**What is a web server and how does it work?**

A server is a computer that runs applications and

services ranging from websites to instant messaging.

It's called a server because it provides a service to another computer and

its user also known as the client.

Is typically stored in something called a data center with hundreds or thousands of

other servers, all running different services connected to the internet.

There are many different systems in data centers to ensure that servers have

continuous power, continuous internet connection and

are kept called 24 hours per day.

Did you know that there are data centers located all around the world.

Many websites use these to allow you to access your content quickly from the data

center nearest to you.

The data center servers are built based on the service purpose.

For example, if the server is only to be used for

storing images, it might have a lot of hard drive space.

Whereas a server computing complex calculations might need a fast processor

and a lot of memory.

This is usually referred to as a server hardware.

The physical components of a server.

Once the server hardware is installed in the data center,

a piece of code can now run.

The code that runs on the hardware is commonly known as software.

One way I like to remember this is to think of hardware as something you can

physically touch and

is difficult to change as you need to physically replace components.

Software is soft or easy to change.

You just replace the code running on the server.

The surface is a server can run can vary but

in this video you will learn about a type of service software known as a web server.

A web server has many functions which includes website storage and

administration, data storage, security and managing email.

Another primary function is to handle something known as a web request.

When you open a browser on your device and type the name of the website,

it's the job of the web server to send you back to that website's content.

This process is known as the request response cycle and

you will learn more about it later.

Web servers are also designed to respond to thousands of requests from clients

per second.

In this video, you learned about web servers and

how they consist of hardware and software.

You also learned that one of the primary roles of a web server is to respond to web

requests from the client.

**What are websites and webpages?**

You may have heard the terms webpages and websites, but what's the difference?

A web page is a document that displays images, texts, videos and other content in

the web browser, a website is a collection of webpages that link together.

You've likely visited many websites this week,

in fact you're watching a video on one right now,

let's explore an example where you visit your favorite encyclopedia website.

When you arrive at the homepage, it contains many links to different articles

clicking on one of those links brings you to a new webpage article and

that article links to even more articles and other web pages.

If all these webpages have a similar address in the web browser's address,

it can be safe to assume that they belong to the same website.

However, the links on the page do not all have to link to the same website,

they can also link to other websites.

For example, when you visit your favorite search engine and

search for a phrase, the search results are a list of links to other websites.

Did you know that as I'm talking to you,

thousands of websites are being launched to the internet.

Every day, hundreds of thousands of websites are launched, to put things into

perspective the Great Pyramid of Giza has 2.3 million blocks of stone.

So think about it,

in one week there are as many new webpages as stones in the Great Pyramid.

With current internet speeds, it would take three million years to download all

the webpages on the internet and

all of these billions of varied webpages rely on the same core technologies.

Most people interact with websites in some form daily, but

few know how webpage actually works.

I mean, what's it made up of and just how does that webpage get from the web server

to what you see on your screen or device?

Well, let's explore that now.

In its most basic form, a webpage is just a text document,

you can open and edit with any text editor, but

developers usually use more sophisticated tools for working with webpages.

If you want to work with a webpage, you need to know about three technologies and

understand how they interact, their HTML, CSS and JavaScript.

HTML structures the content you see, CSS controls the colors and style and

JavaScript is responsible for the user interaction.

As a web developer you will work with these technologies every day and

understand how they work together.

I always like to imagine it like this, think of a shot, HTML is the actual

building, the structure, CSS is the interior decoration and

landscaping outside and JavaScript is just like the business,

the services offered and the people coming in and out.

Let me give you a brief overview of each of these technologies now.

HTML stands for hypertext markup language,

it works by using something called markup tags.

These tags describe the content that is displayed in the browser window,

this content can be things like headings, paragraphs, images and

even multimedia elements such as audio and video,

the way html describes the content is known as markup.

CSS is short for cascading style sheets and adds visual enhancements like colors

and layout to the web page, this is commonly known as styling.

It works by enhancing the HTML elements and telling them how to display.

Have you ever noticed a friend changed their color or style of their hair?

Well, your friend's hair is like the HTML and the color and style is CSS.

The last technology is JavaScript,

which is a programming language built into the browser.

JavaScript provides web developers with tools for interactivity, data processing,

control and action.

Have you ever tried to log to a website only to be told that the information you

provided was incorrect or browse your favorite video streaming site and

seen content update in real time?

Well, that's JavaScript in action,

JavaScript is the powerhouse of a web page.

It has the ability to manipulate the content that you see on the screen as

you interact with it.

In fact, without JavaScript websites would be kind of boring and

very limited in terms of what you can do, okay,

you now know about the essential technologies of web page contains.

But how exactly does this code get translated to display the content that you

see on your screen?

When a copy of that webpage is sent from the web server to your browser,

each line of code is processed in sequential order from first to last.

As each line is interpreted, the browser creates the building blocks,

which is the visual representation you see on the screen.

This creation process is known as page rendering, the response from the web

server must be a complete web page in order to fulfill the request,

to show the page in the browser.

You will learn more about page rendering from the additional reading at the end of

this lesson.

And there you have it,

you now have joined the exclusive group of developers who know how web page works.

You learned about the technologies that make up a web page and

how they interact with each other.

You also learned the page is rendered by the web browser to display what the end

user sees on the screen.

In the time since you started watching this video,

another few 1000 websites launched, every one of them is another example of modern

web developers building and adding to the ever growing internet.

Are you ready to become one of them?

**What is a web browser and how does it work?**

You use a web browser every day on your devices

and you probably have one open

right now watching this video.

But do you really know how they work?

In this video, you will learn what a web browser is and

describe how it receives and displays

content. Let's get started.

A web browser, or browser for short,

is a software application that you

use to browse the World Wide Web.

It works by sending a request to a web server and then

receives a response containing the content that

is to be displayed on the screen of your device.

Once the browser is open on your device,

there is the address bar where you input

the address of the website that you want to visit.

The address is commonly known as

the Uniform Resource Locator or URL for short.

The URL contains the protocol

or the HTTP, the domain name,

usually the name of the website,

and the file path, or the

path to the page that is displayed.

When you make a request using this URL,

the browser and server communicate using a protocol

known as the Hypertext Transfer Protocol or HTTP.

Once the web browser receives the content,

it displays it on the screen of your device.

This exchange of information is made possible

by something known as the request response cycle.

Now, let me demonstrate this using

an example many of us are

familiar with, searching the web.

First, you open a web browser,

which is a software application.

Next, you type the name of your favorite search engine.

The name you type contains

something called a domain name.

Then when you press Enter,

the web browser sends a request across a network and

connects to another computer on

the Internet called a web server.

The web server is a special type of computer that

allows other computers to make requests for data.

The web server responds by

sending a webpage back to the browser.

Once the browser receives all the response information,

the search engine webpage is made visible.

The web page is a coded document that is rendered by

the browser and then presented

visually to you, the end-user.

Now that the search engine webpage

is loaded in the browser,

you interact with that page to

search for what you want to find.

For example, type restaurants near

me in the search bar and press the search button.

Once again, the browser prepares and sends

a request to the same search engine web server.

This time, the request contains

the added instructions to search for

the phrase restaurants near me.

The search engine web server processes the request by

taking the keyword and

using it to find the requested data.

This data is stored in something called a database,

which is connected to the web server.

The web server then picks up

that data and sends it back to the browser.

When the web browser receives

the full response from the web server,

it again renders a visible webpage

with links to some restaurants,

a map showing nearby locations and

more related information such as reviews,

lists, and even reservations.

Success, and all made

possible by computers having a conversation.

You now learned how the request response cycle works.

This web requests cycle is very similar for

a number of other activities we do online,

via chatting with friends,

watching our favorite movie,

or sharing files with our work colleagues.

That is just a brief summary of how the Internet works.

The next time you search for something online,

think of all the steps and data that gets

exchanged to complete your request.

**Web hosting**

By now, you know that websites and files are

stored on web servers located in datacenters.

But what if you wanted to create your own website?

Do you really need your own datacenter

with specialized hardware and software?

Thankfully, the answer is no.

Developers can launch websites to

the Internet using something known as web hosting.

Web hosting is a service where you place

your website and files on

the hosting companies web server.

You're essentially renting the space in

return for stable and secure storage.

You don't need to be accompanied to use a web host.

Individuals can rent space too.

In this video, you will learn

about the different types of web hosting

services that you can use for

your websites and web applications.

First, let me share with you some

of the different hosting options available.

These can include shared hosting,

virtual private hosting,

dedicated hosting, and Cloud hosting.

Let's explore each of

these hosting types in a little more detail now.

The cheapest form of web hosting

is known as shared hosting.

You pay for a location on a web server

containing many web hosting accounts with shared hosting.

This means that you also share

the service processing power,

memory, and bandwidth with

other websites that might slow your performance.

This option is best for

a small website with a small number of visitors.

Many developers also use this as

a low-cost sandbox environment to practice

deploying or hosting their personal websites.

Some companies offer free shared hosting,

but with limitations and often

have advertisements embedded in the webpages.

Sites with more considerable demands use

virtual private surface or VPS.

A VPS is a virtual server with dedicated CPU,

memory, and bandwidth resources.

It will be running on a hardware server with

other VPS instances but as

the resources are fixed per VPS instance,

your website is unlikely to be impacted by

the performance of other VPS instances.

A VPS instance will be

more expensive than shared hosting.

The next option up is to use dedicated hosting.

This will be a hardware server

that is dedicated to you only.

All hardware, CPU, memory,

and bandwidth resources are yours to use.

Generally, this option is more

expensive than a VPS hosting.

The last type of web hosting is

something you may have heard of.

Cloud hosting and the Cloud has grown in popularity over

the last decade and is often

mentioned in various news and services you use.

With Cloud hosting, your website

is run in something called a Cloud environment,

which spans across multiple physical and virtual servers.

If a physical or virtual server fails,

your website will run on

a different server and stay online.

The main advantage of Cloud hosting is that you can use

as many resources as you

need without hardware limitations.

However, you pay based on resource use.

For example, if you transfer

a file from the Cloud to a web browser,

you'll pay for the bandwidth used for

that transfer at a fractional cent cost per megabyte.

While this can quickly become more expensive,

is allows websites and web applications

to scale their costs as popularity grows.

This is how many of the major web applications operate.

In this video, you learned about web hosting and

the different hosting options

available to individuals and companies.

Soon you will build your very first website.

Are you excited to get it

hosted so you can share it with the world?

For more information on web hosting and Services,

please see the additional reading

at the end of this lesson.

**Introduction to Internet Protocols**

Email is a common communication method that we all know about.

But before existed the alternative was to send mail to one another through

the postal system.

There is a surprising parallel between the postal system and

how the computer sends and receives data across the internet every day.

Let's compare how they both work.

Data sent across the internet is quite an important part of our

everyday lives and

it wouldn't be possible without Internet Protocol addresses or IP addresses.

A useful way to think of IP addresses is that they function much like

addresses in a postal system that make it possible for

packets of information to be delivered to you.

And like with the postal system things can go wrong.

But let's first go over how things work.

Before we think about how they can go wrong in this video you will learn what

IP addresses are and explore how computers send data across the internet.

You probably learned how computers connect to each other to form networks and

how these networks connect to each other to form the internet.

When you send data between computers across the internet,

a common way of understanding that data is needed by the computers and

networks that the data travels across.

What makes that possible is the Internet Protocol.

Version four and version six are currently the two most widely used

standards of internet protocol.

Think of the old fashioned postal system again when you send a letter to a friend

you need their address otherwise they won't receive your letter.

Computers work in a similar way.

Every computer on a network is assigned an IP address.

In protocol version four an IP address contains four octet.

It's separated by periods or dots.

For example 192.0.2.235.

In protocol version six.

An IP address contains eight groups of hexadecimal digits separated by a colon.

For example

4527:0a00:1567:0200:ff00:0042:8329.

When you send data across a network,

you send the data as a series of messages called IP packets.

Also known as data grams at a high level IP packets

contain a header and a payload or the data.

Think of that old fashioned postal system again, when you send a letter.

You not only include the recipient's address but

also your own address in case a return location is needed.

IP packets are the same.

They include the destination IP address and source IP address.

These addresses are in the header along with some additional information to help

deliver the packet.

And the payload contains the data of the packet and

some of the other protocols which will cover in a moment.

Earlier I mentioned that things can go wrong with the postal system.

When sending multiple letters to a friend it's possible they may arrive out

of order.

It's possible that a package will get damaged or

if you're really unlucky a letter could get lost.

These issues can happen to IP packets too they can arrive out of order,

become damaged or corrupted to in transit or be dropped or lost during transit.

To solve these problems,

the payload part of the packets contains other protocols too.

You can think of them as another message inside the payload of the IP packet.

The two most common protocols are the Transmission Control Protocol

referred to as TCP and the User Datagram Protocol, also known as UDP.

TCP can solve all three of the previously mentioned issues but

at the cost of a small delay when sending the data.

This protocol is used for sending the data that must arrive correctly and

in order such as a text or image files.

UDP solves the corrupt packet issue but

packets can still arrive out of order or not arrive at all.

This protocol is used for sending data that can tolerate some data loss such

as voice calls or live video streaming.

Both of these protocols contain payloads that contain further protocols inside

of them and there you have it.

The internet uses internet protocols much like an old fashioned postal system.

These protocols can help to make sure that data

arrives in order does not become corrupted or lost or dropped during transit.

Now you're able to explain how IP addresses work and

how computers send data across the internet.

**Introduction to HTTP**

Have you ever noticed the lock icon

beside the URL in your web browser?

This means the secure version of HTTP is being used.

HTTP is a core operational protocol

of the world wide web.

It is what enables your web browser to

communicate with a web server that hosts a website.

HTTP is the communication protocol

you use whenever you browse the web.

HTTP stands for Hypertext Transfer Protocol

is a protocol used for transferring

web resources such as HTML documents,

images, styles, and other files.

HTTP is a request response based protocol.

A web browser or client sends

an HTTP request to a server,

and the webserver sends

the HTTP response back to the browser.

Next, let's start exploring

the makeup of an HTTP request.

An HTTP requests consists of a method,

path, version, and headers.

The HTTP method describes

the type of action that the client was to perform.

The primary or the most commonly used HTTP methods

are GET,

POST, PUT and DELETE.

The GET method is used to

retrieve information from the given server.

The POST request is used to send data to the server.

The PUT method updates

whatever currently exist on the website with

something else and

the DELETE method removes the resource.

The path is the representation of

where the resource is stored on the webserver.

For example, if you requested

an image at https://example.com/index.html,

the path would be /index.html.

There are multiple versions of the HTTP protocol.

I won't explore these right now,

but I want you to be aware that Version

1.1 and 2.0 are the most used.

Finally, there are the headers.

Headers contain additional information

about the request,

and the client that is making the request.

For certain requests methods,

the requests will also contain a body

of content that the client is sending.

Now, let's cover some details

about the makeup of an HTTP response.

HTTP responses follow a format

similar to the request format.

Following the header, the response

will optionally contain

a message body consisting of

the response contents such as the HTML document,

the image file, and so forth.

