ⍰146

Hexadecimal number colors:
>Hexadecimal model is the shortcut for rgb model

>Hexadecimal system ranges from 0 - 15
0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f

Syn: #RRGGBB 1,2 red 3,4 green 5,6 blue
ex: #1a4b68
#RGB
Ex: #3d7

Note: in real time "Hexadecimal model" is recommended.

these colors we can use for foreground color, background color, border color etc..
for setting colors we have some attributes, those are

color to set/to change foreground color (text color)
background-color to set/to change background color
border-color to set/to change border color (line color)
box-shadow to set/to change shadow color
text-shadow to set/to change text shadow color

Note: all these are CSS attributes. Support by Most of html tags

Gradient colors

background: #FC466B; /* fallback for old browsers */
background: -webkit-linear-gradient(to bottom, #3F5EFB, #FC466B); Chrome 10-25, Safari 5.1-6
background: **linear-gradient**(to bottom, #3F5EFB, #FC466B); W3C, IE 10+/ Edge, Firefox 16+, Chrome 26+, Opera 12+, Safari 7+

linear-gradient(direction, color1, color2,…color-n)
dir: to left (r=>l)
to right (l=>r)
to top (b=>t)
to bottom (t=>b)

background: **linear-gradient**(to bottom, #3F5EFB 40%, #FC466B 60%);

-webkit-linear-gradient(to left, #3F5EFB, #FC466B);
linear-gradient(to left, #3F5EFB, #FC466B);

background: radial-gradient(circle, rgba(2,0,36,1) 0%, rgba(38,38,162,1) 60%, rgba(0,212,255,1) 100%);

radial-gradient(shape, color1, color2, …color-n)

radial-gradient(circle, rgb(131,58,180) 0%, rgb(29,166,65) 50%, rgb(252,176,69) 100%);

radial-gradient(circle, rgba(166,29,142,1) 57%, rgba(100,180,111,1) 78%, rgba(69,252,96,1) 100%);

Note: while applying gradient colors we have to use "**background**" property in place of "**background-color**".

working with list tags

these tags are used to display data/info in points wise.
html supports three types of list, those are

> ol
> ul
> dl

List's:

Ordered list numbering
Unordered list bulleting

ol tag

>ol stands for "Ordered List".
>it is used to display the text(names, colors, team names, course name...) with numbering.
>it supports 5types numbering, those are **1, A, a, i, I**. by default it displaying in number.
> by using "ol" tag we can create ordered list
> ol is paired tag & block level element

li tag

> li stands for "list item"
> li is sub tag of ol tag
> li tag is used to print text/data in points wise
> li is paired tag & block level elements
Syn:

```
<ol attributes>
    <li> text </li>
    <li> text </li>
    <li> text </li>
   ...
</ol>
```

ol attributes:

type : which type numbering to display (Default is 1)
start : from where u want to start numbering (default is 1)
reversed : to displaying numbers in desc order

li attributes:

value : used for restarting numbering with specified value

ul tag

>ul stands for "Un-Ordered List".
>it is used to display the list of items(names, colors, team names, course name...) with bulleting.
>it supports 3types bulleting, those are **dot, circle, square**. by default is dot.
>by using "ul" tag we can create un-ordered list items
> ul is paired tag
>"li" tag used for creating list items
Syn:

```
<ul type="dot/circle/square">
  <li> text </li>
  <li> text </li>
  <li> text </li>
  ...
</ul>
```

dl tag

>dl stands for Definition list (since html5 description list)

>dl tag used for to display definitions/full forms (collection of definitions)
>its paired tag
> "dt" and "dd" are sub tags of "dl" tag
> "dt" stands for definition title, "dd" stands for definition data.
> dt & dd are paired
Syn:

```
<dl>
  <dt>title/word</dt>
    <dd>information</dd>
  <dt>title/word</dt>
    <dd>information</dd>
  <dt>title/word</dt>
    <dd>information</dd>
  ...
</dl>
```

fieldset tag

> this tag used for drawing a line/border around elements/tags.
> its paired tag and block level
> we can draw any no.of borders

Syn:

```
<fieldset attributes>
    designing/text
</fieldtset>
```

attributes:

align : align of elements, it supports 3 alignments center, left, right
left is default align
border : style of line, thickness of line, color of line
width : width of box (size in %)

legend tag

>legend tag used for set title/heading for fieldset
>legend is sub tag of fieldset tag
>its paired tag

Syn: <legend attributes>Heading</legend>

attributes:

align :align of elements, it supports 3 alignments center, left, right

left is default align

color :

div tag

>div is a container, means its grouping elements/controls/components of html.
>inside div tag we can place any content like normal text or images.
>div tag is used to divide web page as no.of subpages/parts, each part is rep as div.
>for better maintained, effective design of web page and simplifying css code.
>its paired tag, and block level element

Syn: <div attributes>
contents

>one webpage may contains any no.of div tags.

display:flex; <== it displaying all elements side-by-side row wise (default setting)
flex-wrap:wrap; <== it align element to next line
flex-direction <== it used to specifiy direction (order) of flex elements
flex-direction:row|row-reverse|column|cloumn-reverse;
flex-flow <== it combination of felx-wrap & flex-direction attributes.
flex-flow: direction wrap;

display:grid; <== it displaying all elements in rowsXcols
grid-template-columns <== no.of columns to display (width of columns)
grid-template-columns:col1 col2 col3....;

:auto auto auto auto; <== 4columns

:300px 400px 250px; <== 3columns

:30% 30%; <== 2columns

:30% auto 400px;

grid-column-gap: Npx; <== it provides a gap between column to column

grid-row-gap: Npx; <== it provides a gap between row to row

grid-gap:Xpx; <== it provides a gap between row-row & col-col with same size

Note: its applicable on nested tags, means outer tag only we can apply grid

table tag

>table tag is used to display the data in form rows & cols in the web page.
> a table is a collection of rows, each row is collection of cells/col/field.
> a table is represented as <table> tag, a row represented as <tr> tag, a col heading is represented as <th> tag, data rep as <td> tag.
> table heading is represented as <caption> tag.
> <thead> tag is rep of table head part, <tbody> tag is rep of table body part and <tfoot> tag Is rep of table footer part.

table □ it just comb rows & cols

caption □ main heading of table

tr □ table row

th □ table heading (col heading)

td □ table data (col data)

thead □ table head section

tbody □ table body section

tfoot □ table footer section

> all these 8tags are paired tags

Table, tr, caption, thead, tbody & tfoot are block level tags

Th & td are inline tags

Syn:

```
<table>
  <tr>
    <th>heading</th> or <td>data</td>

    <th>heading</th> or <td>data</td>
  </tr>
  <tr>
     <th>heading</th> or <td>data</td>
    <th>heading</th> or <td>data</td>
```

```
    </tr>
    ...
  </table>
```

NOte:
<th> and <td> are sub tags of <tr>
<tr> is sub tag of <table>

table attributes:
border : border of table (0 means no border, 1-n border req)
align : alignment of table
width : width of table (%)
...

th & td attributes:
colspan : specifies the no.of columns to merge/expend
rowspan : specifies the no.of rows to merge/expend
...

Module 2. Bootstrap Scaffolding

What Is Bootstrap?

Bootstrap is an open source product from Mark Otto and Jacob Thornton who, when it was initially released, were both employees at Twitter. There was a need to standardize the frontend toolsets of engineers across the company. In the launch blog post, Mark Otto introduced the project like this:

In the earlier days of Twitter, engineers used almost any library they were familiar with to meet front-end requirements. Inconsistencies among the individual applications made it difficult to scale and maintain them. Bootstrap began as an answer to these challenges and quickly accelerated during Twitter's first Hackweek. By the end of Hackweek, we had reached a stable version that engineers could use across the company.

— Mark Otto *https://dev.twitter.com/*

Since Bootstrap launched in August 2011, it has taken off in popularity. It has evolved from being an entirely CSS-driven project to include a host of JavaScript plugins and icons that go hand in hand with forms and buttons. At its base, it allows for responsive web design and features a robust 12-column, 940px-wide grid. One of the highlights is the build tool on Bootstrap's website, where you can customize the build to suit your needs, choosing which CSS and JavaScript features you want to include on your site. All of this allows frontend web development to be catapulted forward, building on a stable foundation of forward-looking design and development. Getting started with Bootstrap is as simple as dropping some CSS and JavaScript into the root of your site.

For someone starting a new project, Bootstrap comes with a handful of useful elements. Normally, when I start a project, I start with tools like Eric Meyer's Reset CSS and get going on my web project. With Bootstrap, you just need to include the *bootstrap.css* CSS file and, optionally, the *bootstrap.js* JavaScript file into your website and you are ready to go.

Bootstrap File Structure

```
bootstrap/

        ├── css/

        |    ├── bootstrap.css
```

```
        |   ├── bootstrap.min.css
        ├── js/
        |   ├── bootstrap.js
        |   ├── bootstrap.min.js
        ├── img/
        |   ├── glyphicons-halflings.png
        |   ├── glyphicons-halflings-white.png
        └── README.md
```

The Bootstrap download includes three folders: css, js, and img. For simplicity, add these to the root of your project. Minified versions of the CSS and JavaScript are also included. It is not necessary to include both the uncompressed and the minified versions. For the sake of brevity, I use the uncompressed version during development and then switch to the compressed version in production.

Basic HTML Template

Normally, a web project looks something like this:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Bootstrap 101 Template</title>
        </head>
        <body>
                <h1>Hello, world!</h1>
        </body>
</html>
```

With Bootstrap, we include the link to the CSS stylesheet and the JavaScript:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Bootstrap 101 Template</title>
                <link href="css/bootstrap.min.css" rel="stylesheet">
        </head>
        <body>
                <h1>Hello, world!</h1>
                <script src="js/bootstrap.min.js"></script>
        </body>
</html>
```

NOTE

Don’t forget the HTML5 Doctype.

By including <!DOCTYPE html>, all modern browsers are put into standards mode.

Global Styles

With Bootstrap, a number of items come prebuilt. Instead of using the old reset block that was part of the Bootstrap 1.0 tree, Bootstrap 2.0 uses Normalize.css, a project from Nicolas Gallagher that is part of the HTML5 Boilerplate. This is included in the *bootstrap.css* file.

In particular, the following default styles give special treatment to typography and links:

- `margin` has been removed from the body, and content will snug up to the edges of the browser window.
- `background-color: white;` is applied to the body.
- Bootstrap is using the `@baseFontFamily`, `@baseFontSize`, and `@baseLineHeight` attributes as our typographic base. This allows the height of headings and other content around the site to maintain a similar line height.
- Bootstrap sets the global link color via `@linkColor` and applies link underlines only on `:hover`.

NOTE

Remember, if you don't like the colors or want to change a default, this can be done by changing the globals in any of the *.less* files. To do this, update the *scaffolding.less* file or overwrite colors in your own stylesheet.

Typography

Documentation and examples for Bootstrap typography, including global settings, headings, body text, lists, and more.

Global settings

Bootstrap sets basic global display, typography, and link styles. When more control is needed, check out the textual utility classes.

- Use a native font stack that selects the best `font-family` for each OS and device.
- For a more inclusive and accessible type scale, we assume the browser default root `font-size` (typically 16px) so visitors can customize their browser defaults as needed.
- Use the `$font-family-base`, `$font-size-base`, and `$line-height-base` attributes as our typographic base applied to the `<body>`.
- Set the global link color via `$link-color` and apply link underlines only on `:hover`.
- Use `$body-bg` to set a `background-color` on the `<body>` (`#fff` by default).

These styles can be found within `_reboot.scss`, and the global variables are defined in `_variables.scss`. Make sure to set `$font-size-base` in `rem`.

Headings

All HTML headings, `<h1>` through `<h6>`, are available.

Heading	Example
`<h1></h1>`	h1. Bootstrap heading
`<h2></h2>`	h2. Bootstrap heading
`<h3></h3>`	h3. Bootstrap heading
`<h4></h4>`	h4. Bootstrap heading
`<h5></h5>`	h5. Bootstrap heading
`<h6></h6>`	h6. Bootstrap heading

Copy

```
<h1>h1. Bootstrap heading</h1>
<h2>h2. Bootstrap heading</h2>
<h3>h3. Bootstrap heading</h3>
<h4>h4. Bootstrap heading</h4>
<h5>h5. Bootstrap heading</h5>
<h6>h6. Bootstrap heading</h6>
```

`.h1` through `.h6` classes are also available, for when you want to match the font styling of a heading but cannot use the associated HTML element.

h1. Bootstrap heading

h2. Bootstrap heading

h3. Bootstrap heading

h4. Bootstrap heading

h5. Bootstrap heading

h6. Bootstrap heading

Copy

```
<p class="h1">h1. Bootstrap heading</p>
<p class="h2">h2. Bootstrap heading</p>
<p class="h3">h3. Bootstrap heading</p>
<p class="h4">h4. Bootstrap heading</p>
<p class="h5">h5. Bootstrap heading</p>
<p class="h6">h6. Bootstrap heading</p>
```

Customizing headings

Use the included utility classes to recreate the small secondary heading text from Bootstrap 3.

Fancy display heading With faded secondary text

Copy

```
<h3>
  Fancy display heading
  <small class="text-muted">With faded secondary text</small>
</h3>
```

Display headings

Traditional heading elements are designed to work best in the meat of your page content. When you need a heading to stand out, consider using a **display heading**—a larger, slightly more opinionated heading style. Keep in mind these headings are not responsive by default, but it's possible to enable responsive font sizes.

Display 1

Display 2

Display 3

Display 4

Copy

```
<h1 class="display-1">Display 1</h1>
<h1 class="display-2">Display 2</h1>
<h1 class="display-3">Display 3</h1>
```

```
<h1 class="display-4">Display 4</h1>
```

Lead

Make a paragraph stand out by adding `.lead`.

Vivamus sagittis lacus vel augue laoreet rutrum faucibus dolor auctor. Duis mollis, est non commodo luctus.

Copy

```
<p class="lead">
  Vivamus sagittis lacus vel augue laoreet rutrum faucibus dolor auctor. Duis
mollis, est non commodo luctus.
</p>
```

Inline text elements

Styling for common inline HTML5 elements.

You can use the mark tag to highlight text.

~~This line of text is meant to be treated as no longer accurate.~~

This line of text is meant to be treated as an addition to the document.

This line of text will render as underlined

<small>This line of text is meant to be treated as fine print.</small>

This line rendered as bold text.

This line rendered as italicized text.

Copy

```
<p>You can use the mark tag to <mark>highlight</mark> text.</p>
<p><del>This line of text is meant to be treated as deleted text.</del></p>
<p><s>This line of text is meant to be treated as no longer accurate.</s></p>
<p><ins>This line of text is meant to be treated as an addition to the
document.</ins></p>
<p><u>This line of text will render as underlined</u></p>
<p><small>This line of text is meant to be treated as fine print.</small></p>
<p><strong>This line rendered as bold text.</strong></p>
<p><em>This line rendered as italicized text.</em></p>
```

`.mark` and `.small` classes are also available to apply the same styles as `<mark>` and `<small>` while avoiding any unwanted semantic implications that the tags would bring.

While not shown above, feel free to use `<b>` and `<i>` in HTML5. `<b>` is meant to highlight words or phrases without conveying additional importance while `<i>` is mostly for voice, technical terms, etc.

Text utilities

Change text alignment, transform, style, weight, and color with our text utilities and color utilities.

Abbreviations

Stylized implementation of HTML's `<abbr>` element for abbreviations and acronyms to show the expanded version on hover. Abbreviations have a default underline and gain a help cursor to provide additional context on hover and to users of assistive technologies.

Add `.initialism` to an abbreviation for a slightly smaller font-size.

attr

HTML

Copy

```
<p><abbr title="attribute">attr</abbr></p>
<p><abbr title="HyperText Markup Language" class="initialism">HTML</abbr></p>
```

Blockquotes

For quoting blocks of content from another source within your document. Wrap `<blockquote class="blockquote">` around any HTML as the quote.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

Copy

```
<blockquote class="blockquote">
  <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
</blockquote>
```

Naming a source

Add a `<footer class="blockquote-footer">` for identifying the source. Wrap the name of the source work in `<cite>`.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

Someone famous in *Source Title*

Copy

```
<blockquote class="blockquote">
  <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
  <footer class="blockquote-footer">Someone famous in <cite title="Source Title">Source Title</cite></footer>
</blockquote>
```

Alignment

Use text utilities as needed to change the alignment of your blockquote.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

Someone famous in *Source Title*

Copy

```
<blockquote class="blockquote text-center">
  <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
  <footer class="blockquote-footer">Someone famous in <cite title="Source Title">Source Title</cite></footer>
</blockquote>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

Someone famous in *Source Title*

Copy

```
<blockquote class="blockquote text-right">
  <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.</p>
  <footer class="blockquote-footer">Someone famous in <cite title="Source Title">Source Title</cite></footer>
</blockquote>
```

Lists

Unstyled

Remove the default `list-style` and left margin on list items (immediate children only). **This only applies to immediate children list items**, meaning you will need to add the class for any nested lists as well.

- Lorem ipsum dolor sit amet
- Consectetur adipiscing elit
- Integer molestie lorem at massa
- Facilisis in pretium nisl aliquet
- Nulla volutpat aliquam velit
    - Phasellus iaculis neque
    - Purus sodales ultricies
    - Vestibulum laoreet porttitor sem
    - Ac tristique libero volutpat at
- Faucibus porta lacus fringilla vel
- Aenean sit amet erat nunc
- Eget porttitor lorem

Copy

```
<ul class="list-unstyled">
  <li>Lorem ipsum dolor sit amet</li>
  <li>Consectetur adipiscing elit</li>
  <li>Integer molestie lorem at massa</li>
```

```
  <li>Facilisis in pretium nisl aliquet</li>
  <li>Nulla volutpat aliquam velit
    <ul>
      <li>Phasellus iaculis neque</li>
      <li>Purus sodales ultricies</li>
      <li>Vestibulum laoreet porttitor sem</li>
      <li>Ac tristique libero volutpat at</li>
    </ul>
  </li>
  <li>Faucibus porta lacus fringilla vel</li>
  <li>Aenean sit amet erat nunc</li>
  <li>Eget porttitor lorem</li>
</ul>
```

Inline

Remove a list's bullets and apply some light `margin` with a combination of two classes, `.list-inline` and `.list-inline-item`.

- Lorem ipsum
- Phasellus iaculis
- Nulla volutpat

Copy

```
<ul class="list-inline">
  <li class="list-inline-item">Lorem ipsum</li>
  <li class="list-inline-item">Phasellus iaculis</li>
  <li class="list-inline-item">Nulla volutpat</li>
</ul>
```

Description list alignment

Align terms and descriptions horizontally by using our grid system's predefined classes (or semantic mixins). For longer terms, you can optionally add a `.text-truncate` class to truncate the text with an ellipsis.

Description lists

A description list is perfect for defining terms.

Euismod

Vestibulum id ligula porta felis euismod semper eget lacinia odio sem nec elit.

Donec id elit non mi porta gravida at eget metus.

Malesuada porta

Etiam porta sem malesuada magna mollis euismod.

Truncated term is truncated

Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.

Nesting

Nested definition list

Aenean posuere, tortor sed cursus feugiat, nunc augue blandit nunc.

Copy

```
<dl class="row">
  <dt class="col-sm-3">Description lists</dt>
  <dd class="col-sm-9">A description list is perfect for defining terms.</dd>

  <dt class="col-sm-3">Euismod</dt>
  <dd class="col-sm-9">
    <p>Vestibulum id ligula porta felis euismod semper eget lacinia odio sem nec elit.</p>
    <p>Donec id elit non mi porta gravida at eget metus.</p>
  </dd>

  <dt class="col-sm-3">Malesuada porta</dt>
  <dd class="col-sm-9">Etiam porta sem malesuada magna mollis euismod.</dd>

  <dt class="col-sm-3 text-truncate">Truncated term is truncated</dt>
  <dd class="col-sm-9">Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.</dd>

  <dt class="col-sm-3">Nesting</dt>
  <dd class="col-sm-9">
    <dl class="row">
      <dt class="col-sm-4">Nested definition list</dt>
      <dd class="col-sm-8">Aenean posuere, tortor sed cursus feugiat, nunc augue blandit nunc.</dd>
    </dl>
  </dd>
</dl>
```

Responsive font sizes

Bootstrap v4.3 ships with the option to enable responsive font sizes, allowing text to scale more naturally across device and viewport sizes. RFS can be enabled by changing the `$enable-responsive-font-sizes` Sass variable to `true` and recompiling Bootstrap.

To support RFS, we use a Sass mixin to replace our normal `font-size` properties. Responsive font sizes will be compiled into `calc()` functions with a mix of `rem` and viewport units to enable the responsive scaling behavior. More about RFS and its configuration can be found on its GitHub repository.

Default Grid System

The default Bootstrap grid (see Figure 1-1) system utilizes 12 columns, making for a 940px-wide container without responsive features enabled. With the responsive CSS file added, the grid adapts to be 724px or 1170px wide, depending on your viewport. Below 767px viewports, such as the ones on tablets and smaller devices, the columns become fluid and stack vertically. At the default width, each column is 60 pixels wide and offset 20 pixels to the left. An example of the 12 possible columns is in Figure 1-1.

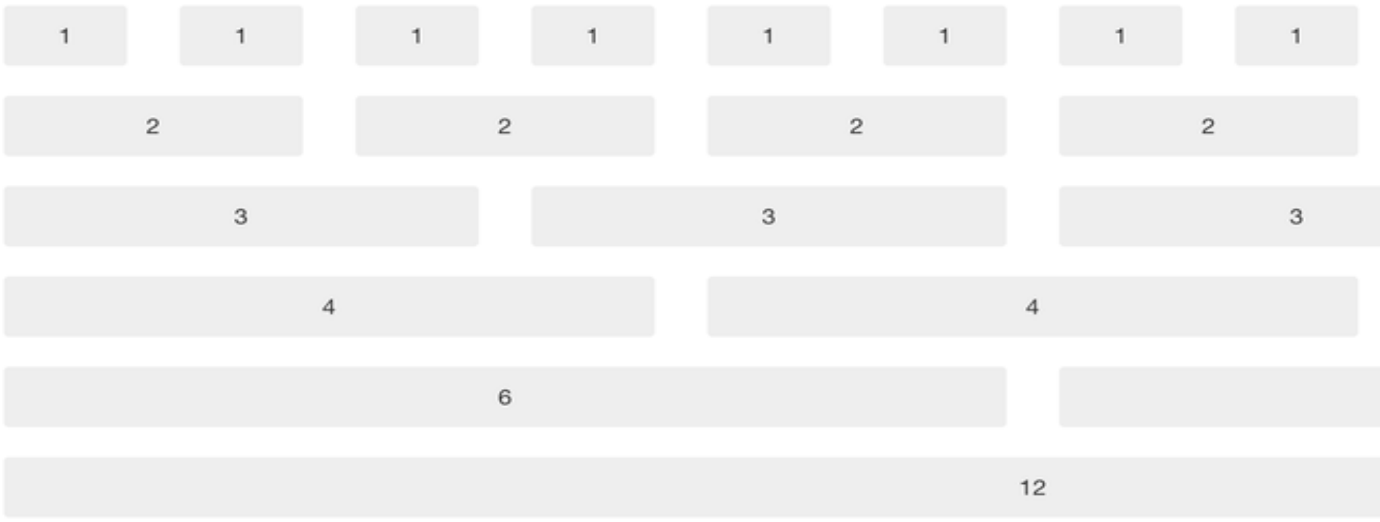


Figure 1-1. Default grid

Basic Grid HTML

To create a simple layout, create a container with a `<div>` that has a class of `.row` and add the appropriate amount of `.span*` columns. Since we have a 12-column grid, we just need the amount of `.span*` columns to equal 12. We could use a 3-6-3 layout, 4-8, 3-5-4, 2-8-2… we could go on and on, but I think you get the gist.

The following code shows `.span8` and `.span4`, which adds up to 12:

```
<div class="row">
  <div class="span8">...</div>
  <div class="span4">...</div>
</div>
```

Offsetting Columns

You can move columns to the right using the `.offset*` class. Each class moves the span over that width. So an `.offset2` would move a `.span7` over two columns (see Figure 1-2):

```
<div class="row">
  <div class="span2">...</div>
  <div class="span7 offset2">...</div>
</div>
```

Figure 1-2. Offset grid

Nesting Columns

To nest your content with the default grid, inside of a `.span*`, simply add a new `.row` with enough `.span*` that it equals the number of spans of the parent container (see Figure 1-3):

```
<div class="row">
  <div class="span9">
    Level 1 of column
    <div class="row">
      <div class="span6">Level 2</div>
      <div class="span3">Level 2</div>
    </div>
  </div>
</div>
```

Figure 1-3. Nesting grid

Fluid Grid System

The fluid grid system uses percentages instead of pixels for column widths. It has the same responsive capabilities as our fixed grid system, ensuring proper proportions for key screen resolutions and devices. You can make any row "fluid" by changing `.row` to `.row-fluid`. The column classes stay exactly the same, making it easy to flip between fixed and fluid grids. To offset, you operate in the same way as the fixed grid system—add `.offset*` to any column to shift by your desired number of columns:

```
<div class="row-fluid">
  <div class="span4">...</div>
  <div class="span8">...</div>
</div>

<div class="row-fluid">
  <div class="span4">...</div>
  <div class="span4 offset2">...</div>
</div>
```

Nesting a fluid grid is a little different. Since we are using percentages, each `.row` resets the column count to 12. For example, if you were inside a `.span8`, instead of two `.span4` elements to divide the content in half, you would use two `.span6` divs (see Figure 1-4). This is the case for responsive content, as we want the content to fill 100% of the container:

```
<div class="row-fluid">
  <div class="span8">
              <div class="row">
```

```
                    <div class="span6">...</div>
                    <div class="span6">...</div>
                </div>
    </div>
</div>
```

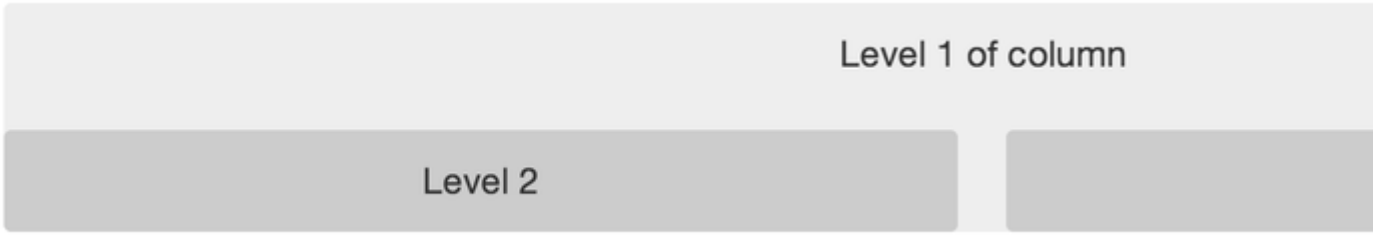


Figure 1-4. Nesting fluid grid

Container Layouts

To add a fixed-width, centered layout to your page, simply wrap the content in `<div class="container">...</div>`. If you would like to use a fluid layout but want to wrap everything in a container, use the following: `<div class="container-fluid">...</div>`. Using a fluid layout is great when you are building applications, administration screens, and other related projects.

Responsive Design

To turn on the responsive features of Bootstrap, you need to add a `<meta>` tag to the `<head>` of your web page. If you haven't downloaded the compiled source, you will also need to add the responsive CSS file. An example of required files looks like this:

```
<!DOCTYPE html>
<html>
        <head>
                <title>My amazing Bootstrap site!</title>
                <meta name="viewport" content="width=device-width,
                initial-scale=1.0">
                <link href="/css/bootstrap.css" rel="stylesheet">
                <link href="/css/bootstrap-responsive.css" rel="stylesheet">
```

```
    </head>
```

WARNING

If you get started and find that the Bootstrap responsive features aren't working, make sure that you have these tags. The responsive features aren't added by default at this time because not everything needs to be responsive. Instead of encouraging developers to remove this feature, the authors of Bootstrap decided that it was best to enable it as needed.

What Is Responsive Design?

Responsive design is a method for taking all of the existing content that is on the page and optimizing it for the device that is viewing it. For example, the desktop not only gets the normal version of the website, but it might also get a widescreen layout, optimized for the larger displays that many people have attached to their computers. Tablets get an optimized layout, taking advantage of their portrait or landscape layouts. And then with phones, you can target their much narrower width. To target these different widths, Bootstrap uses CSS media queries to measure the width of the browser viewport and then, using conditionals, changes which parts of the stylesheets are loaded. Using the width of the browser viewport, Bootstrap can then optimize the content using a combination of ratios or widths, but it mostly relies on *min-width* and *max-width* properties.

At the core, Bootstrap supports five different layouts, each relying on CSS media queries. The largest layout has columns that are 70 pixels wide, contrasting with the 60 pixels of the normal layout. The tablet layout brings the columns to 42 pixels wide, and when narrower than that, each column goes fluid, meaning the columns are stacked vertically and each column is the full width of the device (see Table 1-1).

Table 1-1. Responsive media queries

Label	**Layout width**	**Column width**	**Gutter wi**
Large display	1200px and up	70px	30px
Default	980px and up	60px	20px
Portrait tablets	768px and up	42px	20px
Phones to tablets	767px and below	Fluid columns, no fixed widths	
Phones	480px and below	Fluid columns, no fixed widths	

To add custom CSS based on the media query, you can either include all rules in one CSS file via the media queries below, or use entirely different CSS files:

```
/* Large desktop */
@media (min-width: 1200px) { ... }

/* Portrait tablet to landscape and desktop */
@media (min-width: 768px) and (max-width: 979px) { ... }

/* Landscape phone to portrait tablet */
@media (max-width: 767px) { ... }

/* Landscape phones and down */
@media (max-width: 480px) { ... }
```

For a larger site, you might want to divide each media query into a seperate CSS file. In the HTML file, you can call them with the `<link>` tag in the head of your document. This is useful for keeping file sizes smaller, but it does potentially increase the HTTP requests if the site is responsive. If you are using LESS to compile the CSS, you can have them all processed into one file:

```
<link rel="stylesheet" href="base.css" />
<link rel="stylesheet" media="(min-width: 1200px)" href="large.css" />
<link rel="stylesheet" media="(min-width: 768px) and (max-width: 979px)"
      href="tablet.css" />
<link rel="stylesheet" media="(max-width: 767px)" href="tablet.css" />
<link rel="stylesheet" media="(max-width: 480px)" href="phone.css" />
```

Helper classes

Bootstrap also includes a handful of helper classes for doing responsive development (see Table 1-2). Use these sparingly. A couple of use cases that I have seen involve loading custom elements based on certain layouts. Perhaps you have a really nice header on the main layout, but on mobile you want to pare it down, leaving only a few of the elements. In this scenario, you could use the `.hidden-phone` class to hide either parts or entire dom elements from the header.

Table 1-2. Media queries helper classes

Class	Phones	Tablets	Desktops
.visible-phone	Visible	Hidden	Hidden
.visible-tablet	Hidden	Visible	Hidden
.visible-desktop	Hidden	Hidden	Visible
.hidden-phone	Hidden	Visible	Visible
.hidden-tablet	Visible	Hidden	Visible
.hidden-desktop	Visible	Visible	Hidden

There are two major ways that you could look at doing development. The mantra that a lot of people are shouting now is that you should start with mobile, build to that platform, and let the desktop follow. Bootstrap almost forces the opposite, where you would create a full-featured desktop site that "just works."

If you are looking for a strictly mobile framework, Bootstrap is still a great resource.

Alerts

Provide contextual feedback messages for typical user actions with the handful of available and flexible alert messages.

Examples

Alerts are available for any length of text, as well as an optional dismiss button. For proper styling, use one of the eight **required** contextual classes (e.g., `.alert-success`). For inline dismissal, use the alerts jQuery plugin.

A simple primary alert—check it out!

A simple secondary alert—check it out!

A simple success alert—check it out!

A simple danger alert—check it out!

A simple warning alert—check it out!

A simple info alert—check it out!

A simple light alert—check it out!

A simple dark alert—check it out!

Copy

```
<div class="alert alert-primary" role="alert">
  A simple primary alert—check it out!
</div>
<div class="alert alert-secondary" role="alert">
  A simple secondary alert—check it out!
</div>
<div class="alert alert-success" role="alert">
  A simple success alert—check it out!
</div>
<div class="alert alert-danger" role="alert">
  A simple danger alert—check it out!
</div>
<div class="alert alert-warning" role="alert">
  A simple warning alert—check it out!
</div>
<div class="alert alert-info" role="alert">
  A simple info alert—check it out!
</div>
<div class="alert alert-light" role="alert">
  A simple light alert—check it out!
</div>
<div class="alert alert-dark" role="alert">
  A simple dark alert—check it out!
</div>
```

Conveying meaning to assistive technologies

Using color to add meaning only provides a visual indication, which will not be conveyed to users of assistive technologies – such as screen readers. Ensure that information denoted by the color is either obvious from the content itself (e.g. the visible text), or is included through alternative means, such as additional text hidden with the `.sr-only` class.

Link color

Use the `.alert-link` utility class to quickly provide matching colored links within any alert.

A simple primary alert with **an example link**. Give it a click if you like.

A simple secondary alert with **an example link**. Give it a click if you like.

A simple success alert with **an example link**. Give it a click if you like.

A simple danger alert with **an example link**. Give it a click if you like.

A simple warning alert with **an example link**. Give it a click if you like.

A simple info alert with **an example link**. Give it a click if you like.

A simple light alert with **an example link**. Give it a click if you like.

A simple dark alert with **an example link**. Give it a click if you like.

Copy