HTTP status codes indicate if

the HTTP requests successfully completed.

The code values are in the range of a

100-599 and a grouped by purpose.

The status message is

a text representation of the status code.

During your web browsing,

have you ever encountered pages that display

404 error not found or 500 errors?

Server is not responding?

These are HTTP status codes.

I want to briefly explain to you

about the status codes and their grouping.

There are five groups of status codes.

They're grouped by the first digit of the error number.

Informational is grouped from 100-199.

Successful responses are grouped from 200-299.

Redirection message are from 300-399.

Client error responses ranged from

400-499 and server error responses are from 500- 599.

Information responses are

provisional responses sent by the server.

These responses are interim before the actual response.

The most common inflammation response is 100 Continue,

which indicates that the web client should continue to

request or ignore the response

if the request is already finished.

Successful responses indicate that

the request was successfully processed by the web server,

with the most common success response

being 200 Ok. You're

receiving these responses every day when you

receive content successfully from a website.

The meaning of Ok depends on the HTTP method.

If the method is GET,

it means that the resource is found and is

included in the body of the HTTP response.

If it's POST, it means that

the resource was successfully

transmitted to the webserver.

If it's PUT, the resource was

successfully transmitted to the webserver.

Finally, if the method is DELETE,

it means the resource was deleted.

Redirection responses indicate to the web client

that the requested resource

has been moved to a different path.

The most common response codes used are

301 Moved Permanently and 302 Found.

The difference between the redirection messages 301 and

302 is that 302 indicates a temporary redirection.

The resource has been temporarily moved.

When web browsers receive these responses,

they will automatically submit

the request for the resource at the new path.

Client error responses indicate

that the requests contained

bad syntax or content

and cannot be processed by the webserver.

The most common codes used are,

400 is used where the web browser or

client submitted bad data to the webserver.

401 is used to indicate that the user

must log into an account

before the request can be processed.

403 is used to indicate the request was valid,

but that the webserver is refusing to process it.

This is often used to indicate that a user does not have

sufficient permissions to execute

an action in a web application.

404 is used to indicate that

the request resource was not found on the webserver.

Server error responses indicate that a failure

occurred on the webserver while

trying to process the request.

The most common code used is 500 Internal Server Error,

which is a generic error status

indicating that the server fail to process the request.

Now, have you ever bought something

online and needed to enter your credit card information?

You wouldn't want someone else to get

this information from the HTTP request.

This is where HTTPS is involved.

HTTPS is the secure version of HTTP.

It is used for secure communication between two computers

so that nobody else can see

the information being sent and received.

It does this by using something called encryption.

We won't cover encryption right now.

Like an HTTP,

the requests and responses still behave

in the same way and have the same content.

The big difference is before the content is sent,

it is turned into a secret code.

Only the other computer can turn

the secret code back into its original content.

If someone else was to look at the code,

it wouldn't be understandable.

You use HTTPS every day.

This is the lock icon you see

beside the URL in your web browser.

Before I finish, I want to leave you

with a brief summary of HTTP.

Firstly, it is a protocol

used by web clients and web servers.

It works to transfer web resources such as

HTML files and as

the foundation of any data exchanges on the web.

Also, remember that by using HTTPS,

we send the information securely.

Requests are sent by the client,

usually a web browser,

and the server replies with responses which may be

the return of an image or an HTML page.

HTTP requests have a syntax that includes method,

path, versions, and headers.

HTTP responses follow a similar format to the request.

An HTTP status codes indicate whether

the HTTP requests successfully completed.

The status code is a three-digit number that

corresponds with groups representing

different types of results.

Now you know how the HTTP protocol request

and response cycle works.

You know about its methods and the elements that

make up an HTTP request. Good job.

**HTTP examples**

This reading explores the contents of HTTP requests and responses in more depth.

**Request Line**

Every HTTP request begins with the request line.

This consists of the HTTP method, the requested resource and the HTTP protocol version.

*GET /home.html HTTP/1.1*

In this example, *GET* is the HTTP method, */home.html* is the resource requested and HTTP 1.1 is the protocol used.

**HTTP Methods**

HTTP methods indicate the action that the client wishes to perform on the web server resource.

Common HTTP methods are:

| **HTTP Method** | **Description** |
| --- | --- |
| GET | The client requests a resource on the web server. |
| POST | The client submits data to a resource on the web server. |
| PUT | The client replaces a resource on the web server. |
| DELETE | The client deletes a resource on the web server. |

**HTTP Request Headers**

After the request line, the HTTP headers are followed by a line break.

There are various possibilities when including an HTTP header in the HTTP request. A header is a case-insensitive name followed by a*:* and then followed by a value.

Common headers are:

* The *Host* header specifies the host of the server and indicates where the resource is requested from.
* The *User-Agent* header informs the web server of the application that is making the request. It often includes the operating system (Windows, Mac, Linux), version and application vendor.
* The *Accept* header informs the web server what type of content the client will accept as the response.
* The *Accept-Language* header indicates the language and optionally the locale that the client prefers.
* The *Content-type* header indicates the type of content being transmitted in the request body.

**HTTP Request Body**

HTTP requests can optionally include a request body. A request body is often included when using the HTTP POST and PUT methods to transmit data.

**HTTP Responses**

When the web server is finished processing the HTTP request, it will send back an HTTP response.

The first line of the response is the status line. This line shows the client if the request was successful or if an error occurred.

*HTTP/1.1 200 OK​*

The line begins with the HTTP protocol version, followed by the status code and a reason phrase. The reason phrase is a textual representation of the status code.

**HTTP Status Codes**

The first digit of an HTTP status code indicates the category of the response: Information, Successful, Redirection, Client Error or Server Error.

The common status codes you'll encounter for each category are:

*1XX Informational*

| **Status Code** | **Reason Phrase** | **Description** |
| --- | --- | --- |
| 100 | Continue | The server received the request headers and should continue to send the request body. |
| 101 | Switching Protocols | The client has requested the server to switch protocols and the server has agreed to do so. |
|  |  |  |

*2XX Successful*

| **Status Code** | **Reason Phrase** | **Description** |
| --- | --- | --- |
| 200 | OK | Standard response returned by the server to indicate it successfully processed the request. |
| 201 | Created | The server successfully processed the request and a resource was created. |
| 202 | Accepted | The server accepted the request for processing but the processing has not yet been completed. |
| 204 | No Content | The server successfully processed the request but is not returning any content. |
|  |  |  |

*3XX Redirection*

| **Status Code** | **Reason Phrase** | **Description** |
| --- | --- | --- |
| 301 | Moved Permanently | This request and all future requests should be sent to the returned location. |
| 302 | Found | This request should be sent to the returned location. |
|  |  |  |

*4XX Client Error*

| **Status Code** | **Reason Phrase** | **Description** |
| --- | --- | --- |
| 400 | Bad Request | The server cannot process the request due to a client error, e.g., invalid request or transmitted data is too large. |
| 401 | Unauthorized | The client making the request is unauthorized and should authenticate. |
| 403 | Forbidden | The request was valid but the server is refusing to process it. This is usually returned due to the client having insufficient permissions for the website, e.g., requesting an administrator action but the user is not an administrator. |
| 404 | Not Found | The server did not find the requested resource. |
| 405 | Method Not Allowed | The web server does not support the HTTP method used. |
|  |  |  |

*5XX Server Error*

| **Status Code** | **Reason Phrase** | **Description** |
| --- | --- | --- |
| 500 | Internal Server Error | A generic error status code given when an unexpected error or condition occurred while processing the request. |
| 502 | Bad Gateway | The web server received an invalid response from the Application Server. |
| 503 | Service Unavailable | The web server cannot process the request. |

**HTTP Response Headers**

Following the status line, there are optional HTTP response headers followed by a line break.

Similar to the request headers, there are many possible HTTP headers that can be included in the HTTP response.

Common response headers are:

* The *Date* header specifies the date and time the HTTP response was generated.
* The *Server* header describes the web server software used to generate the response.
* The *Content-Length* header describes the length of the response.
* The *Content-Type* header describes the media type of the resource returned (e.g. HTML document, image, video).

**HTTP Response Body**

Following the HTTP response headers is the HTTP response body. This is the main content of the HTTP response.

This can contain images, video, HTML documents and other media types.

**Intro to HTML, CSS and Javascript**

The web pages you visit every day are

based on three core technologies,

HTML, CSS, and JavaScript.

Together, they enable you to create web pages and

applications so you can

offer any content you have seen online.

In this video, I will demonstrate

two examples you can create using these technologies.

This is to help you understand

how they interact with one another.

Don't worry, you won't need to deal with

the details of the three languages just yet.

In the first example,

I'm building a web page that displays a digital clock.

It shows the hour, minutes, and seconds.

The time is updated every second.

I will work with three files,

an HTML, a CSS,

and a JavaScript file and

you will explore the purpose of each one of them.

To create the clock element,

Let's first define our HTML code

in a file called clock.html.

The HTML code has an element that

describes the content in hours, minutes, and seconds.

If I only use the HTML file,

the content would be shown without any

style positioning or sizing,

just simply as a display of six zeros representing

time in the format of hours: minutes: seconds.

To apply styles to the HTML element,

the HTML code references a CSS file called styles.css.

The web browser retrieves the

styles.css file and processes it.

The CSS code provides styling for the clock.

It tells the browser the position, size, color,

background, and font type,

and size of each element on the screen.

With that information,

the browser updates the content and

the sequence of zeros and

colons now display a digital clock.

Finally, we need to ensure

the clock updates with the correct time.

This is where JavaScript comes in.

The HTML references a JavaScript file called clock.js.

The browser will retrieve clock.js and process it.

The JavaScript file contains code that

every second updates the content of the hour,

minute, and second elements.

With the three files created and linked together,

your clock is fully functional.

The clock example demonstrates

JavaScript dynamically updating content.

JavaScript can also be used for interaction.

The next example demonstrates how

JavaScript can be used interactively.

It's a web page that plays a video.

Below the video, there is

a button that plays or pauses the video.

The button contains a play icon.

My HTML page describes the content,

which is the video element and the Play button element.

The HTML code references

a CSS file that the browser retrieves.

It applies the styling to the video and button.

The code in the HTML file

also references a JavaScript file.

The browser will retrieve the

videoplayer.js file and process it.

In our example, the code in

the JavaScript file performs four actions.

Firstly, it registers a listener on

the button that will execute

some code when the button is clicked.

The second function is that when the code runs,

it checks the current state of the video.

The result of that check is,

if the video is currently stopped,

it begins playing the video and

changes the buttons icon to a stop icon.

Or if the video is currently playing,

it stops playing the video and

changes the buttons icon to a play icon.

Using the three files you

create a fully interactive video player.

A summary of your functioning video player application

is when the user first sees a page,

the video will be stopped by default.

When they click the Play button,

the button will change to

a stop icon and the video will begin playing.

When they click the button again,

it will change back to a play button

and the video will stop playing.

I hope those examples give you an idea of how

the three core files link and work together.

In both the clock and video

examples the HTML file references

the CSS and the JavaScript files.

The CSS file is called on

to format and style your application,

and the code in the JavaScript file implements

the core functions of the app and user interactivity.

**Other Internet Protocols**

Hypertext Transfer Protocols (HTTP) are used on top of Transmission Control Protocol (TCP) to transfer webpages and other content from websites. This reading explores other protocols commonly used on the Internet.

**Dynamic Host Configuration Protocol (DHCP)**

You've learned that computers need IP addresses to communicate with each other. When your computer connects to a network, the Dynamic Host Configuration Protocol or DCHP as it is commonly known, is used to assign your computer an IP address. Your computer communicates over User Datagram Protocol (UDP) using the protocol with a type of server called a DHCP server. The server keeps track of computers on the network and their IP addresses. It will assign your computer an IP address and respond over the protocol to let it know which IP address to use. Once your computer has an IP address, it can communicate with other computers on the network.

**Domain Name System Protocol (DNS)**

Your computer needs a way to know with which IP address to communicate when you visit a website in your web browser, for example, *meta.com*. The Domain Name System Protocol, commonly known as DNS, provides this function. Your computer then checks with the DNS server associated with the domain name and then returns the correct IP address.

**Internet Message Access Protocol (IMAP)**

Do you check your emails on your mobile or tablet device? Or maybe you use an email application on your computer? Your device needs a way to download emails and manage your mailbox on the server storing your emails. This is the purpose of the Internet Message Access Protocol or IMAP.

**Simple Mail Transfer Protocol (SMTP)**

Now that your emails are on your device, you need a way to send emails. The Simple Mail Transfer Protocol, or SMTP, is used. It allows email clients to submit emails for sending via an SMTP server. You can also use it to receive emails from an email client, but IMAP is more commonly used.

**Post Office Protocol (POP)**

The Post Office Protocol (POP) is an older protocol used to download emails to an email client. The main difference in using POP instead of IMAP is that POP will delete the emails on the server once they have been downloaded to your local device. Although it is no longer commonly used in email clients, developers often use it to implement email automation as it is a more straightforward protocol than IMAP.

**File Transfer Protocol (FTP)**

When running your websites and web applications on the Internet, you'll need a way to transfer the files from your local computer to the server they'll run on. The standard protocol used for this is the File Transfer Protocol or FTP. FTP allows you to list, send, receive and delete files on a server. Your server must run an FTP Server and you will need an FTP Client on your local machine. You'll learn more about these in a later course.

**Secure Shell Protocol (SSH)**

When you start working with servers, you'll also need a way to log in and interact with the computer remotely. The most common method of doing this is using the Secure Shell Protocol, commonly referred to as SSH. Using an SSH client allows you to connect to an SSH server running on a server to perform commands on the remote computer. All data sent over SSH is encrypted. This means that third parties cannot understand the data transmitted. Only the sending and receiving computers can understand the data.

**SSH File Transfer Protocol (SFTP)**

The data is transmitted insecurely when using the File Transfer Protocol. This means that third parties may understand the data that you are sending. This is not right if you transmit company files such as software and databases. To solve this, the SSH File Transfer Protocol, alternatively called the Secure File Transfer Protocol, can be used to transfer files over the SSH protocol. This ensures that the data is transmitted securely. Most FTP clients also support the SFTP protocol.

**Webpages, Websites and Web Apps**

When you do something online,

you are likely to encounter web pages,

websites, and web applications.

But how do they differ? As a developer,

how do you decide which one to create?

Let's explore the main features of each one.

A typical web page is one single page that

consists of HTML, CSS, and JavaScript.

It displays images, text,

videos, and other content in the web browser.

If a web page is one single page then a website is

a collection of web pages that link

together under one domain name.

You've likely visited many websites this week.

You know, when you visit

your favorite encyclopedia website

and the homepage has a lot of

links to different articles,

clicking on one of those links brings you

to an article on a new web page,

and that article links to more articles,

and other web pages.

Well, since all of

these web pages exist under the same domain name,

they are a website.

The technical term for a link,

you click on a hyperlink.

This is because they link to hypertext content.

Remember that HTML is Hypertext Markup Language.

However, links themselves,

don't have to link to the same website.

They can also link to other websites.

For example when you go to

your favorite search engine and search for a phrase,

the search results are a list of links to other websites.

You'll explore more about hyperlinks later.

By now, you should probably know

the difference between a web page and the websites,

but what about a web application?

This is where the definitions gets a little more blurred.

The terms website and

web application are often used interchangeably.

The key difference between a website,

and web application is the level of

interactivity, and dynamic content.

The easy way to remember this is that a website

is more informative and

a web application is more interactive.

Think of ordering food online.

Let's say you would like to order

some food and you go to your favorite site,

the browser then displays a web page,

you select some food from the menu and submit your order.

Compare this to for example,

a company website that displays information

about themselves and the services they provide.

In the food ordering example,

the content displayed is specific

to your user account and location,

the web application displays content

based on the user's input and interaction.

Whereas with the company website,

the user simply views the content and

this content is the same for

everyone who visits the website.

You should now be able to distinguish between web pages,

websites, and web applications.

You know that web pages at

a particular domain make up a website,

and that the key difference between websites,

and web applications is the level of

interactivity, and dynamic content.

**Developer tools**

If a car breaks down, you can open the hood to examine the engine and

to find out what's gone wrong.

As a developer, if your front end isn't working as expected,

you can also open the hood to check what's going wrong.

In fact, it's not just your own code that you can investigate.

How about viewing other people's code.

By the end of this video you'll be able to access and

make use of common web browser developer tools.

Most web browsers come equipped with a set of developer tools that allow developers

to inspect their HTML, CSS and Javascript code.

Also, to trace http request to the web server, investigate performance issues and

review web page security.

Let's find out more by exploring the homepage of the Little Lemon Cafe.

To begin, I've just opened the Little Lemon Cafes web page on my chrome browser.

On the homepage, I can view their menu.

But I want to explore the HTML structure of this menu.

To do that, I need to open the developer tools.

To open the developer tools in chrome,

press the F12 key on your keyboard for PC or command option J on Mac.

Alternatively, you can right click on the web page and select inspect.

There are various tabs on the top row of developer tools that provide

different functionality.

In this video,

I'll give you a high level explanation of some of the most used tools.

1st, let's open the console tab.

This tab outputs, javascript logs and errors from your web application.

The sources tab shows all content resolved for the current page.

It includes HTML, CSS, Javascript, images and videos.

With the network tab, you can inspect the timeline and

details of http requests and responses for the web page.

The performance tab shows what the web browser is doing over time.

It is useful if your web application is running slow because you can pinpoint

the functions that are taking the most time.

The memory tab displays the parts of your code that are consuming the most

resources.

Finally, let's check the most used tab the elements tab.

You can use this tab to inspect the documents, HTML elements and

their properties.

For example, when I hover over an element in the elements tab,

it highlights that element in the browser pane.

On the right side of the panel, there are tabs for

inspecting the details of the elements further.

This panel shows us what CSS is applied to an element and

the result of the element displayed in the browser.

We will explore these details in a later lesson.

For now, you just need to know that if you click on an HTML element,

then the information for that element will appear in the tab.

Now, I'm going to demonstrate a fun trick.

If you double click the HTML, you can edit it in the web browser.

For example, if I select the Our Menu body element, then I can change the first item

in the menu from chicken Burger, to Turkey Burger.

This doesn't change the content on the web server.

It only updates it for me while the web pages open.

If I open the web page again, it will reset.

The Web browser developer tools are a valuable part of your development toolkit

to help you investigate and

diagnose problems and you now know how to access and make use of them.

Great work.

**Frameworks and libraries**

You are developing solutions but you need

to save some time and build faster.

What if some of your build problems

have already been solved for you?

Well it's true someone has already figured out

many key development processes and they're contained in

frameworks and libraries that are used

every day in software development,

so what exactly are frameworks and libraries?

Let's say you are not a developer,

but instead you work as a carpenter.

You make chairs and sell them online.

As a carpenter you don't design

a new hammer for every chair you make.

It makes much more sense to use an existing hammer,

but of course you are a developer.

As such it's important for you to

know that to speed up development,

developers use already developed frameworks

and libraries in their application development.

These might be open source,

meaning that the source code is

freely-available for anyone to modify and build from.

There are thousands of open source libraries and

frameworks available or there might be proprietary,

ones that are licensed or developed internally.

Many developers use the terms framework

and library interchangeably,

so what's the difference between them?

Libraries are reusable pieces of

code that can be used by your application.

They are purpose-built to

provide a specific functionality.

To give a more technical example,

you're building a small e-commerce website.

When a user wants to register they

need to provide their email address.

Email addresses while easy to

read can be complicated to validate.

In fact email addresses are

defined across several technical specifications.

That's a lot of reading just to validate an email.

Even if you do read through all the specification,

there's no point in spending hours

or even days implementing their standards

because you have access to

so many readily-available libraries

to validate email addresses.

It is for specific functionality like

validating an email address that libraries are useful.

A developers simply uses the library

to access the functionality they require,

as a result they can have more time to

continue focusing on

the development of their application.

Frameworks on the other hand provide

a structure for developers to build with.

Consider this in the context of our carpenter analogy.

As a carpenter you create a lot of different chairs,

therefore there would be

a blueprint for each chair to speed up building them.

You can decide the type of wood to use,

but the dimensions and

style of the chair are always the same.

Frameworks act as a structure where the developer

provides their own code

that the framework interacts with.

For example, there are

many frameworks for developing web applications.

These frameworks handle functionality

that is common to all web applications

such as receiving HTTP requests

and sending HTTP responses.

The developer then adds their own code that

implements the functionality of the web application.

For instance with the e-commerce website example,

a framework would handle receiving HTTP requests.

The developer would implement

code that processes the request and returns

a response from which the framework

would send a response over HTTP.

Now let's compare how the frameworks relate to libraries.

Most frameworks use many libraries.

The libraries that the framework uses

can be used for your application.

If you wish, your application

can also use other libraries.

You also need to consider when to use

a framework and when to use a library.

Frameworks are considered opinionated

and libraries are considered unopinionated.

This is defined as the degree of freedom

available to the developer to

choose how to code a feature.

The opinionatedness will vary between frameworks,

but by definition they will always

be more opinionated than a library.

The benefit of this is that they can

replace libraries as needed.

For example when new technologies become

available frameworks to find

the libraries flow and control of an application,

whereas with the libraries

those are left to the developer to decide.

As with everything there are

advantages and disadvantages to both.

Frameworks are a great way to reduce

development time and to enforce

a structure on how code is written.

They have many best practices already in place and

contain most of what is needed

to develop an application,

however, sometimes you may find that the way you need to

code something doesn't fit

into the structure of the framework.

Other times you may find that some of

the libraries the framework uses may

conflict with a library that you are required to

use and cause compatibility issues.

If an application is built without a framework,

the developer will need to decide on the set of libraries

they wish to use to achieve

the functionality they must deliver.

They will also need to take care that

the selected libraries can work together.

The upside to this is that they can

replace libraries as needed.

For example, if a new better library is released,

the developer can replace the usage of the old library.

This is much easier than replacing a framework.

Frameworks and libraries give you the opportunity

to reuse existing web app functions.

This can result in faster development, fewer errors,

and more time for you to spend on

the essential features of your application.

Instead of reinventing the wheel,

you can use frameworks and libraries that are designed

specifically to help your web app development processes.

**APIs and services**

Every day you access information on your phone, like reading the news, purchasing

goods and services or communicating with friends over social media.

But how is all this information transferred behind the scenes?

Your favorite websites and apps.

Probably use API's and as a web developer, you'll discover that API's developer

friendly, easily accessible and a very valuable and useful development tool.

A PI is the acronym for application programming interface.

An API is a set of functions and procedures for

creating applications that access the features or data of an operating system,

application or other service.

If this still sounds a bit vague, just remember that the term API,

is intentionally open too many applications and use cases.

As a web developer, a lot of the day to day job involves working with API's.

Some common API's that web developers work with include Browser,

API REST API and Sensor-Based API.

Over the next few minutes, you'll explore each of these API types and

review a few specific examples.

To begin with, here's a brief outline of how a piecewise functions.

In Software development,

API's are often the bridge between different components or systems.

This earns them names like gateway or middleware.

The term is used widely to represent many different tools and systems.

Let's consider some examples of different API use cases.

One common type of API, is Browser or Web APIs,

which are built into the browser itself.

They extend the functionality of the browser by adding new services and

are designed to simplify complex functions and provide easy syntax for

building advanced features.

A good example, is the DOM API.

The DOM API turns the html document into a tree of nodes that

are represented as JavaScript objects.

Another example, is the geolocation API that returns coordinates of where

the browser is located.

There are also other API's available for fetching data known as Fetch API drawing,

graphics or Canvas API keeping history or history API.

And client side storage also known as Web Storage API.

Another critical type of API for web development is the RESTful or REST API.

This kind of API provides data for popular web and mobile apps.

These are also called web servers.

Let's explore REST in a bit more detail.

REST or representational state transfer,

is a set of principles that help build highly efficient API's.

One of the main responsibilities of these kinds of API's is sending and

receiving data to and from a centralized database.

We can query our own REST API or third party ones.

One last type of API,

that you might encounter as a web developer is a Sensor-Based API.

This is what the internet of things also known as IOT is based on.

These are actual physical senses that are interconnected with each other.

The sensors can communicate through API and track and respond to physical data.

Some examples are Philips hue, smart lights and node bots.

That's a lot of API to think about.

Fortunately, for web developers the most common data API is a RESTtful API which

as you've learned is a web server that provides responses to requests.

These API web servers are designed to provide the data backbone for

a web client like a web page or mobile app.

This means that these API's must be able to accomplish things like getting data or

get, creating data.

Also referred to as post updating data or put and deleting data or delete.

API issues, REST principles and

good design practices to create discoverable interfaces.

This helps us get the exact response expected.

But exactly how do they work?

Here's a closer description of their activity.

These API's use endpoints to specify how different resources can be accessed.

The endpoint is built into the URL when accessing the API.

Once the endpoint is hit, the API performs whatever service side

processing is needed to build the response.

Two common forms of response are full web page is a data form based on

JavaScript called Jason.

In this video, you explored some API's and as a web developer,

you will frequently work with many different types of API's.

You will often use API's to extend the abilities of systems or

to act as a bridge between different components.

**What is a an IDE?**

Think of a group of construction workers,

every worker has a toolbox that helps them get their job done.

As a developer, you'll also use many tools.

One of the main tools in your toolbox is the integrated development

environment or IDE.

By the end of this video you'll be able to identify an IDE and

explain the benefits of using an IDE during development.

An integrated development environment or IDE is software for building applications.

An IDE is just like a text editor except instead of writing documents

you're writing code.

There are many IDEs available, some are specific to one programming

language while others support many languages in one IDE.

Let's explore some common IDE features.

Here I am working within an IDE.

First let's cover syntax highlighting.

To improve readability for developers, IEDs have syntax highlighting.

What this means, is that special keywords of the programming language

are highlighted in different colors so that the developer can quickly

differentiate these keywords from other texts.

For example, if you're writing JavaScript code without syntax highlighting,

it could be harder to identify keywords from other texts.

With syntax highlighting, that gets much easier because the JavaScript keywords and

variables are colored differently.

Now, let's explore error highlighting.

Just like checking spelling in a text document,

IDEs can highlight mistakes you make in your programming code.

For example, if I delete the equal symbol where it's needed,

my IDE will highlight the error.

Another feature of IDEs is also complete.

When you're typing a message on your phone, it suggests words as you type.

An IDE's autocomplete is a similar feature.

Since programming languages have special keywords,

IDEs can offer suggestions to autocomplete words as you start typing them.

Additionally, another feature called IntelliSense can make IEDs very smart and

even able to understand your code.

They can detect variables and functions and

offer them as suggestions during autocomplete.

For example, if I have a JavaScript function named myFunction defined at

the top of the JavaScript file, then as soon as I start typing the letter m my

IDE suggest this function as an autocompletion.

Then there is refactoring.

Since IEDs understand your code, they can help you if you need to change it.

To demonstrate how refactoring works,

let's continue with the myFunction function that I defined a moment ago.

In the code, the function is then called multiple times.

It can also be called in the code of other files too.

But what if you need to rename this function?

You would need to rename it in every file that uses the function ensuring that you

update those files to use the new name.

This process is known as refactoring,

changing the structure of the code without changing the functionality.

Doing this manually is very time consuming and prone to error.

If you mistyped the new function name in one place, the application will break.

Since the IDE understand your code, it can assist with refactoring and

automatically update the function name across all files.

That saves a lot of time.

Let's rename our function now.

I just right click on the function and select rename symbol.

Then I change it from myFunction to ourFunction.

The IED then updates all references of that function name.

IDEs come with a lot of other features to help investigate bugs and

collaborate with other developers.

Many even allow you to extend their functionality using plugins and

extensions, but that's beyond the scope of this lesson.

We have explored some features of IDEs in this video.

You now know how IDEs operate as part of the developers toolbox to write code more

effectively.

Well done.

**What is Hyper Text Markup Language?**

When a team of builders begin constructing a building.

One of their key tasks is to assemble the frame.

It doesn't matter what kind of building it is,

it always starts with a frame of beams and rafters.

Once in place, the frame

guides the rest of the construction.

If you compare a building with a web page

an HTML document is a lot like the frame.

It's a basic structure assembled with

different components such as tags and elements.

In this video, you will learn about

HTML documents and the difference

between HTML tags and elements.

To begin, let me tell you

a short story about where it all started.

Sir Tim Berners-Lee, a physicist,

often thought about how he and his colleagues

from around the world would share information.

He was so used to browsing information

online that it might seem a silly thing to ponder about.

Actually, that question was very relevant at

the time and started the development of HTML.

Would you like to explore the very first web page?

I know just where to find it.

The first web page was created in

1999 by Sir Tim Berners-Lee.

He worked at the European Organization for

Nuclear Research or CERN,

and originally proposed HTML.

The first version of HTML was released in

1991 along with the first web browser and web server.

HTML stands for Hypertext Markup Language.

But what does that mean exactly?

Hypertext is text which contains links to other text.

Markup refers to tags and

elements used within a document.

For now, let's just focus on some of the basics of HTML.

HTML is simply a text file with

a specific structure that consists of elements and tags.

Also take note that HTML files

usually have a dot HTML suffix.

For instance, when you visit a website,

the first page that is returned to the browser

is often called index.html.

Now, let's explore what HTML tags and elements are.

Each HTML element consists of

an opening tag enclosed in angle brackets.

For example, to create a paragraph,

you type <p> for paragraph.

Most elements are paired with a closing tag,

which has a forward slash off to the left angle bracket.

For example, you close the paragraph element with </p>.

HTML elements usually have some content inside them.

For example, between the opening

and closing tags of a paragraph,

you add the text of the paragraph you want to write.

HTML elements can also contain other elements.

For example, you can add an italics element

inside a paragraph element

to make texts appear in italics.

But elements can also be empty or self-closing,

meaning they do not have a closing HTML tag.

One example of a self-closing element

is the line break tag.

You can add a line break tag in a paragraph tag to

move content to the following line by typing <br>.