```
<div class="alert alert-primary" role="alert">
  A simple primary alert with <a href="#" class="alert-link">an example link</a>.
Give it a click if you like.
</div>
<div class="alert alert-secondary" role="alert">
  A simple secondary alert with <a href="#" class="alert-link">an example
link</a>. Give it a click if you like.
</div>
<div class="alert alert-success" role="alert">
  A simple success alert with <a href="#" class="alert-link">an example link</a>.
Give it a click if you like.
</div>
<div class="alert alert-danger" role="alert">
  A simple danger alert with <a href="#" class="alert-link">an example link</a>.
Give it a click if you like.
</div>
<div class="alert alert-warning" role="alert">
  A simple warning alert with <a href="#" class="alert-link">an example link</a>.
Give it a click if you like.
</div>
<div class="alert alert-info" role="alert">
  A simple info alert with <a href="#" class="alert-link">an example link</a>.
Give it a click if you like.
</div>
<div class="alert alert-light" role="alert">
  A simple light alert with <a href="#" class="alert-link">an example link</a>.
Give it a click if you like.
</div>
<div class="alert alert-dark" role="alert">
  A simple dark alert with <a href="#" class="alert-link">an example link</a>.
Give it a click if you like.
</div>
```

Additional content

Alerts can also contain additional HTML elements like headings, paragraphs and dividers.

Well done!

Aww yeah, you successfully read this important alert message. This example text is going to run a bit longer so that you can see how spacing within an alert works with this kind of content.

Whenever you need to, be sure to use margin utilities to keep things nice and tidy.

Copy

```
<div class="alert alert-success" role="alert">
  <h4 class="alert-heading">Well done!</h4>
  <p>Aww yeah, you successfully read this important alert message. This example
text is going to run a bit longer so that you can see how spacing within an alert
works with this kind of content.</p>
  <hr>
```

```
  <p class="mb-0">Whenever you need to, be sure to use margin utilities to keep
things nice and tidy.</p>
</div>
```

Dismissing

Using the alert JavaScript plugin, it's possible to dismiss any alert inline. Here's how:

- Be sure you've loaded the alert plugin, or the compiled Bootstrap JavaScript.
- If you're building our JavaScript from source, it requires `util.js`. The compiled version includes this.
- Add a dismiss button and the `.alert-dismissible` class, which adds extra padding to the right of the alert and positions the `.close` button.
- On the dismiss button, add the `data-dismiss="alert"` attribute, which triggers the JavaScript functionality. Be sure to use the `<button>` element with it for proper behavior across all devices.
- To animate alerts when dismissing them, be sure to add the `.fade` and `.show` classes.

You can see this in action with a live demo:

```
<div class="alert alert-warning alert-dismissible fade show" role="alert">
  <strong>Holy guacamole!</strong> You should check in on some of those fields
below.
  <button type="button" class="close" data-dismiss="alert" aria-label="Close">
    <span aria-hidden="true">&times;</span>
  </button>
</div>
```

JavaScript behavior

Triggers

Enable dismissal of an alert via JavaScript:

Copy

```
$('.alert').alert()
```

Or with `data` attributes on a button **within the alert**, as demonstrated above:

Copy

```
<button type="button" class="close" data-dismiss="alert" aria-label="Close">
  <span aria-hidden="true">&times;</span>
</button>
```

Note that closing an alert will remove it from the DOM.

Methods

Method	Description
`$().alert()`	Makes an alert listen for click events on descendant elements which have the `data-dismiss="alert"` at (Not necessary when using the data-api's auto-initialization.)

Method	Description
`$().alert('close')`	Closes an alert by removing it from the DOM. If the `.fade` and `.show` classes are present on the elemen will fade out before it is removed.
`$().alert('dispose')`	Destroys an element's alert.

Copy

```
$('.alert').alert('close')
```

Events

Bootstrap's alert plugin exposes a few events for hooking into alert functionality.

Event	Description
`close.bs.alert`	This event fires immediately when the `close` instance method is called.
`closed.bs.alert`	This event is fired when the alert has been closed (will wait for CSS transitions to complete).

Copy

```
$('#myAlert').on('closed.bs.alert', function () {
  // do something...
})
```

Cards

Bootstrap's cards provide a flexible and extensible content container with multiple variants and options.

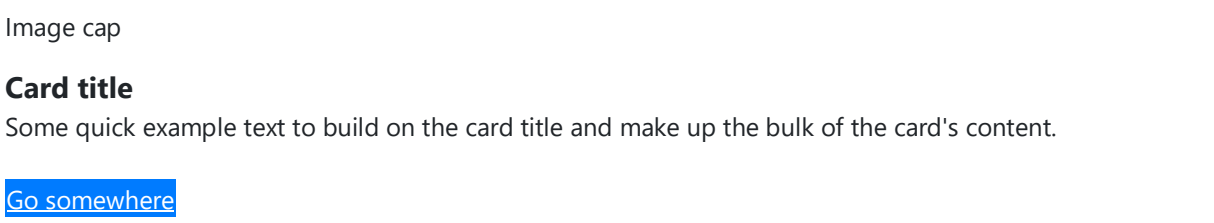
About

A **card** is a flexible and extensible content container. It includes options for headers and footers, a wide variety of content, contextual background colors, and powerful display options. If you're familiar with Bootstrap 3, cards replace our old panels, wells, and thumbnails. Similar functionality to those components is available as modifier classes for cards.

Example

Cards are built with as little markup and styles as possible, but still manage to deliver a ton of control and customization. Built with flexbox, they offer easy alignment and mix well with other Bootstrap components. They have no `margin` by default, so use spacing utilities as needed.

Below is an example of a basic card with mixed content and a fixed width. Cards have no fixed width to start, so they'll naturally fill the full width of its parent element. This is easily customized with our various sizing options.



Copy

```
<div class="card" style="width: 18rem;">
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Content types

Cards support a wide variety of content, including images, text, list groups, links, and more. Below are examples of what's supported.

Body

The building block of a card is the `.card-body`. Use it whenever you need a padded section within a card.

Copy

```
<div class="card">
  <div class="card-body">
    This is some text within a card body.
  </div>
</div>
```

Titles, text, and links

Card titles are used by adding `.card-title` to a `<h*>` tag. In the same way, links are added and placed next to each other by adding `.card-link` to an `<a>` tag.

Subtitles are used by adding a `.card-subtitle` to a `<h*>` tag. If the `.card-title` and the `.card-subtitle` items are placed in a `.card-body` item, the card title and subtitle are aligned nicely.

Card title

Card subtitle

Some quick example text to build on the card title and make up the bulk of the card's content.

Card link Another link

Copy

```
<div class="card" style="width: 18rem;">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <h6 class="card-subtitle mb-2 text-muted">Card subtitle</h6>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
    <a href="#" class="card-link">Card link</a>
    <a href="#" class="card-link">Another link</a>
  </div>
</div>
```

Images

`.card-img-top` places an image to the top of the card. With `.card-text`, text can be added to the card. Text within `.card-text` can also be styled with the standard HTML tags.

Image cap

Some quick example text to build on the card title and make up the bulk of the card's content.

Copy

```
<div class="card" style="width: 18rem;">
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
```

List groups

Create lists of content in a card with a flush list group.

- Cras justo odio
- Dapibus ac facilisis in
- Vestibulum at eros

Copy

```
<div class="card" style="width: 18rem;">
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Cras justo odio</li>
    <li class="list-group-item">Dapibus ac facilisis in</li>
    <li class="list-group-item">Vestibulum at eros</li>
  </ul>
</div>
```

Featured

- Cras justo odio
- Dapibus ac facilisis in
- Vestibulum at eros

Copy

```
<div class="card" style="width: 18rem;">
  <div class="card-header">
    Featured
  </div>
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Cras justo odio</li>
    <li class="list-group-item">Dapibus ac facilisis in</li>
    <li class="list-group-item">Vestibulum at eros</li>
  </ul>
</div>
```

Kitchen sink

Mix and match multiple content types to create the card you need, or throw everything in there. Shown below are image styles, blocks, text styles, and a list group—all wrapped in a fixed-width card.

Image cap

Card title

Some quick example text to build on the card title and make up the bulk of the card's content.

- Cras justo odio
- Dapibus ac facilisis in
- Vestibulum at eros

Card link Another link

Copy

```
<div class="card" style="width: 18rem;">
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Cras justo odio</li>
    <li class="list-group-item">Dapibus ac facilisis in</li>
    <li class="list-group-item">Vestibulum at eros</li>
  </ul>
  <div class="card-body">
    <a href="#" class="card-link">Card link</a>
    <a href="#" class="card-link">Another link</a>
  </div>
</div>
```

Header and footer

Add an optional header and/or footer within a card.

Featured

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

Copy

```
<div class="card">
  <div class="card-header">
    Featured
  </div>
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Card headers can be styled by adding `.card-header` to `<h*>` elements.

Featured

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

Copy

```
<div class="card">
  <h5 class="card-header">Featured</h5>
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Quote

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

Someone famous in *Source Title*

Copy

```
<div class="card">
  <div class="card-header">
    Quote
  </div>
  <div class="card-body">
    <blockquote class="blockquote mb-0">
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere
erat a ante.</p>
      <footer class="blockquote-footer">Someone famous in <cite title="Source
Title">Source Title</cite></footer>
    </blockquote>
```

```
  </div>
</div>
```

Featured

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

2 days ago

Copy

```
<div class="card text-center">
  <div class="card-header">
    Featured
  </div>
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
  <div class="card-footer text-muted">
    2 days ago
  </div>
</div>
```

Sizing

Cards assume no specific `width` to start, so they'll be 100% wide unless otherwise stated. You can change this as needed with custom CSS, grid classes, grid Sass mixins, or utilities.

Using grid markup

Using the grid, wrap cards in columns and rows as needed.

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

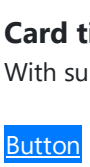
Copy

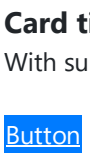
```
<div class="row">
  <div class="col-sm-6">
    <div class="card">
      <div class="card-body">
        <h5 class="card-title">Special title treatment</h5>
```

```
        <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
        <a href="#" class="btn btn-primary">Go somewhere</a>
      </div>
    </div>
  </div>
  <div class="col-sm-6">
    <div class="card">
      <div class="card-body">
        <h5 class="card-title">Special title treatment</h5>
        <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
        <a href="#" class="btn btn-primary">Go somewhere</a>
      </div>
    </div>
  </div>
</div>
```

Using utilities

Use our handful of available sizing utilities to quickly set a card's width.





Copy

```
<div class="card w-75">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Button</a>
  </div>
</div>

<div class="card w-50">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Button</a>
  </div>
</div>
```

Using custom CSS

Use custom CSS in your stylesheets or as inline styles to set a width.

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

Copy

```
<div class="card" style="width: 18rem;">
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Text alignment

You can quickly change the text alignment of any card—in its entirety or specific parts—with our text align classes.

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

Copy

```
<div class="card" style="width: 18rem;">
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>

<div class="card text-center" style="width: 18rem;">
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

```
<div class="card text-right" style="width: 18rem;">
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Navigation

Add some navigation to a card's header (or block) with Bootstrap's nav components.

- Active
- Link
- Disabled

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

Copy

```
<div class="card text-center">
  <div class="card-header">
    <ul class="nav nav-tabs card-header-tabs">
      <li class="nav-item">
        <a class="nav-link active" href="#">Active</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled" href="#" tabindex="-1" aria-
disabled="true">Disabled</a>
      </li>
    </ul>
  </div>
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

- Active
- Link
- Disabled

Special title treatment

With supporting text below as a natural lead-in to additional content.

Go somewhere

Copy

```
<div class="card text-center">
  <div class="card-header">
    <ul class="nav nav-pills card-header-pills">
      <li class="nav-item">
        <a class="nav-link active" href="#">Active</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Link</a>
      </li>
      <li class="nav-item">
        <a class="nav-link disabled" href="#" tabindex="-1" aria-
disabled="true">Disabled</a>
      </li>
    </ul>
  </div>
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-in to
additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

Images

Cards include a few options for working with images. Choose from appending “image caps” at either end of a card, overlaying images with card content, or simply embedding the image in a card.

Image caps

Similar to headers and footers, cards can include top and bottom “image caps”—images at the top or bottom of a card.

Image cap

Card title
This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Last updated 3 mins ago
Card title
This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Last updated 3 mins ago
Image cap

Copy

```
<div class="card mb-3">
  <img src="..." class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
```

```
    <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This content is a little bit longer.</p>
    <p class="card-text"><small class="text-muted">Last updated 3 mins
ago</small></p>
  </div>
</div>
<div class="card">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This content is a little bit longer.</p>
    <p class="card-text"><small class="text-muted">Last updated 3 mins
ago</small></p>
  </div>
  <img src="..." class="card-img-top" alt="...">
</div>
```

Image overlays

Turn an image into a card background and overlay your card's text. Depending on the image, you may or may not need additional styles or utilities.

Card image

Card title
This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Last updated 3 mins ago
Copy

```
<div class="card bg-dark text-white">
  <img src="..." class="card-img" alt="...">
  <div class="card-img-overlay">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This content is a little bit longer.</p>
    <p class="card-text">Last updated 3 mins ago</p>
  </div>
</div>
```

Note that content should not be larger than the height of the image. If content is larger than the image the content will be displayed outside the image.

Horizontal

Using a combination of grid and utility classes, cards can be made horizontal in a mobile-friendly and responsive way. In the example below, we remove the grid gutters with `.no-gutters` and use `.col-md-*` classes to make the card horizontal at the `md` breakpoint. Further adjustments may be needed depending on your card content.

Image

Card title
This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Last updated 3 mins ago
Copy

```
<div class="card mb-3" style="max-width: 540px;">
  <div class="row no-gutters">
    <div class="col-md-4">
      <img src="..." class="card-img" alt="...">
    </div>
    <div class="col-md-8">
      <div class="card-body">
        <h5 class="card-title">Card title</h5>
        <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This content is a little bit longer.</p>
        <p class="card-text"><small class="text-muted">Last updated 3 mins
ago</small></p>
      </div>
    </div>
  </div>
</div>
```

Card styles

Cards include various options for customizing their backgrounds, borders, and color.

Background and color

Use text and background utilities to change the appearance of a card.

Header

Primary card title
Some quick example text to build on the card title and make up the bulk of the card's content.
Header

Secondary card title
Some quick example text to build on the card title and make up the bulk of the card's content.
Header

Success card title
Some quick example text to build on the card title and make up the bulk of the card's content.
Header

Danger card title
Some quick example text to build on the card title and make up the bulk of the card's content.
Header

Warning card title
Some quick example text to build on the card title and make up the bulk of the card's content.
Header

Info card title
Some quick example text to build on the card title and make up the bulk of the card's content.
Header

Light card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Header

Dark card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Copy

```
<div class="card text-white bg-primary mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body">
    <h5 class="card-title">Primary card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card text-white bg-secondary mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body">
    <h5 class="card-title">Secondary card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card text-white bg-success mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body">
    <h5 class="card-title">Success card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card text-white bg-danger mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body">
    <h5 class="card-title">Danger card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card text-white bg-warning mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body">
    <h5 class="card-title">Warning card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card text-white bg-info mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body">
    <h5 class="card-title">Info card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card bg-light mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body">
```

```
    <h5 class="card-title">Light card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card text-white bg-dark mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body">
    <h5 class="card-title">Dark card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
```

Conveying meaning to assistive technologies

Using color to add meaning only provides a visual indication, which will not be conveyed to users of assistive technologies – such as screen readers. Ensure that information denoted by the color is either obvious from the content itself (e.g. the visible text), or is included through alternative means, such as additional text hidden with the `.sr-only` class.

Border

Use border utilities to change just the `border-color` of a card. Note that you can put `.text-{color}` classes on the parent `.card` or a subset of the card's contents as shown below.

Header

Primary card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Header

Secondary card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Header

Success card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Header

Danger card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Header

Warning card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Header

Info card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Header

Light card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Header

Dark card title

Some quick example text to build on the card title and make up the bulk of the card's content.

Copy

```
<div class="card border-primary mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body text-primary">
    <h5 class="card-title">Primary card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card border-secondary mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body text-secondary">
    <h5 class="card-title">Secondary card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card border-success mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body text-success">
    <h5 class="card-title">Success card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card border-danger mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body text-danger">
    <h5 class="card-title">Danger card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card border-warning mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body text-warning">
    <h5 class="card-title">Warning card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card border-info mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body text-info">
    <h5 class="card-title">Info card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
<div class="card border-light mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body">
    <h5 class="card-title">Light card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
```

```
</div>
<div class="card border-dark mb-3" style="max-width: 18rem;">
  <div class="card-header">Header</div>
  <div class="card-body text-dark">
    <h5 class="card-title">Dark card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
</div>
```

Mixins utilities

You can also change the borders on the card header and footer as needed, and even remove their `background-color` with `.bg-transparent`.

Header

Success card title
Some quick example text to build on the card title and make up the bulk of the card's content.
Footer

Copy

```
<div class="card border-success mb-3" style="max-width: 18rem;">
  <div class="card-header bg-transparent border-success">Header</div>
  <div class="card-body text-success">
    <h5 class="card-title">Success card title</h5>
    <p class="card-text">Some quick example text to build on the card title and
make up the bulk of the card's content.</p>
  </div>
  <div class="card-footer bg-transparent border-success">Footer</div>
</div>
```

Card layout

In addition to styling the content within cards, Bootstrap includes a few options for laying out series of cards. For the time being, **these layout options are not yet responsive**.

Card groups

Use card groups to render cards as a single, attached element with equal width and height columns. Card groups use `display: flex;` to achieve their uniform sizing.

Image cap

Card title
This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Last updated 3 mins ago
Image cap

Card title
This card has supporting text below as a natural lead-in to additional content.

Last updated 3 mins ago

Image cap

Card title

This is a wider card with supporting text below as a natural lead-in to additional content. This card has even longer content than the first to show that equal height action.

Last updated 3 mins ago

Copy

```
<div class="card-group">
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This content is a little bit longer.</p>
      <p class="card-text"><small class="text-muted">Last updated 3 mins
ago</small></p>
    </div>
  </div>
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This card has supporting text below as a natural lead-
in to additional content.</p>
      <p class="card-text"><small class="text-muted">Last updated 3 mins
ago</small></p>
    </div>
  </div>
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This card has even longer content than the
first to show that equal height action.</p>
      <p class="card-text"><small class="text-muted">Last updated 3 mins
ago</small></p>
    </div>
  </div>
</div>
```

When using card groups with footers, their content will automatically line up.

Image cap

Card title

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Last updated 3 mins ago

Image cap

Card title

This card has supporting text below as a natural lead-in to additional content.

Last updated 3 mins ago

Image cap

Card title

This is a wider card with supporting text below as a natural lead-in to additional content. This card has even longer content than the first to show that equal height action.

Last updated 3 mins ago

Copy

```
<div class="card-group">
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This content is a little bit longer.</p>
    </div>
    <div class="card-footer">
      <small class="text-muted">Last updated 3 mins ago</small>
    </div>
  </div>
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This card has supporting text below as a natural lead-
in to additional content.</p>
    </div>
    <div class="card-footer">
      <small class="text-muted">Last updated 3 mins ago</small>
    </div>
  </div>
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This card has even longer content than the
first to show that equal height action.</p>
    </div>
    <div class="card-footer">
      <small class="text-muted">Last updated 3 mins ago</small>
    </div>
  </div>
</div>
```

Card decks

Need a set of equal width and height cards that aren't attached to one another? Use card decks.

Image cap

Card title

This is a longer card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Last updated 3 mins ago

Image cap

Card title
This card has supporting text below as a natural lead-in to additional content.

Last updated 3 mins ago
Image cap

Card title
This is a wider card with supporting text below as a natural lead-in to additional content. This card has even longer content than the first to show that equal height action.

Last updated 3 mins ago
Copy

```
<div class="card-deck">
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This is a longer card with supporting text below as a
natural lead-in to additional content. This content is a little bit longer.</p>
      <p class="card-text"><small class="text-muted">Last updated 3 mins
ago</small></p>
    </div>
  </div>
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This card has supporting text below as a natural lead-
in to additional content.</p>
      <p class="card-text"><small class="text-muted">Last updated 3 mins
ago</small></p>
    </div>
  </div>
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This card has even longer content than the
first to show that equal height action.</p>
      <p class="card-text"><small class="text-muted">Last updated 3 mins
ago</small></p>
    </div>
  </div>
</div>
```

Just like with card groups, card footers in decks will automatically line up.

Image cap

Card title
This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.
Last updated 3 mins ago

Image cap

Card title

This card has supporting text below as a natural lead-in to additional content.

Last updated 3 mins ago

Image cap

Card title

This is a wider card with supporting text below as a natural lead-in to additional content. This card has even longer content than the first to show that equal height action.

Last updated 3 mins ago

Copy

```
<div class="card-deck">
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This content is a little bit longer.</p>
    </div>
    <div class="card-footer">
      <small class="text-muted">Last updated 3 mins ago</small>
    </div>
  </div>
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This card has supporting text below as a natural lead-
in to additional content.</p>
    </div>
    <div class="card-footer">
      <small class="text-muted">Last updated 3 mins ago</small>
    </div>
  </div>
  <div class="card">
    <img src="..." class="card-img-top" alt="...">
    <div class="card-body">
      <h5 class="card-title">Card title</h5>
      <p class="card-text">This is a wider card with supporting text below as a
natural lead-in to additional content. This card has even longer content than the
first to show that equal height action.</p>
    </div>
    <div class="card-footer">
      <small class="text-muted">Last updated 3 mins ago</small>
    </div>
  </div>
</div>
```

Forms

Examples and usage guidelines for form control styles, layout options, and custom components for creating a wide variety of forms.

Overview

Bootstrap's form controls expand on our Rebooted form styles with classes. Use these classes to opt into their customized displays for a more consistent rendering across browsers and devices.

Be sure to use an appropriate `type` attribute on all inputs (e.g., `email` for email address or `number` for numerical information) to take advantage of newer input controls like email verification, number selection, and more.

Here's a quick example to demonstrate Bootstrap's form styles. Keep reading for documentation on required classes, form layout, and more.

Email addressWe'll never share your email with anyone else.

Password

Check me out

Submit

Copy

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1" aria-
describedby="emailHelp">
    <small id="emailHelp" class="form-text text-muted">We'll never share your
email with anyone else.</small>
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1">
  </div>
  <div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="exampleCheck1">
    <label class="form-check-label" for="exampleCheck1">Check me out</label>
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Form controls

Textual form controls—like `<input>`s, `<select>`s, and `<textarea>`s—are styled with the `.form-control` class. Included are styles for general appearance, focus state, sizing, and more.

Be sure to explore our custom forms to further style `<select>`s.

Email address

Example select

Example multiple select

Example textarea

Copy

```
<form>
  <div class="form-group">
    <label for="exampleFormControlInput1">Email address</label>
    <input type="email" class="form-control" id="exampleFormControlInput1"
placeholder="name@example.com">
  </div>
  <div class="form-group">
    <label for="exampleFormControlSelect1">Example select</label>
    <select class="form-control" id="exampleFormControlSelect1">
      <option>1</option>
      <option>2</option>
      <option>3</option>
      <option>4</option>
      <option>5</option>
    </select>
  </div>
  <div class="form-group">
    <label for="exampleFormControlSelect2">Example multiple select</label>
    <select multiple class="form-control" id="exampleFormControlSelect2">
      <option>1</option>
      <option>2</option>
      <option>3</option>
      <option>4</option>
      <option>5</option>
    </select>
  </div>
  <div class="form-group">
    <label for="exampleFormControlTextarea1">Example textarea</label>
    <textarea class="form-control" id="exampleFormControlTextarea1"
rows="3"></textarea>
  </div>
</form>
```

For file inputs, swap the `.form-control` for `.form-control-file`.

Example file input

Copy

```
<form>
  <div class="form-group">
    <label for="exampleFormControlFile1">Example file input</label>
    <input type="file" class="form-control-file" id="exampleFormControlFile1">
  </div>
</form>
```

Sizing

Set heights using classes like `.form-control-lg` and `.form-control-sm`.

Copy

```
<input class="form-control form-control-lg" type="text" placeholder=".form-
control-lg">
<input class="form-control" type="text" placeholder="Default input">
<input class="form-control form-control-sm" type="text" placeholder=".form-
control-sm">
```

Copy

```
<select class="form-control form-control-lg">
  <option>Large select</option>
</select>
<select class="form-control">
  <option>Default select</option>
</select>
<select class="form-control form-control-sm">
  <option>Small select</option>
</select>
```

Readonly

Add the `readonly` boolean attribute on an input to prevent modification of the input's value. Read-only inputs appear lighter (just like disabled inputs), but retain the standard cursor.

Copy

```
<input class="form-control" type="text" placeholder="Readonly input here..."
readonly>
```

Checkboxes and radios

Default checkboxes and radios are improved upon with the help of `.form-check`, **a single class for both input types that improves the layout and behavior of their HTML elements**. Checkboxes are for selecting one or several options in a list, while radios are for selecting one option from many.

Disabled checkboxes and radios are supported. The `disabled` attribute will apply a lighter color to help indicate the input's state.

Checkboxes and radio buttons support HTML-based form validation and provide concise, accessible labels. As such, our `<input>`s and `<label>`s are sibling elements as opposed to an `<input>` within a `<label>`. This is slightly more verbose as you must specify `id` and `for` attributes to relate the `<input>` and `<label>`.

Default (stacked)

By default, any number of checkboxes and radios that are immediate sibling will be vertically stacked and appropriately spaced with `.form-check`.

- [ ] Default checkbox
- [ ] Disabled checkbox

Copy

```
<div class="form-check">
  <input class="form-check-input" type="checkbox" value="" id="defaultCheck1">
  <label class="form-check-label" for="defaultCheck1">
    Default checkbox
  </label>
</div>
<div class="form-check">
  <input class="form-check-input" type="checkbox" value="" id="defaultCheck2"
disabled>
  <label class="form-check-label" for="defaultCheck2">
    Disabled checkbox
  </label>
</div>
```

- (•) Default radio
- ( ) Second default radio
- ( ) Disabled radio

Copy

```
<div class="form-check">
  <input class="form-check-input" type="radio" name="exampleRadios"
id="exampleRadios1" value="option1" checked>
  <label class="form-check-label" for="exampleRadios1">
    Default radio
  </label>
</div>
<div class="form-check">
  <input class="form-check-input" type="radio" name="exampleRadios"
id="exampleRadios2" value="option2">
  <label class="form-check-label" for="exampleRadios2">
    Second default radio
  </label>
</div>
<div class="form-check">
  <input class="form-check-input" type="radio" name="exampleRadios"
id="exampleRadios3" value="option3" disabled>
  <label class="form-check-label" for="exampleRadios3">
    Disabled radio
  </label>
</div>
```

Inline

Group checkboxes or radios on the same horizontal row by adding `.form-check-inline` to any `.form-check`.

- [ ] 1

2

3 (disabled)

Copy

```
<div class="form-check form-check-inline">
  <input class="form-check-input" type="checkbox" id="inlineCheckbox1"
value="option1">
  <label class="form-check-label" for="inlineCheckbox1">1</label>
</div>
<div class="form-check form-check-inline">
  <input class="form-check-input" type="checkbox" id="inlineCheckbox2"
value="option2">
  <label class="form-check-label" for="inlineCheckbox2">2</label>
</div>
<div class="form-check form-check-inline">
  <input class="form-check-input" type="checkbox" id="inlineCheckbox3"
value="option3" disabled>
  <label class="form-check-label" for="inlineCheckbox3">3 (disabled)</label>
</div>
```

1

2

3 (disabled)

Copy

```
<div class="form-check form-check-inline">
  <input class="form-check-input" type="radio" name="inlineRadioOptions"
id="inlineRadio1" value="option1">
  <label class="form-check-label" for="inlineRadio1">1</label>
</div>
<div class="form-check form-check-inline">
  <input class="form-check-input" type="radio" name="inlineRadioOptions"
id="inlineRadio2" value="option2">
  <label class="form-check-label" for="inlineRadio2">2</label>
</div>
<div class="form-check form-check-inline">
  <input class="form-check-input" type="radio" name="inlineRadioOptions"
id="inlineRadio3" value="option3" disabled>
  <label class="form-check-label" for="inlineRadio3">3 (disabled)</label>
</div>
```

Without labels

Add `.position-static` to inputs within `.form-check` that don't have any label text. Remember to still provide some form of label for assistive technologies (for instance, using `aria-label`).

Copy

```
<div class="form-check">
  <input class="form-check-input position-static" type="checkbox"
id="blankCheckbox" value="option1" aria-label="...">
</div>
<div class="form-check">
```

```
  <input class="form-check-input position-static" type="radio" name="blankRadio"
id="blankRadio1" value="option1" aria-label="...">
</div>
```

Layout

Since Bootstrap applies `display: block` and `width: 100%` to almost all our form controls, forms will by default stack vertically. Additional classes can be used to vary this layout on a per-form basis.

Form groups

The `.form-group` class is the easiest way to add some structure to forms. It provides a flexible class that encourages proper grouping of labels, controls, optional help text, and form validation messaging. By default it only applies `margin-bottom`, but it picks up additional styles in `.form-inline` as needed. Use it with `<fieldset>`s, `<div>`s, or nearly any other element.

Example label

Another label

Copy

```
<form>
  <div class="form-group">
    <label for="formGroupExampleInput">Example label</label>
    <input type="text" class="form-control" id="formGroupExampleInput"
placeholder="Example input placeholder">
  </div>
  <div class="form-group">
    <label for="formGroupExampleInput2">Another label</label>
    <input type="text" class="form-control" id="formGroupExampleInput2"
placeholder="Another input placeholder">
  </div>
</form>
```

Horizontal form

Create horizontal forms with the grid by adding the `.row` class to form groups and using the `.col-*-*` classes to specify the width of your labels and controls. Be sure to add `.col-form-label` to your `<label>`s as well so they're vertically centered with their associated form controls.

At times, you maybe need to use margin or padding utilities to create that perfect alignment you need. For example, we've removed the `padding-top` on our stacked radio inputs label to better align the text baseline.

Email

Password

Radios

First radio

Second radio

Third disabled radio

Checkbox

Example checkbox

Sign in

Copy

```
<form>
  <div class="form-group row">
    <label for="inputEmail3" class="col-sm-2 col-form-label">Email</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="inputEmail3">
    </div>
  </div>
  <div class="form-group row">
    <label for="inputPassword3" class="col-sm-2 col-form-label">Password</label>
    <div class="col-sm-10">
      <input type="password" class="form-control" id="inputPassword3">
    </div>
  </div>
  <fieldset class="form-group">
    <div class="row">
      <legend class="col-form-label col-sm-2 pt-0">Radios</legend>
      <div class="col-sm-10">
        <div class="form-check">
          <input class="form-check-input" type="radio" name="gridRadios"
id="gridRadios1" value="option1" checked>
          <label class="form-check-label" for="gridRadios1">
            First radio
          </label>
        </div>
        <div class="form-check">
          <input class="form-check-input" type="radio" name="gridRadios"
id="gridRadios2" value="option2">
          <label class="form-check-label" for="gridRadios2">
            Second radio
          </label>
        </div>
        <div class="form-check disabled">
          <input class="form-check-input" type="radio" name="gridRadios"
id="gridRadios3" value="option3" disabled>
          <label class="form-check-label" for="gridRadios3">
            Third disabled radio
          </label>
        </div>
      </div>
    </div>
  </fieldset>
  <div class="form-group row">
    <div class="col-sm-2">Checkbox</div>
    <div class="col-sm-10">
```

```
      <div class="form-check">
        <input class="form-check-input" type="checkbox" id="gridCheck1">
        <label class="form-check-label" for="gridCheck1">
          Example checkbox
        </label>
      </div>
    </div>
  </div>
  <div class="form-group row">
    <div class="col-sm-10">
      <button type="submit" class="btn btn-primary">Sign in</button>
    </div>
  </div>
</form>
```

Horizontal form label sizing

Be sure to use `.col-form-label-sm` or `.col-form-label-lg` to your `<label>`s or `<legend>`s to correctly follow the size of `.form-control-lg` and `.form-control-sm`.

Email

Email

Email

Copy