At the end of a self-closing tag,

you simply add a right angle bracket.

You can also close the right angle bracket

by typing a forward slash right before it.

Now that you know what elements and tags are,

you will explore what HTML standards are.

The rules and structure for elements and tags

are known as the HTML specification.

The HTML specification is

maintained by the World Wide Web Consortium,

or W3C as it is commonly known.

Whenever the HTML specification changes

a new version of HTML is standardized,

the current version is HTML5.

To summarize, HTML elements with

their opening and closing tags in

angle brackets build up an HTML document.

These elements form the structure of a web page

and describe to the web browser what to display.

For example, the browser reads an HTML page that has

an image tags display an image file called icon.png.

Next, it reads a p tag

to display a paragraph under that image.

But the browser reads the HTML document and

displays the web page to the user in a very basic format.

Now, if you want to tell

the browser how to display the web page,

you should use CSS.

You will learn more about CSS in this course.

By now, you have learned how HTML tags are used

to create elements that

build the structure of a web page.

You'll have an opportunity to explore

HTML in action in this course.

**HTML documents**

In this video, I will guide you to

the blueprints of a HTML document.

I will create a very simple webpage

for a restaurant called Little Lemon.

The great thing about HTML documents

is that you don't need the web to view them.

Yes, they don't have to be hosted on

a web server for you to view them in your web browser.

You can save HTML files on

your computer and viewed them

locally with your web browser,

almost like eating takeaways at home.

By the end of this video, you'll be

able to identify the structure

of a HTML document and create a basic webpage,.

To create a HTML file and Visual Studio Code.

Right-click in the Explorer panel and select New File.

I'll name it index.html.

By now you know that HTML documents

are made up of elements and tags.

But before I can add elements or tags,

I first need to create

a standard HTML structure

which starts with the DOCTYPE declaration.

This notifies the web browser that is a HTML documents.

Next, I create the HTML tag,

which is also known as the HTML root element.

Inside the HTML tag,

I add two main elements.

The head and the body elements.

Is important to note that nothing inside

the head element is displayed

on the webpage in the web browser.

In the head element, you add information or

metadata about the HTML documents.

It is not part of the content of the webpage.

For example, inside the head tag,

you always create the title elements.

This is the title that is displayed

in the web browser tab.

I can also link to CSS files and

define Meta tags in the head section.

Meta tags can, for instance,

specify the description of the web page,

keywords for search engines

and the order of the web page.

Next up, I add the content of

the webpage inside the body tag.

The body can contain elements such as headings,

paragraphs, images, and videos.

Let me start by adding a main heading on

the webpage by using the H1 tag.

Inside the H1 tag,

I typed the heading, Our Menu.

That's good. Now I need to start creating my menu.

I want to add two items.

To do that, I add two subheadings.

For this, I use the H2 tag.

The menu items in the subheadings are

falafel and pasta salad.

In order to make sure everything is correct,

I open my index.html

file in the browser to check my webpage.

But first I need to save the file by pressing

the keys Control and S or if you're on a Mac,

Command and S. Now,

I right-click on my index.html file in

the explorer panel and select Reveal in File Explorer.

If you're not using Windows,

this may be called revealing

finder or reveal in file browser.

Now the file browser opens and lists the file.

I double-click on index.html

and it opens in the web browser.

Great, it's all there.

The title that displays in the browser tab

matches the title that I added to the head section

and the content is displayed in

the browser window but it is still a bit empty.

I go back to Visual Studio Code

and add a short description for each dish.

I do this by adding

a paragraph tag below each subheading H2.

Let's imagine a nice falafel.

I type chickpea, herbs, and spices.

What goes into the pasta salad again?

Lettuce, vegetables, and mozzarella.

Yummy. I save the file again to update the changes.

Again, I open it in my web browser.

There you go. Now the ingredients

are also included in the menu.

Some of the best websites are websites about food.

Whenever you need some inspiration,

search for food website ideas online,

you will be fascinators.

Now that you know some basic HTML tags,

try to identify where they were using those websites.

There are many HTML tags

that you can use in your webpages.

But now you're familiar with

the few elementary ones and I

encourage you to start practicing with

them. Best of luck.

**Simple HTML tags**

There are many tags available in HTML. Here you will learn about common tags that you'll use as a developer.

**Headings**

Headings allow you to display titles and subtitles on your webpage.

<body>

  <h1>Heading 1</h1>

  <h2>Heading 2</h2>

  <h3>Heading 3</h3>

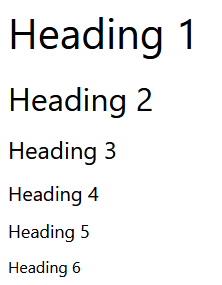
  <h4>Heading 4</h4>

  <h5>Heading 5</h5>

  <h6>Heading 6</h6>

</body>

The following displays in the web browser:



**Paragraphs**

Paragraphs contain text content.

<p>

   This paragraph

   contains a lot of lines

   but they are ignored.

</p>

The following displays in the web browser:

Paragraph style displayed in browser 

**Note** that putting content on a new line is ignored by the web browser.

**Line Breaks**

As you've learned, line breaks in the paragraph tag line are ignored by HTML. Instead, they must be specified using the *<br>* tag. The *<br>* tag does not need a closing tag.

<p>

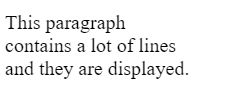
   This paragraph<br>

   contains a lot of lines<br>

   and they are displayed.

</p>

The following displays in the web browser:



**Strong**

Strong tags can be used to indicate that a range of text has importance.

<p>

   No matter how much the dog barks: <strong>don't feed him chocolate</strong>.

</p>

The following displays in the web browser:

Text with strong tag displayed in browser 

**Bold**

Bold tags can be used to draw the reader's attention to a range of text.

<p>

   The primary colors are <b>red</b>, <b>yellow</b> and <b>blue</b>.

</p>

The following displays in the web browser:

Bold text displayed in browser 

Bold tags should be used to draw attention but not to indicate that something is more important. Consider the following example:

The three core technologies of the Internet are <b>HTML</b>, <b>CSS</b> and <b>Javascript</b>.

The following displays in the web browser:

Bold text displayed in browser 

**Emphasis**

Emphasis tags can be used to add emphasis to text.

<p>

   Wake up <em>now</em>!

</p>

The following displays in the web browser:

Text with emphasis tag displayed in browser 

**Italics**

Italics tags can be used to offset a range of text.

<p>

   The term <i>HTML</i> stands for HyperText Markup Language.

</p>

The following displays in the web browser:

Italic text displayed in browser 

**Emphasis vs. Italics**

By default both tags will have the same visual effect in the web browser. The only difference is the meaning.

Emphasis tags stress the text contained in them. Let's explore the following example:

I <em>really</em> want ice cream.

The following displays in the web browser:

Text with emphasis tag displayed in browser. 

Italics represent off-set text and should be used for technical terms, titles, a thought or a phrase from another language, for example:

My favourite book is <i>Dracula</i>.

The following displays in the web browser:

Italic text displayed in browser

Screen readers will not announce any difference if an *italics* tag is used.

**Lists**

You can add lists to your web pages. There are two types of lists in HTML.

Lists can be unordered using the *<ul>* tag. List items are specified using the *<li>* tag, for example:

<ul>

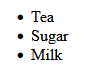
   <li>Tea</li>

   <li>Sugar</li>

   <li>Milk</li>

</ul>

This displays in the web browser as:



Lists can also be ordered using the *<ol>* tag. Again, list items are specified using the *<li>* tag.

<ol>

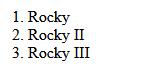
   <li>Rocky</li>

   <li>Rocky II</li>

   <li>Rocky III</li>

</ol>

This displays as the following in the web browser.



**Div tags**

A *<div>* tag defines a content division in a HTML document. It acts as a generic container and has no effect on the content unless it is styled by CSS.

The following example shows a *<div>* element that contains a paragraph element:

<div>

   <p>This is a paragraph inside a div</p>

</div>

This displays as the following in the web browser.

Div displayed in browser img12

It can be nested inside other elements, for example:

<div>

   <div>

      <p>This is a paragraph inside a div that’s inside another div</p>

   </div>

</div>

This displays in the web browser as:

Div inside a dive displayed in browser 

As mentioned, the div has no impact on content unless it is styled by CSS. Let’s add a small CSS rule that styles all divs on the page.

Don't worry about the meaning of the CSS just yet, you'll explore CSS further in a later lesson. In summary, you're applying a rule that adds a border and some visual spacing to the element.

<style>

   div {

      border: 1px solid black;

      padding: 2px;

   }

</style>

<div>

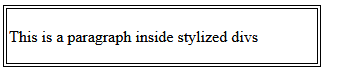
   <div>

      <p>This is a paragraph inside stylized divs</p>

   </div>

</div>

This displays in the web browser as:



Div elements are an important part of building webpages. More advanced usage of div elements will be explored in another course.

**Comments**

If you want to leave a comment in the code for other developers, it can be added as:

*<!-- This is a comment -->*

The comment will not be displayed in the web browser.

**Linking documents**

By now you should know how to build a basic webpage but

how do you build a website?

Remember, websites consists of multiple webpages linked together.

To link pages together.

You use the anchor tag anchor tag create hyperlinks or

links as they are commonly known.

In this video,

you will learn how to link different web pages together to create your own website.

In my project folder.

I now create a second webpage for

Little Lemon that will contain the address of the restaurants.

I right click in the explorer panel to create a new file and

name it location.html.

I use the same file structure created in my index to

HTML email but add new content.

I add a new H one tag for the heading and type our location.

Next I add a paragraph tag and insert the address

123 Rome Road, Main Districts capital city.

I press control and S to save the file or command and S on Mac.

Now that my new file with the restaurant location is created.

Let's navigate back to index.html to add a link from there to location.html.

Below the last paragraph tag I add the anchor or

a tag followed by the href tribute or hypertext reference.

I typed the file name, location.html so that my index.html file will link to it.

Within a tag I can also add a descriptive text that is

displayed as a link name on my web page.

So I type our location.

The text our location will now display on the web page in the browser as

a clickable link.

Let's test see if it works.

I saved the file to update it with the changes and open it in my web browser by

right clicking on the index.html file and selecting reveal in file Explorer.

If you're not on Windows, this may be called reveal in finder or

revealing file browser.

I then double click on the index file.

The words are location are displayed in blue by the web browser, success.

If I click on the blue link, the browser opens the location.html file.

You can now link different HTML files together to create your website.

Best of luck.

**Adding images to a webpage with HTML**

A picture is worth 1000 words as they say.

I think you will agree that a website without pictures wouldn't provide a very

good user experience.

Especially websites about food.

My example website about Little Lemon could certainly use a few images.

In this video you will learn how to add images to HTML documents using the image

or IMG tag as it is commonly known.

I have to image files in my project folder,

one name falafel.jpeg and another named salad.jpeg.

Similar to what I did with document links,

I have to specify the link to the image file using the image tag.

Images are not inserted into web pages.

Instead we use image tags in HTML to link to image files.

The image tag then creates a placeholder for the image on the web page.

To add an image tag, I type img and inside it I add

the src attribute or source attribute as it is known.

I type falafel.jpeg As a source for

one image, And salad.jpeg,

As a source for the other image.

Let's save the file and open it in the web browser.

The images are displayed but they are way too big.

Luckily there are several ways to set the size of an image but

in this video I will just demonstrate how to do this in HTML.

Right after the sourcing in the image tag,

you can specify the dimensions of an image.

I first add the width attribute, And set it to 240 pixels.

Then I add the height attribute and

set it to 135 pixels.

I'll set the other image to the same dimensions.

Let's save and view it in the browser again.

Much better, the images are smaller now.

But there is one important attribute that you should always include with image tags.

It is good practice to add a short description for images.

It helps improve accessibility for people using assistive technologies and

it can also improve search engine rankings.

You can add image descriptions with the alternative text attribute also known as

the altar tribute.

Still within the image tag after the dimensions attributes you can add

the altar tribute.

I will type a short description, A falafel and for

the other image I will type, A pasta salad.

I will save the Html file again and open it in the web browser.

The all text is not displayed anywhere on the site but

will be read by assistive technologies such as screen readers.

Screen readers and other accessibility tools use the altar tribute to provide

information to their users.

Now you know how to add images to websites with the image tag.

Little Lemon website is now ready for everyone who uses the web.

Have fun while you make your sites colorful by adding images

**Use HTML to work with data in tables**

My website for the lemon is coming on well,

but some key information is still missing. The prices.

The best way to add information like

a price list is by using an HTML table.

You might be familiar with columns and

rows in spreadsheets software.

HTML tables look similar and allow you to

neatly organize content in rows and columns.

Websites almost always have information that needs to

be organized in a way that makes

it quick and easy to read.

For example, the schedule of an event,

available sizes of clothing items,

or the specs of a camera.

In this video, I will demonstrate how to create

a symbol table for the prices of little lemon.

I want to add a table with the prices to

the bottom of the index.html page.

So I already have it open.

Still in the body elements below the anchor elements,

I add the table tag.

Inside of it I add to table row tags or tea or tags,

as they're commonly referred to.

This will create a row for each of the dishes.

Now I need to add table data tags or

td tags so that I can include the prices.

Table data tags define the contents of table cells.

So I need to add two of those inside each row,

one for the dish and one for its price.

In the first row, I type

Falafel inside the first table data tag.

In the second table data tag,

I add the price, let's say $10.

I do the same with the second row.

I type Pasta Salad inside the first table data tag,

and a price of $12 inside the second table data tag.

Now I will save the file and open it in my web browser.

Great. The prices display at the bottom of the webpage.

However, the table is not

displayed like a regular table at the moment,

but I will style it soon.

To make it more clear for my users,

I'm going to add headers to the columns now.

To do this, I add a new row to

the top of the table with the table row tag.

Then use the table header tag or

th tag for the heading cells.

I need to add two table header tags.

I type dish in the first column heading,

and price in the second heading.

I save the HTML and refresh it

again to check the update in the browser.

The information that's presented much

clearer now that the columns have headings,

but it is not displayed like a table, yet.

I will add a bit of CSS to

improve the styling of the table.

Don't worry about the details just yet.

But I would like to add a little bit of magic

now to get you excited about styling.

I will add a style tag.

I will add a one pixel border to the table.

Let's save the file again and open it in the browser.

Now the table has a border around all the cells.

This is just a quick method of styling a table.

Later you will learn how to do this

using best practices with CSS.

Congratulations. You now know how to

add tables to HTML files.

I use tables extensively and I am sure you will too.

**What are forms?**

Today you can order almost anything directly to your home.

But did you know that one of the functionalities that makes online shopping

possible is HTML forms.

Without them,

you would not be able to enter your credit card details during checkout.

But forms are not limited to e commerce.

You can use forms practically every day on the internet.

When you log into your favorite website, you do it using a form.

In this video, I'll take you through the steps to create a form and

introduce you to a few different input types.

But first, let's briefly explore how forms work.

To put it simply when a user enters data on a website an HTML form

submits an HTTP request containing the data to the server.

But how do you create a form in the first place?

You define forms with the html form tags.