```
<form>
  <div class="form-group row">
    <label for="colFormLabelSm" class="col-sm-2 col-form-label col-form-label-
sm">Email</label>
    <div class="col-sm-10">
      <input type="email" class="form-control form-control-sm" id="colFormLabelSm"
placeholder="col-form-label-sm">
    </div>
  </div>
  <div class="form-group row">
    <label for="colFormLabel" class="col-sm-2 col-form-label">Email</label>
    <div class="col-sm-10">
      <input type="email" class="form-control" id="colFormLabel" placeholder="col-
form-label">
    </div>
  </div>
  <div class="form-group row">
    <label for="colFormLabelLg" class="col-sm-2 col-form-label col-form-label-
lg">Email</label>
    <div class="col-sm-10">
      <input type="email" class="form-control form-control-lg" id="colFormLabelLg"
placeholder="col-form-label-lg">
    </div>
  </div>
</form>
```

Column sizing

As shown in the previous examples, our grid system allows you to place any number of `.col`s within a `.row` or `.form-row`. They'll split the available width equally between them. You may also pick a subset of your columns to take up more or less space, while the remaining `.col`s equally split the rest, with specific column classes like `.col-7`.

Copy

```
<form>
  <div class="form-row">
    <div class="col-7">
      <input type="text" class="form-control" placeholder="City">
    </div>
    <div class="col">
      <input type="text" class="form-control" placeholder="State">
    </div>
    <div class="col">
      <input type="text" class="form-control" placeholder="Zip">
    </div>
  </div>
</form>
```

Auto-sizing

The example below uses a flexbox utility to vertically center the contents and changes `.col` to `.col-auto` so that your columns only take up as much space as needed. Put another way, the column sizes itself based on the contents.

Name 

Username

@

 Remember me

Submit

Copy

```
<form>
  <div class="form-row align-items-center">
    <div class="col-auto">
      <label class="sr-only" for="inlineFormInput">Name</label>
      <input type="text" class="form-control mb-2" id="inlineFormInput"
placeholder="Jane Doe">
    </div>
    <div class="col-auto">
      <label class="sr-only" for="inlineFormInputGroup">Username</label>
      <div class="input-group mb-2">
        <div class="input-group-prepend">
          <div class="input-group-text">@</div>
        </div>
        <input type="text" class="form-control" id="inlineFormInputGroup"
placeholder="Username">
      </div>
```

```
    </div>
    <div class="col-auto">
      <div class="form-check mb-2">
        <input class="form-check-input" type="checkbox" id="autoSizingCheck">
        <label class="form-check-label" for="autoSizingCheck">
          Remember me
        </label>
      </div>
    </div>
    <div class="col-auto">
      <button type="submit" class="btn btn-primary mb-2">Submit</button>
    </div>
  </div>
</form>
```

You can then remix that once again with size-specific column classes.

Name

Username

@

Remember me

Submit

Copy

```
<form>
  <div class="form-row align-items-center">
    <div class="col-sm-3 my-1">
      <label class="sr-only" for="inlineFormInputName">Name</label>
      <input type="text" class="form-control" id="inlineFormInputName"
placeholder="Jane Doe">
    </div>
    <div class="col-sm-3 my-1">
      <label class="sr-only" for="inlineFormInputGroupUsername">Username</label>
      <div class="input-group">
        <div class="input-group-prepend">
          <div class="input-group-text">@</div>
        </div>
        <input type="text" class="form-control" id="inlineFormInputGroupUsername"
placeholder="Username">
      </div>
    </div>
    <div class="col-auto my-1">
      <div class="form-check">
        <input class="form-check-input" type="checkbox" id="autoSizingCheck2">
        <label class="form-check-label" for="autoSizingCheck2">
          Remember me
        </label>
      </div>
    </div>
    <div class="col-auto my-1">
      <button type="submit" class="btn btn-primary">Submit</button>
    </div>
```

```
  </div>
</form>
```

And of course custom form controls are supported.

Preference Choose...

Remember my preference

Submit

Copy

```
<form>
  <div class="form-row align-items-center">
    <div class="col-auto my-1">
      <label class="mr-sm-2 sr-only"
for="inlineFormCustomSelect">Preference</label>
      <select class="custom-select mr-sm-2" id="inlineFormCustomSelect">
        <option selected>Choose...</option>
        <option value="1">One</option>
        <option value="2">Two</option>
        <option value="3">Three</option>
      </select>
    </div>
    <div class="col-auto my-1">
      <div class="custom-control custom-checkbox mr-sm-2">
        <input type="checkbox" class="custom-control-input"
id="customControlAutosizing">
        <label class="custom-control-label" for="customControlAutosizing">Remember
my preference</label>
      </div>
    </div>
    <div class="col-auto my-1">
      <button type="submit" class="btn btn-primary">Submit</button>
    </div>
  </div>
</form>
```

Inline forms

Use the `.form-inline` class to display a series of labels, form controls, and buttons on a single horizontal row. Form controls within inline forms vary slightly from their default states.

- Controls are `display: flex`, collapsing any HTML white space and allowing you to provide alignment control with spacing and flexbox utilities.
- Controls and input groups receive `width: auto` to override the Bootstrap default `width: 100%`.
- Controls **only appear inline in viewports that are at least 576px wide** to account for narrow viewports on mobile devices.

You may need to manually address the width and alignment of individual form controls with spacing utilities (as shown below). Lastly, be sure to always include a `<label>` with each form control, even if you need to hide it from non-screenreader visitors with `.sr-only`.

Name Username

@

Remember me

Submit

Copy

```
<form class="form-inline">
  <label class="sr-only" for="inlineFormInputName2">Name</label>
  <input type="text" class="form-control mb-2 mr-sm-2" id="inlineFormInputName2"
placeholder="Jane Doe">

  <label class="sr-only" for="inlineFormInputGroupUsername2">Username</label>
  <div class="input-group mb-2 mr-sm-2">
    <div class="input-group-prepend">
      <div class="input-group-text">@</div>
    </div>
    <input type="text" class="form-control" id="inlineFormInputGroupUsername2"
placeholder="Username">
  </div>

  <div class="form-check mb-2 mr-sm-2">
    <input class="form-check-input" type="checkbox" id="inlineFormCheck">
    <label class="form-check-label" for="inlineFormCheck">
      Remember me
    </label>
  </div>

  <button type="submit" class="btn btn-primary mb-2">Submit</button>
</form>
```

Custom form controls and selects are also supported.

Preference Choose...

Remember my preference

Submit

Copy

```
<form class="form-inline">
  <label class="my-1 mr-2" for="inlineFormCustomSelectPref">Preference</label>
  <select class="custom-select my-1 mr-sm-2" id="inlineFormCustomSelectPref">
    <option selected>Choose...</option>
    <option value="1">One</option>
    <option value="2">Two</option>
    <option value="3">Three</option>
  </select>

  <div class="custom-control custom-checkbox my-1 mr-sm-2">
    <input type="checkbox" class="custom-control-input" id="customControlInline">
    <label class="custom-control-label" for="customControlInline">Remember my
preference</label>
  </div>
```

```
  <button type="submit" class="btn btn-primary my-1">Submit</button>
</form>
```

Alternatives to hidden labels

Assistive technologies such as screen readers will have trouble with your forms if you don't include a label for every input. For these inline forms, you can hide the labels using the `.sr-only` class. There are further alternative methods of providing a label for assistive technologies, such as the `aria-label`, `aria-labelledby` or `title` attribute. If none of these are present, assistive technologies may resort to using the `placeholder` attribute, if present, but note that use of `placeholder` as a replacement for other labelling methods is not advised.

Help text

Block-level help text in forms can be created using `.form-text` (previously known as `.help-block` in v3). Inline help text can be flexibly implemented using any inline HTML element and utility classes like `.text-muted`.

Associating help text with form controls

Help text should be explicitly associated with the form control it relates to using the `aria-describedby` attribute. This will ensure that assistive technologies—such as screen readers—will announce this help text when the user focuses or enters the control.

Help text below inputs can be styled with `.form-text`. This class includes `display: block` and adds some top margin for easy spacing from the inputs above.

Password Your password must be 8-20 characters long, contain letters and numbers, and must not contain spaces, special characters, or emoji.

Copy

```
<label for="inputPassword5">Password</label>
<input type="password" id="inputPassword5" class="form-control" aria-
describedby="passwordHelpBlock">
<small id="passwordHelpBlock" class="form-text text-muted">
  Your password must be 8-20 characters long, contain letters and numbers, and
must not contain spaces, special characters, or emoji.
</small>
```

Inline text can use any typical inline HTML element (be it a `<small>`, `<span>`, or something else) with nothing more than a utility class.

Password Must be 8-20 characters long.

Copy

```
<form class="form-inline">
  <div class="form-group">
    <label for="inputPassword6">Password</label>
    <input type="password" id="inputPassword6" class="form-control mx-sm-3" aria-
describedby="passwordHelpInline">
    <small id="passwordHelpInline" class="text-muted">
      Must be 8-20 characters long.
    </small>
  </div>
</form>
```

Disabled forms

Add the `disabled` boolean attribute on an input to prevent user interactions and make it appear lighter.

Copy

```
<input class="form-control" id="disabledInput" type="text" placeholder="Disabled input here..." disabled>
```

Add the `disabled` attribute to a `<fieldset>` to disable all the controls within.

Disabled input

Disabled select menu

Can't check this

Submit

Copy

```
<form>
  <fieldset disabled>
    <div class="form-group">
      <label for="disabledTextInput">Disabled input</label>
      <input type="text" id="disabledTextInput" class="form-control" placeholder="Disabled input">
    </div>
    <div class="form-group">
      <label for="disabledSelect">Disabled select menu</label>
      <select id="disabledSelect" class="form-control">
        <option>Disabled select</option>
      </select>
    </div>
    <div class="form-group">
      <div class="form-check">
        <input class="form-check-input" type="checkbox" id="disabledFieldsetCheck" disabled>
        <label class="form-check-label" for="disabledFieldsetCheck">
          Can't check this
        </label>
      </div>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
  </fieldset>
</form>
```

Caveat with anchors

By default, browsers will treat all native form controls (`<input>`, `<select>` and `<button>` elements) inside a `<fieldset disabled>` as disabled, preventing both keyboard and mouse interactions on them. However, if your form also includes `<a ... class="btn btn-*">` elements, these will only be given a style of `pointer-events: none`. As noted in the section about disabled state for buttons (and specifically in the sub-section for anchor elements), this CSS property is not yet standardized and isn't fully supported in Internet Explorer 10, and won't prevent keyboard users from being able to focus or activate these links. So to be safe, use custom JavaScript to disable such links.

Cross-browser compatibility

While Bootstrap will apply these styles in all browsers, Internet Explorer 11 and below don't fully support the `disabled` attribute on a `<fieldset>`. Use custom JavaScript to disable the fieldset in these browsers.

Validation

Provide valuable, actionable feedback to your users with HTML5 form validation–available in all our supported browsers. Choose from the browser default validation feedback, or implement custom messages with our built-in classes and starter JavaScript.

We currently recommend using custom validation styles, as native browser default validation messages are not consistently exposed to assistive technologies in all browsers (most notably, Chrome on desktop and mobile).

How it works

Here's how form validation works with Bootstrap:

- HTML form validation is applied via CSS's two pseudo-classes, `:invalid` and `:valid`. It applies to `<input>`, `<select>`, and `<textarea>` elements.
- Bootstrap scopes the `:invalid` and `:valid` styles to parent `.was-validated` class, usually applied to the `<form>`. Otherwise, any required field without a value shows up as invalid on page load. This way, you may choose when to activate them (typically after form submission is attempted).
- To reset the appearance of the form (for instance, in the case of dynamic form submissions using AJAX), remove the `.was-validated` class from the `<form>` again after submission.
- As a fallback, `.is-invalid` and `.is-valid` classes may be used instead of the pseudo-classes for server side validation. They do not require a `.was-validated` parent class.
- Due to constraints in how CSS works, we cannot (at present) apply styles to a `<label>` that comes before a form control in the DOM without the help of custom JavaScript.
- All modern browsers support the constraint validation API, a series of JavaScript methods for validating form controls.
- Feedback messages may utilize the browser defaults (different for each browser, and unstylable via CSS) or our custom feedback styles with additional HTML and CSS.
- You may provide custom validity messages with `setCustomValidity` in JavaScript.

With that in mind, consider the following demos for our custom form validation styles, optional server side classes, and browser defaults.

Custom styles

For custom Bootstrap form validation messages, you'll need to add the `novalidate` boolean attribute to your `<form>`. This disables the browser default feedback tooltips, but still provides access to the form validation APIs in JavaScript. Try to submit the form below; our JavaScript will intercept the submit button and relay feedback to you. When attempting to submit, you'll see the `:invalid` and `:valid` styles applied to your form controls.

Custom feedback styles apply custom colors, borders, focus styles, and background icons to better communicate feedback. Background icons for `<select>`s are only available with `.custom-select`, and not `.form-control`.

First name Mark

Last name Otto

Username

@

City

State Choose...

Zip

Agree to terms and conditions

Submit form

Copy

```
<form class="needs-validation" novalidate>
  <div class="form-row">
    <div class="col-md-4 mb-3">
      <label for="validationCustom01">First name</label>
      <input type="text" class="form-control" id="validationCustom01" value="Mark"
required>
      <div class="valid-feedback">
        Looks good!
      </div>
    </div>
    <div class="col-md-4 mb-3">
      <label for="validationCustom02">Last name</label>
      <input type="text" class="form-control" id="validationCustom02" value="Otto"
required>
      <div class="valid-feedback">
        Looks good!
      </div>
    </div>
    <div class="col-md-4 mb-3">
      <label for="validationCustomUsername">Username</label>
      <div class="input-group">
        <div class="input-group-prepend">
          <span class="input-group-text" id="inputGroupPrepend">@</span>
        </div>
        <input type="text" class="form-control" id="validationCustomUsername"
aria-describedby="inputGroupPrepend" required>
        <div class="invalid-feedback">
          Please choose a username.
        </div>
      </div>
    </div>
  </div>
  <div class="form-row">
    <div class="col-md-6 mb-3">
      <label for="validationCustom03">City</label>
      <input type="text" class="form-control" id="validationCustom03" required>
      <div class="invalid-feedback">
```

```
        Please provide a valid city.
      </div>
    </div>
    <div class="col-md-3 mb-3">
      <label for="validationCustom04">State</label>
      <select class="custom-select" id="validationCustom04" required>
        <option selected disabled value="">Choose...</option>
        <option>...</option>
      </select>
      <div class="invalid-feedback">
        Please select a valid state.
      </div>
    </div>
    <div class="col-md-3 mb-3">
      <label for="validationCustom05">Zip</label>
      <input type="text" class="form-control" id="validationCustom05" required>
      <div class="invalid-feedback">
        Please provide a valid zip.
      </div>
    </div>
  </div>
  <div class="form-group">
    <div class="form-check">
      <input class="form-check-input" type="checkbox" value="" id="invalidCheck"
required>
      <label class="form-check-label" for="invalidCheck">
        Agree to terms and conditions
      </label>
      <div class="invalid-feedback">
        You must agree before submitting.
      </div>
    </div>
  </div>
  <button class="btn btn-primary" type="submit">Submit form</button>
</form>

<script>
// Example starter JavaScript for disabling form submissions if there are invalid
fields
(function() {
  'use strict';
  window.addEventListener('load', function() {
    // Fetch all the forms we want to apply custom Bootstrap validation styles to
    var forms = document.getElementsByClassName('needs-validation');
    // Loop over them and prevent submission
    var validation = Array.prototype.filter.call(forms, function(form) {
      form.addEventListener('submit', function(event) {
        if (form.checkValidity() === false) {
          event.preventDefault();
          event.stopPropagation();
        }
        form.classList.add('was-validated');
      }, false);
    });
  }, false);
})();
</script>
```

Browser defaults

Not interested in custom validation feedback messages or writing JavaScript to change form behaviors? All good, you can use the browser defaults. Try submitting the form below. Depending on your browser and OS, you'll see a slightly different style of feedback.

While these feedback styles cannot be styled with CSS, you can still customize the feedback text through JavaScript.

First name Mark

Last name Otto

Username

@

City

State Choose...

Zip

Agree to terms and conditions

Submit form

Copy

```
<form>
  <div class="form-row">
    <div class="col-md-4 mb-3">
      <label for="validationDefault01">First name</label>
      <input type="text" class="form-control" id="validationDefault01"
value="Mark" required>
    </div>
    <div class="col-md-4 mb-3">
      <label for="validationDefault02">Last name</label>
      <input type="text" class="form-control" id="validationDefault02"
value="Otto" required>
    </div>
    <div class="col-md-4 mb-3">
      <label for="validationDefaultUsername">Username</label>
      <div class="input-group">
        <div class="input-group-prepend">
          <span class="input-group-text" id="inputGroupPrepend2">@</span>
        </div>
        <input type="text" class="form-control" id="validationDefaultUsername"
aria-describedby="inputGroupPrepend2" required>
      </div>
    </div>
  </div>
  <div class="form-row">
```

```
    <div class="col-md-6 mb-3">
      <label for="validationDefault03">City</label>
      <input type="text" class="form-control" id="validationDefault03" required>
    </div>
    <div class="col-md-3 mb-3">
      <label for="validationDefault04">State</label>
      <select class="custom-select" id="validationDefault04" required>
        <option selected disabled value="">Choose...</option>
        <option>...</option>
      </select>
    </div>
    <div class="col-md-3 mb-3">
      <label for="validationDefault05">Zip</label>
      <input type="text" class="form-control" id="validationDefault05" required>
    </div>
  </div>
  <div class="form-group">
    <div class="form-check">
      <input class="form-check-input" type="checkbox" value="" id="invalidCheck2"
required>
      <label class="form-check-label" for="invalidCheck2">
        Agree to terms and conditions
      </label>
    </div>
  </div>
  <button class="btn btn-primary" type="submit">Submit form</button>
</form>
```

Server side

We recommend using client-side validation, but in case you require server-side validation, you can indicate invalid and valid form fields with `.is-invalid` and `.is-valid`. Note that `.invalid-feedback` is also supported with these classes.

First name Mark

Looks good!

Last name Otto

Looks good!

Username

@

Please choose a username.

City

Please provide a valid city.

State Choose...

Please select a valid state.

Zip

Please provide a valid zip.

Agree to terms and conditions

You must agree before submitting.

Submit form

Copy

```
<form>
  <div class="form-row">
    <div class="col-md-4 mb-3">
      <label for="validationServer01">First name</label>
      <input type="text" class="form-control is-valid" id="validationServer01"
value="Mark" required>
      <div class="valid-feedback">
        Looks good!
      </div>
    </div>
    <div class="col-md-4 mb-3">
      <label for="validationServer02">Last name</label>
      <input type="text" class="form-control is-valid" id="validationServer02"
value="Otto" required>
      <div class="valid-feedback">
        Looks good!
      </div>
    </div>
    <div class="col-md-4 mb-3">
      <label for="validationServerUsername">Username</label>
      <div class="input-group">
        <div class="input-group-prepend">
          <span class="input-group-text" id="inputGroupPrepend3">@</span>
        </div>
        <input type="text" class="form-control is-invalid"
id="validationServerUsername" aria-describedby="inputGroupPrepend3" required>
        <div class="invalid-feedback">
          Please choose a username.
        </div>
      </div>
    </div>
  </div>
  <div class="form-row">
    <div class="col-md-6 mb-3">
      <label for="validationServer03">City</label>
      <input type="text" class="form-control is-invalid" id="validationServer03"
required>
      <div class="invalid-feedback">
        Please provide a valid city.
      </div>
    </div>
    <div class="col-md-3 mb-3">
      <label for="validationServer04">State</label>
      <select class="custom-select is-invalid" id="validationServer04" required>
        <option selected disabled value="">Choose...</option>
        <option>...</option>
      </select>
```

```
      <div class="invalid-feedback">
        Please select a valid state.
      </div>
    </div>
    <div class="col-md-3 mb-3">
      <label for="validationServer05">Zip</label>
      <input type="text" class="form-control is-invalid" id="validationServer05"
required>
      <div class="invalid-feedback">
        Please provide a valid zip.
      </div>
    </div>
  </div>
  <div class="form-group">
    <div class="form-check">
      <input class="form-check-input is-invalid" type="checkbox" value=""
id="invalidCheck3" required>
      <label class="form-check-label" for="invalidCheck3">
        Agree to terms and conditions
      </label>
      <div class="invalid-feedback">
        You must agree before submitting.
      </div>
    </div>
  </div>
  <button class="btn btn-primary" type="submit">Submit form</button>
</form>
```

Supported elements

Validation styles are available for the following form controls and components:

- `<input>`s and `<textarea>`s with `.form-control` (including up to one `.form-control` in input groups)
- `<select>`s with `.form-control` or `.custom-select`
- `.form-check`s
- `.custom-checkbox`s and `.custom-radio`s
- `.custom-file`

Carousel

A slideshow component for cycling through elements—images or slides of text—like a carousel.

How it works

The carousel is a slideshow for cycling through a series of content, built with CSS 3D transforms and a bit of JavaScript. It works with a series of images, text, or custom markup. It also includes support for previous/next controls and indicators.

In browsers where the Page Visibility API is supported, the carousel will avoid sliding when the webpage is not visible to the user (such as when the browser tab is inactive, the browser window is minimized, etc.).

The animation effect of this component is dependent on the `prefers-reduced-motion` media query. See the reduced motion section of our accessibility documentation.
Please be aware that nested carousels are not supported, and carousels are generally not compliant with accessibility standards.

Lastly, if you're building our JavaScript from source, it requires `util.js`.

Example

Carousels don't automatically normalize slide dimensions. As such, you may need to use additional utilities or custom styles to appropriately size content. While carousels support previous/next controls and indicators, they're not explicitly required. Add and customize as you see fit.

The `.active` class needs to be added to one of the slides otherwise the carousel will not be visible. Also be sure to set a unique id on the `.carousel` for optional controls, especially if you're using multiple carousels on a single page. Control and indicator elements must have a `data-target` attribute (or `href` for links) that matches the id of the `.carousel` element.

Slides only

Here's a carousel with slides only. Note the presence of the `.d-block` and `.w-100` on carousel images to prevent browser default image alignment.

Third slide

Copy

```
<div id="carouselExampleSlidesOnly" class="carousel slide" data-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
        <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
</div>
```

With controls

Adding in the previous and next controls:

First slide

Copy

```
<div id="carouselExampleControls" class="carousel slide" data-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleControls" role="button"
data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleControls" role="button"
data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

With indicators

You can also add the indicators to the carousel, alongside the controls, too.

Third slide

Copy

```
<div id="carouselExampleIndicators" class="carousel slide" data-ride="carousel">
  <ol class="carousel-indicators">
    <li data-target="#carouselExampleIndicators" data-slide-to="0"
class="active"></li>
    <li data-target="#carouselExampleIndicators" data-slide-to="1"></li>
    <li data-target="#carouselExampleIndicators" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleIndicators" role="button"
data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
```

```
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleIndicators" role="button"
data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

Jumbotron

Lightweight, flexible component for showcasing hero unit style content.

A lightweight, flexible component that can optionally extend the entire viewport to showcase key marketing messages on your site.

Hello, world!

This is a simple hero unit, a simple jumbotron-style component for calling extra attention to featured content or information.

It uses utility classes for typography and spacing to space content out within the larger container.

Copy

```
<div class="jumbotron">
  <h1 class="display-4">Hello, world!</h1>
  <p class="lead">This is a simple hero unit, a simple jumbotron-style component
for calling extra attention to featured content or information.</p>
  <hr class="my-4">
  <p>It uses utility classes for typography and spacing to space content out
within the larger container.</p>
  <a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a>
</div>
```

To make the jumbotron full width, and without rounded corners, add the `.jumbotron-fluid` modifier class and add a `.container` or `.container-fluid` within.

Fluid jumbotron

This is a modified jumbotron that occupies the entire horizontal space of its parent.

Copy

```
<div class="jumbotron jumbotron-fluid">
  <div class="container">
    <h1 class="display-4">Fluid jumbotron</h1>
    <p class="lead">This is a modified jumbotron that occupies the entire
horizontal space of its parent.</p>
  </div>
</div>
```

Pagination

Documentation and examples for showing pagination to indicate a series of related content exists across multiple pages.

Overview

We use a large block of connected links for our pagination, making links hard to miss and easily scalable—all while providing large hit areas. Pagination is built with list HTML elements so screen readers can announce the number of available links. Use a wrapping `<nav>` element to identify it as a navigation section to screen readers and other assistive technologies.

In addition, as pages likely have more than one such navigation section, it's advisable to provide a descriptive `aria-label` for the `<nav>` to reflect its purpose. For example, if the pagination component is used to navigate between a set of search results, an appropriate label could be `aria-label="Search results pages"`.

- Previous
- 1
- 2
- 3
- Next

Copy

```
<nav aria-label="Page navigation example">
  <ul class="pagination">
    <li class="page-item"><a class="page-link" href="#">Previous</a></li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item"><a class="page-link" href="#">Next</a></li>
  </ul>
</nav>
```

Working with icons

Looking to use an icon or symbol in place of text for some pagination links? Be sure to provide proper screen reader support with `aria` attributes.

- «
- 1
- 2
- 3
- »

Copy

```
<nav aria-label="Page navigation example">
  <ul class="pagination">
    <li class="page-item">
      <a class="page-link" href="#" aria-label="Previous">
        <span aria-hidden="true">&amp;laquo;</span>
      </a>
```

```
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#" aria-label="Next">
        <span aria-hidden="true">&raquo;</span>
      </a>
    </li>
  </ul>
</nav>
```

Disabled and active states

Pagination links are customizable for different circumstances. Use `.disabled` for links that appear un-clickable and `.active` to indicate the current page.

While the `.disabled` class uses `pointer-events: none` to *try* to disable the link functionality of `<a>`s, that CSS property is not yet standardized and doesn't account for keyboard navigation. As such, you should always add `tabindex="-1"` on disabled links and use custom JavaScript to fully disable their functionality.

- Previous
- 1
- 2 (current)
- 3
- Next

Copy

```
<nav aria-label="...">
  <ul class="pagination">
    <li class="page-item disabled">
      <a class="page-link" href="#" tabindex="-1" aria-
disabled="true">Previous</a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item active" aria-current="page">
      <a class="page-link" href="#">2 <span class="sr-only">(current)</span></a>
    </li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#">Next</a>
    </li>
  </ul>
</nav>
```

You can optionally swap out active or disabled anchors for `<span>`, or omit the anchor in the case of the prev/next arrows, to remove click functionality and prevent keyboard focus while retaining intended styles.

- Previous
- 1
- 2 (current)
- 3
- Next

Copy

```
<nav aria-label="...">
  <ul class="pagination">
    <li class="page-item disabled">
      <span class="page-link">Previous</span>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item active" aria-current="page">
      <span class="page-link">
        2
        <span class="sr-only">(current)</span>
      </span>
    </li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#">Next</a>
    </li>
  </ul>
</nav>
```

Sizing

Fancy larger or smaller pagination? Add `.pagination-lg` or `.pagination-sm` for additional sizes.

- 1 (current)
- 2
- 3

Copy

```
<nav aria-label="...">
  <ul class="pagination pagination-lg">
    <li class="page-item active" aria-current="page">
      <span class="page-link">
        1
        <span class="sr-only">(current)</span>
      </span>
    </li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
  </ul>
</nav>
```

- 1 (current)
- 2
- 3

Copy

```
<nav aria-label="...">
  <ul class="pagination pagination-sm">
    <li class="page-item active" aria-current="page">
      <span class="page-link">
        1
        <span class="sr-only">(current)</span>
      </span>
    </li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
```

```
    <li class="page-item"><a class="page-link" href="#">3</a></li>
  </ul>
</nav>
```

Alignment

Change the alignment of pagination components with flexbox utilities.

- Previous
- 1
- 2
- 3
- Next

Copy

```
<nav aria-label="Page navigation example">
  <ul class="pagination justify-content-center">
    <li class="page-item disabled">
      <a class="page-link" href="#" tabindex="-1" aria-
disabled="true">Previous</a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#">Next</a>
    </li>
  </ul>
</nav>
```

- Previous
- 1
- 2
- 3
- Next

Copy

```
<nav aria-label="Page navigation example">
  <ul class="pagination justify-content-end">
    <li class="page-item disabled">
      <a class="page-link" href="#" tabindex="-1" aria-
disabled="true">Previous</a>
    </li>
    <li class="page-item"><a class="page-link" href="#">1</a></li>
    <li class="page-item"><a class="page-link" href="#">2</a></li>
    <li class="page-item"><a class="page-link" href="#">3</a></li>
    <li class="page-item">
      <a class="page-link" href="#">Next</a>
    </li>
  </ul>
</nav>
```

What is Scripting?

A script is program code that doesn't need pre-processing (e.g. compiling) before being run. In the context of a Web browser, scripting usually refers to program code written in JavaScript that is executed by the browser when a page is loaded, or in response to an event triggered by the user.

Scripting can make Web pages more dynamic. For example, without reloading a new version of a page it may allow modifications to the content of that page, or allow content to be added to or sent from that page. The former has been called DHTML (Dynamic HTML), and the latter AJAX (Asynchronous JavaScript and XML).

API lib predefine programs

What scripting interfaces are available?

The most basic scripting interface developed at **W3C** is the **DOM**, the **D**ocument **O**bject **M**odel which allows programs and scripts to dynamically access and updates the content, structure and style of documents. DOM specifications form the core of DHTML.

Modifications of the content using the DOM by the user and by scripts trigger events that developers can make use of to build rich user interfaces.

Types of script: Scripts are classified into the following two types.

- Client-side script
- Server-side script

Client-side script: These scripts are getting executed within the web Browser (client). Here we don't need any software. These scripts are used for client-side validations (data verification & data validations)

Ex: JavaScript, VBScript, typescript, etc…

Server-side script: A script which executes in server machine with support of the web-server/app-server software's like **IIS**(Internet information services), Tomcat, JBOSS, etc. These scripts are used for server-side validations (authentication & authorization).

Ex: php, jsp, asp.net, nodeJS, cgi, perl etc…

What are the differences between script and language?

Script	Language
Weakly or loosely typed programming And lightweight	Strong or closely typed programming and HW
Easy to understand compare to PL	Complex to understand compare to Script
External libraries not required	Required
No special compiler required	Special compiler mandatory
Client side validation	Server/client side validation/verifications
Ex: JavaScript, VBScript, TypeScript, Perl, Shell etc.	Ex: C, CPP, vb.net, Java etc.

JavaScript Introduction

✓ In **1995**, JavaScript was created by a Netscape developer named **"Brendan Eich"**. First, it was called **Mocha**. Later, it was renamed **LiveScript**.

✓ **Netscape** first introduced a JavaScript interpreter in **Navigator**2.

Mocha(1995) → LiveScript → JavaScript(1997)

Why is it called JavaScript?

When JavaScript was created, it initially had another name: "LiveScript". But Java was very popular at that time, so it was decided that positioning a new language as a "younger brother" of Java would help. But as it evolved, JavaScript became a fully independent language with its own specification called ECMAScript, and now it has no relation to Java at all.

ECMAScript/ES JS

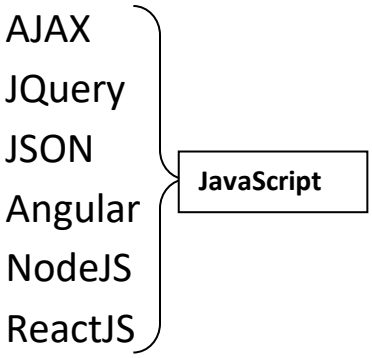
✓ Later JavaScript became an **ECMA**(**E**uropean **C**omputer **M**anufacturers **A**ssociation **S**cript) (ES) standard in 1997. **ECMAScript** is the official name of the language.

✓ **JavaScript** is implementation of **ES**; **ES** is the specification of JavaScript.

RBI SBI, HDFC, ICICI customer

ES JS Programmer

✓ JavaScript is originally it's not programming language.



These tech & frameworks using JavaScript as their PL lib

- JavaScript is a **Speed, light weight, Interoperability, Extended Functionality, dynamic**, **loosely typed**, **single threaded**, **free ware** and **open-source**.

✓ **JavaScript** is an object-based or **prototype-based** programming.

It's not OOPS

✓ JavaScript is client-side (browser-side) programming. That means it executes on the browser.

✓ It can also be used in server-side by using **NodeJS**

✓ JavaScript Translator is responsible to translate the JavaScript code which is embedded in browser.

Parser's:

Html code (high) => html parser => machine code

Css code (high) => css parser => machine code

JS code (high) => js parser => machine code

Every browser they have own JS engine:

V8 Chrome and Opera.

SpiderMonkey Firefox.
Chakra IE
JavaScriptCore, and SquirrelFish Safari
V8 Edge

✓ JavaScript is “interpreter-based” programming, means the code will be converted into machine language line-by-line. JavaScript interpreter is already embedded in Browsers.

✓ JavaScript is a case sensitive programme.
✓ To work with JavaScript, we don’t need to install any software.

Why we Use JavaScript?

Using HTML/CSS, we can only design a web page but it’s not supported to perform logical operations **such as calculations, decision making and repetitive tasks, dynamically displaying output, reading inputs from the user, and updating content on webpage at client side**. Hence to perform these entire tasks at client side we need to use JavaScript.

Where it is used?

There are so many web applications running on the web that are using JavaScript like Google, Facebook, twitter, amazon, YouTube etc.

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

1. Client-side verifications and validation
2. Dynamic drop-down menus
3. Displaying date and time
4. Build forms that respond to user input without accessing a server.
5. Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)
6. Manipulate HTML "layers" including hiding, moving, and allowing the user to drag them around a browser window.

etc...

Limitations of JavaScript

Client-Side JavaScript have some limitations which are given below;

1. Client-side JavaScript does not allow reading and writing of files.
2. It cannot be used for networking applications.
3. It doesn't have any multithreading or multiprocessing capabilities.
4. it doesn't support db connections.

JavaScript Versions

Version	Officeal Name	Release Date
1	ECMAScript 1	June-1997
2	ECMAScript 2	June-1998

3	ECMAScript 3	Dec-1998
4	ECMAScript 4	2004 (not released)
5	ECMAScript 5	Dec-2009
5.1	ECMAScript 5.1	June-2011
6	ECMAScript	June-2015
7	ECMAScript	June-2016
8	ECMAScript	June-2017

how many ways to imp js?

JS we can develop/imp in 3 ways, but in 4place.

those are:

- inline scripting
- internal scripting
- external scripting

> inline scripting

inline script nothing but writing code within the tag, by using event/dynamic attributes

for this we need tag & event attribute

onclick, onsubmit, onfocus, oninput, onload, etc..

Syn: **<tag attributes event="js code" event=”js” event=”js”>**

>internal scripting

Internal script is nothing but html code and javascript code both are placed in the same file, but not in same line.

Internal script must be implemented inside **<script>** tag, <script> is a paired tag.

> scripting in head sec

head is first executed part of html, hence javascript is also executes first.