Forms also have an optional form attribute called action.

Actions specifies the URL or path that the form should submit the request to.

When the action attribute is not specified,

it submits the request to the same path as the current web page.

There is also the FORM method,

with the FORM method you can specify the HTTP method to use for the HTTP request.

There are two HTTP methods to submit the form data, GET and POST.

Remember the GET HTTP method retrieves information from the server.

The POST HTTP method sends data to the server.

When a user submits a form, one of these HTTP methods is used.

We'll explore how this is done on a later course.

Now let's explore how to add fields that the user can enter data into.

First, let's add a simple text field.

For example, a user name field.

The fields in a form are specified by input tags.

Note that the input tag does not need a closing tag.

But if I want to,

I could close the tag by adding a forward slash before the end of the input tag.

When I use the text input type, it displays a text field on the web page.

But a text box on its own isn't very user friendly, so

I will now add a label above it.

By adding the label tag,

the form will now display the word user name above the input field.

Next, let's add a password field.

In this case I won't use the text input type since I don't want the password to be

visible.

HTML has an input type specific for

passwords that will mask the user's input data.

The user name will be visible and the password will be masked.

Finally, I need to add a button so the user can submit the form.

So I used the input type, submit.

When the user clicks the submit button,

the HTTP request containing the form content will be sent to the web server.

In summary, you use the form tag at the start of a form where after you indicate

the method that should be used.

You can create labels for input types with the label tag,

you use the text input type for a text input like a user name.

For a password, you use the password input type.

To add a submit button to a form, you use the submit input type.

There are many different input types available.

For example, you can add checkboxes to a form.

To do this, you use the checkbooks input type.

Each box can be checked or unchecked.

You use the name and value attributes to configure how the data is sent to

the server, you will explore this in more depth in a later course.

Radio buttons on the other hand are like checkboxes except

only one button can be checked in on the group.

Checking one radio button will uncheck all the other radio buttons.

There are also other types of inputs such as the number, email and

file upload types.

However, some input fields do not use the input tags.

For instance the multi line text field.

The text input type that I showed you earlier is only for

single line text content.

To allow users to enter multiple lines of text,

I need to use the text area tag instead of text input type.

Another type of input that does not use the input tag is a drop down list which

contains items that the user can select.

This time, you use the select tag and for the items on the list you use option tags.

You now know how HTML forms work and the different input types, developers use.

You covered form tags such as text, password, checkbox,

radio, text area and select.

I can't imagine my life without online shopping.

Next time you order something online, notice all the types of input the form

includes and how you, as a user interacts with it.

You might notice that the user experience of some forms is better than others.

I'm sure that you are now able to choose appropriate input types for

the forms that you create.

**Introduction to the DOM**

Imagine your favorite social media site.

If it was just an HTML document, users would be able to scroll the page,

look at pictures and read text but they wouldn't be able to log in like posts or

get notifications for new messages.

In order to allow users to do these things, they need to be able to interact

with objects on the page such as the reaction icons or the comment button.

Let me put it differently, an HTML document must be represented in

a certain way, so that JavaScript code can query and update it,

to do this we use the document object model.

In this video, I will unpack what the DOM is, how it is created and

how JavaScript interacts with the DOM to make sites dynamic.

When your web browser receives an HTML page, it constructs a DOM to represent it.

DOM stands for Document Object Model and it is simply a tree, structure or

model of the objects in your HTML file.

To understand the DOM, I want you to think of a simple HTML page, remember that

an HTML document has opening and closing tags for the document and the head.

And inside the head tags you have title tags, then you have the body and

inside the body there might be elements such as div tags.

Okay, now, let me guide you through how a DOM would be generated,

the DOM has a series of objects each representing a single HTML element.

At the root of the DOM is the html object and it contains the head and body object.

From there, the head object houses the title object and

the title object contains its text object.

The body object contains the two div objects, the first div houses,

the h1 object which again houses its text object.

The second div object contains the paragraph object which contains its text

object.

In summary, all the elements in the HTML file are represented as objects in

the document object model.

I just took you through a very simple webpage,

webpages typically have hundreds of elements.

Can you imagine how complex the DOM of a highly interactive modern web page is?

You as a developer can use JavaScript to access and

modify the DOM to make your webpages dynamic.

In fact, in a later course you will learn how to access and

modify the structure of the DOM and its elements in different ways.

Now that you know what the DOM is and how it is referenced,

let me briefly explain common DOOM and JavaScript uses.

For each element, you can access the HTML attributes and content, this means you can

update the existing content that is displayed in the web browser.

For instance, you could write some JavaScript to update the seconds,

minutes and hours of a digital clock on a website.

Or if you develop a movie streaming website, you can write code that responds

to user actions such as clicking mouse over scrolling and so on.

In this way you can allow a preview of the movie to play when

the user hovers the mouse over the movie's poster.

Or you can think of a log in page, when users click the Login button,

you can disable the button so that it cannot be clicked again.

The web application would bring them to a new web page or

would re-enable the button, if a log in error occurred.

You can even add DOM objects to dynamically add new HTML content to a live

web page.

For instance, you can add an error message to a form if the user inserts invalid

data, DOM objects can also be deleted which will

remove the corresponding HTML displayed in the browser.

Think of a to-do list on a website,

you can remove an item from the list when the user clicks on it.

Another major use of JavaScript and the DOM is to animate the HTML elements.

This can be quite complex depending on the animation but

there are many libraries available that aim to simplify this.

For example, when a page first loads, you can fade in the page contents or

when a user receives an instant message,

a notification can pop up in the web browser.

Many JavaScript libraries and frameworks rely on the DOM,

one of these libraries is the react library.

By now, you know what the DOM is, how it is constructed and the wonderful things

web developers can do with it to create modern interactive websites.

Next time you use your favorite social media site think about how the DOM is

working behind the scenes to change the color of that like button.

**Web accessibility**

as a web developer you should try to ensure that everyone can access and

enjoy the internet almost like an architect that needs to design

a building with ramps and rails so that it is accessible by all.

You need to build a website in a certain way to improve accessibility.

When the Web accessibility Initiative or W.

A. I.

was launched in 1997.

The creator of the World Wide Web.

Sir, tim Berners lee said the power of the Web is in its universality access

by everyone regardless of disability is an essential aspect.

Web accessibility aims to allow people with

disabilities to understand navigate and interact with websites.

A common misunderstanding is that it refers to visual disabilities but

in fact it includes all disabilities that affect interaction with websites

such as audio and visual disabilities, cognitive and

neurological disabilities and physical and speech disabilities.

The W three C.

Web accessibility initiative developed specifications and

supporting resources for accessibility.

These are considered international standards for web accessibility.

In fact web accessibility is becoming a requirement in many industries In 2016.

The European Union approved the web accessibility directive which requires

websites and mobile applications of public section bodies to be accessible to those

with disabilities.

Let's explore some examples of how accessibility is provided to those with

different disabilities.

People living with disabilities often use assistive technologies to aid them in

browsing the web screen reader software can read the content of websites and

everything that is happening on the device.

Screen readers are used not only by blind and vision impaired users but

also by those with reading or learning difficulties.

Speech recognition software can turn spoken words into computer commands or

dictate inputting text.

This is useful for people who may not be able to use a mouse or

keyboard due to physical or neurological disabilities, subtitles and video

scripts and videos provide assistance to those with audio and visual disabilities.

There are many steps that you can take to improve accessibility.

It is good practice to think of accessibility from the beginning of

a project, it is a lot harder to rework the project to be accessible later on.

If you use the correct html structure and appropriate elements as a foundation, it

will greatly improve accessibility in this course you will learn to structure your

html properly and follow best practices to support assistive technologies for

example having text that is not contained within proper tags like paragraph or

heading tags makes it harder for assistive technologies to interact with the content.

Similarly using multiple line breaks to break up text and

add space also presents barriers to accessibility.

These are just simple examples to improve web accessibility.

However, in the future you'll probably develop very complex user interfaces and

to make them accessible may not be as straightforward

as just having correct html.

Fortunately one of the tasks of the W AI is to define the accessible rich internet

application or aria.

Specification.

The aria specification outlines different techniques to improve accessibility for

complex web apps.

Now, you know about different assistive technologies and that you should use

the correct html structure and appropriate elements to improve accessibility.

**Selecting and styling**

I'd like you to think of a physical building and its structure,

interior, exterior and decorations in a similar fashion.

When it comes to websites and web apps you can think of html as being the frame and

structure.

CSS, however, is the paint, wallpaper, fixtures, artwork and

overall style or look and feel.

In other words, CSS tells the web browser how to display html elements on screen.

When using CSS, there are five elements you need to understand.

A declaration block starts with a left curly bracket and

ends with the matching right curly bracket.

In between these curly brackets are the style declarations.

Let's cover these elements one by one.

The first element of the CSS rule is the selector.

This indicates which html element or elements we want to style.

For example, you can create a rule that uses the h1 selector to change the color

of all heading one tags in a web page to gray.

In this case, hi is the selector.

Color is property and gray is the value of the property.

Now let's focus on the structure of the declaration block.

Each declaration is made up of two parts a property and a value.

For example, in this case the properties that you want to change are the color and

the background color of the header or selector.

Once you have decided which properties you want to define,

then you must assign each property of value.

The color property gets a value of blue and

the background color property gets a value of gray.

This will result in all heading one tags displaying blue text on a gray background

in the browser.

By now, you have learned that CSS tells the web browser how to display

html elements on screen.

Now, let's work through an example to change a set of

heading elements using different CSS rules.

I will share a pro tip demonstrating how to install and

use the VS code extension live preview to speed up your workflow.

Okay, so I'm back in visual studio code here.

Did you know that you can add additional functionality by using extensions.

If you click on the extensions tab,

you can install the live preview extension by Microsoft.

If you are running VS code locally, you can install and use this extension.

Once installed, right click on your html file in the explorer panel and

select live preview, show preview.

When you update an html file,

you immediately notice the changes in visual studio code.

In this video, I will demonstrate how to create a CSS file and

style some html elements.

Before I start styling elements,

let's create a simple html document named index.html.

To demonstrate how CSS works, I will use an example created previously.

Next I create a file named style.CSS.

For my web page to use the CSS file,

I need to link it to the head element of the html file.

To link to a style sheet,

you use the link tag which must be assigned to attributes rel and h ref.

The rel attribute is assigned to style sheet and

the href attribute is assigned to the name of the style sheet file.

In this example, it's style.CSS.

I am now ready to apply CSS to the index.html file using style

rules that I will write in the style.CSS file.

The selector I'm creating applies the CSS rule to all h1 elements.

This type of selector is called an html type selector.

The properties in the declaration block allow me to change

how the selected elements are displayed.

For example,

I'm setting the color property of all the h1 elements to display text and purple.

I save the file using Ctrl nd S, notice that this change occurs immediately,

and there is no need to refresh the webpage.

So now what if I want to only style a single h1 element?

I can add an ID attribute to the tag h1 that I want to style.

And in my CSS file, I create a rule for that ID.

Let's do that now.

First, I create an ID rule using the hash symbol and then the name header one.

Then I define the CSS properties of that rule to change the color value to green.

I apply the rule and save the file using Ctrl and S.

Notice that the text for the heading, chapter one on the web page turns green.

You may be wondering,

why does the first CSS rule not apply to the h1 element with the ID?

This is because of something called CSS precedents and specificity.

This is a complex statement, but

basically the browser will use the most precise selector for an html element.

CSS has a set of hierarchy rules which dictate which rule will

apply to an element.

In this example,

the ID rule takes precedence over the html type selector rule.

You can learn more about CSS selector rules and

precedents from the additional reading at the end of this lesson.

Congratulations, you have now learned about the selection and styling in CSS.

Let's quickly recap.

After choosing a selector for your styles,

you create a declaration block using opening and closing curly brackets.

Inside of the code block, you write a declaration which consists of

a property value pair ending in a semi colon.

And all these parts together.

The selector code block and declaration make up a CSS rule.

I encourage you to add some CSS rules to your html

documents to practice selecting and styling.

**Different types of selectors**

When styling a web page, there are many types of selectors available that allow developers to be as broad or as specific as they need to be when selecting HTML elements to apply CSS rules to.

Here you will learn about some of the common CSS selectors that you will use as a developer.

**Element Selectors**

The element selector allows developers to select HTML elements based on their element type.

For example, if you use *p* as the selector, the rule will apply to all *p* elements on the webpage.

*HTML*

<p>Once upon a time...</p>

<p>In a hidden land...</p>

*CSS*

p {

  color: blue;

}​

**ID Selectors**

The ID selector uses the id attribute of an HTML element. Since the id is unique within a webpage, it allows the developer to select a specific element for styling. ID selectors are prefixed with a *#* character.

*HTML*

​<span id="latest">New!</span>

*CSS*

#latest {

  background-color: purple;

}​​

**Class Selectors**

Elements can also be selected based on the class attribute applied to them. The CSS rule has been applied to all elements with the specified class name. The class selector is prefixed with a *.* character.

In the following example, the CSS rule applies to both elements as they have the *navigation* CSS class applied to them.

*HTML*

​<a class="navigation">Go Back</a>

​<p class="navigation">Go Forward</p>

*CSS*

.navigation {

  margin: 2px;

}​​

**Element with Class Selector**

A more specific method for selecting HTML elements is by first selecting the HTML element, then selecting the CSS class or ID.

The example below selects all *p* elements that have the CSS class *introduction* applied to them.

*HTML*

<p class="introduction"></a>

*CSS*

p.introduction {

  margin: 2px;

}​​

**Descendant Selectors**

Descendant selectors are useful if you need to select HTML elements that are contained within another selector.

Let's explore an example.

You have the following HTML structure and CSS rule.

*HTML*

<div id="blog">

  <h1>Latest News</h1>

  <div>

    <h1>Today's Weather</h1>

    <p>The weather will be sunny</p>

  </div>

  <p>Subscribe for more news</p>

</div>

<div>

  <h1>Archives</h1>

</div>

*CSS*

#blog h1​ {

  color: blue;

}

* The CSS rule will select all *h1* elements that are contained within the element with the ID *blog*. The CSS rule will not apply to the *h1* element containing the text *Archives*.
* The structure of a descendant selector is a CSS selector, followed by a single space character, followed by another CSS selector.
* Multiple descendants can also be selected. For example, to select all *h1* elements that are descendants of *div* elements which are descendants of the *blog* element, the selector is specified as follows.

*CSS*

#blog div h1​ {

  color: blue;

}

**Child Selectors**

Child selectors are more specific than descendant selectors. They only select elements that are immediate descendants (children) of a selector (the parent).

For example, you have the following HTML structure:

*HTML*

<div id="blog">

  <h1>Latest News</h1>

  <div>

    <h1>Today's Weather</h1>

    <p>The weather will be sunny</p>

  </div>

  <p>Subscribe for more news</p>

</div>

If you wanted to style the *h1* element containing the text *Latest News*, you can use the following child selector:

*CSS*

#blog > h1​ {

  color: blue;

}

This will select the element with the ID *blog* (the parent), then it will select all *h1* elements that are contained directly in that element (the children). The structure of the child selector is a CSS selector followed by the child combinator symbol *>* followed by another CSS selector.