```
<head>
    <script type="text/javascript”>
        JS code
    </script>
</head>
```

> scripting in body sec

body level script is executed after head section

```
<body>
    <script type="text/javascript”>
       JS code
    </script>
</body>
```

> external scripting

> external script is nothing but html code and javascript code designed in separate files

> type js code in sep file and save that file with "filename.js"

> re-use

>while writing external script don’t use **<script>** tag and event attribute.

External file Syn:

```
function fun-name()
{
    Steps
}
```

Fun

OR

```
{
    Steps
}
```

block

Note: external file should be saved with an extension “.js”

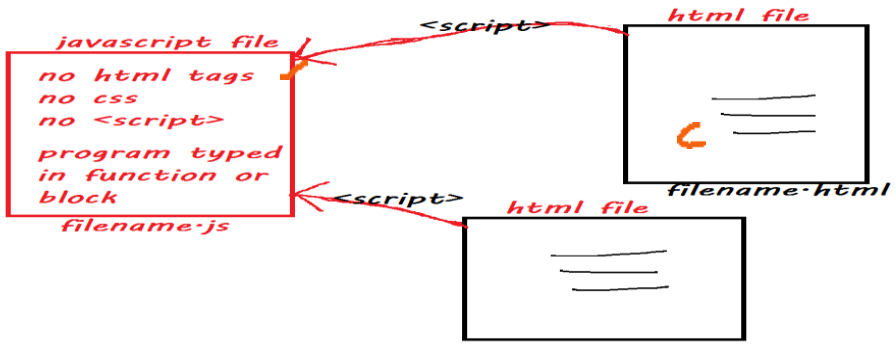
>we can access external script by using <script> tag from html.

> from either head nor body section

Syn:

```
<script src="filename1.js"> </script>
<script src="filename2.js"> </script>
```

link



Comments in JavaScript

Comment is nothing but it is a statement which is not display on browser window. It is useful to understand which code is written for what purpose.

Comments are useful in every programming language to deliver message. It is used to add information about the code, warnings or suggestions so that the end user or other developer can easily interpret the code.

Types of Comments:

There are two types of comments are in JavaScript

1. Single-line Comment ex: **//comment**
2. Multi-line Comment ex: **/* comments */**

Single-line Comment

It is represented by double forward slashes //. It can be used before any statement.

Example:

```
<script>
// It is single line comment
document.write("Hello Javascript");
</script>
```

Multi-line Comment

It can be used to add single as well as multi line comments.

It is represented by forward slash / with asterisk * then asterisk with forward slash.

Example:

```
<script>
```

/* It is multi line comment.

It will not be displayed */

document.write("Javascript multiline comment");

</script>

JS keyword/reserve words, functions/methods, classes, objects

DOM => loading page

window & document both are implicit objects created by browser, @time loading a web page.

"window" is base object for all JS objects.

"window" object used for interacting with browser window.

"document" is the sub object of window.

"document" object used for interacting with web page/web document.

"console" object used for interacting with browser console.

write() method: The write() method writes HTML expressions or javascript code to a document without line breaking.

Syn: document.write(val1, val2, val3, ….);
document.write(exp1 + exp2 + exp3 + ...);

writeln() method: The writeln() method writes HTML expressions or javascript code to a document with line breaking.

Syn: document.writeln(exp1, exp2, exp3, ...);
document.writeln(exp1 + exp2 + exp3 + ...);

log() method: The log() method writes HTML expressions or javascript code on browser's console (press F12) with line break.

Syn: console.log(exp1, exp2, exp3, ...);
console.log(exp1 + exp2 + exp3 + ...);

Example:

```
<html>
<head>
<script type='text/javascript'>
        document.write("<h1>hello world!</h1><p> have a nice day ! </p>");
</script>
</head>
</html>
```

Can we use HTML tags in write() method?

Yes, we can use tags in write().

writeln() method: The writeln() method is similar to the write method, with the addition of writing a newline character after each statement.

Syn:document.writeln(exp1,exp2,exp3 ...)

Example:

```
<!DOCTYPE html>
<html>
<body>
<pre>
        <script type='text/javascript'>
                document.writeln("Welcome to JS");
                document.writeln("Welcome to JS");
        </script>
</pre>
</head>
</html>
```

Note: You have to place writeln() in pre tag to see difference between write() and writeln().

Writeln() actually produces the output in new line (\n) but browser will not detect the \n as linebreak, hence to show it correctly and keep format as it is we will use pre tag.

Example:

```
<!DOCTYPE html>
<html>
<head>
	<script type='text/javascript'>
		document.write("<h1 style='color:blue; font-size:30px; font-family:tahoma'> Welcome To JS</h1>");
		document.write("<font color='green' size='16px' face='Arial'> Welcome To JS</font>");
	</script>
</head>
<body>
</body>
</html>
```

Note:

- the above type of code is known as DHTML
- In JavaScript a string should be in single or double quotes.
- Double quotes inside using single quotes are valid, single quotes inside using double quotes valid.

Example

```
<!DOCTYPE html>
<html>
<body>
	<script>
		document.write("JavaScript is client side script");
		document.write("<br>");
		document.write("JavScript is 'ECMA' Implementation<br>");
		document.write('JavScript released by NetScape<br>');
		document.write('NetScape release "Mocha"<br>');
	//document.write('NetScape release 'Mocha'<br>'); Error
	//document.write("NetScape release "Mocha"<br>"); Error
	</script>
```

</body>
</html>

JavaScript string with escape sequences: An escape character is consisting of backslash "/" symbol with an alphabet. The following are frequently using escape characters.

1. \n : inserts a new line
2. \t : inserts a tab space
3. \r : carriage return
4. \b : backspace
5. \f : form feed
6. \' : single quote
7. \”: double quote
8. \\ : Backslash

Difference between window.document.write&document.write:
There is no difference between these two statements,window is highest level object. It contains child objects & their methods

child object/sub object
|
window.document.write();
| | |
browser page method

document.write();
| |
page method

browser is default object, master object, super object.

write() is a method related to document object

ex:

```
<head>
	<script type='text/javascript'>
		window.document.write("livescript is javascript");
		document.write("<br>");
		document.write('livescript is javascript');
	</script>
</head>
```

JavaScript semicolon(;):

In javascript every statements ends with semicolon(;). It is an optional notation.

ex:

```
<head>
	<script type='text/javascript'>
		document.write=("livescript is javascript");
	</script>
</head>
```

ex:

```
<head>
	<script type='text/javascript'>
		document.write("javascript");
		document.write('livescript');
		document.write('livescript is javascript');
	</script>
</head>
```

Note:

1) In the above script semicolon(;) is mandatory.
2) It is a good programming practice to use the semicolon.

JavaScript place in HTML file:
There is a flexibility given to include javascript code any where in a html document but the follow ways are most prefered in the live environment.

- script in <head>-----</head> section
- script in <body>-----</body> section
- script in <body>-----</body>&<head>-----</head> section
- script in & external file & then include in <head>-----</head> section.

ex:

```
<head>
	<script type='text/javascript'>
		document.write("welcome to head section");
	</script>
</head>
<body>
	<script language="javascript">
		document.write("welcome to the body section");
	</script>
</body>
```

External javascript:
Javascript can also be placed in a external files,these files contains javascript code. This code we can apply on different webpages. External javascript files extensions is .js

Note:
1) External script cann't contain the <script></script> tags.
2) To use an external script,point to the .js file in the "src" attribute of the <script> tag.

how to run external js:
Creating javascript file:

```
document.write("<h1 style='color:blue'> welcome to external javascript....!</h1>");
```

```
document.write("<br/>");
document.write("thank u----");
```

save with **example.js** extension @any location.

creating html files:

```
<html>
	<head>
		<script type="text/javascript" src="example.js"></script>
	</head>
	<body>
	</body>
</html>
```

save with .html or .htm extensions.

Javascript code: It is a sequence of javascript statements, each statement is executed by the browser in the sequence they are returned.

```
<head>
	<script type="text/javascript">
		document.write("<p>This is paragraph</p>") //js code
	</script>
</head>
```

Javascript blocks: JavaScript sentences can be group together in blocks. Blocks starts with a left curly bracket { and end with a right curly bracket }. The purpose of your block is to make the sequence of statement excute together.

```
<head>
	<script type="text/javascript">
		{
			document.write(This is a block);
		}
	</script>
</head>
```

JavaScript dialog boxes: JavaScript has 3 kinds of dialog boxes.

1. Alert box
2. Confirm box
3. Prompt box

Alert box: An alert box is often used if you want to make sure information comes through the user. When an alert box pops up, the user will have to click "ok" to proceed.

Syn: window.alert("message"/expr);

ex:

```
<body>
	<script type='text/javascript'>
		alert("invalid entry");
	</script>
</body>
```

Note: html tags we can't use in alert() function.

How to display multiple line on the alert:

We cann't the use
 tag here because alert is a method of the windows object, that can't be interpret html tag. Instead, we use the new line escape character.

```
<head>
	<script type="text/javascript">
-		alert("javascript \n is\n a\n client-side \n programming \n language");
	</script>
</head>
```

ex: **Alert with functions**

```
<head>
	<script type='text/javascript'>
		functionmyAlert(){
```

```
                alert("javascript \n is \n a \n client-side \n programming \n language");
                alert("1 \n \t 2 \n \t \t3");
            }
        </script>
</head>
<body>
        <p> click the button to display alert messages ....</p>
        <button onclick="myAlert()"> click me</button>
</body>
```

confirm box:

It is often used, if you want the user to verify and accept something. When a confirm box pops up, the user will have to click either "ok" or "cancel" to proceed. If the user clicks "**ok**" the box returns "**true"**. If the user clicks **"cancel"** the box returns **"false".**

Syntax: window.confirm("message");

ex:

```
<head>
        <script type='text/javascript'>
            confirm("click ok or cancel");
        </script>
</head>
```

ex:

```
<head>
        <script type='text/javascript'>
            var x=confirm("click ok or cancel");
            alert("user selected option is:"+x);
        </script>
</head>
```

ex:

```
<head>
        <script type='text/javascript'>
```

```
            var x=confirm("click ok or cancle");
            alert("user selected option is:"+x);
            if(x==true) {
                  alert("user clicked on OK button");
            }
            else{
                  alert("user clicked on cancel button");
            }
      </script>
</head>
```

ex: **confirm with function**

```
<head>
      <script type='text/javascript'>
            functionmyConfirm(){
                  var x=confirm("click ok or cancel");
                  alert("user selected option is:"+x);
                  if(x==true) {
                        alert("user clicked on ok button");
                  }
            }
      </script>
</head>
<body>
      <p> click the button to display the user selected result..</p>
      <button onclick="myConfirm()">confirm</button>
</body>
```

Prompt Box: It is used to, if you want the user to input a value while entering a page. When a prompt box pops up the user will have to click either "**ok**" or "**cancel**" to proceed after entering an input value. If the user clicks "ok" the box returns the **value/empty**. If the user clicks "cancel" the box returns **"null".**

Syntax: window.prompt("sometext", defaultvalue);

ex:

```
<head>
	<script type='text/javascript'>
		prompt("Enter Any Number:");
	</script>
</head>
```

ex:

```
<head>
	<script type='text/javascript'>
		varMyVal=prompt("Enter Any Number:");
		alert("User Entered value is:"+MyVal);
	</script>
</head>
```

Note: these 3methods are provided by window object.

External JavaScript with popup boxes:

step1: *Create a required js file*

```
functionMyAlert(){
	alert("welcome to externaljs");
}
functionMyConfirm(){
	confirm("click ok or cancel");
}
functionMyPrompt(){
	prompt("Enter Any Value");
}
```

Save with .js extension @ any location....!!

step2: preparing required html file.

```
<html>
	<head>
		<script type="text/javascript" src="myscript.js">
```

```
        </script>
    </head>
    <body>
        <p>Click the button to display alert message..</p>
        <button onclick="MyAlert()">Alert</button>
        <p>click the button to display confirm message...</p>
        <button onclick="MyConfirm()">confirm</button>
        <p>click the button to display prompt value..</p>
        <button onclick="MyPrompt()">prompt</button>
    </body>
</html>
```

JS Naming Conventions

JS case: mixed

class name TitleCase/Capitalilze

ex: SimlaGreenApple, YellowMango

fun/method 1st word is lowercase, rest of words(2-n) are TitleCase/Capitalilze

ex: apple(), simlaGreenApple()

variables 1st word is lowercase, rest of words(2-n) are TitleCase/Capitalilze

Ex: apple, simlaGreenApple

constants uppercase

Ex: SIMLAGREENAPPLE, PI, EXP

JavaScript Reserved Words:

The following are reserved words in JavaScript. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract, boolean, break, byte, case, catch, char, class, const, continue, debugger, default, delete, do, double, else, enum, export, extends, false, final, finally, float, for, function, goto, if, implements, import, instanceof, int, interface, let, long, native, new, null, package, private, protected, public, return, short, static, super, switch, synchronized, this, throw, throws, transient, true, try, typeof, var, void, volatile, while, with.

Variable Declaration

variable is a reference name of a memory block.

variables are created or stored in RAM(stack area).

variables are used to store/to hold a value for reuse purpose.

Java script did not provide any **data types** for declaring variables and a variable in javascript can store any type of value. Hence java script is loosely typed programme.

We can use a variable directly without declaring it in javascript, it's called dynamic programming.

how to declare a variable?

we can define vars in JS Three ways, those are:

> by using "**var**" keyword

Syn: **var** varname; declaration

OR

var varname=value; initialization

> by using "**let**" keyword (since js6) ES6

Syn: **let** varname;

OR

let varname=value; init

> by using "**const**" keyword (since js6) ES6

Syn: **const** varname=value; initialization

Rules to declaring a variable

- Name should start with an alphabet (a to z or A to Z), underscore(_), or dollar($) sign.
- After first character we can use digits (0 to 9).
- variables are case sensitive. for example, a and A are different variables.
- space is not allowed, means name should be single word.
- special chars (symbols) are not allowed in name, except _ and $.
- keywords we can't use as a name.

for example:

var eid;	~~var 1a;~~
var total;	var a1;
var _b;	~~var book-id;~~
~~var a@;~~	var studentid;
~~var #b;~~	~~var case;~~
var book_id;	var a$1

where do we declare variables?

We can declare variables in open script tag, with in function or with in block.

Example of Variable declaration in JavaScript

```
<script>
var a=10;
var b=20;
var c=a+b;
document.write(c);
</script>
```

Output

30

Types of Variable in JavaScript

- Local Variable
- Outer/Global Variable

Local Variable

A variable which is declared inside block or function is called local variable. It is accessible within the function or block only.

For example:

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

or

Example

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

Global Variable

var is declared with in script tag but outside function & block those are global variables.

these global variables are accessible from anywhere in program.

declared with **window** object is known as global variable.

For example:

```
<script>
var value=10;//global variable
function a()
```

```
{
alert(value);
}
function b()
{
alert(value);
}
</script>
```

Declaring global variable through window object

The best way to declare global variable in javascript is through the window object.

Syntax

```
window.var=value;
```

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

For example:

```
function m()
{
window.value=200; //declaring global variable by window object
}
function n()
{
alert(window.value); //accessing global variable from other function
}
```

In JS we can declare the variables the following two ways.

1. Implicit declaration
2. Explicit declaration

Implicit declaration: In every scripting it is the default declaration.

ex: y=100;

Explicit declaration: All programming languages default declaration
ex: int a=5;

Scripts are able to support implicit declaration but languages are only explicit declaration.
Note: Explicit declaration is always recommended as a good programming practice.

Javascript datatypes:

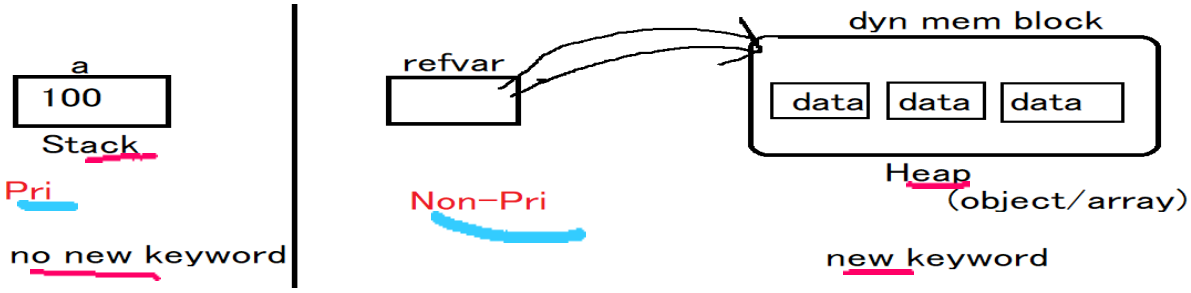
In javascript data types are classified into the following two cat.

1. primitive datatypes
2. non-primitive datatypes

primitive data types: Primitive datatypes allow to store data directly.
These datatypes allow us to store only 1 value @time.
These are popularly known as non-reference
Javascript has a five primitive data types.

1. string **Ex:** "siva" 'apples123' "Dno: 1-2-3/1a"
2. number **Ex:** 10 -25 100.56 -3.7
3. boolean **Ex:** true false
4. undefined **Ex:** value not assigned or datatype is not identified
5. object **Ex:** null { }

Non-primitive datatypes: Non-primitive datatypes allow to store reference(address) of data.
These datatypes allow us to store more than 1 value @time.
These are popularly known as reference or composite data types.
Ex: class, array



vat st1 = new String();

Primitive data types:

Strings: In javascript a String should be within a single or double quote.

var name="nit";

var name='nit';

Number: Javascript has only one type of numbers, they can be return with or without decimals

var x1=34.00; with decimals

var x2=34 without decimals

Boolean: It is used to represent a Boolean value,These are as follows.

var x = true //equivalent to true, yes or on

var y = false //equivalent to false, no or off

undefined: It is a value of variable with no value.

var x; //now x is undefined

Null: variables can be emptied by setting the value to null.

ex: var x=null; //now x is null

typeof

typeof is predefine function, and it's used to identify datatype of a variable or value.

Syn: **typeof** var-name

typeof value

Dynamic data types: Javascript has dynamic types. This means that the same variable can be used as different types.
ex:
var x; //now x is undefined
var x=5; // now x is a number
var x="ram"; // now x is a String

<!-- example on variable declaration -->

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <title>JS Ex13</title>
 </head>
 <body>
   <h1> Demo on difference between var and let </h1>
       <script>
     var a=10;  //define
     document.write(a +"<br>");
          var a=20.56;   //re-defination
          document.write(a +"<br>");
     var a="apple";  //re-defination
          document.write(a +"<br>");

          let n=101;  //define
     document.write(n +"<br>");
          //let n=202;  re-defination ==> Error: Identifier 'n' has already been
declared
          n=202;  //changing value
     document.write(n +"<br>");
       </script>
 </body>
</html>
```

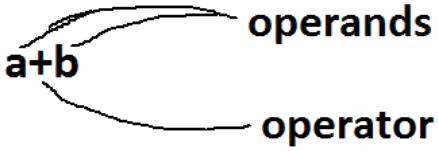
Non-primitive data types: When a variable is declared with the keyword **new**, the variable is an object.
new is used for dynamic memory allocations (for creating objects and arrays).
these datatypes are also called as reference datatype.
Ex:

```
var st = new String();
var x = new Number();
let y = new Boolean();
let a = [ ];
```

here **LHS** are reference variables, and **RHS** are objects.
reference variables are storing address of dynamic memory (object)

JavaScript operators

operator is a symbol (special char) and it is used to perform certain operation(task).
every operator is a symbol, but every symbol is not operator.
every operator requires some values, those are called as operands.
Ex:



Expression
Its combination of one operator and some operands

Javascript supports the following list of operators:

- arithmetic operators
- relational operators
- logical operators
- assignment operators
- incre/decre operators
- ternary operator
- concatenation operator etc...

Arithmetic operators: using these operators we can perform the basic math calculations.

operators are:

operator	Description	example
+	addition	j+12
-	subtraction	j-22
*	multiplication	j*7
/	division	j/3
%	modulus	j%6
**	power	x**y x^y

relational operators: these operators are used to provide comparison between two operands. these are boolean operators (true/false).

operators are:

operator	Description	example
==	is equal to	j==42
!=	is not equal to	j!=17
>	is greater than	j>0
<	is less than	j<100
>=	is greater than or equal to	j>=23
<=	is less than or equal to	j<=13
===		a===b
!==		a!==b

Logical operators: these operators are used to perform multiple comparisons @time. these are boolean operators (true/false).

operators are:

operator	Description	example
&&	And	j==1 **&&** k==2
||	OR	j<100 **||** j>0
!	Not	!(j==k)

And			**Or**			**Not**	
Cond1	**Cond2**	**Result**	**Cond1**	**Cond2**	**Result**	**Cond**	**Result**
T	**T**	**T**	**T**	**T**	**T**	**T**	**F**

T	**F**	**F**	**T**	**F**	**T**	**F**	**T**
F	**T**	**F**	**F**	**T**	**T**		
F	**F**	**F**	**F**	**F**	**F**		

assignment operators: these operators are used to store/assign value.
operator is:

operator	Description	example
=	store	a=10

shorthand:

+=	addition & assign	a+=10
-=	subtract & assign	a-=5
*=	product & assign	a*=20
/=	division & assign	a/=7
%=	modulus & assign	a%=6

unary operators: these operators are used to increment or to decrement a value. operators are ++ and --

++ (increment) ==> it adding 1 to an existing value **Ex:** a++ or ++a

-- (decrement) ==> it subtracting 1 from an existing value **Ex:** a-- or --a

ternary operator: this operator is used to decision making operation. operator is **?:,** this operator also called as conditional operator.

(condition) ? statement1 : statement2

concatenation operator: this operator is used to concatenation multiple strings then formed into a single string. one operand should be string to perform concatenation. resultant value comes in string format.
operator is +.

Ex: **"rama"+"rao" ==> "ramarao"**
"mangos"+123 ==> "mangos123"
true+"siva" "truesiva"

parseInt()

predefine function => window
text based int converts into number format
"100" → 100
"10.78" → 10
"rama" → NaN (Not a Number)
Syn:

window.parseInt("value")

parseFloat()

predefine function => window
text based float converts into number-based float
“100" → 100.0
"10.78" → 10.78
"rama" → NaN (Not a Numeric)
Syn:

window.parseFloat("value")

Note: both are global functions

Control Statement

control statements are used to control(change) execution flow of program based on user input data.

types:

> conditional statements (dm)

> loops (iterations)

> un-conditional (branching)

Conditional Statements:

If Statement

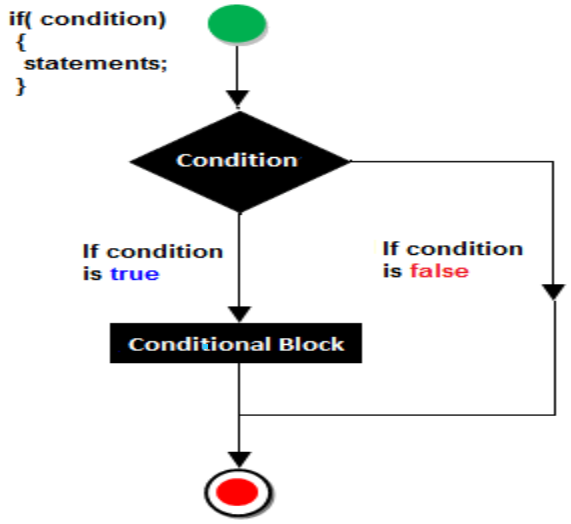
The if statement is used to perform decision making operations. means if condition is true, it executes some statements. if condition is false, it executes some other statements.

There are three forms of if statement.

- simple if
- If else
- if else if (ladder if)

If statement

if is most basic statement of Decision-making statements. It tells to program to execute a certain part of code only if particular condition or test case is true.

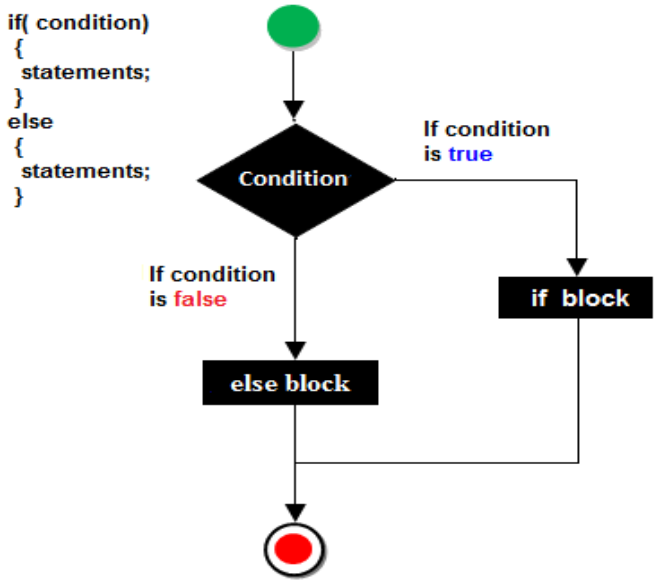


Example

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

if-else statement

In general, it can be used to execute one block of statement among two blocks.



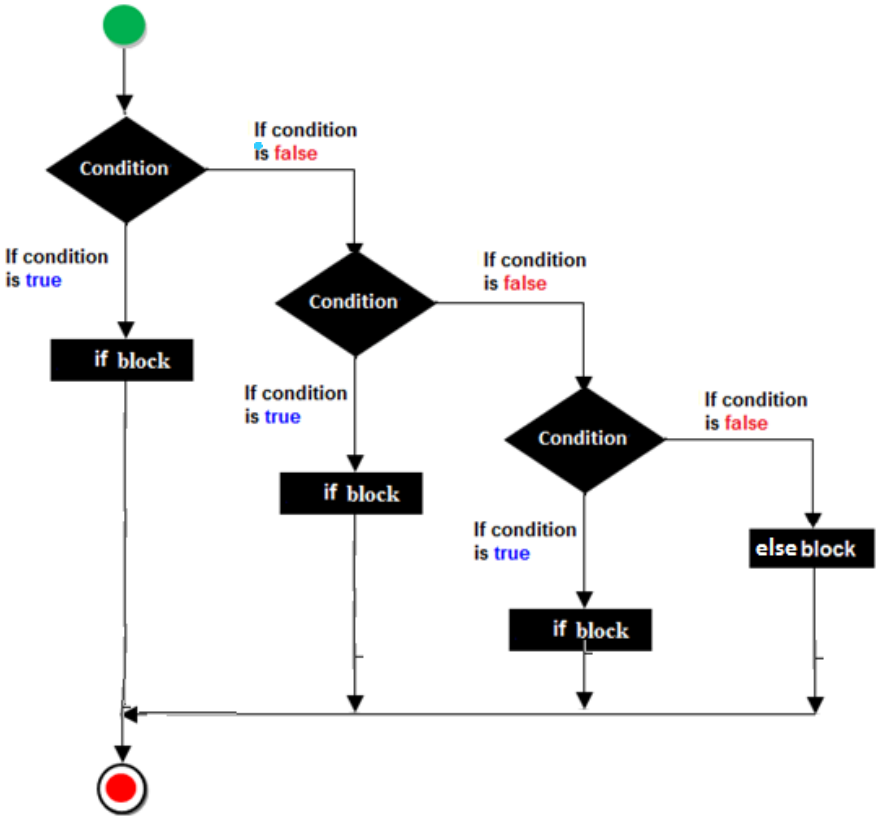
Example of if else statement

```
<script>
var a=40;
if(a%2==0)
{
document.write("a is even number");
}
else{
document.write("a is odd number");
}
</script>
```

Result
a is even number

JavaScript If...else if statement

It evaluates the content only if expression is true from several expressions.



Syntax

```
if(expression1)
{
//content to be evaluated if expression1 is true
}
else
if(expression2)
{
//content to be evaluated if expression2 is true
}
else
{
//content to be evaluated if no expression is true
```

```
}
```
Example of if..else if statement

```
<script>
var a=40;
if(a==20)
{
document.write("a is equal to 20");
}
else if(a==5)
{
document.write("a is equal to 5");
}
else if(a==30)
{
document.write("a is equal to 30");
}
else
{
document.write("a is not equal to 20, 5 or 30");
}
</script>
```

switch statement

> switch is selection statement, but it's not decision making.
> its better performance.

Syn:

```
switch(var/expr)
{
  case value: statements...
              break;
  case value: statements...
              break;
  case ...
  default:  statements...
```

}

Looping Statement

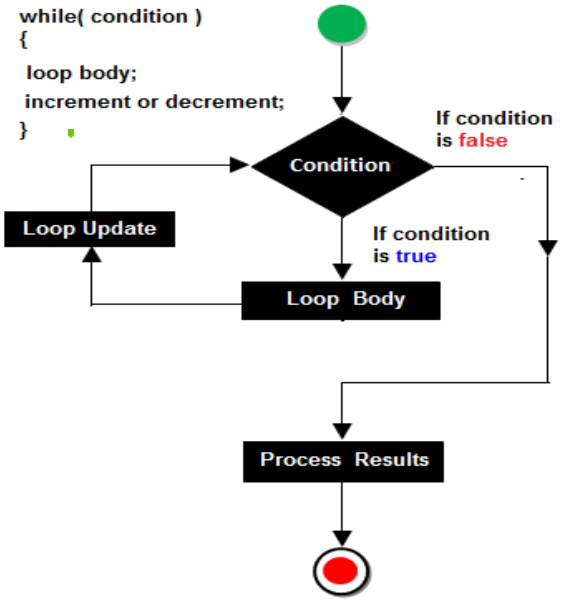
Set of instructions given to the interpreter to execute until condition becomes false is called loops. The basic purpose of loop is min code repetition.

The way of the repetition will be forming a circle that's why repetition statements are called loops. Some loops are available In JavaScript which are given below.

- while loop (top testing/entry level)
- for loop
- do-while (bottom testing/exit level)

while loop

When we are working with “while loop” always pre-checking process will be occurred. Pre-checking process means before evolution of statement block condition part will be executed. “While loop” will repeat in clock wise direction or anti-clock wise direction.

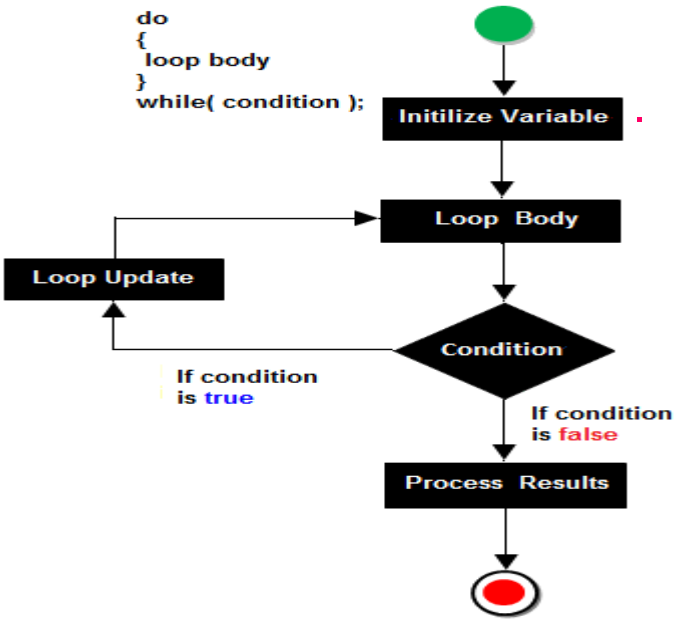


Example of while loop

```
<script>
var i=10;
while (i<=13)
{
document.write(i + "<br/>");
i++;
}
</script>
```

do-while loop

In implementation when we need to repeat the statement block at least 1 then go for do-while. In do-while loop post checking of the statement block condition part will be executed.

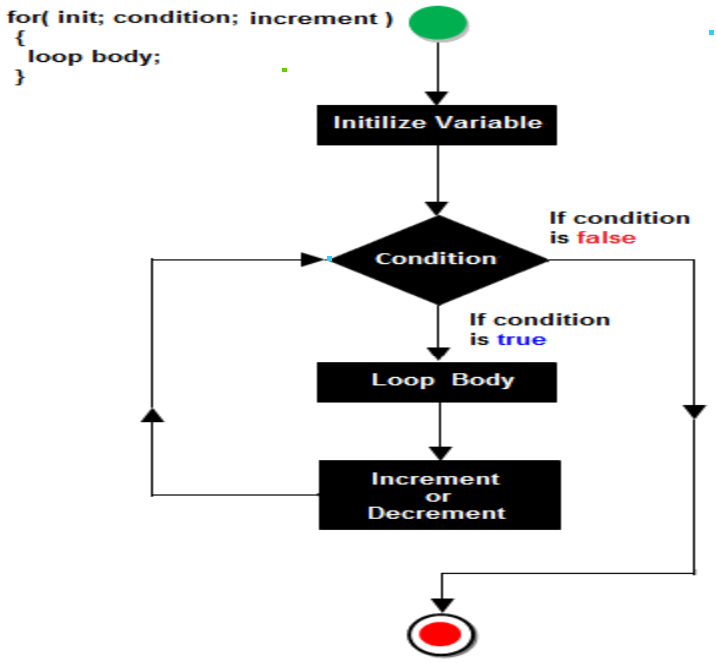


Example of do-while loop

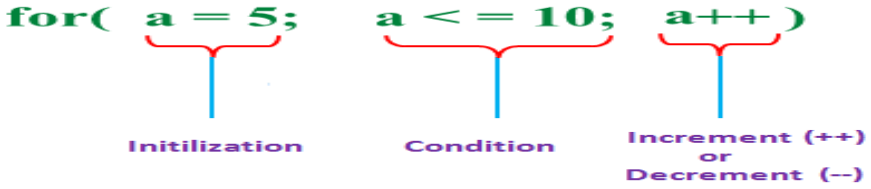
```
<script>
var i=11;
do{
  document.write(i + "<br/>");
  i++;
}while (i<=15);
</script>
```

for Loop

For loop is a simplest loop first we initialized the value then check condition and then increment and decrements occurred.



Steps of for loop



Example of for loop

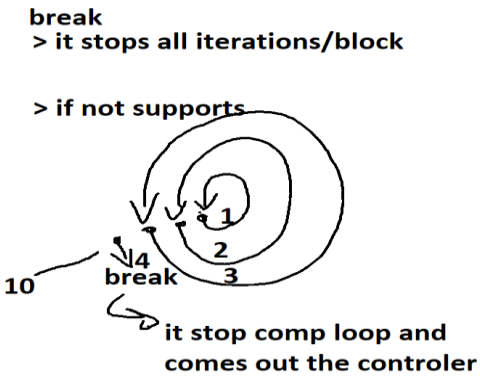
```
<script>
for (i=1; i<=5; i++)
{
document.write(i + "<br/>")
}
</script>
```

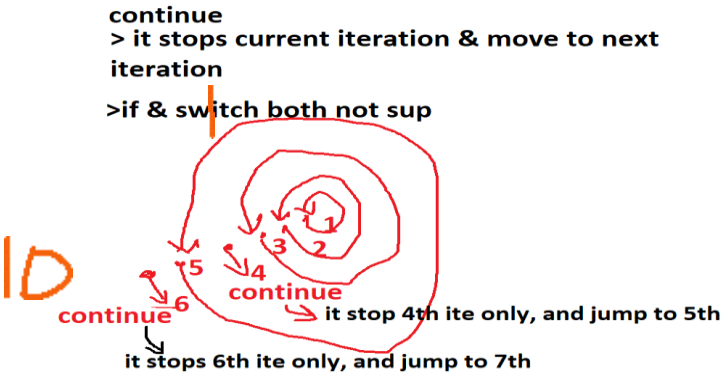
Unconditional statements

These are used to jump/skip statements execution

Types:

- break
- continue
- return





<noscript> tag: It is used to provide an alternate contains for users when script is disabled or not supporting, It is a paired tag. It is always declared within the body section.

syntax: <noscript>------</noscript>

ex:

```
<head>
	<script type='text/javascript'>
		alert("welcome to js");
	</script>
</head>
<body>
	<noscript>
		<p style='color:red'>oops your browser not supporting javascript
		update/change the script settings and try..</p>
	</noscript>
</body>
```

Arrays

> array is collection of elements (values).

> array allows sim type of values (homogeneous) as well as diff types of values, means one array can store group numbers, strings, booleans etc...

> storing group of values with same refname is called array.

adv:
- > arrays are simplyfying coding when work with group of values.
- > easy transporting data
- > also used for data maintenance in application

> arrays we can create local scope or outer scope.

> arrays are belongs to reference/non-primitive datatype.

> arrays are created dynamically, and arrays are created in heap area.

>primitive dt stores data but non-primitive stores address of data.

Syn:
array creation:
Approch1 (using Literals [ ]):
- let/var/const array = [ ]; <= 1st
- let/var/const array = [val1, val2, val3, ...]; <= 2nd

Approch2 (using new kw):
- var array = new Array(); <= 3rd
- var array = new Array(val1,val2,...); <= 4th

datatype array[size]; <== c/c++
datatype array[] = new datatype[size]; <== java

accessing array:
- array[index]
- index is a slno of memory block, its start 0.

set value:
- array[index]=value;

size of array:
- array.length ==> predefine property, it returns size of array
- array.length=N; ==> it reset size of array

push()
add a new element @end of array
- array.push(newvalue)

pop()
  it returns ele of array (R -> L), it removes popped ele
        array.pop()

shift()
  it returns ele of array (L -> R), it removes shifted ele
        array.shift();

unshift()
  add a new element @begining of array
        array.unshift(value);

indexOf()
  finding given ele ava in an array or not
  if found => index, 1st occurence
  if not found => -1
  by def search starts from 0th index or search starts from given index.

lastIndexOf()
  finding given ele ava in an array or not
  if found => index, last occurence
  if not found => -1

include()
   it searching the given ele found or not
   if found => true
   not found => false

sort()
   it sorting an array in asce order

reverse()
   it re-arrange ele of array in reverse order

splice()
  it used to remove/delete ele from an array based given index
        array.splice(st-index, no.of elements)
  it used to insert ele in array based given index
        array.splice(index, 0, newvalue)
  it used to overwrite eles of array

join()

MDA
---
storing group of ele in tabler (row & col) format is called MDA (2DA).
mda is a coll of sda's

array creation:

```
        var array=[ [val1, val2, ...],
                    [val1, val2, ...],
                    ...
                  ];
accessing array:
        array[rowind][colind]

set value:
        array[rowind][colind]=value;

size of array:
        array.length  => it returns no.of rows
        array[rowind].length  => it returns no.of cols


for in loop
----------
> it used to get elements from an array based on index
> this loop extracting elements in forward direction only (back not sup)
> we can't start the loop from middle of array (random access not poss)
Syn:
    for(var in array)
    {
      code
    }


for of loop
----------
> it used to get elements from an array based on value
> this loop extracting elements in forward direction only (back not sup)
> we can't start the loop from middle of array (random access not poss)
Syn:
    for(var of array)
    {
      code
    }


forEach loop
------------
> this loop used to get elements from an array based on value
> this loop extracting elements in forward direction only (back not sup)
> we can't start the loop from middle of array (random access not poss)
Syn:
    array.forEach(function(variable){
                      code
                  });
```

Working with Functions

Quite often we need to perform a similar action in many places of the script.
For example, we need to show a message when a visitor logs in, logs out and maybe somewhere else.

Functions are the main “building blocks” of the program. We’ve already seen examples of built-in functions, like **`alert(message)`**, **`prompt(message, default)`** and **`confirm(question)`**. But we can create functions of our own as well.

>function is a named block; it consists group of statements
>function is used to perform specific task/operation

adv:

> Reusable means they allow the code to be called many times without repetition.
> reduce length of code
> Easy maintenance code (readability, easy debugging, modification of code, ...)

we can develop functions either internal or external

internal => within the **script** tag
external => in **sep file**, but no script tag (any no.of funs)

how to define a function?

by using "**function**" keyword we can define/develop functions.

Syn:

```
function fun-name(parameters)
{
  local dec
  statements
  return value;
}
```

Where to call a function?

we can call a function, from diff places, those are

> from script tag
> from another function
> event attribute

How to calling:

```
fun-name()
fun-name(arg1, arg2, …)
```

Function Declaration

To create a function, we can use a *function declaration*.

The `function` keyword goes first, then goes the *name of the function*, then a list of *parameters* between the parentheses (comma-separated, empty in the example above) and finally the code of the function, also named “the function body”, between curly braces.

```
function name(parameters) {
  ...body...
}
calling Syn:
     fun-name()
     fun-name(arg1, arg2,...)

function showMessage() {
  alert( 'Hello everyone!' );
}

showMessage();
showMessage();
```

The call `showMessage()` executes the code of the function. Here we will see the message two times.

This example clearly demonstrates one of the main purposes of functions: to avoid code duplication.

Local variables

A variable declared inside a function is only visible inside that function.

For example:

```
function showMessage() {
  let message = "Hello, I'm JavaScript!"; // local variable

  alert( message );
}

showMessage(); // Hello, I'm JavaScript!

alert( message ); // <-- Error! The variable is local to the function
```

Outer variables

A function can access an outer variable as well, for example:

```
let userName = 'Siva';

function showMessage() {
  let message = 'Hello, ' + userName;
  alert(message);
}

showMessage(); // Hello, Siva
```

The function has full access to the outer variable. It can modify it as well.

For Example:

```
let userName = 'Siva';

function showMessage()
{
  userName = "Kumar";   //changed the outer variable

  let message = 'Hello, ' + userName;
  document.write(message);
}

document.write( userName ); // Siva before the function call

showMessage();

document.write( userName ); // Kumar, the value was modified by
the function
```

The outer variable is only used if there's no local one.
If a same-named variable is declared inside the function then it *shadows* the outer one. For Example, in the code below the function uses the local `userName`. The outer one is ignored:

```
let userName = 'Siva';

function showMessage() {
  let userName = "Kumar";    // declare a local variable

  let message = 'Hello, ' + userName;    // Kumar
  document.write(message);
}

// the function will create and use its own userName
showMessage();

document.write( userName ); // Siva, unchanged, the function did
not access the outer variable
```

Global variables

Variables declared outside of any function, such as the outer `userName` in the code above, are called *global*.
Global variables are visible from any function (unless shadowed by locals).
It's a good practice to minimize the use of global variables. Modern code has few or no global. Most variables reside in their functions. Sometimes though, they can be useful to store project-level data.

Parameters

We can pass arbitrary data to functions using parameters (also called *function arguments*).
Note: while declaring parameters don't use **let, const** and **var** keywords.
Syn:

function fun-name **(**param1, param2, param3… **)** //parameters (formal)

{
  Code
}

Calling:

fun-name(var1/val,var2,var3 …); //arguments (actual)

for Example:

```
function showMessage(from, text) {   // arguments: from, text
  alert(from + ': ' + text);
}

showMessage('Siva', 'Hello!'); // call1
showMessage('Siva', "What's up?"); // call2
```

When the function is called, the given values are copied to local variables `from` and `text`. Then the function uses them.

Here's one more example: we have a variable `from` and pass it to the function. Please note: the function changes `from`, but the change is not seen outside, because a function always gets a copy of the value:

```
function showMessage(from, text) {

  from = '*' + from + '*';

  document.write( from + ': ' + text );
}

let from = "Siva";

showMessage(from, "Hello");

// the value of "from" is the same, the function modified a
local copy
document.write( from ); // Ann
```

Default values

function fun-name(param=value, param=value, param=value) { }
function fun-name(param, param=value, param=value) { }
function fun-name(param, param, param=value) { }
function fun-name(param=value, param, param) { } X
function fun-name(param=value, param=value, param) { } X
function fun-name(param, param=value, param) { } X

If a parameter is not provided, then its value becomes `undefined`.

For instance, the aforementioned function `showMessage(from, text)` can be called with a single argument:

```
showMessage("Siva");
```

That's not an error. Such a call would output `"Siva: undefined"`. There's no `text`, so it's assumed that `text === undefined`.

If we want to use a "default" `text` in this case, then we can specify it after `=`:

```
function showMessage(from, text ="no data given") {
  document.write( from + ": " + text );
}

showMessage("Siva"); // Siva: no text given
```

Now if the `text` parameter is not passed, it will get the value `"no data given"`

Here `"no data given"` is a string, but it can be a more complex expression, which is only evaluated and assigned if the parameter is missing. So, this is also possible:

```
function showMessage(from, text = anotherFunction())
{
  // anotherFunction() only executed if no text given
  // its result becomes the value of text
}
```

Evaluation of default parameters

In JavaScript, a default parameter is evaluated every time the function is called without the respective parameter.

In the example above, `anotherFunction()` is called every time `showMessage()` is called without the `text` parameter.

Default parameters old-style

Old editions of JavaScript did not support default parameters. So there are alternative ways to support them, that you can find mostly in the old scripts.

For instance, an explicit check for being `undefined`:

```
function showMessage(from, text) {
  if (text === undefined) {
    text = 'no text given';
  }

  alert( from + ": " + text );
}
```

Or the `||` operator:

```
function showMessage(from, text) {
  // if text is no value then text gets the "default" value
  text = text || 'no text given';
  alert(text);
}
```

Returning a value

A function can return a value back into the calling code as the result.

Syn: **return var/value/expr;**

The simplest example would be a function that sums two values:

```
function sum(a, b) {
  return a + b;
}

let result =
sum(1, 2);
alert( result ); // 3
```

The directive `return` can be in any place of the function. When the execution reaches it, the function stops, and the to the calling code value is returned (assigned to `result` above).

There may be many occurrences of `return` in a single function. For instance:

```
function checkAge(age) {
  if (age >= 18) {
    return true;
  } else {
    return confirm('Do you have permission from your parents?');
  }
}

let age = prompt('How old are you?', 18);

if ( checkAge(age) ) {
  alert( 'Access granted' );
} else {
  alert( 'Access denied' );
}
```

It is possible to use `return` without a value. That causes the function to exit immediately.

For example:

```
function showMovie(age) {
  if ( !checkAge(age) ) {
    return;
  }

  alert( "Showing you the movie" ); // (*)
  // ...
}
```

In the code above, if `checkAge(age)` returns `false`, then `showMovie` won't proceed to the `alert`.

A function with an empty `return` or without it returns `undefined`

If a function does not return a value, it is the same as if it returns `undefined`:

```
function doNothing() { /* empty */ }

alert( doNothing() === undefined ); // true
```

An empty `return` is also the same as `return undefined`:

```
function doNothing() {
  return;
}

alert( doNothing() === undefined ); // true
```

Never add a newline between `return` and the value

For a long expression in `return`, it might be tempting to put it on a separate line, like this:

```
return
 (some + long + expression + or + whatever * f(a) + f(b))
```

That doesn't work, because JavaScript assumes a semicolon after `return`. That'll work the same as:

```
        return;
 (some + long + expression + or + whatever * f(a) + f(b))
```

So, it effectively becomes an empty return.

If we want the returned expression to wrap across multiple lines, we should start it at the same line as `return`. Or at least put the opening parentheses there as follows:

```
return (
  some + long + expression
  + or +
  whatever * f(a) + f(b)
  )
```

And it will work just as we expect it to.

Naming a function

Functions are actions. So their name is usually a verb. It should be brief, as accurate as possible and describe what the function does, so that someone reading the code gets an indication of what the function does.

It is a widespread practice to start a function with a verbal prefix which vaguely describes the action. There must be an agreement within the team on the meaning of the prefixes.

For instance, functions that start with `"show"` usually show something.

Function starting with…

- `"get…"` – return a value,
- `"calc…"` – calculate something,
- `"create…"` – create something,
- `"check…"` – check something and return a boolean, etc.

Examples of such names:

```
showMessage(..)     // shows a message
getAge(..)          // returns the age (gets it somehow)
calcSum(..)         // calculates a sum and returns the result
createForm(..)      // creates a form (and usually returns it)
checkPermission(..) // checks a permission, returns true/false
```

With prefixes in place, a glance at a function name gives an understanding what kind of work it does and what kind of value it returns.

One function – one action

A function should do exactly what is suggested by its name, no more.
Two independent actions usually deserve two functions, even if they are usually called together (in that case we can make a 3rd function that calls those two).
A few examples of breaking this rule:

- `getAge` – would be bad if it shows an `alert` with the age (should only get).
- `createForm` – would be bad if it modifies the document, adding a form to it (should only create it and return).
- `checkPermission` – would be bad if it displays the `access granted/denied` message (should only perform the check and return the result).

These examples assume common meanings of prefixes. You and your team are free to agree on other meanings, but usually they're not much different. In any case, you should have a firm understanding of what a prefix means, what a prefixed function can and cannot do. All same-prefixed functions should obey the rules. And the team should share the knowledge.

Ultrashort function names

Functions that are used *very often* sometimes have ultrashort names.

For example, the jQuery framework defines a function with `$`. The Lodash library has its core function named `_`.

These are exceptions. Generally functions names should be concise and descriptive.

Functions Comments

Functions should be short and do exactly one thing. If that thing is big, maybe it's worth it to split the function into a few smaller functions. Sometimes following this rule may not be that easy, but it's definitely a good thing.
A separate function is not only easier to test and debug – its very existence is a great comment!

For instance, compare the two functions `showPrimes(n)` below. Each one outputs prime numbers up to `n`.

The first variant uses a label:

```
function showPrimes(n) {
  nextPrime: for (let i = 2; i < n; i++) {

    for (let j = 2; j < i; j++) {
      if (i % j == 0) continue nextPrime;
    }

    alert( i ); // a prime
  }
}
```

The second variant uses an additional function `isPrime(n)` to test for primarily:

```
function showPrimes(n) {

  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;

    alert(i);  // a prime
  }
}

function isPrime(n) {
```

```
  for (let i = 2; i < n; i++) {
    if ( n % i == 0) return false;
  }
  return true;
}
```

The second variant is easier to understand, isn't it? Instead of the code piece we see a name of the action (`isPrime`). Sometimes people refer to such code as *self-describing*.

So, functions can be created even if we don't intend to reuse them. They structure the code and make it readable.

Summary

A function declaration looks like this:

```
function name(parameters, delimited, by, comma) {
  /* code */
}
```

- Values passed to a function as parameters are copied to its local variables.
- A function may access outer variables. But it works only from inside out. The code outside of the function doesn't see its local variables.
- A function can return a value. If it doesn't, then its result is `undefined`.

To make the code clean and easy to understand, it's recommended to use mainly local variables and parameters in the function, not outer variables.

It is always easier to understand a function which gets parameters, works with them and returns a result than a function which gets no parameters, but modifies outer variables as a side-effect.

Function naming:

- A name should clearly describe what the function does. When we see a function call in the code, a good name instantly gives us an understanding what it does and returns.
- A function is an action, so function names are usually verbal.
- There exist many well-known function prefixes like `create…`, `show…`, `get…`, `check…` and so on. Use them to hint what a function does.

Functions are the main building blocks of scripts. Now we've covered the basics, so we actually can start creating and using them. But that's only the beginning of the path. We are going to return to them many times, going more deeply into their advanced features.

Object Oriented Programming in JavaScript

Langs: POP OOPS class based

prototype /object based

Object Oriented Programming (OOP) is a programming paradigm (programming structure), which is based on the concept of "object".

- Object represents a physical component.
- Object is a real-time entity.
  - We can see
  - We can touch
  - We can use

Ex:  etc…

- Object is an instance of a class, nothing but memory block (one copy of class)

- Object is a collection of members:
  1. **properties (**variables or fields)
  2. **methods (**functions**)**

- **Properties:** details about the object. Properties are the variables which are stored inside the Object. Properties are used to store data about specific person, product or thing.
  Ex: array.length=5
- **Methods:** to perform manipulations on the properties. Methods are the functions stored inside the object. **Methods read values from properties**, **write values into properties,** to **perform logical operations.**
  Ex: array.sort() array.push() array.pop()

Note: objects are used to data maintenance

Array	**Object**
Seq	random
Index base	properties
[ ]	**{ }**

Example:

Car is an object:
-properties
- Car model: I20
- Car colour : white
- Car no: 5579

-methods
- Start()
- Change gear()
- Stop()

Person is an object:
-**properties**
> name: siva
> age: 50
> gen: male

-**methods**
> sleep()
> eat()
> talk()

- In the above example the "car" object has three properties called "car model, car colour, car no", which have respective values.

We have two types of OOP languages:

1. **class-based** Object-Oriented Programming

   ex: java, .net, python, cpp etc…

2. **prototype-based(object based)** Object-Oriented Programming

   ex: javascript, typescript, perl, php, …

"**Object**" is a predefine class, every class/object should be derived from "**Object**" class prototype.

Creating objects:

we can create objects in 2 ways: -

1. with object literals { }
2. by using constructor function new

Object literals

- Object literals are represented as curly braces **{ }**, which can include properties and methods.
- The property and values are separated with : symbol
- The method-name and body are separated with : symbol

Syntax:

```
v/l/c refname = {"property":value, property:value, ...,
                 "method-name": function() { body },
                  method-name() { body } };
```

Note: methods we can write in any of 3forms, i.e like normal function or expression or arrow

how to access?

refname.property
refname.property=value
refname.method-name()

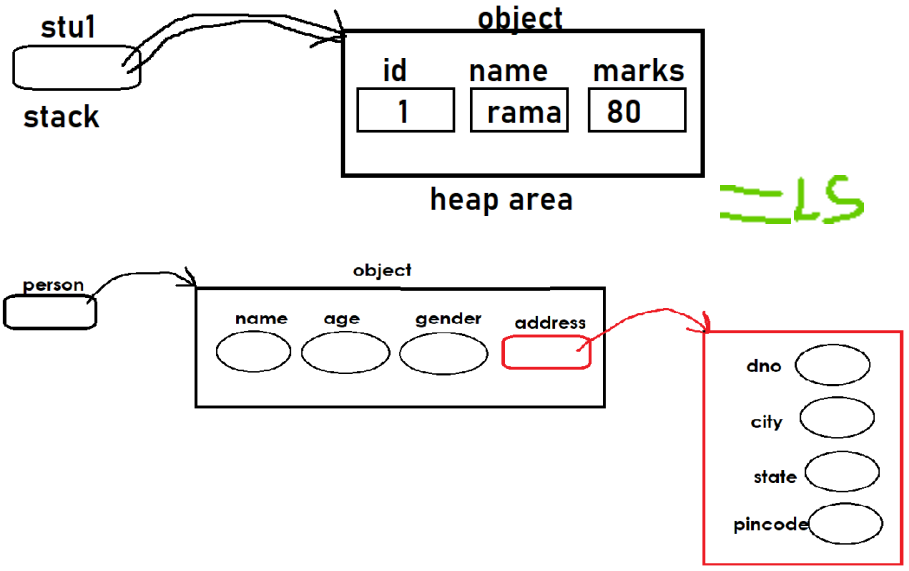
Note: every class and every object should be derived from a class called "Object" class(lib class).

Example 1 on object literals (properties)

```
<Html>
<head>
 <title> object literals </title>
</head>
<h1> object literals</h1>
<script>
   var stu1= {"id" : 1, name : "ram", marks: 80};
 var stu2= {id : 2, name : "sam", marks: 90};

 document.write("Id       :"+stu1.id+"<br>");
 document.write("Name :"+ stu1.name+"<br>");
```

```
 document.write("Marks:"+ stu1.marks+"<br>");
 document.write(stu1+"<br><br>");

 document.write("Id       :"+ stu2.id+"<br>");
 document.write("Name :"+ stu2.name+"<br>");
 document.write("Marks:"+ stu2.marks+"<br>");
 document.write(stu2);
</script>
</body>
</html>
```

RS



Example 2 oncreating object with literals

```
<Html>
<head>
<title>object literals</title>
</head>
<body>
<h1> object with properties and methods </h1>
<script>
   var stu1= { "id" : 1, name : "ram", marks: 30, "getResult": function() {
                                                    if(stu1.marks>=40)
                                                            return "Pass";
                                                    else
                                                            return "Fail";
                                                   },
                        };

 document.write("Id       :"+ stu1.id +"<br>");
 document.write("Name :"+ stu1.name +"<br>");
 document.write("Marks:"+ stu1.marks +"<br>");
 document.write("Result is          :"+ stu1.getResult() +"<br>");
 document.write(stu1+"<br><br>");
```

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

Note: The “this” keyword represents/substitutes the current working object. For example, if it is called for the first time, the “this” key word represents the first object; if it is called second time, it represents the second object.

If we want access properties inside method or constructor function, we should use “this” keyword.

Constructor function

Constructor is a function that receives an empty (created by new keyword) object, initializes properties and methods to the object.

Constructor functions technically are regular functions. There are three conventions though:

1. They are named with capital letter first.
2. They should be executed only with "new" keyword, while object creation.
3. constructor functions don’t return any value, hence no return statement.

Syn:

```
function Const-name()  constructor developing
{
  this.property1=value;  initializing code
  this.property2=value;
  ...
  this.method-name = function(){
                        code
                     };
  this.method-name = function(){
                        code
                     };
  ...
}
```

Lit:	**no identifier**	**:**	**,**	**no this**
Const:	**idenentifier**	**=**	**;**	**this**

Object Syn:

```
refname = new Const-name();  constructor calling
refname = new Const-name(args);
```

<!-- example on creating object with constructor -->

```
<Html>
<head>
<title>constructor</title>
</head>
<body>
```

```
<h1> object with constructor </h1>
<script>
 function Book(name, year)
 {
  this.name = name;
  this.year = '(' + year + ')';
 }

 var firstBook = new Book("Html", 2014);
 var secondBook = new Book("JavaScript", 2013);
 var thirdBook = new Book("CSS", 2010);

 document.write(firstBook.name, firstBook.year + "<br>");
 document.write(secondBook.name, secondBook.year + "<br>");
 document.write(thirdBook.name, thirdBook.year + "<br>");

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

<!-- example on creating object with constructor -->

```
<Html>
<head>
<title>Constructor</title>
</head>
<body>
<h1> Constructor with properties and methods </h1>
<script>
  function Student(id,name,total)
{
    //initializing
this.id=id;
    this.name=name;
    this.total=total;
    this.getResult = function() {
                        if(this.total>=40)
                                return "Pass";
                        else
                                return "Fail";
          }; //end of method
  } //end of const
        var stu = new Student(11, "Ram", 88);
        document.write("Id       :"+ stu.id +"<br>");
        document.write("Name :"+ stu.name +"<br>");
        document.write("Marks:"+ stu.total +"<br>");
        document.write("Result is          :"+ stu.getResult() +"<br>");
</script>
</body>
</html>
```

Using the new keyword is essential

It's important to remember to use the new keyword before all constructors. If you accidentally forget new, you will be modifying the global object instead of the newly created object. Consider the following example:

<!-- example on this & instanceof keywrod -->

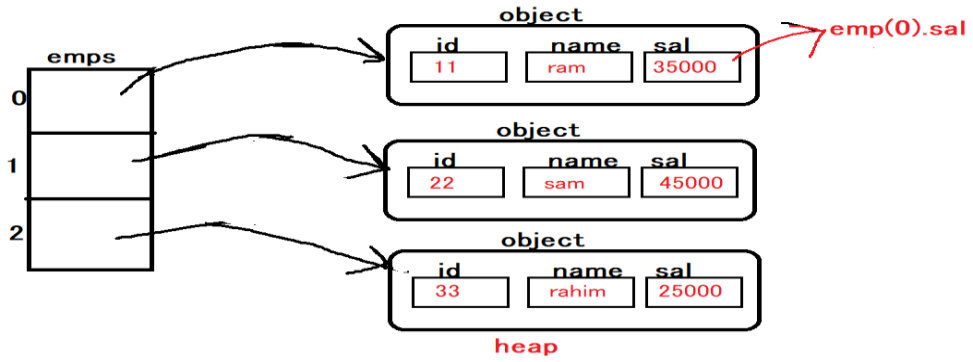
```
<html>
<head>
<title>Document</title>
</head>
<body>
<script>
        function Book(name, year)
        {
          console.log(this);
          this.name = name;
          this.year = year;
        }

        var myBook = Book("js book", 2014);
        console.log(myBook instanceof Book);
        console.log(window.name, window.year);

        var myBook = new Book("js book", 2014);
        console.log(myBook instanceof Book);
        console.log(myBook.name, myBook.year);
</script>
</body>
</html>
```

<!-- Object Array Literals -->

```
<Html>
<body>
<h1> Creating Object Array with Literals </h1>
<script>
  //object array
  var emps =[ { id:11, name:"ram", sal:35000 },
                                  { id:22, name:"sam", sal:45000 },
                                  { id:33, name:"rahim", sal:25000 }
                                 ];

  //retrieving data from array
  for(i=0; i<emps.length; i++){
    document.write(emps[i].id, emps[i].name, emps[i].sal+"<br>");
  }

  document.write(emps);
</script>
</body>
</html>
```

<!-- exmaple on object arrays -->

```
<html>
 <head>
  <script>
          function totalValue(prods) //user define function
          {
           let inventory_value = 0;
           for(let i=0; i<prods.length; i+=1) {
                    amt= prods[i].inventory * prods[i].unit_price;
                    inventory_value +=amt;
                    document.write(prods[i].name, amt, "<br>");
            }
           return inventory_value;
          }
  </script>
 </head>
<body>
 <script>
        let products = [ { name: "chair", inventory: 5, unit_price: 45},
                         { name: "table", inventory: 10, unit_price: 120},
                         { name: "sofa", inventory: 2, unit_price: 500}
                       ];

                                  //array passing as an param to fun
        document.write("Total Bill Amt  :"+  totalValue(products)   );
 </script>
</bodY>
</html>
```



<!-- exmaple on object arrays with constructor -->

```
<html>
<body>
<h1> Creating Object Array with Constructor Function </h1>
<script>
        function Movie(name,hero,dir)  //constructor
        {
                this.name=name;
                this.hero=hero;
                this.dir=dir;
        }
```

```
  //object array
  var movies=[ new Movie("Bharath","Mahesh","Siva"),
                  new Movie("ASVR","Ntr","Trivikram")
               ];

  //retrieving data from array
  for(i=0; i<movies.length; i++){
    document.write(movies[i].name, movies[i].hero, movies[i].dir+"<br>");
  }
    document.write(movies);
</script>
</body>
</html>
```

prototype

the "prototype" generally represents model of the object (structure), which contains list of properties and methods of the object.

"prototype" is a predefine attribute.

All JavaScript objects inherit properties and methods from a prototype:

- Student objects inherit from Student.prototype
- Date objects inherit from Date.prototype
- Array objects inherit from Array.prototype

The **Object.prototype** is on the top of the prototype inheritance chain:
Date objects, Array objects, and Person objects inherit from **Object.prototype**.

Adding Properties and Methods to Objects:

Sometimes you want to add new **properties or methods** to an existing object literal of a given type.
Sometimes you want to add new **properties or methods** to an object constructor.

Syn:

Constructor Syn:

Constructor.prototype.new-property = value;

Constructor.prototype.new-method = function() { code };

Literal Syn:

Object.prototype.new-property = value;

Object.prototype.new-method = function() { code };

<!-- exmaple on prototype -->

```
<html>
<body>
<script>
function Product(name,qty,unitPrice)  //constructor function (class)
  {
    this.name=name;
    this.qty=qty;
    this.unitPrice=unitPrice;
}
```

```
   //adding new property to an existing object
         Product.prototype.discount=10;
   //adding new method to an existing object
   Product.prototype.getAmount=function(){
                                                   return this.qty*this.unitPrice;
                                              };

   //creating object
   let p = new Product("Soap",2,42.50);

         document.write("Name :"+ p.name +"<br>");
         document.write("Qty    :"+ p.qty +"<br>");
         document.write("UnitPrice       :"+ p.unitPrice +"<br>");
         document.write("TotalAmt        :"+ p.getAmount() +"<br>");
         document.write("Discount        :"+ p.discount +"<br>");
         document.write("BillingAmt      :"+ (p.getAmount()-p.discount) +"<br>");
</script>
</bodY>
</html>
```

Inheritance

> the process of creating a new object based on another object prototype (exists) is called "inheritance".

OR

> the process of creating a new constructor function based on another constructor function (exists already) is called "inheritance".

> one object is deriving from an existing object.

> hence all the properties and methods of the 1st constructor (parent) is inherited into the 2nd constructor (child).

> sharing

> by calling 1st object's constructor from 2nd object's constructor function.

Syn:

```
function ConstructorP(parameters)   //parent
{
  properties
  methods
}
function ConstructorC(parameters)   //child
{
  ConstructorP();  inheritance
        OR
  ConstructorP.call(this, parameters);   inheritance
  properties
```

```
        methods
        //here we access CP properties & methods directly
    }
```

call()

call() is a predefine function, it's used to call parent constructor function from child constructor function.

React

We will build a small game during this tutorial. **You might be tempted to skip it because you're not building games — but give it a chance.** The techniques you'll learn in the tutorial are fundamental to building any React app, and mastering it will give you a deep understanding of React.

Tip

This tutorial is designed for people who prefer to **learn by doing**. If you prefer learning concepts from the ground up, check out our step-by-step guide. You might find this tutorial and the guide complementary to each other.

The tutorial is divided into several sections:

- Setup for the Tutorial will give you **a starting point** to follow the tutorial.
- Overview will teach you **the fundamentals** of React: components, props, and state.
- Completing the Game will teach you **the most common techniques** in React development.
- Adding Time Travel will give you **a deeper insight** into the unique strengths of React.

You don't have to complete all of the sections at once to get the value out of this tutorial. Try to get as far as you can — even if it's one or two sections.

What Are We Building?

In this tutorial, we'll show how to build an interactive tic-tac-toe game with React.

You can see what we'll be building here: **Final Result**. If the code doesn't make sense to you, or if you are unfamiliar with the code's syntax, don't worry! The goal of this tutorial is to help you understand React and its syntax.

We recommend that you check out the tic-tac-toe game before continuing with the tutorial. One of the features that you'll notice is that there is a numbered list to the right of the game's board. This list gives you a history of all of the moves that have occurred in the game, and it is updated as the game progresses.

You can close the tic-tac-toe game once you're familiar with it. We'll be starting from a simpler template in this tutorial. Our next step is to set you up so that you can start building the game.

Prerequisites

We'll assume that you have some familiarity with HTML and JavaScript, but you should be able to follow along even if you're coming from a different programming language. We'll also assume that you're familiar with programming concepts like functions, objects, arrays, and to a lesser extent, classes.

If you need to review JavaScript, we recommend reading this guide. Note that we're also using some features from ES6 — a recent version of JavaScript. In this tutorial, we're using arrow functions, classes, `let`, and `const` statements. You can use the Babel REPL to check what ES6 code compiles to.

Setup for the Tutorial

There are two ways to complete this tutorial: you can either write the code in your browser, or you can set up a local development environment on your computer.

Setup Option 1: Write Code in the Browser

This is the quickest way to get started!
First, open this **Starter Code** in a new tab. The new tab should display an empty tic-tac-toe game board and React code. We will be editing the React code in this tutorial.
You can now skip the second setup option, and go to the Overview section to get an overview of React.

Setup Option 2: Local Development Environment

This is completely optional and not required for this tutorial!

Optional: Instructions for following along locally using your preferred text editor

1.
2.
3.
4.
5.
6.

Help, I'm Stuck!

If you get stuck, check out the community support resources. In particular, Reactiflux Chat is a great way to get help quickly. If you don't receive an answer, or if you remain stuck, please file an issue, and we'll help you out.

Overview

Now that you're set up, let's get an overview of React!

What Is React?

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called "components".
React has a few different kinds of components, but we'll start with `React.Component` subclasses:

```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
```

```
  }
}

// Example usage: <ShoppingList name="Mark" />
```

We'll get to the funny XML-like tags soon. We use components to tell React what we want to see on the screen. When our data changes, React will efficiently update and re-render our components.
Here, ShoppingList is a **React component class**, or **React component type**. A component takes in parameters, called `props` (short for "properties"), and returns a hierarchy of views to display via the `render` method.
The `render` method returns a *description* of what you want to see on the screen. React takes the description and displays the result. In particular, `render` returns a **React element**, which is a lightweight description of what to render. Most React developers use a special syntax called "JSX" which makes these structures easier to write. The `<div />` syntax is transformed at build time to `React.createElement('div')`. The example above is equivalent to:

```
return React.createElement('div', {className: 'shopping-list'},
  React.createElement('h1', /* ... h1 children ... */),
  React.createElement('ul', /* ... ul children ... */)
);
```

See full expanded version.
If you're curious, `createElement()` is described in more detail in the API reference, but we won't be using it in this tutorial. Instead, we will keep using JSX.
JSX comes with the full power of JavaScript. You can put *any* JavaScript expressions within braces inside JSX. Each React element is a JavaScript object that you can store in a variable or pass around in your program.
The `ShoppingList` component above only renders built-in DOM components like `<div />` and `<li />`. But you can compose and render custom React components too. For example, we can now refer to the whole shopping list by writing `<ShoppingList />`. Each React component is encapsulated and can operate independently; this allows you to build complex UIs from simple components.

Inspecting the Starter Code

If you're going to work on the tutorial **in your browser,** open this code in a new tab: **Starter Code**. If you're going to work on the tutorial **locally,** instead open `src/index.js` in your project folder (you have already touched this file during the setup).

This Starter Code is the base of what we're building. We've provided the CSS styling so that you only need to focus on learning React and programming the tic-tac-toe game.

By inspecting the code, you'll notice that we have three React components:

- Square
- Board
- Game

The Square component renders a single `<button>` and the Board renders 9 squares. The Game component renders a board with placeholder values which we'll modify later. There are currently no interactive components.

Passing Data Through Props

To get our feet wet, let's try passing some data from our Board component to our Square component.

We strongly recommend typing code by hand as you're working through the tutorial and not using copy/paste. This will help you develop muscle memory and a stronger understanding.
In Board's `renderSquare` method, change the code to pass a prop called `value` to the Square:

```
class Board extends React.Component {
  renderSquare(i) {
    return <Square value={i} />;  }
}
```

Change Square's `render` method to show that value by replacing `{/* TODO */}` with `{this.props.value}`:

```
class Square extends React.Component {
  render() {
    return (
      <button className="square">
        {this.props.value}      </button>
    );
  }
}
```

A Brief Overview of React Router and Client-Side Routing

What is React Router?

React Router is an API for React applications. Most current code is written with React Router 3, although version 4 has been released. React Router uses dynamic routing.

When we say dynamic routing, we mean routing that takes place ***as your app is rendering****, not in a configuration or convention outside of a running app. That means almost everything is a component in React Router. — [React Training]*

Why use React Router?

React Router, and dynamic, client-side routing, allows us to build a single-page web application with navigation without the page refreshing as the user navigates. React Router uses component structure to call components, which display the appropriate information.

By preventing a page refresh, and using Router or Link, which is explained in more depth below, the flash of a white screen or blank page is prevented. This is one increasingly common way of having a more seamless user experience. React router also allows the user to utilize browser functionality like the back button and the refresh page while maintaining the correct view of the application.

What happens when you need to navigate TWO routing system?

An API is any place where a piece of code talks to another piece of code, but we often use it to mean somebody's external resource that gives me values, or our own internal database resource(s).

If you are using a frontend and a backend, and you are potentially writing in multiple languages that don't necessarily have the same routing conventions, don't worry! The backend functions just as an API, and the user really doesn't interact with it at all. The routes that used to manage the user experience and the routes that used to manage queries to the database are not the same.

Using React Router

First, install React Router, using either yarn or npm.

```
yarn add react-router-dom
npm install react-router-dom
```

Note that in the documentation and in the API, the actual component is called Browser Router . Some people prefer to simply refer to the component as Router, so you may see it aliased or choose to alias it in code, in which case it will be referred to as <Router> as long as it as been imported with an alias.

```
import { BrowserRouter } from 'react-router-dom'//alternatively, aliasedimport { BrowserRouter as Router} from 'react-router-dom'
```

When not using React Router, App is often the highest parent component in React apps. With React Router, however, the Router component needs to be the highest parent. This just lets all of the component use the power of Router, because as a parent, it passes down all of its props to its children, and thus the entire application.

To set this up most simply, in index.js, include:

```
ReactDom.render(<Router><App /></Router>)
```

React Router <Link>

Using <Link> functions similarly to using an <a> tag, but, as mentioned above, prevents a page refresh, and looks like a React component (because it is!).

For example, if you want a link to contact information,

```
<Link to= "/contact">Contact Us</Link>
```

Using Link will navigate the user to /contact, and the URL will change, without the page reloading!

Where does this get included in the application? The Router component functions like a control center, and connects the route path (link) with the React component that should appear on the page.

```
<Route path="/contact" component={ContactPage} />
```

What does this accomplish? The "component=" syntax is like a reference for a callback function. Since components are functions, and functions can be referenced, then you can reference a component in the same way!

At its most basic, this is client-side routing

Additional Notes

1. Setting the path to just “/” — it effectively wraps, or shows up on every page. This is particularly helpful for things like a navigation bar, a footer, a sign in/out toggle button . So, at any point that the path has “/” in it — it is a match and the component will render.

2. Relatedly, in some systems, such as rails, you could have had a route like

```
books.com/novels/1
```

which would show one novel, with /books being the index page, and the /books/1 is the show page with index 1 (the first book). It doesn’t work quite the same with React Router! To get a component to something to show up ONLY at a specific route, use the “exact” keyword.

```
Route exact path='/books'
```

will be all the books at books.com, or some sort of index page, and will *not* show at path=“dogs/1”

Since React Router allows you to set your own URLs, it can be helpful to construct a list of routes along with domain modeling and wireframing and consider if the URL make sense in the context of how an application would be structured and how a user interacts with it.

3. You cannot pass down props in a function reference! Instead, use the syntax.

```
render={() => <Text myExample='Marcella' />}
```

To access those values, you can then use

```
<p> Hello world from {props.myExample} </p>
```

Setting up

Make sure you have `Nodejs` and `npm` installed on your machine (at least version 8 or higher) along with a code editor and a web browser (preferably Chrome or Firefox).
Create a new project using `create-react-app`:

```shell
npx create-react-app react-routing
```

shell
Clean up the project template by removing `logo.svg`, all its relevant imports, and some boilerplate inside `App.js`. Your `App.js` should look like this:

```
import React from 'react';

```

```
3function App() {
4  return (
5    <div className="App">
6
7    </div>
8  );
9}
10
11export default App;
```

jsx

Install the `react-router-dom` package using the following command inside the root directory:

```
1npm install react-router-dom
```

Styles

Put the following styles inside `index.css`:

```
1body {
2  margin: 0;
3  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI',
'Roboto', 'Oxygen',
4    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans',
'Helvetica Neue',
5    sans-serif;
6  -webkit-font-smoothing: antialiased;
7  -moz-osx-font-smoothing: grayscale;
8}
9
10.App {
11  max-width: 400px;
12  margin: 30px auto;
13  text-align: center;
14}
15
16nav{
17  padding: 5px;
18  background: rgb(151, 150, 150);
```

```
  color: #ffffff;
}
nav ul {
  padding: 0;
}
nav ul li {
  display: inline-block;
  margin: 0 10px;
  cursor: pointer;
  padding: 5px;
  font-weight: 600;
}

.nav-link{
  color: #ffffff;
  text-decoration: none;
}
```

css

Creating the UI Components

All UI components will render an `<h2>` indicating which component is currently rendered by that route.
Simply import React from the core, create a stateless functional component, and make sure to export it in the end. This is how `About.js` should look:

```
import React from 'react';

const About=()=>{
    return(
        <div className="about">
            <h2>This is the about page</h2>
        </div>
    )
}

export default About;
```

jsx

Similarly, `Contact.js` will look like this:

```jsx
import React from 'react';

const Contact=()=>{
    return(
        <div className="contact">
            <h2>This is the contact page</h2>
        </div>
    )
}

export default Contact;
```

jsx

Finally, you can create the `Home` component in the same manner:

```jsx
import React from 'react';

const Home=()=>{
    return(
        <div className="home">
            <h2>This is the home page</h2>
        </div>
    )
}

export default Home;
```

jsx

The Router Module

The three important components of the Router Module are `<BrowserRouter/>` , `<Switch/>`, and `<Route/>`.

Everything inside `<BrowserRouter/>` indicates information about Routing. It's a wrapper component for all the components that will be using client-side routing to render themselves, and it informs React about that.

`<Switch/>` ensures that only one route is handled at a time, and the `<Route/>` component tells us which component will be rendered on which route.

For a large application, it's more logical to create a separate component, but for now, it's convenient to put all routes inside `App.js`.

```jsx
import React from 'react';
```

```jsx
import Navbar from './Components/Navbar';
import { BrowserRouter, Switch, Route } from 'react-router-dom';
import Home from './Components/Home';
import Contact from './Components/Contact';
import About from './Components/About';

function App() {
  return (
    <div className="App">
        <BrowserRouter>
          <Navbar/>
            <Switch>
              <Route exact path="/" component={Home}></Route>
              <Route exact path="/contact" component={Contact}></Route>
              <Route exact path="/about" component={About}></Route>
            </Switch>
        </BrowserRouter>
    </div>
  );
}

export default App;
```

jsx

The `<Route/>` tag takes in two props. One is `path`, which simply takes a string indicating the route for that component and a `component` prop that outputs that component or renders it when the user goes to that particular route. The `exact` property ensures that the component is rendered only when the route matches the exact string mentioned in the path and no substring or superstring of that route will be able to render that component.

The Navbar Component

Lastly, to ensure that these routes are triggered from the UI, set up these routes inside `Navbar.js` . Instead of the traditional anchor (`<a/>`) tag, `react-router-dom` provides us with the `<Link/>` tag, which works exactly like the anchor tag but prevents the default page reloading action.

```jsx
import React from 'react';
```

```
2import { Link } from 'react-router-dom';
3
4const Navbar=()=>{
5    return(
6          <nav >
7
8       <ul>
9         <li><Link to="/" className="nav-
link">Home</Link></li>
10         <li><Link to="/about" className="nav-
link">About</Link></li>
11         <li><Link to="/contact" className="nav-
link">Contact</Link></li>
12       </ul>
13    </nav>
14    )
15}
16
17export default Navbar;
```

jsx

Testing

Run `npm start` to spin a local development server. You can see the app up and running on localhost:3000 in your browser (`create-react-app` automatically does this for you). You can click the different navigation links on the navbar to see how each component renders when its corresponding route is initiated.
Note that the version of `react-router-dom` used in this example is v. 5. If you get any depreciated warnings or errors, you can use the exact version of this library used in this guide by updating your `package.json` file and running the command `npm i`.

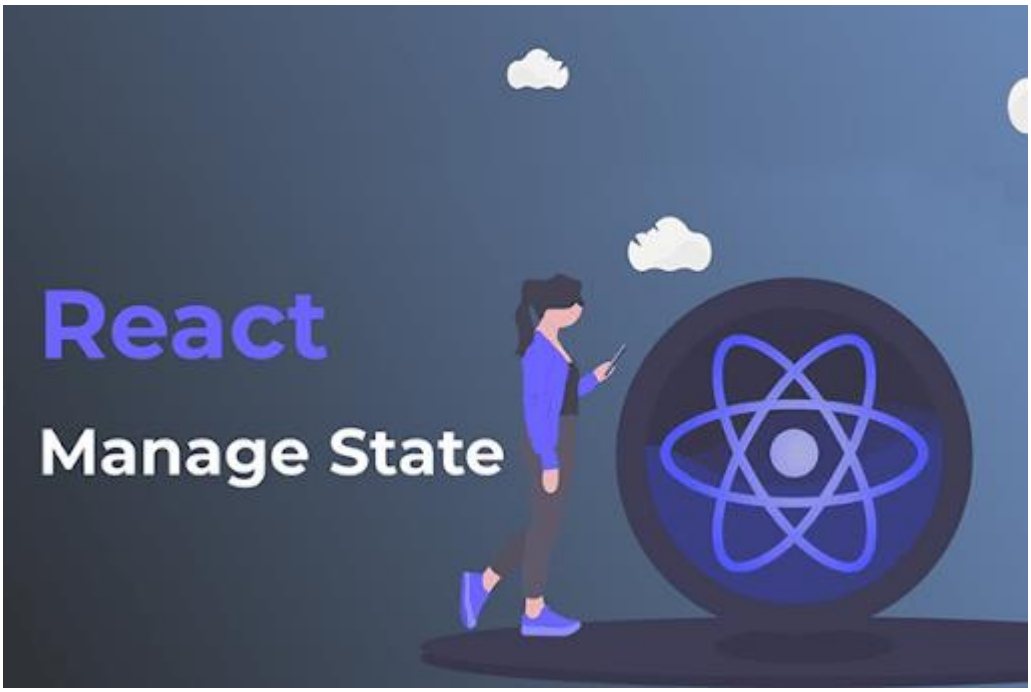
```
1{
2 ..//
3
4     "react-dom": "^16.13.1",
5   "react-router-dom": "^5.1.2",
6
7  ..//
8
9}
```

React state management: What is it and why to use it?

By Versha Gupta

Learn about React state management and why we need state management and how to use it with React hooks and Redux in the best possible way.

ReactReduxHooks



Biggest Challenge in React application is the management of state for frontend developers. In large applications, React alone is not sufficient to handle the complexity which is why some developers use React hooks and others use state management libraries such as Redux.

In this post, We are going to take a closer look at both React hooks and Redux to manage the state.

What is React State Management?

React components have a built-in state object. The state is encapsulated data where you store assets that are persistent between component renderings.

The state is just a fancy term for a JavaScript data structure. If a user changes state by interacting with your application, the UI may look completely different afterwards, because it's represented by this new state rather than the old state.

Make a state variable responsible for one concern to use efficiently.

Why do you need React state management?

React applications are built using components and they manage their state internally and it works well for applications with few components, but when the application grows bigger, the complexity of managing states shared across components becomes difficult.

Here is a simple example of an e-commerce application, in which the status of multiple components will change when purchasing a product.

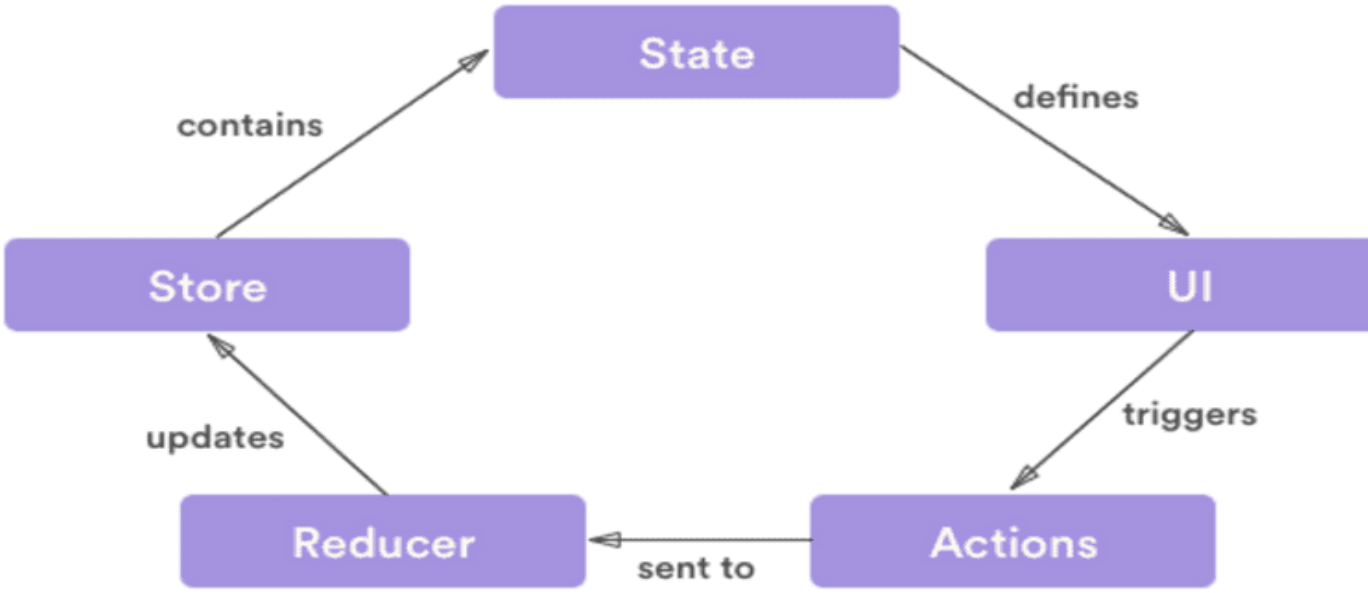
- Add that product to the shopping list
- Add product to customer history
- trigger count of purchased products

If developers do not have scalability in mind then it is really hard to find out what is happening when something goes wrong. This is why you need state management in your application.

Let’s discuss how to use react state management using react hooks and redux

What is Redux?

Redux was created to resolve this particular issue. it provides a central store that holds all states of your application. Each component can access the stored state without sending it from one component to another. Here is a simple view of how Redux works.



There are three building parts: actions, store, and reducers. Let’s briefly discuss what each of them does.

Actions in Redux

Actions are payloads of information that send data from your application to your store. Actions are sent using `store.dispatch()`. Actions are created via an action creator. Here is an example action that represents adding a new todo item:

```
{
type: "ADD_TODO",
payload: {text:"Hello Foo"}
 }
```

Here is an example of its action creator:

```
ocnst addTodo = (text) => {
  return {
     type: "ADD_TODO",
     text
  };
}
```

Reducers in Redux

Reducers specify how the application's state changes in response to actions sent to the store. An example of how Reducer works in Redux is as follows:

```
  const TODOReducer= (state = {}, action) => {
  switch (action.type) {
    case "ADD_TODO":
      return {
        ...state,
        ...action.payload
      };
    default:
```

```
      return state;
  }
};
```

Store in Redux

The store holds the application state. You can access stored state, update the state, and register or unregister listeners via helper methods.

Let's create a store for our TODO app:

```
const store = createStore(TODOReducer);
```

In other words, Redux gives you code organization and debugging superpowers. This makes it easier to build more maintainable code, and much easier to track down the root cause when something goes wrong.

What is React Hook?

These are functions that hook you into React state and features from function components. Hooks don't work inside classes and it allows you to use React features without writing a class.

Hooks are backwards-compatible, which means it doesn't keep any breaking changes. React provides some built-in Hooks like `useState`, `UseEffect` and `useReducer` etc. You can also make custom hooks.

React Hook Rules

- Call hook at the top level only means that you need to call inside a loop, nested function, or conditions.
- React function components are called hooks only.

Please see the following examples of some react hooks as follows:

What is useState and how to use it

`useState` is a Hook that Lets you add React state to function components. Example: Declaring a State Variable in class and initialize count state with 0 by setting this.state to {count:0}.

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
```

In a function component, we have no this, so we can't assign or read this.state. Instead, we call the `useState` Hook directly inside our component:

```
function Example() {
```

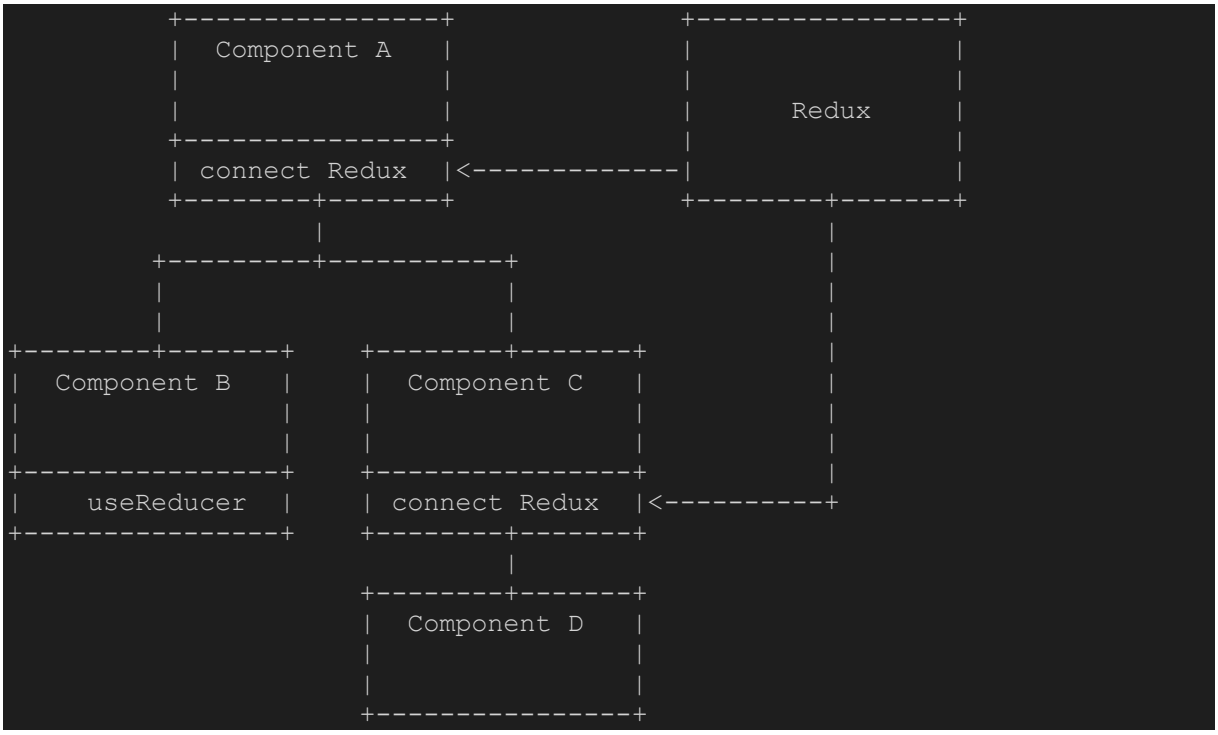
```
    const [count, setCount] = useState(0);
}
```

We declare a state variable called count and set it to 0. React will remember its current value between re-renders, and provide the most recent one to our function. If we want to update the current count, we can call setCount.

what is useReducer and how to use it

`useReducer` is a hook I use sometimes to manage the state of the application. It is very similar to the *useState* hook, just more complex. *useReducer* hook uses the same concept as the reducers in Redux. It is basically a pure function, with no side-effects.
Example of useReducer:

useReducer creates an independent component co-located state container within your component. Whereas Redux creates a global state container that hangs somewhere above your entire application.



Below an example of todo items is completed or not using the useReducer react hook.

See the following function which is a reducer function for managing state transitions for a list of items:

```
const todoReducer = (state, action) => {
     switch (action.type) {
       case "ADD_TODO":
         return state.map(todo => {
           if (todo.id === action.id) {
             return { ...todo, complete: true };
           } else {
             return todo;
           }
         });
```

```
        case "REMOVE_TODO":
          return state.map(todo => {
            if (todo.id === action.id) {
              return { ...todo, complete: false };
            } else {
              return todo;
            }
          });
        default:
          return state;
      }
    };
```

There are two types of actions which are equivalent to two states. they used to toggle the complete boolean field and additional payload to identify incoming action.

The state which is managed in this reducer is an array of items:

```
const initialTodos = [
      {
        id: "t1",
        task: "Add Task 1",
        complete: false
      },
      {
        id: "t2",
        task: "Add Task 2",
        complete: false
      }
    ];
```

In code, The useReducer hook is used for complex state and state transitions. It takes a reducer function and an initial state as input and returns the current state and a dispatch function as output

```
 const [todos, dispatch] = React.useReducer(
    todoReducer,
    initialTodos
  );
```

Complete file:

```
import React from "react";

const initialTodos = [
  {
    id: "t1",
    task: "Add Task 1",
    complete: false
  },
  {
    id: "t2",
    task: "Add Task 2",
    complete: false
  }
];
const todoReducer = (state, action) => {
  switch (action.type) {
    case "ADD_TODO":
      return state.map(todo => {
```

```
        if (todo.id === action.id) {
          return { ...todo, complete: true };
        } else {
          return todo;
        }
      });
    case "REMOVE_TODO":
      return state.map(todo => {
        if (todo.id === action.id) {
          return { ...todo, complete: false };
        } else {
          return todo;
        }
      });
    default:
      return state;
  }
};
const App = () => {
  const [todos, dispatch] = React.useReducer(todoReducer, initialTodos);

  const handleChange = todo => {
    dispatch({
      type: todo.complete ? "REMOVE_TODO" : "ADD_TODO",
      id: todo.id
    });
  };
  return (
    <ul>
      {todos.map(todo => (
        <li key={todo.id}>
          <label>
            <input
              type="checkbox"
              checked={todo.complete}
              onChange={() => handleChange(todo)}
            />
            {todo.task}
          </label>
        </li>
      ))}
    </ul>
  );
};

export default App;
```

Let’s do a similar example with Redux.

Store in *App.js*.

```
import React from "react";
import { Provider } from "react-redux";
import { createStore } from "redux";
import rootReducer from "./reducers";
import Todo from "./Components/TODO";
const store = createStore(rootReducer);

function App() {
  return (
    <div className="App">
      <Provider store={store}>
```

```
        <Todo />
      </Provider>
    </div>
  );
}

export default App;
```

Actions in *actions/index.js*.

```
export const addTodo = id => ({
  type: "ADD_TODO",
  id
});

export const removeTodo = id => ({
  type: "REMOVE_TODO",
  id
});
```

Reducers in *reducers/index.js*.

```
const initialTodos = [
  {
    id: "t1",
    task: "Add Task 1",
    complete: false
  },
  {
    id: "t2",
    task: "Add Task 2",
    complete: false
  }
];
const todos = (state = initialTodos, action) => {
  switch (action.type) {
    case "ADD_TODO":
      return state.map(todo => {
        if (todo.id === action.id) {
          return { ...todo, complete: true };
        } else {
          return todo;
        }
      });
    case "REMOVE_TODO":
      return state.map(todo => {
        if (todo.id === action.id) {
          return { ...todo, complete: false };
        } else {
          return todo;
        }
      });
    default:
      return state;
  }
};

export default todos;
```

FIle components/Todo.js

```
import React from "react";
import { connect } from "react-redux";
import { addTodo, removeTodo } from "../../redux/actions/authActions";
```

```
const Todo = ({ todos, addTodo, removeTodo }) => {
  const handleChange = todo => {
    if (todo.complete) {
      removeTodo(todo.id);
    } else {
      addTodo(todo.id);
    }
  };

  return (
    <ul>
      {todos.map(todo => (
        <li key={todo.id}>
          <label>
            <input
              type="checkbox"
              checked={todo.complete}
              onChange={() => handleChange(todo)}
            />
            {todo.task}
          </label>
        </li>
      ))}
    </ul>
  );
};

const mapStateToProps = state => ({ todos: state.auth.todos });
const mapDispatchToProps = dispatch => {
  return {
    addTodo: id => dispatch(addTodo(id)),
    removeTodo: id => dispatch(removeTodo(id))
  };
};

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(Todo);
```

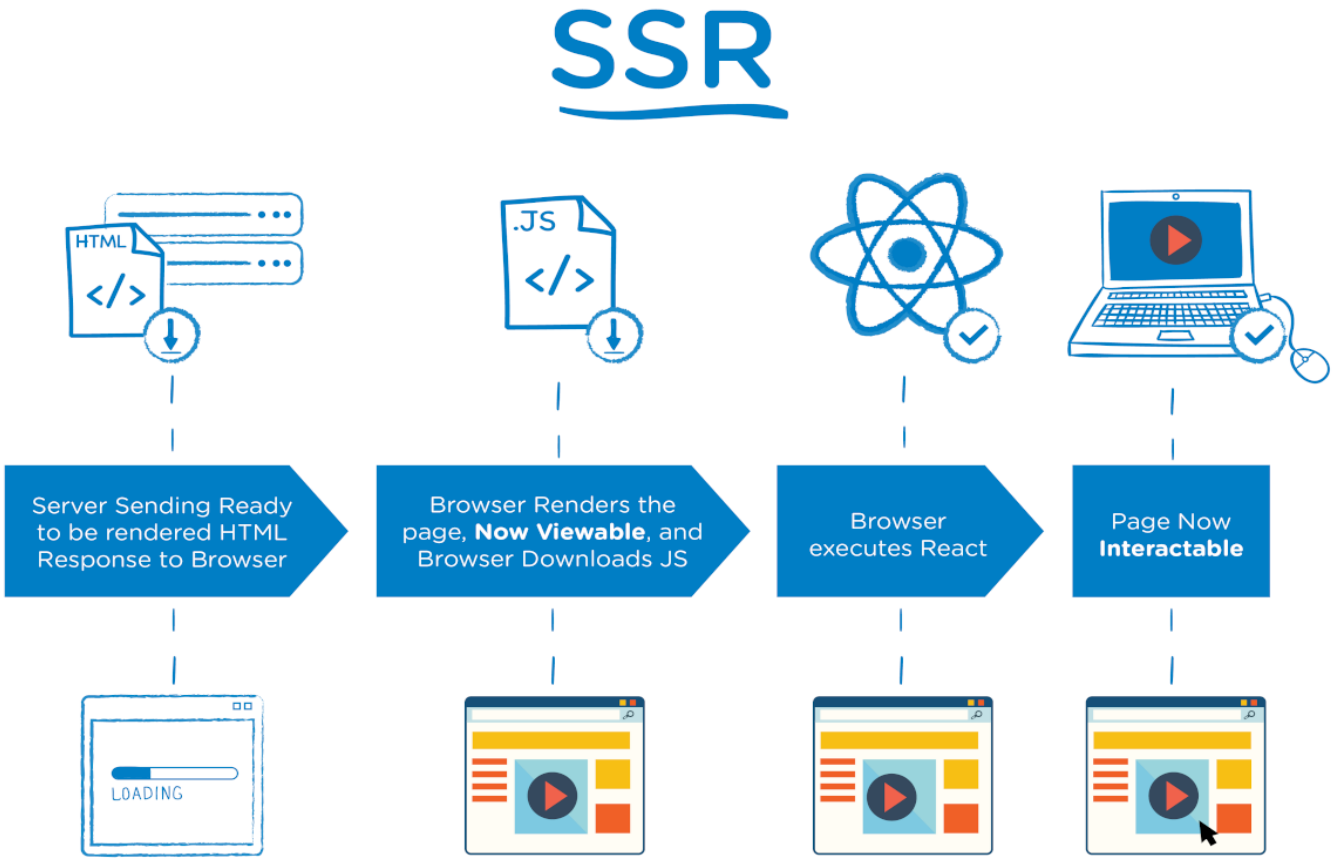
React offers react hooks which can be used as an alternative for `connect()`. You can use built-in hooks mainly useState, UseReducer and useContext and because of these you often may not require Redux.
But for large applications, you can use both redux and react hooks. it works great! React Hook is a useful new feature, and the addition of React-Redux with Redux-specific hooks is a great step towards simplifying Redux development.\

That is exactly what can be done these days which had made server-side rendering gain traction again! So, let us understand what Server-Side rendering in React JS is, the pros and cons of using it and how to go about setting it up.