**Note** that this will not go beyond a single depth level. Therefore, the CSS rule will **not** be applied to the *h1* element containing the text *Today's Weather*.

**:hover Pseudo-Class**

A special keyword called a pseudo-class allows developers to select elements based on their state. Don't worry too much about what that means right now. For now, let's look at how the hover pseudo-class allows you to style an element when the mouse cursor hovers over the element.

The simplest example of this is changing the color of a hyperlink when it is hovered over. To do this, you add the *:hover* pseudo-class to the end of the selector. The following example will change the color of the hyperlink to orange when it is hovered over.

*CSS*

:hover {

  color: orange;

}

This pseudo-class is very useful for creating visual effects based on user interaction.

**Other Selectors**

There are many other CSS selectors available to style your webpage.

**Text and color in CSS**

As you design websites, you'll be working a lot with colors and text. There are many different ways to display text and equally many ways to define colors.

This reading covers how text and color work in CSS.

**Color**

Colors are used in many CSS properties, for example:

p {

  color: blue;

}​

From CSS Version 3, there are five main ways to reference a color.

* By RGB value,
* By RGBA value,
* By HSL value,
* By hex value and
* By predefined color names.

**RGB value**

* RGB is a color model that adds the colors red (R), green (G) and blue (B) together to create colors. This is based on how the human eye sees colors.
* Each value is defined as a number between *0* and *255*, representing the intensity of that color.
* For example, the color red would have the RGB value of *255,0,0* since the intensity of the red color would be 100% while blue and green would be 0%.
* The color black then would be *0,0,0* and the color white *255,255,255*.

When using RGB values in CSS, they can be defined using the *rgb* keyword:

p {

  color: rgb(255, 0, 0);

}​

**RGBA value**

RGBA is an extension of RGB that add an alpha (A) channel. The alpha channel represents the opacity, or transparency, of the color.

Similar to RGB, this is specified in CSS using the *rgba* keyword:

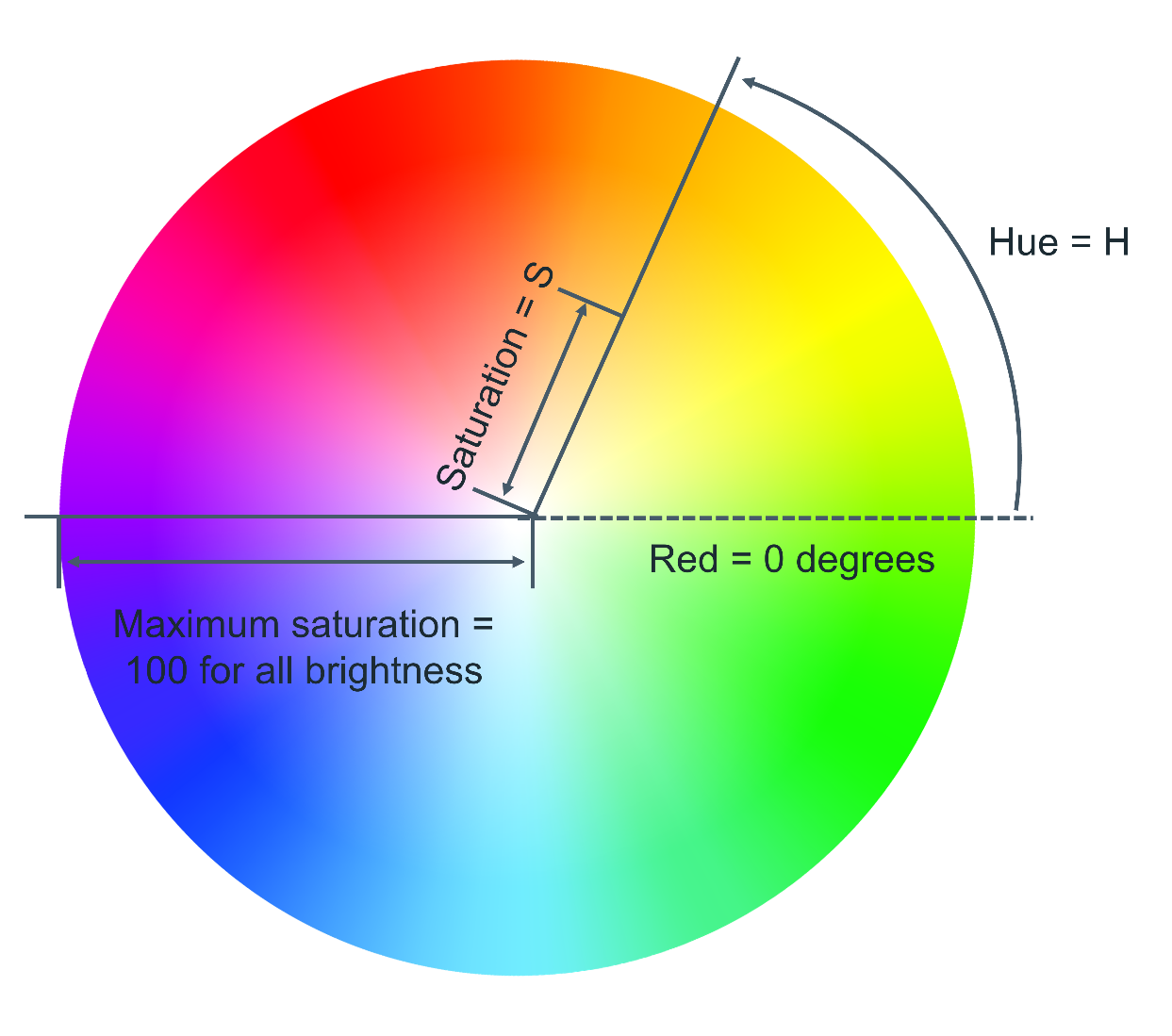
p {

  color: rgba(255, 0, 0, 128);

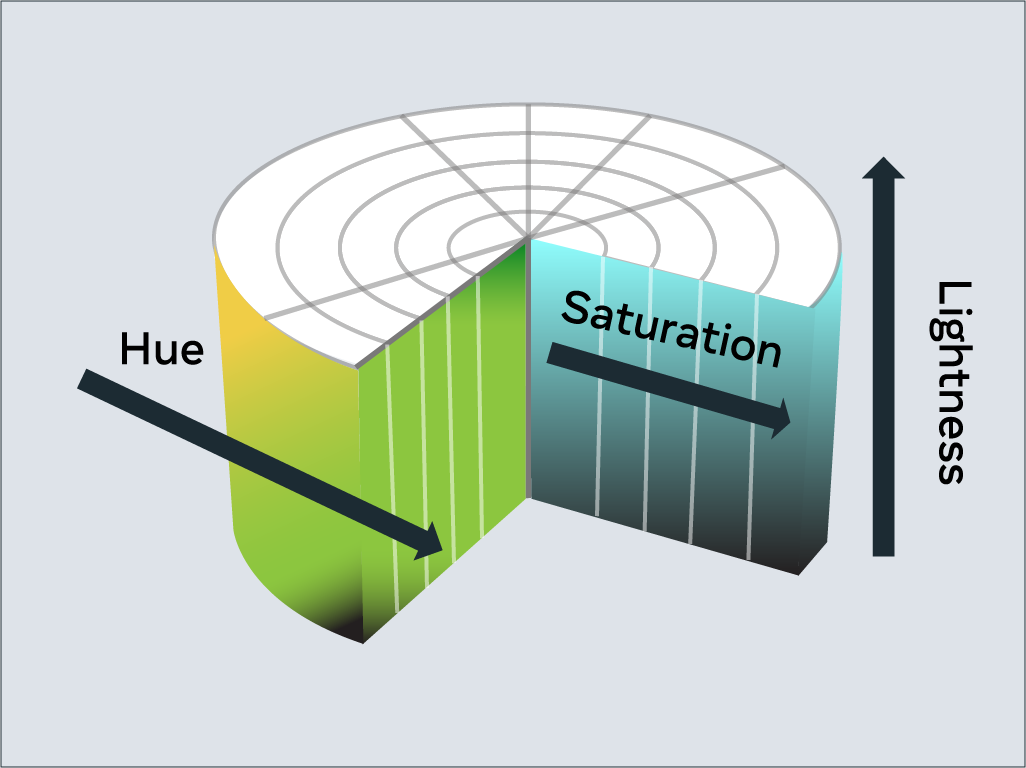
}​

**HSL value**

* HSL is a newer color model defined as Hue (H), Saturation (S) and Lightness (L). The aim of the model is to simplify mental visualization of the color that the value represents.
* Think of a rainbow that has been turned into a full circle. This represents the Hue. The Hue value is the degree value on this circle, from 0 degrees to 360 degrees. 0 is red, 120 is green and 240 is blue.



* Saturation is the distance from the center of the circle to its edge. The saturation value is represented by a percentage from 0% to 100% where 0% is the center of the circle and 100% is its edge. For example, 0% will mean that the color is more grey and 100% represents the full color.
* Lightness is the third element of this color model. Think of it as turning the circle into a 3D cylinder where the bottom of the cylinder is more black and toward the top is more white. Therefore, lightness is the distance from the bottom of the cylinder to the top. Again, lightness is represented by a percentage from 0% to 100% where 0% is the bottom of the cylinder and 100% is its top. In other words, 0% will mean that the color is more black and 100% is white.



In CSS, you use the *hsl* keyword to define a color with HSL.

p {

  color: hsl(0, 100%, 50%);

}​

**Hex value**

* Colors can be specified using a hexadecimal value. If you're unfamiliar with hexadecimal, think of it as a different number set.
* Decimal is what you use every day. Digits range from *0* to *9* before tens and hundreds are used.
* Hexadecimal is similar, except it has 16 digits. This is counted as *0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F*.
* In fact, you can convert between decimal and hexadecimal. Decimal *10* is equal to hexadecimal *A*. Hexadecimal *F* is equal to decimal *15*.
* Hexadecimal can also go to tens and hundreds. For example, decimal *16* is equal to hexadecimal *10*, with *10* being the next number after *F*.
* It can be a little confusing at first but don't worry, there are plenty of converters available if you get stuck.
* Colors specified using hexadecimal are prefixed with a *#* symbol followed by the RGB value in hexadecimal format.
* For example, the color red which is RGB *255,0,0* would be written as hexadecimal *#FF0000*.
* Again don't worry if you get stuck, there are plenty of converters available for this too!

**Predefined color names**

Modern web browsers support 140 predefined color names. These color names are for convenience purposes and can be mapped to equivalent hex/RGB/HSL values.

Some common color names available are listed below.

* black
* silver
* gray
* white
* maroon
* red
* purple
* fuchsia
* green
* lime
* olive
* yellow
* navy
* blue
* teal
* aqua

**Text**

With CSS there are many ways to change how text is displayed. In this section, you'll learn the most common text manipulation CSS properties.

**Text Color**

The *color* property sets the color of text. The following CSS sets the text color for all paragraph elements to red.

p {

  color: red;

}​

**Text Font and Size**

* There are many different fonts to display text on your computer. In simple terms, a font is a collection of text characters written in a specific style and size.
* If you've used a word processor before, you're probably familiar with the fonts Times New Roman and Calibri.

To set the font used by text in CSS you use the *font-family* property.

p {

  font-family: "Courier New", monospace;

}​

Since computers vary in what fonts they have installed, it is recommended to include several fonts when using the *font-family* property. These are specified in a fallback order, meaning that if the first font is not available, it will check for the second font. If the second font is not available, then it will check for the third font and so on. If none of the fonts are available, it will use the browser's default font.

To set the size of the font, the *font-size* property is used.

p {

  font-family: "Courier New", monospace;

  font-size: 12px;

}​

**Text Transformation**

Text transformation is useful if you want to ensure the correct capitalization of the text content. In the example below, the CSS rule will change all text in paragraph elements to uppercase using the *text-transform* property:

p {

  text-transform: uppercase;

}​

The most commonly used values for the *text-transform* property are: *uppercase*, *lowercase*, *capitalize* and *none*. The default value used is *none,* which means the text displays as it was written in the HTML document.

**Text Decoration**

The *text-decoration* property is useful to apply additional decoration to text such as underlining and line-through (strikethrough).

p {

  text-decoration: underline;

}​

It is possible to set the color, thickness and styling of the decoration too. In the example below, the underline will be a solid red line that is 5 pixels thick.

p {

  text-decoration: underline red solid 5px;

}​

If this is confusing, don't worry. These properties can be individually set using the *text-decoration-line*, *text-decoration-color*, *text-decoration-style* and *text-decoration-thickness* properties. Let's use the same example again and define it using the individual properties:

p {

  text-decoration-line: underline;

  text-decoration-color: red;

  text-decoration-style: solid;

  text-decoration-thickness: 5px;

}​

The most common *text-decoration-line* values used are: *underline*, *overline*, *line-through* and *none*. None is the default value to use no text decoration.

There are many styles available for the *text-decoration style* property; *solid*, *double*, *dotted*, *dashed* and *wavy*. The *text-decoration-style* property requires the decoration line to be defined. If the decoration style is not specified, *solid* will be used.

**Box model introduction**

Just like a busy day at a restaurant,

a website can easily look squashed and busy,

with waiters having to maneuver their way around and

customers chairs up against each other, it can get messy.

Good restaurants usually have

carefully planned layouts to make

customers feel comfortable and

allow waiters to move around with ease.

Sometimes creative web development

requires thinking inside the box.

In this video, I will demonstrate

how to use the box model in CSS

to create an appealing layout and allow

users eyes to easily flow between content.

When an HTML document and CSS style sheet are downloaded,

the web browser needs to know

how to display the elements on the screen.

To do this, it allocates

a rectangle or box to each element.

CSS rules are applied to the boxes of the elements.

This is known as the box model.

Every box consists of four parts.

The content, the padding,

the border, and finally, the margin.

The content is the actual content of the element,

like the text or the image.

Its size is known as

the content width and content height.

Browsers are clever and by default,

they will calculate the width and

height based on the content itself.

Fortunately, developers can

control the size to manipulate the layout.

You can do this by applying

CSS rules to the content, such as width,

minimum width, maximum width,

height, minimum height, and maximum height.

Let's move on to the padding.

The padding extends the content size.

Its size is known as

the padding box width and the padding box height.

The thickness of the padding is

determined by the padding top,

padding bottom, padding left and padding right.

The padding box width can be calculated as

content width plus padding

left side plus padding right side.

You can calculate the padding box height

by adding the content height,

the padding top side and padding bottom side together.

Next, let me show you how you set the border.

The border goes around the padding and content.

Its size is known as

the border box width and border-box height.

You can set different types of borders,

such as solid border or a dash border.

To understand how this plays out practically,

I will demonstrate the HTML,

the CSS style, and the physical border.

To set the border, you can write border width thin,

border width medium, and border width thick.

Now, let me share with you

a pro developer tip to help you

calculate the size of

your HTML elements when working with the box model.

An HTML element is equal to the size of the border box.

To get the border box width,

you add the padding box width plus

the border left side plus the border right side.

You can calculate the border-box height by adding

the padding box height and the border top side,

as well as the border bottom side.

Lastly, let me show you how to set the margin.

Following on from what you have just learned,

do you have an idea of what the CSS rule might be?

The margin extends the border area to

separate the element from its neighboring elements.

Its size is known as the margin box width,

a margin box height.

To specify the margin sides,

you write, margin top,

followed by the value, the margin bottom,

margin-left, and margin-right follow

the exact same notation.

We can also set the padding,

border and margin with shorthand properties,

but more on this later.

Lastly, let's calculate the size of the margin box.

Can you guess how? The margin box width

can be calculated by adding the border box width,

the margin left side and the margin right-side together.

To get the margin box height at the border box height,

margin top side and margin bottom side together.

To easily remember this,

think of the content as yourself.

The padding is like the thickness of your clothes.

The border is like the silhouette or outline

and the margin is the personal space

between you and another person.

In this video, I've opened the box model for you.

Inside it is the content and around it flows the padding,

the border fits around the padding.

Lastly, the margin is

the empty space keeping elements apart.

You will work with the box model

in every website you develop,

so I encourage you to start

practicing with it in your designs.

**Document flow - block vs. inline**

By now you've learned how CSS styles individual html elements,

but how does the web browser nowhere to place the elements on the screen?

The web browsers normal way of calculating

the position of html elements on the screen is called document flow.

By default, nearly all html elements are organized into one of two

categories namely in block and in line elements.

A block level element will occupy the full horizontal width of its parent element and

the vertical height of its content.

Each block level element will have a new line before and after.

Therefore, multiple block level elements will stack on top of each other

like a stack of boxes.

In line elements only occupy the width and height of their content.

They don't appear on a new line, hence the name in line.

Therefore, multiple in line elements can form a row of elements.

When coding in html, you need to be able to recognize and use block elements.

Some examples of block level elements include the tags, div form and heading.

You also need to be familiar with common in line elements.

These include the tags anchor, image, input label, bold,

italics, emphasis and span.

Let me demonstrate the difference between block and

in line now by using an example of an html file containing div and span elements.

Okay, so, I've opened visual studio code here,

I will now demonstrate an example where you will learn about document flow.

First, I want you to notice that I have a div element containing three

sentences of Lauren ipsum.

A div or divided element is used to divide content into sections.

If you don't speak Latin, don't worry.

Lauren Ipsum is placeholder text that has been used since the 1500s.

Each sentence contains a span element that you may recall are in line elements.

Notice that all text is displayed in an unbroken flow of content in the browser.

In the code I have div and span elements.

I am now going to change the middle sentence from a span to a div tag.

Because the div tag is a block level element, notice that the sentence has been

moved to a new line and the span element after it is also a new line.

It's possible to change elements from block level to in line and vice versa.

This can be done using the display CSS property.

Let me add a CSS file called style dot CSS.

In my html file, let me give the middle sentence div element an Id.

I now add a rule in the CSS file to change the element to an in line element.

To create the CSS rule, I type hash middle and open the curly brackets.

On the next line, I type display colon space in line semi colon.

And on the next line I close the curly brackets,

notice that the div has now changed to an in line element.

If I want to change it back, I can just remove the property.

I have deleted the rule in the code to change the element to an in line element.

Another way to achieve this outcome is to set the display property to block.

I have now changed the value for display to block.

In another video, I will demonstrate to you that there are other values you

can set for the display property to change the layout behavior of an element.

Congratulations, you have now learned about block and in line elements in html.

It's important for developers to be comfortable working with each and

to know the differences.

It's important to remember that block elements begin on a new line and

take up the full width of the page.

And in line elements work within the flow of surrounding content rather than

breaking onto their own line.

**Alignment basics**

Let's explore how to align text and HTML elements using CSS.

Let's first focus on horizontal alignment. Vertical alignment is more difficult so you'll explore that later on.

**Text Alignment**

Aligning text within an HTML element is very simple. To do this, you use the *text-align* CSS property. In the following example, the CSS rule is setting the text of all paragraph elements to be center aligned.

p {

    text-align: center;

}

* Text alignment can be set to *left*, *right*, *center* and *justify*.
* The *justify* alignment spreads the text out so that every line of the text has the same width.
* The default alignment is *left* for languages that are left-to-right such as English. For right-to-left languages such as Arabic, the default alignment is *right*.

**HTML Element Alignment**

HTML element alignment is more complicated than text alignment. To align HTML elements, you must consider the box model and document flow from previous lessons. Aligning an HTML element is done by changing the properties of its box model and how it impacts the document flow.

**HTML Element Center Alignment**

To center align an element, you set a width on the element and push its margins out to fill the remaining available space of the parent element as in the following HTML structure:

<div class="parent">

  <div class="child">

  </div>

</div>

In your CSS, you'll set the *parent* element to have a red border to visualize the space it occupies:

.parent {

  border: 4px solid red;

}

The *child* element will have a width equal to 50% of the *parent* element with a padding of 20 pixels. Note that *padding: 20px* is shorthand for setting the padding top, bottom, left and right to *20px*. To visualize the space it occupies, set the border to green:

.child {

  width: 50%;

  padding: 20px;

  border: 4px solid green;

}

To align the element to the center, set its *margin* property to *auto*. The *auto* will tell the browser to calculate the margin automatically based on the space available.

.child {

  width: 50%;

  padding: 20px;

  border: 4px solid green;

  margin: auto;

}

The result is the *child* element is centered within the *parent* element:



It is important to note that this works because the *div* element is a block-level element.

If you want to align an inline element like *img*, you will need to change it to a block-level element.

Similar to the *div* example, you add the *img* to a parent element:

<div class="parent">

  <img src="photo.png" class="child">

</div>

The CSS rule then changes the *img* element to a block-level element and sets its margin to *auto:*

.child {

  display: block;

  width: 50%;

  margin: auto;

}

To be more precise, in CSS you can set only the left and right margins to auto. This allows you to set the top and bottom margins to specific values if needed.

.child {

  display: block;

  width: 50%;

  margin-left: auto;

  margin-right: auto;

}

**HTML Element Left / Right Alignment**

* The two most common ways to left and right align elements are to use the *float* property and the *position* property.
* The *position* property has several value options that impact how the element displays in the document flow. You'll explore how to use the *position* property later on. For now, let's focus on the *float* property.
* The *float* property sets an element's position relative to the text content within a parent element. Text will wrap around the element.

In the following example, the image will be aligned to the right of the *div* element. The text content will wrap around the image:

*HTML*

<div class="parent">

  <img src="photo.png" class="child"> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur eu odio eget leo auctor porta sit amet sit amet justo. Donec fermentum quam in diam volutpat, at lacinia diam placerat. Aenean quis feugiat sem. Suspendisse a dui massa. Phasellus scelerisque, mi vestibulum iaculis tristique, orci tellus gravida nisi, in pellentesque elit massa ut lorem. Sed elementum ornare nunc vel cursus. Duis sed enim in nulla efficitur convallis sed eget dolor. Curabitur scelerisque eros erat, in vulputate dolor consequat vel. Praesent ac sapien condimentum, ultricies libero at, auctor orci. Curabitur ut augue ac massa convallis faucibus sed in magna. Phasellus scelerisque auctor est a auctor. Nam laoreet sem sapien, porta imperdiet urna laoreet eu. Morbi dolor turpis, congue id bibendum eget, viverra et risus. Quisque vitae erat id tortor ullamcorper maximus.

</div>

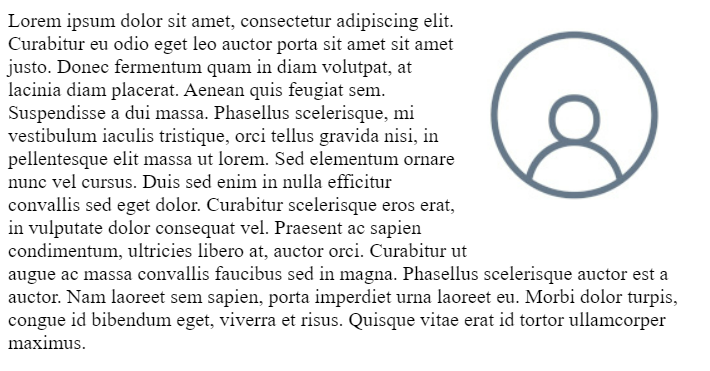
*CSS*

.child {

  float: right;

}

The following displays in the web browser:



**Working with libraries**

Hello and welcome.

Imagine you want to cook a dish you never tried before.

You can try different ingredients,

different cooking times,

or you can just use a recipe found on the Internet.

In a similar way as a developer,

you can try to build your website from scratch

or you can use code that other developers created.

In this video, I'll illustrate how to include

CSS and JavaScript libraries in your HTML file.

I'm also going to explain what dependencies are,

the purpose of a package manager and

JavaScript handlers. Let's get started.

By now, you know you

write code in your application and that

this code interacts with

APIs provided by libraries and frameworks.

Since you'll be deploying your application

to a web server in the end,

you must include these libraries and frameworks with it.

If you do not include them,

your application won't be able to call

the API functions it needs to execute when you run it.

That is why the libraries and

frameworks are often referred to as dependencies.

Your application depends on them.

On the front end, you do this by referencing

the JavaScript or CSS files in your HTML file.

One popular library you can use to develop

user interfaces is the Bootstrap library.

You will learn more about Bootstrap in this course.

But for now, let me demonstrate how

a library such as Bootstrap can be included on a webpage.

Start by opening your HTML file.

Next, add the CSS library for

Bootstrap in the head element using a link tag.

The link tag includes an H ref

attribute to link the CSS library I'm including.

I also included a rail attribute to

specify that the link is for a style sheet.

Bootstrap also provides a JavaScript library

that enables enhanced functionality,

such as drop-downs and tooltips.

You'll likely need those features during development.

Let me demonstrate how to add

the JavaScript library to your webpage.

Open your previous file that

already has the Bootstrap CSS.

Next, add the JavaScript library for

Bootstrap in the body element using a script element.

I then add an SRC attribute to

the script element to specify the link to the library.

Let me add a Bootstrap button to the page to

demonstrate how easy it is in the body,

I add the button element.

Then I add an attribute called

type and the CSS class BTN.

I also add the modifier

btn dash primary, and a description.

Click this button. This will add

a button using Bootstrap's primary color,

blue on the webpage.

This is what the rendered version

of the code will look like.

You can now start using

the additional Bootstrap functionality.

This is quite a simple example.

In day to day development, however,

it can end up being a lot more complicated.

For example, what happens if

your dependency also depends

on other libraries and frameworks?

This is known as a dependency tree.

A project could have hundreds

of dependencies and its tree,

it will take a very long time to download,

setup, and configure all of them.

How do you make sure you're using

the same versions the rest of your team is using?

This is where package managers come in.

A package manager is a tool that

automatically downloads and installs dependencies.

We also refer to dependencies as packages.

A package manager also provides

the capability to publish your own packages.

But let's not worry about that right now.

For each dependency, you can specify a version of

that dependency and the package manager

will download it for you.

If there is a dependency tree,

the package manager takes care of that for you.

It will automatically download all of

the dependency tree so that you can

use APIs without dependency issues.

The most common package manager for front end development

is the Node Package Manager or npm for sure.

You will have an opportunity to explore

using NPM in this program.

Now that all your dependencies are downloaded,

you need to include them in your HTML file.

But adding all of them into an

HTML file would take forever.

This is where you will use a bundling tool.

The purpose of a bundler is to

automatically combine them into a single file.

If you're bundle is significantly large,

many bundles can split

your dependencies into multiple bundles.

There are many bundle is available

such as Gulp and Webpack.

We'll be exploring bundling in more depth later on.

You now know how to include

CSS and JavaScript libraries in your HTML file.

You also know what a dependency is,

what the purpose of a package manager is,

and what JavaScript bundles are.

**Introduction to responsive design**

Have you ever wondered how the size of a website can so

easily adjust between different devices like laptops and mobile phones?

Well, the answer is responsive design.

Responsive design means that a web page can automatically stretch or

shrink depending on the screen is displayed on.

In the world of web development, you use responsive design every day.

Let me share how it works.

In this video,you will be able to explain how responsive design is used to provide

the best user experience based on the device that the website is viewed on.

Describe CSS media queries and screen resolutions,

classified fluid images and differentiate between fixed and fluid grids.

Phone, tablet and computer screens including the one you're probably using

now and made up of thousands of tiny lights called pixels.

The resolution of your screen refers to the number of pixels it contains most

often expressed as the number of horizontal pixels multiplied by the number

of vertical pixels.

For example, a common computer screen resolution is

1920 pixels in width 1080 pixels in height.

Thanks to responsive design websites can be correctly displayed on any of your

devices.

However, today we have many different screens with different resolutions,

which is why responsive design is so important.

And it is complicated by new high resolution screens like the one found on

your mobile phone.

These screens group multiple physical pixels into one logical pixel to display

smoother images and text.

These are often used in your favorite smartphones to give more high definition

visuals like making text images and rounded edges appear smoother and

making the individual pixels less visible.

Given all the possibilities and complicating factors,

it would be challenging to develop websites that appear correctly on all

kinds of devices if it wasn't for responsive design.

Responsive design is a set of three practices that allows a website

to automatically change its visuals.

In other words, to respond based on the device it is displayed on.

It is the combination of three techniques, flexible grids,

fluid images and media queries.

Firstly, flexible grids are made up of columns, gutters and margins.

The space between the columns is called the gutter and the spaces between

the content and the left and right edges of the screen are called margins.

Instead of defining website Element sizes based on pixels, flexible grids

are defined in percentage values, allowing them to adjust depending on screen size.

Next you have fluid images by setting the CSS max width property of images to 100%.

The images will scale down smaller if they're containing column becomes

narrower than the images size but never grow larger.

This enables an image to scale down to fit in a flexibly sized column rather

than overflow it but not grow larger and

become pixelated if the column becomes wider than the image.

Finally, there are media queries that are part of CSS.

They allow developers to query the display size orientation and

aspect ratio to conditionally apply CSS rules.

For example, if you wanted your website background to appear blue on a screen size

less than or equal to 700 pixels.

Like on a mobile phone, you could use a media rule to set the background depending

on the size of the screen.

Remember that I said responsive design is the combination of flexible grids,

fluid images and media queries.

When these elements are used together, you build a website that will automatically

adjust its layout based on the device, thus delivering a responsive grid.

In responsive design,

the pixel value specified is often referred to as the breakpoint.

A breakpoint is the point at which a website's content and

layout will adapt to provide the best possible user experience.

A Breakpoint can function in different ways across three different grids

a fixed grid fluid or for with grids and lastly, hybrid grids.

Let's explore each of these now, firstly,

a fixed grid has fixed with columns and flexible margins.

The fixed grid has a fixed content with that doesn't change in a specific

breakpoint range while the flexible margins occupy the remaining space on

screen.

Then we have fluid or full width grids with fluid with columns and

fixed gutters and side margins.

The fluid grid has a flexible content with that goes edge to edge as per

the screen size.

In a fluid grid, columns either grow or shrink to adapt to the available space.