What is Server-Side Rendering?

Server-side rendering with JavaScript libraries like React is where the server returns a ready to render HTML page and the JS scripts required to make the page interactive. The HTML is rendered immediately with all the static elements. In the meantime, the browser downloads and executes the JS code after which the page becomes interactive. The interactions on this page are now handled by the browser on the client-side. For any new content or new data, the browser sends a request to the server through APIs and only the newly required information is fetched.

In brief, SSR is a process where the server converts the web pages into viewable format before sending them to the browser.



Server-Side Rendering requires using a little more server power, but it gives the end user's browser less work to do. / source: omnisci.com

What are the advantages and disadvantages of SSR in react?

To decide whether to use Server-side rendering in React.js, let us look at some of the benefits and drawbacks of SSR.

ADVANTAGES

- Fast initial loading of the web page since ready to display HTML is provided to the browser.
- Great user experience even if the user has a bad connection, outdated device or JavaScript disabled in the browser because all the basic content is ready to be rendered.
- The content of the web page is indexed quicker resulting in better SEO ranking.
- A great option for static pages since server-side rendering loads the content promptly and efficiently.

DISADVANTAGES

- SSR needs more resources and can be expensive since all the processing is done on the server.
- For complex applications, the high number of server requests can slow down the site.
- Increased load with many users can lead to bottlenecks.
- Setting up SSR can be complicated and tedious.

Impact of SSR on SEO and performance

Today we know that Search Engine Optimization is crucial to driving traffic to web pages. Right now, most search engines other than Google are inadequate for rendering pages before indexing. Even with Google, there are issues in the navigation of sites with client-side rendering.

In server-side rendering, all the elements required for SEO are available in the initial response. Moreover, webpages rendered on the server-side are more accurately indexed since browsers prioritize pages that load faster.

In short, your site will be ranked higher in the search results if it is rendered on the server-side.
On social media platforms, linking to websites with server-side rendering is better for proper representation of the title and thumbnail of the site.

When talking about performance, there are three major parameters that we need to consider:

1. TFB (Time to First Byte) – the amount of time between a link clicked and the first bit of content is received.

2. FCP (First Contentful Paint) – the moment when some requested content is rendered.

3. TTI (Time To Interactive) – the time when the page becomes interactive.

TTFB can be more with server-side rendering since a fully rendered HTML page is sent to the browser, but FCP is much faster which contributes to improved performance and user-friendliness. TTI can depend on the complexity of the web page and how many scripts need to run for rendering it. Most pages have moderate level interactivity which results in low TTI when rendering server-side.

What is required in the environment to run SSR?

The first and foremost requirement for server-side rendering is to have a runtime environment that can implement a web server and handle events.

Node.js is one of the most popular frameworks to set up SSR for a React application and Express is a great option for creating the HTTP server. Next, we need a JavaScript compiler like Babel and a JS module bundler like Webpack, Rollup or a similar tool.

How Do We Unit Test a React.js Component?

There are many strategies we can use to test a React.js component:

- We can verify that a particular function in `props` was called when certain a event is dispatched.
- We can also get the result of the `render` function given the current component's state and match it to a predefined layout.
- We can even check if the number of the component's children matches an expected quantity.

In order to use these strategies, we are going to use two tools that come in handy to work with tests in React.js: Jest and Enzyme.

Using Jest to Create Unit Tests

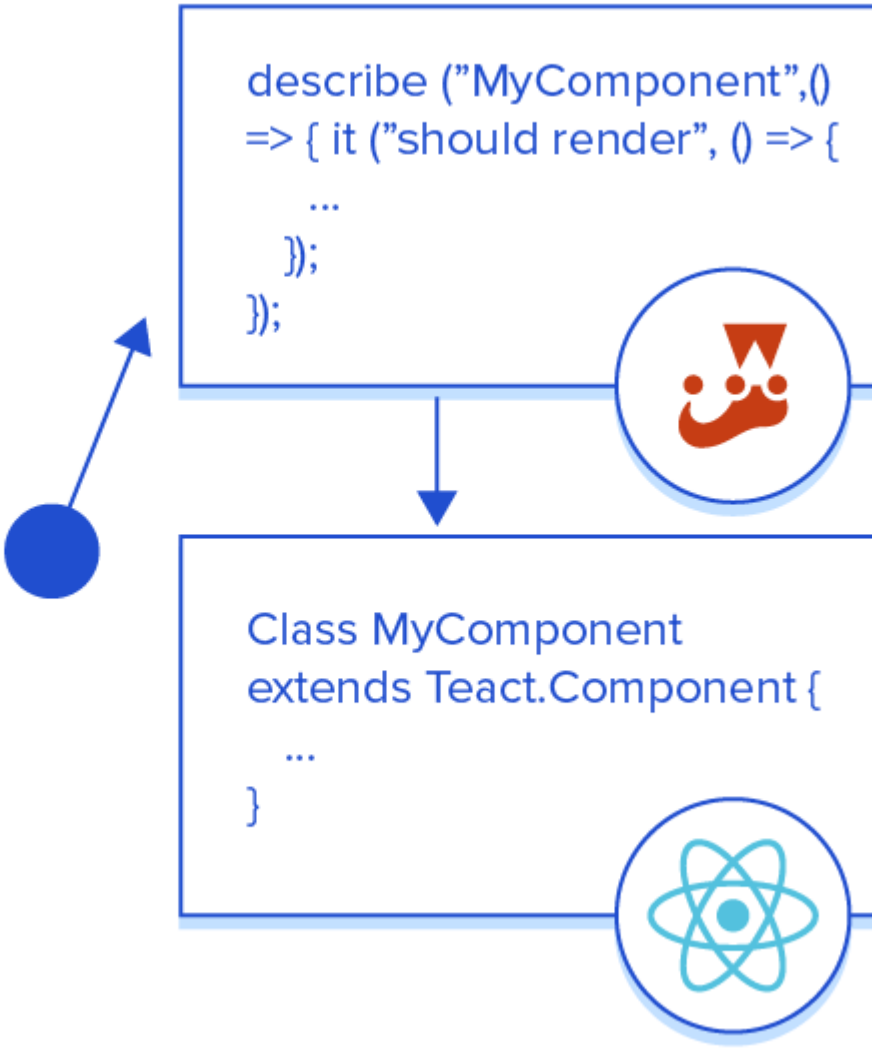
Jest is an open-source test framework created by Facebook that has a great integration with React.js. It includes a command line tool for test execution similar to what Jasmine and Mocha offer. It also allows us to create mock functions with almost zero configuration and provides a really nice set of matchers that makes assertions easier to read.

Furthermore, it offers a really nice feature called “snapshot testing,” which helps us check and verify the component rendering result. We’ll use snapshot testing to capture a component’s tree and save it into a file that we can use to compare it against a rendering tree (or whatever we pass to the `expect` function as first argument.)

Using Enzyme to Mount React.js Components

Enzyme provides a mechanism to mount and traverse React.js component trees. This will help us get access to its own properties and state as well as its children props in order to run our assertions.

Enzyme offers two basic functions for component mounting: `shallow` and `mount`. The `shallow` function loads in memory only the root component whereas `mount` loads the full DOM tree.
We’re going to combine Enzyme and Jest to mount a React.js component and run assertions over it.



Setting Up Our Environment

You can take a look at this repo, which has the basic configuration to run this example.
We're using the following versions:

```
{
  "react": "16.0.0",
  "enzyme": "^2.9.1",
  "jest": "^21.2.1",
  "jest-cli": "^21.2.1",
  "babel-jest": "^21.2.0"
}
```

Creating the React.js Component Using TDD

The first step is to create a failing test which will try to render a React.js Component using the enzyme's shallow function.

```
// MyComponent.test.js
import React from 'react';
import { shallow } from 'enzyme';
import MyComponent from './MyComponent';
describe("MyComponent", () => {
  it("should render my component", () => {
    const wrapper = shallow(<MyComponent />);
  });
});
```

After running the test, we get the following error:

```
ReferenceError: MyComponent is not defined.
```

We then create the component providing the basic syntax to make the test pass.

```
// MyComponent.js
import React from 'react';

export default class MyComponent extends React.Component {
  render() {
    return <div />;
  }
}
```

In the next step, we'll make sure our component renders a predefined UI layout using `toMatchSnapshot` function from Jest.

After calling this method, Jest automatically creates a snapshot file called `[testFileName].snap`, which is added the `__snapshots__` folder.

This file represents the UI layout we are expecting from our component rendering.

However, given that we are trying to do *pure* TDD, we should create this file first and then call the `toMatchSnapshot` function to make the test fail.

This may sound a little confusing, given that we don't know which format Jest uses to represent this layout.

You may be tempted to execute the `toMatchSnapshot` function first and see the result in the snapshot file, and that's a valid option. However, if we truly want to use *pure* TDD, we need to learn how snapshot files are structured. The snapshot file contains a layout that matches the name of the test. This means that if our test has this form:

```
desc("ComponentA", () => {
  it("should do something", () => {
    …
  }
});
```

We should specify this in the exports section: `Component A should do something 1`.
You can read more about snapshot testing here.
So, we first create the `MyComponent.test.js.snap` file.

```
// __snapshots__/MyComponent.test.js.snap
exports[`MyComponent should render initial layout 1`] = `
Array [
<div>
    <input
        type="text"
    />
</div>,
]
`;
```

Then, we create the unit test that will check that the snapshot matches the component child elements.

```
// MyComponent.test.js
...
it("should render initial layout", () => {
    // when
    const component = shallow(<MyComponent />);
    // then
    expect(component.getElements()).toMatchSnapshot();
});
...
```

We can consider `components.getElements` as the result of the render method.
We pass these elements to the `expect` method in order to run the verification against the snapshot file.
After executing the test we get the following error:

```
Received value does not match stored snapshot 1.
Expected:
 - Array [
```

```
    <div>
        <input type="text" />
    </div>,
    ]
Actual:
+ Array []
```

Jest is telling us that the result from `component.getElements` does not match the snapshot. So, we make this test pass by adding the input element in `MyComponent`.

```
// MyComponent.js
import React from 'react';
export default class MyComponent extends React.Component {
  render() {
    return <div><input type="text" /></div>;
  }
}
```

The next step is to add functionality to `input` by executing a function when its value changes. We do this by specifying a function in the `onChange` prop. We first need to change the snapshot to make the test fail.

```
//__snapshots__/MyComponent.test.js.snap
exports[`MyComponent should render initial layout 1`] = `
Array [
<div>
    <input
        onChange={[Function]}
        type="text"
    />
</div>,
]
`;
```

A drawback of modifying the snapshot first is that the order of the props (or attributes) is important.

Jest will alphabetically sort the props received in the `expect` function before verifying it against the snapshot. So, we should specify them in that order. After executing the test we get the following error:

```
Received value does not match stored snapshot 1.
Expected:
 - Array [
    <div>
        onChange={[Function]}
        <input type="text"/>
    </div>,
```

```
    ]
Actual:
+ Array [
    <div>
      <input type="text"  />
    </div>,
    ]
```

To make this test pass, we can simply provide an empty function to `onChange`.

```
// MyComponent.js
import React from 'react';

export default class MyComponent extends React.Component {
  render() {
    return <div><input
      onChange={() => {}}
      type="text" /></div>;
  }
}
```

Then, we make sure that the component's state changes after the `onChange` event is dispatched.

To do this, we create a new unit test which is going to call the `onChange` function in the input by passing an *event* in order to mimic a real event in the UI.

Then, we verify that the component *state* contains a key named `input`.

```
// MyComponent.test.js
...
it("should create an entry in component state", () => {
    // given
    const component = shallow(<MyComponent />);
    const form = component.find('input');
    // when
    form.props().onChange({target: {
      name: 'myName',
      value: 'myValue'
    }});
    // then
    expect(component.state('input')).toBeDefined();
});
```

We now get the following error.

```
Expected value to be defined, instead received undefined
```

This indicates that the component doesn't have a property in the state called `input`.

We make the test pass by setting this entry in the component's state.

```
// MyComponent.js
import React from 'react';

export default class MyComponent extends React.Component {
  render() {
    return <div><input
      onChange={(event) => {this.setState({input: ''})}}
      type="text" /></div>;
  }
}
```

Then, we need to make sure a value is set in the new state entry. We will get this value from the event.

So, let's create a test that makes sure the state contains this value.

```
// MyComponent.test.js
...
  it("should create an entry in component state with the event value", () => {
    // given
    const component = shallow(<MyComponent />);
    const form = component.find('input');
    // when
    form.props().onChange({target: {
      name: 'myName',
      value: 'myValue'
    }});
    // then
    expect(component.state('input')).toEqual('myValue');
  });
  ~~~

Not surprisingly, we get the following error.

~~
Expected value to equal: "myValue"
Received: ""
```

We finally make this test pass by getting the value from the event and setting it as the input value.

```
// MyComponent.js
import React from 'react';

export default class MyComponent extends React.Component {
  render() {
    return <div><input
```

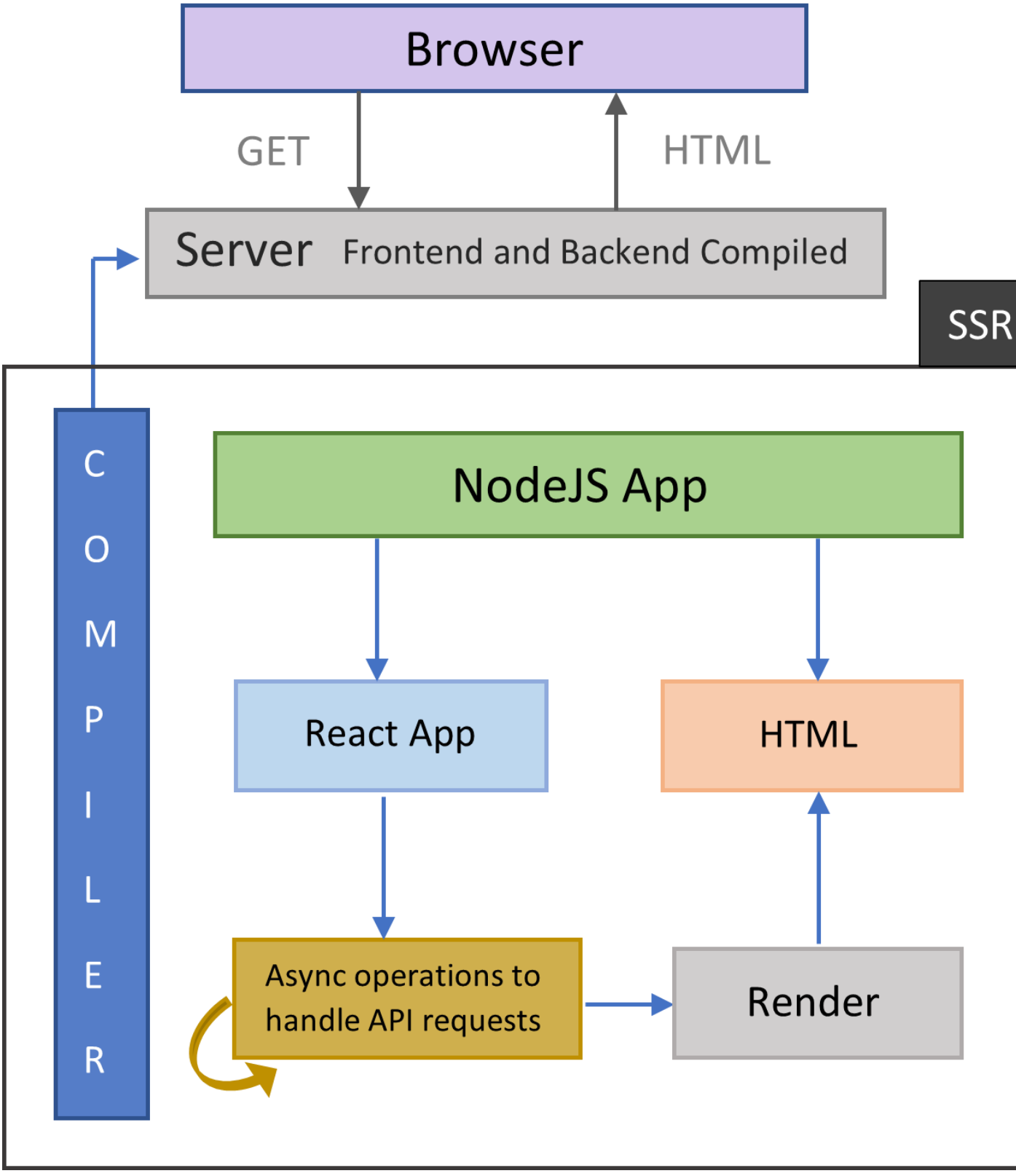
```
      onChange={(event) => {
         this.setState({input: event.target.value})}}
      type="text" /></div>;
  }
}
```

After making sure all tests pass, we can refactor our code.

We can extract the function passed in the `onChange` prop to a new function called `updateState`.

```
// MyComponent.js
import React from 'react';

export default class MyComponent extends React.Component {
  updateState(event) {
    this.setState({
      input: event.target.value
    });
  }
  render() {
    return <div><input
      onChange={this.updateState.bind(this)}
      type="text" /></div>;
  }
}
```

We now have a simple React.js component created using TDD



A schematic representation of SSR application / Based on the source – https://dev.to/alexsergey/server-side-rendering-from-zero-to-hero-2610

Let us see briefly how to set up a simple React JS website with server-side rendering using Express.js. The configuration steps are along the following lines:

1. Create a new folder for the React app.
2. Install the dependencies like Babel.
3. Configure the dependencies installed and set up the packages used by the server.
4. Move all the code to the client directory and create a server directory.
5. Create basic server code with express.
6. Transpile all the code
7. Test the server code.
8. Configure webpack or any module bundler.
9. Build the code.
10. Test and debug.

This can quickly get complex and create issues regarding styles, images, routing, browser history, and so on.

All in all, there are quite a few libraries and packages to install and configure when setting up server-side rendering on your machine, and it can be tiresome to implement everything ourselves.

That is where something like Next.js can help us!

We listed business requirements, defined the **state** structure we need to make the app work, and created a series of action types to describe "what happened" and match the kinds of events that can happen as a user interacts with our app. We also wrote **reducer** functions that can handle updating our `state.todos` and `state.filters` sections, and saw how we can use the Redux `combineReducers` function to create a "root reducer" based on the different "slice reducers" for each feature in our app.

Now, it's time to pull those pieces together, with the central piece of a Redux app: the **store**.

Redux Store

The Redux **store** brings together the state, actions, and reducers that make up your app. The store has several responsibilities:

- Holds the current application state inside
- Allows access to the current state via `store.getState();`
- Allows state to be updated via `store.dispatch(action);`
- Registers listener callbacks via `store.subscribe(listener);`
- Handles unregistering of listeners via the `unsubscribe` function returned by `store.subscribe(listener).`

It's important to note that **you'll only have a single store in a Redux application**. When you want to split your data handling logic, you'll use reducer composition and create multiple reducers that can be combined together, instead of creating separate stores.

Creating a Store

Every Redux store has a single root reducer function. In the previous section, we created a root reducer function using `combineReducers`. That root reducer is currently defined in `src/reducer.js` in our example app. Let's import that root reducer and create our first store.

The Redux core library has **a `createStore` API** that will create the store. Add a new file called `store.js`, and import `createStore` and the root reducer. Then, call `createStore` and pass in the root reducer:

src/store.js

```
import { createStore } from 'redux'
import rootReducer from './reducer'

const store = createStore(rootReducer)

export default store
```

Loading Initial State

`createStore` can also accept a `preloadedState` value as its second argument. You could use this to add initial data when the store is created, such as values that were included in an HTML page sent from the server, or persisted in `localStorage` and read back when the user visits the page again, like this:

storeStatePersistenceExample.js

```
import { createStore } from 'redux'
import rootReducer from './reducer'

let preloadedState
const persistedTodosString = localStorage.getItem('todos')

if (persistedTodosString) {
  preloadedState = {
    todos: JSON.parse(persistedTodosString)
  }
}

const store = createStore(rootReducer, preloadedState)
```

Dispatching Actions

Now that we have created a store, let's verify our program works! Even without any UI, we can already test the update logic.

TIP

Before you run this code, try going back to `src/features/todos/todosSlice.js`, and remove all the example todo objects from the `initialState` so that it's an empty array. That will make the output from this example a bit easier to read.

src/index.js

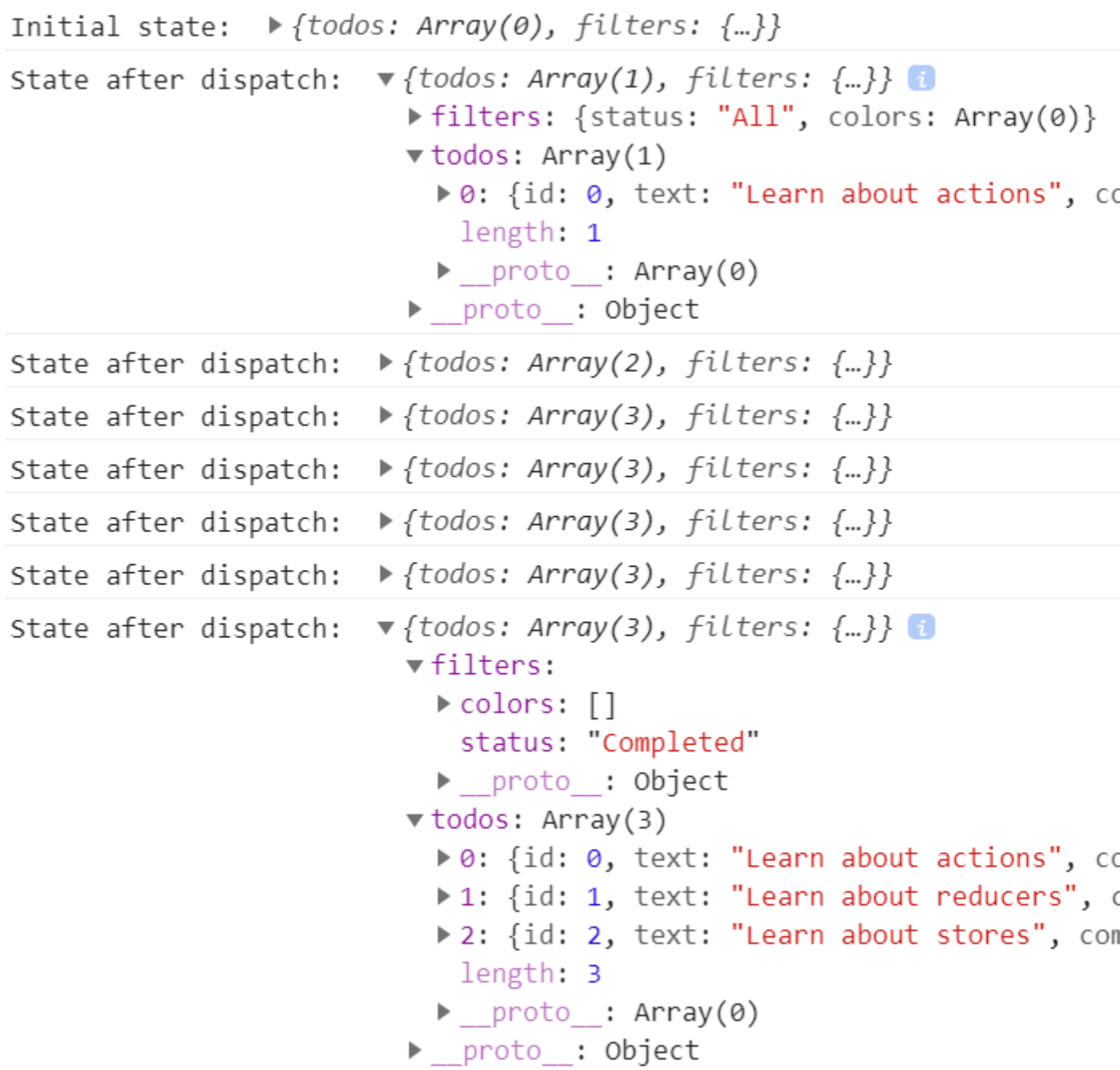
```
// Omit existing React imports

import store from './store'

// Log the initial state
console.log('Initial state: ', store.getState())
// {todos: [....], filters: {status, colors}}

// Every time the state changes, log it
// Note that subscribe() returns a function for unregistering the listener
const unsubscribe = store.subscribe(() =>
  console.log('State after dispatch: ', store.getState())
)

// Now, dispatch some actions

store.dispatch({ type: 'todos/todoAdded', payload: 'Learn about actions' })
store.dispatch({ type: 'todos/todoAdded', payload: 'Learn about reducers'
})
store.dispatch({ type: 'todos/todoAdded', payload: 'Learn about stores' })

store.dispatch({ type: 'todos/todoToggled', payload: 0 })
store.dispatch({ type: 'todos/todoToggled', payload: 1 })

store.dispatch({ type: 'filters/statusFilterChanged', payload: 'Active' })

store.dispatch({
  type: 'filters/colorFilterChanged',
  payload: { color: 'red', changeType: 'added' }
})
```

```
// Stop listening to state updates
unsubscribe()

// Dispatch one more action to see what happens

store.dispatch({ type: 'todos/todoAdded', payload: 'Try creating a store'
})

// Omit existing React rendering logic
```

Remember, every time we call `store.dispatch(action)`:

- The store calls `rootReducer(state, action)`
  - That root reducer may call other slice reducers inside of itself, like `todosReducer(state.todos, action)`
- The store saves the *new* state value inside
- The store calls all the listener subscription callbacks
- If a listener has access to the `store`, it can now call `store.getState()` to read the latest state value

If we look at the console log output from that example, you can see how the Redux state changes as each action was dispatched:



Notice that our app did *not* log anything from the last action. That's because we removed the listener callback when we called `unsubscribe()`, so nothing else ran after the action was dispatched.

We specified the behavior of our app before we even started writing the UI. That helps give us confidence that the app will work as intended.

INFO

If you want, you can now try writing tests for your reducers. Because they're pure functions, it should be straightforward to test them. Call them with an example `state` and `action`, take the result, and check to see if it matches what you expect:

todosSlice.spec.js

```
import todosReducer from './todosSlice'

test('Toggles a todo based on id', () => {
  const initialState = [{ id: 0, text: 'Test text', completed: false }]

  const action = { type: 'todos/todoToggled', payload: 0 }
  const result = todosReducer(initialState, action)
  expect(result[0].completed).toBe(true)
})
```

Inside a Redux Store

It might be helpful to take a peek inside a Redux store to see how it works. Here's a miniature example of a working Redux store, in about 25 lines of code:

miniReduxStoreExample.js

```
function createStore(reducer, preloadedState) {
  let state = preloadedState
  const listeners = []

  function getState() {
    return state
  }

  function subscribe(listener) {
    listeners.push(listener)
    return function unsubscribe() {
      const index = listeners.indexOf(listener)
      listeners.splice(index, 1)
    }
  }

  function dispatch(action) {
    state = reducer(state, action)
    listeners.forEach(listener => listener())
  }

  dispatch({ type: '@@redux/INIT' })

  return { dispatch, subscribe, getState }
}
```

This small version of a Redux store works well enough that you could use it to replace the actual Redux `createStore` function you've been using in your app so far. (Try it and see for yourself!) The actual Redux store implementation is longer and a bit more complicated, but most of that is comments, warning messages, and handling some edge cases.

As you can see, the actual logic here is fairly short:

- The store has the current `state` value and `reducer` function inside of itself
- `getState` returns the current state value
- `subscribe` keeps an array of listener callbacks and returns a function to remove the new callback
- `dispatch` calls the reducer, saves the state, and runs the listeners

- The store dispatches one action on startup to initialize the reducers with their state
- The store API is an object with `{dispatch, subscribe, getState}` inside

To emphasize one of those in particular: notice that `getState` just returns whatever the current `state` value is. That means that **by default, nothing prevents you from accidentally mutating the current state value!** This code will run without any errors, but it's incorrect:

```
const state = store.getState()
// ❌Don't do this - it mutates the current state!
state.filters.status = 'Active'
```

In other words:

- The Redux store doesn't make an extra copy of the `state` value when you call `getState()`. It's exactly the same reference that was returned from the root reducer function
- The Redux store doesn't do anything else to prevent accidental mutations. It *is* possible to mutate the state, either inside a reducer or outside the store, and you must always be careful to avoid mutations.

One common cause of accidental mutations is sorting arrays. **Calling `array.sort()` actually mutates the existing array**. If we called `const sortedTodos = state.todos.sort()`, we'd end up mutating the real store state unintentionally.

TIP

In Part 8: Modern Redux, we'll see how Redux Toolkit helps avoid mutations in reducers, and detects and warns about accidental mutations outside of reducers.

Configuring the Store

We've already seen that we can pass `rootReducer` and `preloadedState` arguments to `createStore`. However, `createStore` can also take one more argument, which is used to customize the store's abilities and give it new powers.

Redux stores are customized using something called a **store enhancer**. A store enhancer is like a special version of `createStore` that adds another layer wrapping around the original Redux store. An enhanced store can then change how the store behaves, by supplying its own versions of the store's `dispatch`, `getState`, and `subscribe` functions instead of the originals.

For this tutorial, we won't go into details about how store enhancers actually work - we'll focus on how to use them.

Creating a Store with Enhancers

Our project has two small example store enhancers available, in the `src/exampleAddons/enhancers.js` file:

- `sayHiOnDispatch`: an enhancer that always logs `'Hi'!` to the console every time an action is dispatched

- `includeMeaningOfLife`: an enhancer that always adds the field `meaningOfLife: 42` to the value returned from `getState()`

Let's start by using `sayHiOnDispatch`. First, we'll import it, and pass it to `createStore`:

src/store.js
```
import { createStore } from 'redux'
import rootReducer from './reducer'
import { sayHiOnDispatch } from './exampleAddons/enhancers'

const store = createStore(rootReducer, undefined, sayHiOnDispatch)

export default store
```

We don't have a `preloadedState` value here, so we'll pass `undefined` as the second argument instead.

Next, let's try dispatching an action:

src/index.js
```
import store from './store'

console.log('Dispatching action')
store.dispatch({ type: 'todos/todoAdded', payload: 'Learn about actions' })
console.log('Dispatch complete')
```

Now look at the console. You should see `'Hi!'` logged there, in between the other two log statements:

The `sayHiOnDispatch` enhancer wrapped the original `store.dispatch` function with its own specialized version of `dispatch`. When we called `store.dispatch()`, we were actually calling the wrapper function from `sayHiOnDispatch`, which called the original and then printed 'Hi'.

Now, let's try adding a second enhancer. We can import `includeMeaningOfLife` from that same file... but we have a problem. **`createStore` only accepts one enhancer as its third argument!** How can we pass *two* enhancers at the same time?

What we really need is some way to merge both the `sayHiOnDispatch` enhancer and the `includeMeaningOfLife` enhancer into a single combined enhancer, and then pass that instead.

Fortunately, **the Redux core includes a `compose` function that can be used to merge multiple enhancers together**. Let's use that here:

src/store.js
```
import { createStore, compose } from 'redux'
import rootReducer from './reducer'
import {
  sayHiOnDispatch,
  includeMeaningOfLife
} from './exampleAddons/enhancers'
```

```
const composedEnhancer = compose(sayHiOnDispatch, includeMeaningOfLife)

const store = createStore(rootReducer, undefined, composedEnhancer)

export default store
```

Now we can see what happens if we use the store:

```
src/index.js
import store from './store'

store.dispatch({ type: 'todos/todoAdded', payload: 'Learn about actions' })
// log: 'Hi!'

console.log('State after dispatch: ', store.getState())
// log: {todos: [...], filters: {status, colors}, meaningOfLife: 42}
```

And the logged output looks like this:

```
Hi!
State after dispatch:  ▼{todos: Array(1), filters: {…}, meaningOfLife
                         ▼filters:
                           ▶colors: []
                             status: "All"
                           ▶__proto__: Object
                           meaningOfLife: 42
                         ▼todos: Array(1)
                           ▶0: {id: 0, text: "Learn about actions", co
                             length: 1
                           ▶__proto__: Array(0)
                         ▶__proto__: Object
```

So, we can see that both enhancers are modifying the behavior of the store at the same time. `sayHiOnDispatch` has changed how `dispatch` works, and `includeMeaningOfLife` has changed how `getState` works.

Store enhancers are a very powerful way to modify the store, and almost all Redux apps will include at least one enhancer when setting up the store.

TIP

If you don't have any `preloadedState` to pass in, you can pass the `enhancer` as the second argument instead:

```
const store = createStore(rootReducer, storeEnhancer)
```

Middleware

Enhancers are powerful because they can override or replace any of the store's methods: `dispatch`, `getState`, and `subscribe`.

But, much of the time, we only need to customize how `dispatch` behaves. It would be nice if there was a way to add some customized behavior when `dispatch` runs.

Redux uses a special kind of addon called **middleware** to let us customize the `dispatch` function.

If you've ever used a library like Express or Koa, you might already be familiar with the idea of adding middleware to customize behavior. In these frameworks, middleware is some code you can put between the framework receiving a request, and the framework generating a response. For example, Express or Koa middleware may add CORS headers, logging, compression, and more. The best feature of middleware is that it's composable in a chain. You can use multiple independent third-party middleware in a single project.

Redux middleware solves different problems than Express or Koa middleware, but in a conceptually similar way. **Redux middleware provides a third-party extension point between dispatching an action, and the moment it reaches the reducer.** People use Redux middleware for logging, crash reporting, talking to an asynchronous API, routing, and more.

First, we'll look at how to add middleware to the store, then we'll show how you can write your own.

Using Middleware

We already saw that you can customize a Redux store using store enhancers. Redux middleware are actually implemented on top of a very special store enhancer that comes built in with Redux, called **`applyMiddleware`**.

Since we already know how to add enhancers to our store, we should be able to do that now. We'll start with `applyMiddleware` by itself, and we'll add three example middleware that have been included in this project.

src/store.js

```
import { createStore, applyMiddleware } from 'redux'
import rootReducer from './reducer'
import { print1, print2, print3 } from './exampleAddons/middleware'

const middlewareEnhancer = applyMiddleware(print1, print2, print3)

// Pass enhancer as the second arg, since there's no preloadedState
const store = createStore(rootReducer, middlewareEnhancer)

export default store
```

As their names say, each of these middleware will print a number when an action is dispatched.

What happens if we dispatch now?

src/index.js

```
import store from './store'

store.dispatch({ type: 'todos/todoAdded', payload: 'Learn about actions' })
// log: '1'
// log: '2'
// log: '3'
```

And we can see the output in the console:

So how does that work?

Middleware form a pipeline around the store's `dispatch` method. When we call `store.dispatch(action)`, we're *actually* calling the first middleware in the pipeline. That middleware can then do anything it wants when it sees the action. Typically, a middleware will check to see if the action is a specific type that it cares about, much like a reducer would. If it's the right type, the middleware might run some custom logic. Otherwise, it passes the action to the next middleware in the pipeline.

Unlike a reducer, **middleware can have side effects inside**, including timeouts and other async logic.

In this case, the action is passed through:

1. The `print1` middleware (which we see as `store.dispatch`)
2. The `print2` middleware
3. The `print3` middleware
4. The original `store.dispatch`
5. The root reducer inside `store`

And since these are all function calls, they all *return* from that call stack. So, the `print1` middleware is the first to run, and the last to finish.

Writing Custom Middleware

We can also write our own middleware. You might not need to do this all the time, but custom middleware are a great way to add specific behaviors to a Redux application.

Redux middleware are written as a series of three nested functions. Let's see what that pattern looks like. We'll start by trying to write this middleware using the `function` keyword, so that it's more clear what's happening:

```
// Middleware written as ES5 functions

// Outer function:
function exampleMiddleware(storeAPI) {
  return function wrapDispatch(next) {
    return function handleAction(action) {
      // Do anything here: pass the action onwards with next(action),
      // or restart the pipeline with storeAPI.dispatch(action)
      // Can also use storeAPI.getState() here
```

```
      return next(action)
    }
  }
}
```

Let's break down what these three functions do and what their arguments are.

- `exampleMiddleware`: The outer function is actually the "middleware" itself. It will be called by `applyMiddleware`, and receives a `storeAPI` object containing the store's `{dispatch, getState}` functions. These are the same `dispatch` and `getState` functions that are actually part of the store. If you call this `dispatch` function, it will send the action to the *start* of the middleware pipeline. This is only called once.
- `wrapDispatch`: The middle function receives a function called `next` as its argument. This function is actually the *next middleware* in the pipeline. If this middleware is the last one in the sequence, then `next` is actually the original `store.dispatch` function instead.
  Calling `next(action)` passes the action to the *next* middleware in the pipeline. This is also only called once
- `handleAction`: Finally, the inner function receives the current `action` as its argument, and will be called *every* time an action is dispatched.

TIP

You can give these middleware functions any names you want, but it can help to use these names to remember what each one does:

- Outer: `someCustomMiddleware` (or whatever your middleware is called)
- Middle: `wrapDispatch`
- Inner: `handleAction`

Because these are normal functions, we can also write them using ES6 arrow functions. This lets us write them shorter because arrow functions don't have to have a `return` statement, but it can also be a bit harder to read if you're not yet familiar with arrow functions and implicit returns.

Here's the same example as above, using arrow functions:

```
const anotherExampleMiddleware = storeAPI => next => action => {
  // Do something in here, when each action is dispatched

  return next(action)
}
```

We're still nesting those three functions together, and returning each function, but the implicit returns make this shorter.

Your First Custom Middleware

Let's say we want to add some logging to our application. We'd like to see the contents of each action in the console when it's dispatched, and we'd like to see what the state is after the action has been handled by the reducers.

INFO

These example middleware aren't specifically part of the actual todo app, but you can try adding them to your project to see what happens when you use them.

We can write a small middleware that will log that information to the console for us:

```
const loggerMiddleware = storeAPI => next => action => {
  console.log('dispatching', action)
  let result = next(action)
  console.log('next state', storeAPI.getState())
  return result
}
```

Whenever an action is dispatched:

- The first part of the `handleAction` function runs, and we print `'dispatching'`
- We pass the action to the `next` section, which may be another middleware or the real `store.dispatch`
- Eventually the reducers run and the state is updated, and the `next` function returns
- We can now call `storeAPI.getState()` and see what the new state is
- We finish by returning whatever `result` value came from the `next` middleware

Any middleware can return any value, and the return value from the first middleware in the pipeline is actually returned when you call `store.dispatch()`. For example:

```
const alwaysReturnHelloMiddleware = storeAPI => next => action {
  const originalResult = next(action);
  // Ignore the original result, return something else
  return 'Hello!'
}

const middlewareEnhancer = applyMiddleware(alwaysReturnHelloMiddleware)
const store = createStore(rootReducer, middlewareEnhancer)

const dispatchResult = store.dispatch({type: 'some/action'})
console.log(dispatchResult)
// log: 'Hello!'
```

Build a React Project with Create React App in 10 Steps:

The package Create React App makes creating and developing React apps a breeze.

It is one of the easiest ways to spin up a new React project and is an ideal choice to use for your own personal projects as well as for serious, large-scale applications.

We're going to cover, step-by-step, how to use all of the major features of Create React App to quickly and easily build your own React projects.

Throughout this guide, I've also included a lot of helpful tips I've learned through building apps with Create React App to make your workflow even easier.

Let's get started.

Want to learn how to create impressive, production-ready apps with React using Create React App? Check out The React Bootcamp.

Tools You Will Need:

- Node installed on your computer. You can download Node at nodejs.org. Create React App requires a Node version of at least 10.
- A package manager called npm. It is automatically included in your installation of Node. You need to have an npm version of at least 5.2.
- A good code editor to work with our project files. I highly recommend using the editor Visual Studio Code. You can grab it at code.visualstudio.com.

Step 1. How to Install Create React App

To use Create React App, we first need to open our terminal or command line on our computer.

To create a new React project, we can use the tool `npx`, provided you have an npm version of at least 5.2.
Note: You can check what npm version you have by running in your terminal `npm -v`

npx gives us the ability to use the `create-react-app` package without having to first install it on our computer, which is very convenient.
Using npx also ensures that we are using latest version of Create React App to create our project:

```
npx create-react-app my-react-app
```

Once we run this command, a folder named "my-react-app" will be created where we specified on our computer and all of the packages it requires will be automatically installed.

Note: Creating a new React app with create-react-app will usually take 2-3 minutes, sometimes more.

Create React App also gives us some templates to use for specific types of React projects.

For example, if we wanted to create a React project that used the tool TypeScript, we could use a template for that instead of having to install TypeScript manually.

To create a React app that uses TypeScript, we can use the Create React App TypeScript template:

```
npx create-react-app my-react-app --template typescript
```

Step 2. Reviewing the Project Structure

Once our project files have been created and our dependencies have been installed, our project structure should look like this:

```
my-react-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
└── src
```

What are each of these files and folders for?

- `README.md` is a markdown file that includes a lot of helpful tips and links that can help you while learning to use Create React App.
- `node_modules` is a folder that includes all of the dependency-related code that Create React App has installed. You will never need to go into this folder.
- `package.json` that manages our app dependencies and what is included in our node_modules folder for our project, plus the scripts we need to run our app.
- `.gitignore` is a file that is used to exclude files and folders from being tracked by Git. We don't want to include large folders such as the node_modules folder
- `public` is a folder that we can use to store our static assets, such as images, svgs, and fonts for our React app.
- `src` is a folder that contains our source code. It is where all of our React-related code will live and is what we will primarily work in to build our app.

Note: A new Git repository is created whenever you make a new project with Create React App. You can start saving changes to your app right away using `git add .` and `git commit -m "your commit message"`.

Step 3. How to Run your React Project

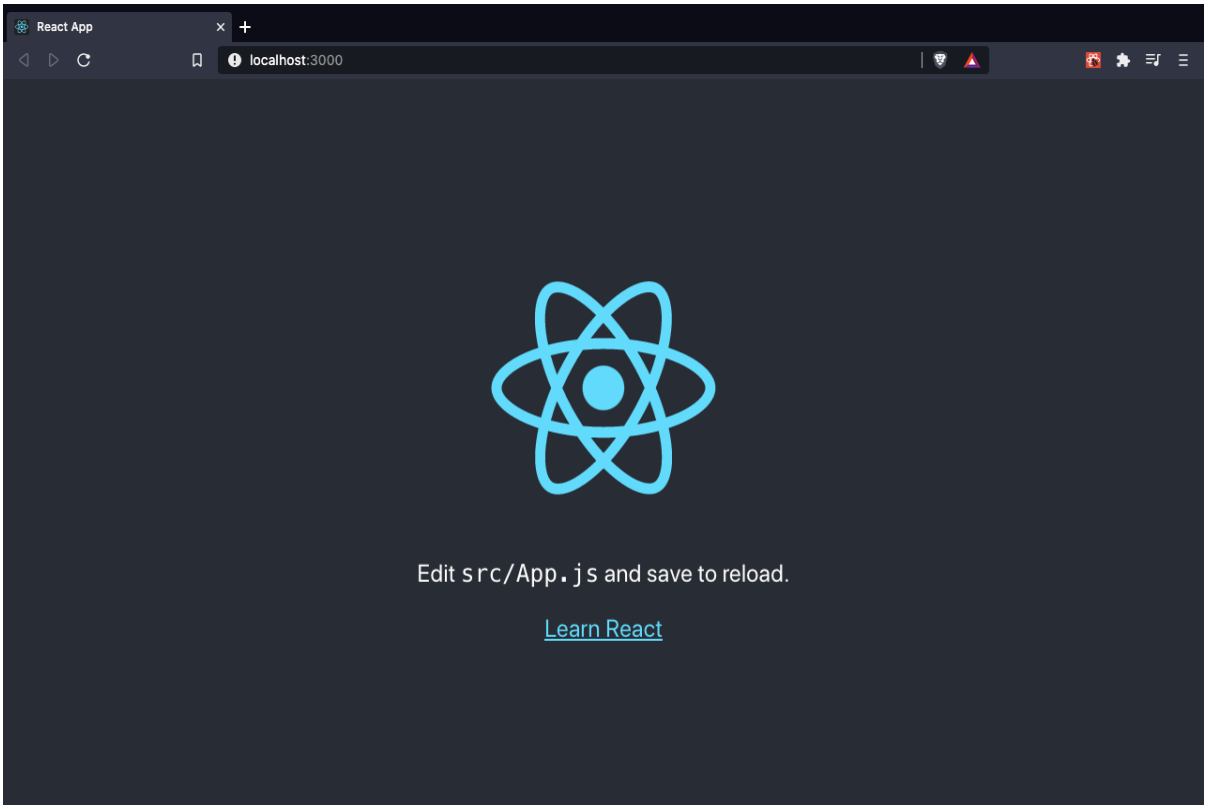
Once you have dragged your project into your code editor, you can open up your terminal (in VSCode, go to View > Terminal).

To start your React project, you can simply run:

```
npm start
```

When we run our project, a new browser tab will automatically open on our computer's default browser to view our app.

The development server will start up on localhost:3000 and, right away, we can see the starting home page for our app.



Where is our app content coming from?

It's coming from the App.js file within the src folder. If we head over to that file, we can start making changes to our app code.

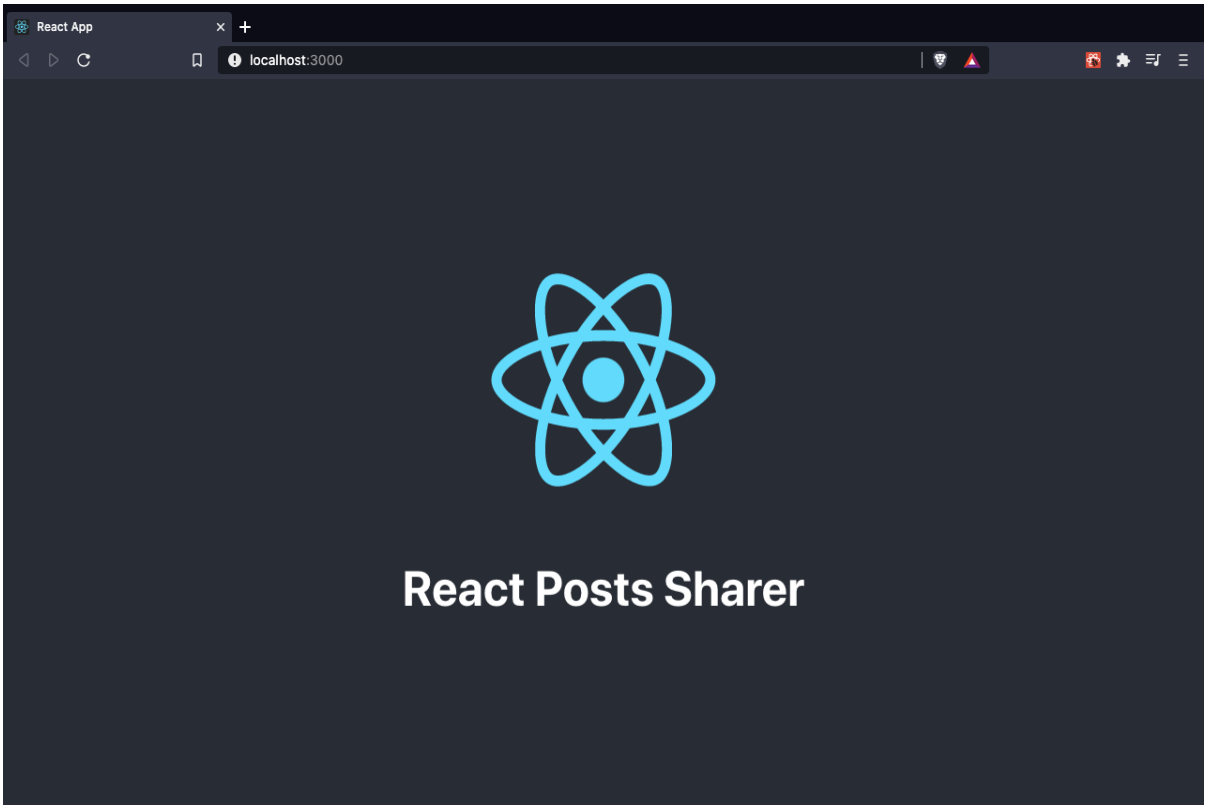
```
// src/App.js

import logo from "./logo.svg";
import "./App.css";

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
```

```
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
          >
            Learn React
          </a>
        </header>
      </div>
    );
}

export default App;
```

In particular, let's remove the `p` and `a` tags, and add an `h1` element with the name of our app, "React Posts Sharer":

```
// src/App.js

import logo from "./logo.svg";
import "./App.css";

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <h1>React Posts Sharer</h1>
      </header>
    </div>
  );
}

export default App;
```

When you save by using Command/Ctrl + S, you will see our page immediately update to look like this:



What is great about the development server is that it automatically refreshes to reflect our changes. There is no need to manually refresh the browser.

Note: The only time you may need to refresh the browser when working with Create React App is when you have an error.

Step 4. How to Run Tests with the React Testing Library

Create React App makes it very simple to test your React app.

It includes all of the packages you need to run tests using the React Testing Library (`@testing-library/react`).
A basic test is included in the file App.test.js in src. It tests that our App component successfully displays a link with the text "learn react".

We can run our tests with the command:

```
npm run test
```

Note: Tests will be run in all files that end in .test.js when you run the command `npm run test`

If we run this, however, our test will fail.

This is because we no longer have a link element, but a title element. To make our test pass we want to get a title element with the text "React Posts Sharer".

```
// src/App.test.js

import { render, screen } from "@testing-library/react";
import App from "./App";

test("renders app title element", () => {
  render(<App />);
  const titleElement = screen.getByText(/React Posts Sharer/i);
  expect(titleElement).toBeInTheDocument();
});
```

Once we run our test again, we see that it passes:

```
PASS  src/App.test.js

  ✓ renders app title element (54 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        2.776 s, estimated 3 s
Ran all test suites related to changed files.
```

Note: When run the test command, you do not need to start and stop it manually. If you have a failing test, you can jump into your app code, fix your error, and once you hit save, all tests will automatically re-run.

Step 5. How to Change the App's Meta Data

How does our project work? We can see how by going to the index.js file.

```
// src/index.js

import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
```

```
  document.getElementById('root')
);
```

The package ReactDOM renders our application (specifically the App component and every component within it), by attaching it to a HTML element with an id value of 'root'.

This element can be found within `public/index.html`.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
  name="description"
  content="Web site created using create-react-app"
 />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

The entire React app is attached to this HTML page using the div with the id of root you see above.

We don't need to change anything within the `body` tags. However, it is useful to change the metadata in the `head` tags, to tell users and search engines about our specific app.
We can see that it includes meta tags for a title, description, and favicon image (the little icon in the browser tab).

You'll also see several other tags like theme-color, apple-touch-icon and manifest. These are useful if users want to add your application to their device or computer's home screen.

In our case, we can change the title to our app name and the description to suit the app we're making:

```
<!DOCTYPE html>
<html lang="en">
```

```
    <head>
        <meta charset="utf-8" />
        <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <meta name="theme-color" content="#000000" />
        <meta
      name="description"
      content="App for sharing peoples' posts from around the web"
    />
        <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
        <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
        <title>React Posts Sharer</title>
    </head>
    <body>
        <noscript>You need to enable JavaScript to run this app.</noscript>
        <div id="root"></div>
    </body>
</html>
```

Step 6. How to Work with Images and Other Assets

If we look at our App component, we see that we are using an `img` element.
What's interesting is that we are importing a file from our src folder, as if it was a variable being exported from that file.

```
// src/App.js

import "./App.css";
import logo from "./logo.svg";

function App() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1>React Posts Sharer</h1>
        </header>
      </div>
    );
}
```

```
export default App;
```

We can import image files and other static assets directly into our React components. This feature comes from Create React App's webpack configuration.

Instead of including static assets directly within our src folder, we also have the option to include them in our public folder.

If we move our logo.svg file from src to public, instead of importing our file by using the import syntax, we can write the following:

```
// src/App.js

import "./App.css";

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src="/logo.svg" className="App-logo" alt="logo" />
        <h1>React Posts Sharer</h1>
      </header>
    </div>
  );
}

export default App;
```

Any file that's placed in the public folder can be used in .js or .css files with the syntax: `/filename.extension`.
What is convenient about Create React App is that we do not need to use an `img` element at all to display this svg.
We can import this svg as a component using the following syntax:

```
// src/App.js

import { ReactComponent as Logo } from "./logo.svg";
import "./App.css";

function App() {
  return (
    <div className="App">
      <header className="App-header">
```

```
          <Logo style={{ height: 200 }} />
          <h1>React Posts Sharer</h1>
        </header>
      </div>
    );
}

export default App;
```

What is happening here? We can import the svg file as a ReactComponent and then rename it to whatever name we like using the `as` keyword.
In other words, we can use our imported svg just like we would a regular component.
Svg files have traditionally been challenging to use in React. This component syntax makes it very easy and allows us to do things such as use inline styles (like you see above, where we set the logo's height to 200px).

Step 7. How to Install Dependencies

For our post sharing app that we're making, let's grab some post data to display in our app from the JSON Placeholder API.

We can use a dependency called `axios` to make a request to get our posts.
To install axios, run:

```
npm install axios
```

Note: You can more easily install packages using the shorthand command `npm i axios` instead of `npm install`

When we install axios, it will be added to our `node_modules` folder.
We can review all dependencies we have installed directly within our package.json file and see that axios has been added to the "dependencies" section:

```
{
  "name": "my-react-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.4",
    "@testing-library/react": "^11.1.0",
    "@testing-library/user-event": "^12.1.10",
    "axios": "^0.21.1",
    "react": "^17.0.1",
    "react-dom": "^17.0.1",
    "react-scripts": "4.0.2",
    "web-vitals": "^1.0.1"
```

```
  }
}
```

We will not include it in this project, but if you are interested in using TypeScript with your existing Create React App project, the process is very simple.

You simply need to install the `typescript` dependency and the appropriate type definitions to use for React development and testing:

```
npm install typescript @types/node @types/react @types/react-dom @types/jest
```

After that, you can simply restart your development server and rename any React file that ends with .js to .tsx and you have a working React and TypeScript project.

Step 8. How to Import Components

Instead of writing all of our code within the App component, let's create a separate component to fetch our data and display it.

We'll call this component Posts, so let's create a folder within src to hold all of our components and put a file within it: Posts.js.

The complete path for our component file is `src/components/Posts.js`.
To fetch our posts, we will request them from JSON Placeholder, put them in a state variable called posts, and then map over them to display their title and body:

```
// src/components/Posts.js

import React from "react";
import axios from "axios";

function Posts() {
  const [posts, setPosts] = React.useState([]);

  React.useEffect(() => {
    axios
      .get("http://jsonplaceholder.typicode.com/posts")
      .then((response) => setPosts(response.data));
  }, []);

  return (
    <ul className="posts">
      {posts.map((post) => (
        <li className="post" key={post.id}>
          <h4>{post.title}</h4>
          <p>{post.body}</p>
```

```
            </li>
          ))}
        </ul>
    );
}

export default Posts;
```

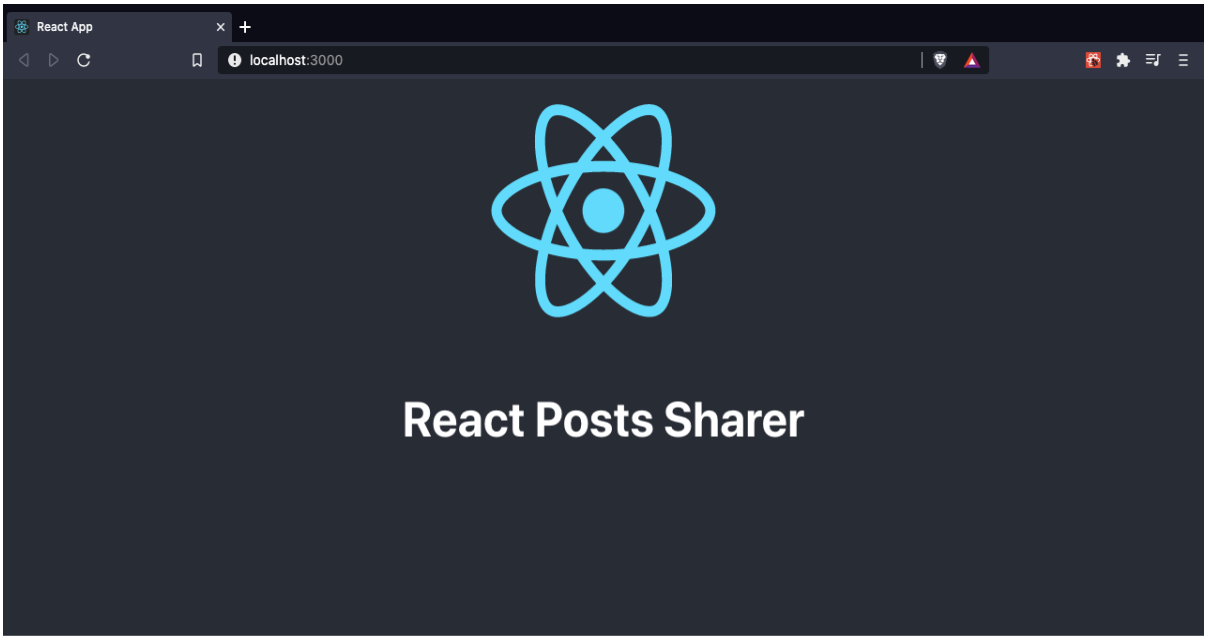
We are fetching and returning our post data from the Posts component, but to see it in our app, we need to import it into the App component.

Let's head back to App.js and import it by going into the components folder and getting the Posts component from Posts.js.

After that, we can place our Posts component under our `header`:

```
// src/App.js

import Posts from "./components/Posts";
import "./App.css";

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src="/logo.svg" className="App-logo" alt="logo" />
        <h1>React Posts Sharer</h1>
      </header>
      <Posts />
    </div>
  );
}

export default App;
```

And we can see all of our fetched posts on our home page below our header:



- **sunt aut facere repellat provident occaecati excepturi optio reprehenderit**

  quia et suscipit suscipit recusandae consequuntur expedita et cum reprehenderit molestiae ut ut quas totam nostrum rerum est autem sunt rem eveniet architecto

- **qui est esse**

  est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla

Step 9: How to Style our App with CSS

Our app could benefit from some improved styles.

Create React App comes with CSS support out of the box. If you head to App.js, you can see at the top that we are importing an App.css file from src.

Note: You can import .css files into any component you like, however these styles will be applied globally to our app. They are not scoped to the component into which the .css file is imported.

Within App.css, we can add some styles to improve our app's appearance:

```
/* src/App.css */

.App {
  text-align: center;
  margin: 0 auto;
  max-width: 1000px;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
```

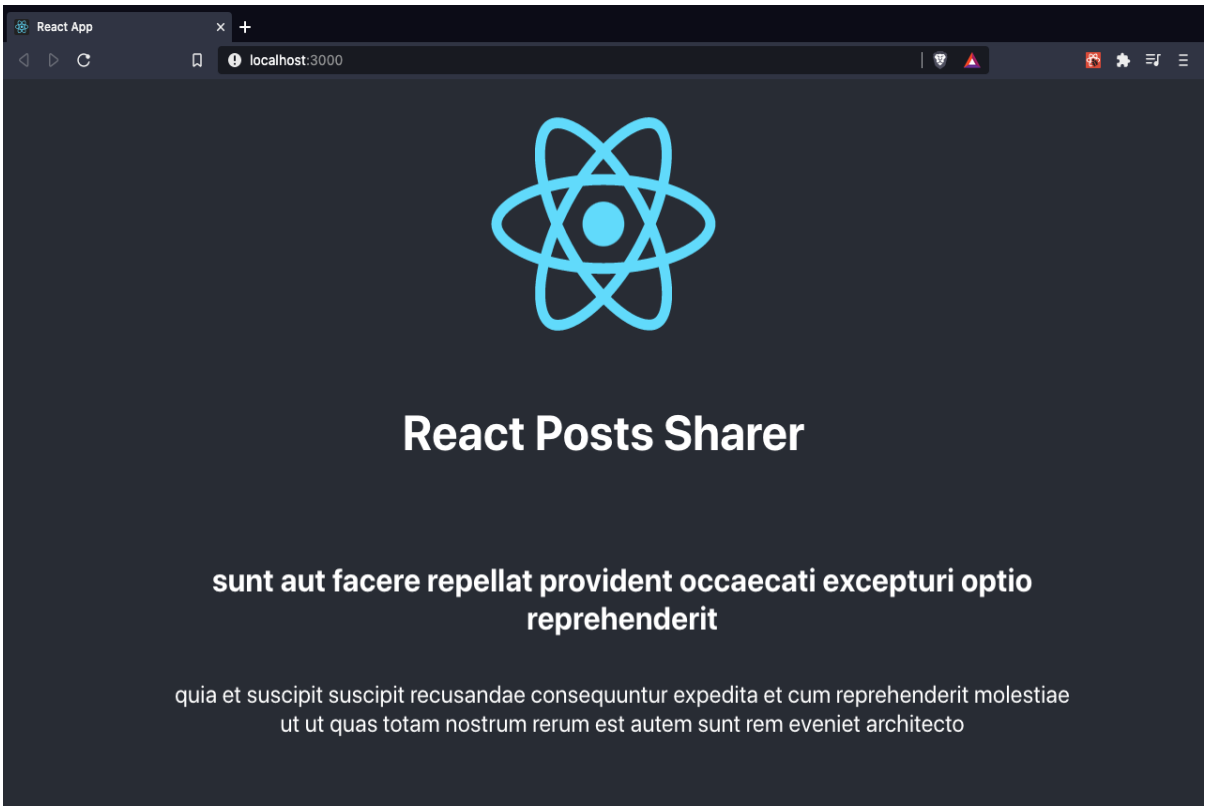
```
}

.App-header {
  margin: 0 auto;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
}

li {
  list-style-type: none;
}

.post {
  margin-bottom: 4em;
}

.post h4 {
  font-size: 2rem;
}
```

There is also another global stylesheet called index.css that has more general style rules.

In it, we can add some additional properties for the body element to make our background dark and our text white:

```
/* src/index.css */

body {
  background-color: #282c34;
  color: white;
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto", "Oxygen",
    "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
```

After adding these styles, we have a much better looking app:



Be aware that it is also very easy to add a more advanced CSS configurations, such as if you want to add CSS modules or SASS to your React app.

More helpful resources for CSS styling are included in your README.md file.

Step 10. How to Build the App and Publish It

Once we are happy with our app and are ready to publish it, we can build it with the following command:

```
npm run build
```

This command will create an optimized production build for our project and will output what files it has generated and how large each file is:

```
Compiled successfully.


File sizes after gzip:


  46.62 KB  build/static/js/2.1500c654.chunk.js
  1.59 KB   build/static/js/3.8022f77f.chunk.js
  1.17 KB   build/static/js/runtime-main.86c7b7c2.js
  649 B     build/static/js/main.ef6580eb.chunk.js
  430 B     build/static/css/main.5ae9c609.chunk.css
```

The output is coming from the build tool Webpack.

It helps to give us an idea of the size of our app files because the size of our .js files in particular can make a large impact on our app's performance.

Each chunk includes a unique string or hash, which will change on every build to make sure any new deployment is not saved (cached) by the browser.

If we did not have this cache-busting hash for each of our files, we likely couldn't see any changes we made to our app.

Finally, we can run our built React project locally with the help of the npm package `serve`. This is helpful to detect any errors we might have with the final version of our project before pushing live to the web.

Like create-react-app, we can use npx to run `serve` without installing it globally on our computer.

```
npx serve
```

Using `serve`, our app will start up on a different development port instead of 3000. In this case, localhost:5000.
And with that, we have a completed React application ready to publish live to the web on any deployment service, such as Netlify, Github Pages, or Heroku!