And finally there are hybrid grids that have both fluid width and

fixed with components.

Many responsive design frameworks provide multiple CSS

rules based on different device sizes to provide the best visual experience.

Now you know the importance of responsive design.

In the next few videos, you're going to learn more about bootstrap,

the world's most Popular framework for building responsive mobile 1st Sites.

**Getting started with Bootstrap**

Part of designing a website is thinking about

how different UI elements and menus are

positioned on the page and in relation to each other.

Bootstrap is a collection of

pre-written code chunks in CSS and JavaScript that allows

you to create websites more quickly than if you

had to create every bit of code from scratch.

Building websites from scratch every

time will be quite time-consuming.

This is where Bootstrap comes in.

In this video, you will learn how to

create a simple Bootstrap webpage.

To do that, I am using the file with

the Bootstrap libraries I have created.

Let's begin by setting up the layout.

The first element I need to add when setting up

a Bootstrap website is the container elements.

I need it before I can use the Bootstrap grid system.

I add HTML div element

and apply the container CSS class provided by Bootstrap.

Next, I add a row for the contents.

Similar to what I had in the container elements,

a row is a HTML div element,

which uses the row CSS class provided by Bootstrap.

Under this row, I want to add two columns,

one for the menu items and one for the prices.

I add two div elements with the col CSS class,

one for each column.

Col is short for column.

Now I want to name the columns.

I do this by adding heading tags in each div.

Within the first column div,

I add a heading 1 tag or H1 tag.

I type the name of the column, our menu.

The second column div element I type prices in a H2 tag.

Under the our menu column,

I will add the name of each dish,

the ingredients used, and an image.

First, I add a heading 2 tag below the our menu H1 tag,

and inside it, I add the name of the dish, falafel.

On the falafel H2 tag,

I add a paragraph tag or p tag.

Inside this tag, I list the ingredients,

chickpea, herbs, and spices.

Now to add an image of a falafel,

I add the IMG or image tag and inside it

add the SRC or source attribute as it is known.

In the source attribute,

I enter the name of the image file I want to link to.

I type falafel.jpeg.

I now add a CSS class called IMG-fluid or image fluid,

as most developers call it.

Instead of having to manually set the image size,

the IMG-fluid class will ensure

my image scales to its parent column's width.

Next, let's add another dish.

I add another heading 2 or H2 tag,

and inside it, I add the name of the dish pasta salad.

Under the pasta salad H2 tag,

I add a paragraph tag or p tag,

and list the ingredients,

lettuce, vegetables, and mozzarella.

Now I add another image tag and

inside it add the source attribute.

In the source attribute, I enter the name

of the image file I want to link to.

I type salad.jpeg.

I also add the same CSS class image-fluid.

Now I save the file by pressing the keys Control and S,

or if you're on a Mac Command and S. Next,

I preview the webpage in live preview.

The dishes are now displayed in columns

and the images are sized correctly.

Now let's add the price table.

Under the price column, I add

a HTML table tag and apply the Bootstrap CSS class table.

Inside the HTML table tag,

I add two table row tags or tr tags.

This will create a row for each of the dishes.

Then I need to add table data tags or td tags.

The table data tags to find the contents of table cells.

I'll need to add two of those inside each table row,

one for the dish name and one for its price.

In the first row, I add falafel to the first td tag,

and in the second td tag,

I add the price, let's say $12.

I do the same with the second row.

I type pasta salad inside

the first td tag and

the price of $10 in the second td tag.

I save the file again by pressing Control and

S or if you're on Mac Command and S. Next,

I preview the webpage in live preview.

The table structure now displays

using Bootstrap's table style.

You've now learned how to make

a simple webpage using Bootstrap.

**Using Bootstrap styles**

As a developer, you were tasked to design a website for the Little Lemon Restaurant.

One of the main requirements is that the website easily adapts to different

devices, platforms and screen sizes.

Does this mean that you have to redesign the website for each device?

No, relax.

You do not have to redesign your website for each device.

In this video, I'll introduce you to Bootstrap CSS class in fixes and

modifiers.

And you will learn how you can use them to save your time.

Let's get started.

Bootstrap has quite a large CSS library built by Bootstraps developers using

thousands of use cases.

How it achieves this is through CSS classes and

their variations through in fixes and modifiers.

It is important that you as a developer, understand in fixes and modifiers and

you will explore each of these terms.

You will use class in fixes for responsive breakpoints in the Bootstrap grid system,

let's go ahead and cover how in fixes are used.

You already know that breakpoints are the triggers in bootstrap for

how your layout changes across device or viewpoint sizes.

Here are the responsive breakpoints available in bootstrap and

how we use them.

Know that each has a specific name.

Extra small is for screens less than 576 pixels wide.

Small is for screens greater than or equal to 576 pixels wide.

In bootstrap CSS rules, this is abbreviated as SM.

Medium is for screens greater than or equal to 768 pixels wide.

In Bootstrap CSS rules, this is abbreviated as MD.

Large is for screens greater than or equal to 992 pixels wide.

In Bootstrap CSS rules, this is abbreviated as LG.

Extra large is for screens greater than or equal to 1200 pixels wide.

In bootstrap CSS rules this is abbreviated as XL.

Extra extra large is the screen is greater than or equal to 1400 pixels wide.

In bootstrap CSS rules, this is abbreviated as XXL.

Did you notice that there is no class in fix or dimension for

extra small on the list?

This is because extra small is the default breakpoint in Bootstrap CSS rules.

As bootstrap is mobile first.

We use the abbreviations for

these breakpoints as in fixes in the CSS rules for the grid system.

This basically means you have to insert the abbreviation into the CSS class name.

I will show you how to use this shortly.

Bootstrap components provide a pre built set of reusable UI styles and elements for

your web applications.

Let's now learn how to use modifiers through an example that uses

an alert element.

I can have an html file where I set an alert primary CSS class that is applied to

the element.

Alert primary displays the alert using Bootstraps primary color which is blue.

The dash primary part of the class is the modifier.

If you want to change this to red for example, for

an error message you will use the danger modifier.

Here is a list of modifiers available in Bootstrap.

Primary, secondary, success, info, warning, danger, light, dark.

Alerts are often used to show information that needs immediate attention from users

such as warnings, errors or confirmation messages.

Bootstrap provides an easy way to create predefined alert messages and

it also has different types of alerts.

The difference between the alerts is just the color used to display the predefined

alert message.

For example,

the primary alert displays the alert using bootstraps primary color which is blue.

On the other hand, the danger alert displays the alert in red.

Now, let's turn our attention to how this works in code.

In this html file, I have a six columns CSS class applied to a div element.

In this example, I want to change this rule for large screens.

To do this, I in fix LG into the col CSS class, by adding col dash LG dash six.

Bootstrap components provide a pre built set of reusable UI styles and elements for

your web applications.

For example, I can add an alert component.

I do this by adding an alert CSS class followed by the contextual class,

alert dash primary.

Now I have to identify the function of the alert.

To do this, I add a roll CSS class directly after the contextual class,

alert dash primary and type alert.

On the next line, I type the name that must be displayed on the alert.

A message.

Alerts are useful for displaying information, warnings or

error messages to users.

The contextual class,

alert primary will display the alert using Bootstraps primary color, which is blue.

The primary part of the class is the modifier.

If I want to change the color to red, for example, for an error message,

I will use the danger modifier.

To change the color to red, I simply replace the modifier Primary with danger.

Now you have learned about CSS class in fixes and modifiers and

how you can use them to save you time.

**Bootstrap grid**

As you have previously learned building a website using responsive design requires

a responsive grid and responsive breakpoints.

Bootstrap provides both of these as part of its library.

The bootstrap grid system helps us to create web page layouts through a series

of rows and columns that house our content.

For the grid, bootstrap uses a 12 column grid system that can be fluid or fixed.

The bootstrap grid system always has a container, rows and columns.

The container is the root element of your grid.

Bootstraps grid system always starts with the container.

The container contains pads and aligns your content.

Its width is determined based on the current responsive breakpoint.

You can add rows and inside each row you can add columns.

Let me demonstrate how we put this all together by using

little lemons website as an example.

All right, I have opened my joe's burger index.html file.

In this web page I have added two columns to my row.

Bootstrap is smart and will set the two columns to span six columns spaces.

If I want to control how many spaces a column uses, I can suffix a number.

Let's shrink the price column to four spaces and

expand the menu column to eight spaces.

To do this, I add the suffix-4 to the coal CSS class above the prices heading.

I also add the suffix-8 to the coal CSS class above our menu heading.

But here's the amazing part of bootstrap.

I can configure my content to use different layouts based on the device

using the breakpoint specific CSS rules.

Let's have the content stacked on top of each other on mobile devices and

have them display side by side on desktop devices.

Since columns will wrapped in euros, if we exceed the 12 column space,

we can use this to stack content.

For mobile devices, I'll set the columns to use 12 spaces each by adding

12 as a suffix to the coal rule for both columns.

Then for desktop I'll set them to six columns each by using a cold- LG rule and

adding the column sizes-6 as a suffix.

I will save the file by pressing control and s or

command s on a Mac and open it on my web browser.

Notice that now the columns are side by side.

Next I opened my web developer tools by pressing the F- 12 key.

In the top left corner, there is a mobile phone tablet button.

Clicking on this button enables the device mode.

This allows me to simulate different devices.

I select to view my website on a mobile phone device, amazing.

My content has stacked vertically on mobile.

Bootstrap has ensured our website will work on a mobile phone thanks to its CSS

rules and grid system.

This will save a lot of development time by not needing to develop different

content layouts per device.

Bootstraps grid system is very powerful and suitable for most development needs

**Working with Bootstrap grid lab**

<!DOCTYPE html>

<html>

<head>

    <title>Little Lemon</title>

    <link href="bootstrap.min.css" rel="stylesheet">

</head>

<body>

    <div class="container">

        <div class="row">

            <div class="col-12">

                <div class="text-center">

                    <img src="logo.png" class="img-fluid">

                </div>

            </div>

        </div>

        <div class="row">

            <div class="col-12">

                <div class="text-center">

                    <h1>Our Menu</h1>

                </div>

            </div>

        </div>

        <div class="row">

            <div class="col-12 col-lg-6">

                <h2>Falafel</h2>

                <p>Chickpea, herbs, spices.</p>

                <h2>Fried Calamari</h2>

                <p>Squid, buttermilk.</p>

            </div>

            <div class="col-12 col-lg-6">

                <h2>Pasta Salad</h2>

                <p>Lettuce, vegetables, mozarella.</p>

                <h2>Greek Salad</h2>

                <p>Cucumbers, onion, feta cheese.</p>

            </div>

        </div>

    </div>

    <script src="bootstrap.bundle.min.js"></script>

</body>

</html>

**Bootstrap components**

Boot trap includes a pre made set of UI elements and

styles to help you build your website.

They range from alert messages to navigation menus.

These are called bootstrap components.

In this video you will learn how to use bootstrap components.

Little lemon have added a new fried calamari dish to the menu and

they want to add it to their website.

They also want to update the website to use bootstrap components.

First, let's add the new dish.

To do this in the menu column, I add a new h2 tag and

inside it I add the name of the dish Fried Calamari.

I didn't add a paragraph tag inside this tag,

I list the ingredients, squid and buttermilk.

Now I want to add an image of the fried calamari.

So I used an image tag with its source attribute and

the image file name calamari.jpeg.

And I add the image dash fluids CSS class

to ensure that the image scales to its parent columns width.

Bootstrap comes with a badge component that I can use to highlight that

the dish is new.

Let's add that now.

Next to Fried Calamari, Inside the h2 element,

I add a span element and assign it a CSS class of badge.

Now I can set the background color of my badge by

adding another class called bg-primary.

The contextual class bg-primary will display the badge

using bootstraps primary color which is blue.

Inside the span I added text new.

Next I'm going to improve the visuals by using the card component for each dish.

With card elements I can style and accommodate different types of contents

like title's subtitle body copy and images for the cards.

Let's have them stack on mobile devices and tile side by side on desktop.

To do this, I add a row below

the our menu h1 tag.

I then add a div element for

each dish of the three dishes.

For each day of I applied a col-12 class for

mobile devices and a col-Lg-6

class for desktop devices.

Inside each div of elements,

I added narrative elements

that uses the card CSS class

by adding one and copying and

pasting it inside other dudes.

Then move the code with the image

tag of each dish to the card class.

Instead of using img-fluid I change each image to card- img-top.

This will improve its visual style within the card.

I then add another day for each dish and give it the card-body css class.

This will contain the text content of the card.

I then moved the header and paragraph of each dish into these card body elements.

Again, to improve the visual style,

I add a card-title css class

to the header of each dish.

In the paragraph elements I add a car-

text css class for each description.

Let's preview in live preview.

Our menu has a much better layout now.

Finally let's update the prices

Under the price column I add a table row or

tr tags for the new dish.

I then add 2 td tags but from now on I will just refer to it as td.

In the first row, I typed fried calamari, in the first td tag and

in the second td tag, I typed the price of $12.

Now I will add an alert to inform our customers about the new dish so

that they can try it out.

To do this I add a div element with the alert css class.

I then add the alert-info css class to change the alert to blue.

Now identify the function of the alert by adding

a role attribute directly after the contextual css class.

For its value I type alert.

On the next line.

I typed text that must be displayed on the alert.

Try our new fried calamari.

Let's see how it looks in live preview.

The website has a much better layout now,

hopefully little lemon will have a lot more orders now.

There are many components available in Bootstrap.

Maybe there are some more you can use to improve your website

**Working with Bootstrap components lab**

<!DOCTYPE html>

<html>

<head>

    <title>Little Lemon</title>

    <link href="bootstrap.min.css" rel="stylesheet">

</head>

<body>

    <div class="container">

        <div class="row">

            <div class="col-12">

                <div class="text-center">

                    <img src="logo.png" class="img-fluid">

                </div>

            </div>

        </div>

        <div class="row">

            <div class="col-12">

                <div class="text-center">

                    <h1>Our Menu</h1>

                </div>

                <div class="alert alert-info" role="alert">

                    Our restaurant will be closed on New Year's Day

                </div>

            </div>

        </div>

        <div class="row">

            <div class="col-12 col-lg-6">

                <h2>Falafel <span class="badge bg-secondary">New</span></h2>

                <p>Chickpea, herbs, spices.</p>

                <h2>Fried Calamari</h2>

                <p>Squid, buttermilk.</p>

            </div>

            <div class="col-12 col-lg-6">

                <h2>Pasta Salad</h2>

                <p>Lettuce, vegetables, mozzarella.</p>

                <h2>Greek Salad</h2>

                <p>Cucumbers, onion, feta cheese.</p>

            </div>

        </div>

        <div class="row">

            <div class="col-12">

                <div class="text-center">

                    <button type="button" class="btn btn-primary">Order Online</button>

                </div>

            </div>

        </div>

    </div>

    <script src="bootstrap.bundle.min.js"></script>

</body>

</html>

**Using Bootstrap documentation**

Bootstrap comes with detailed documentation on setting up and using the features available in its library. The documentation is clear and has many code examples to help you get started.

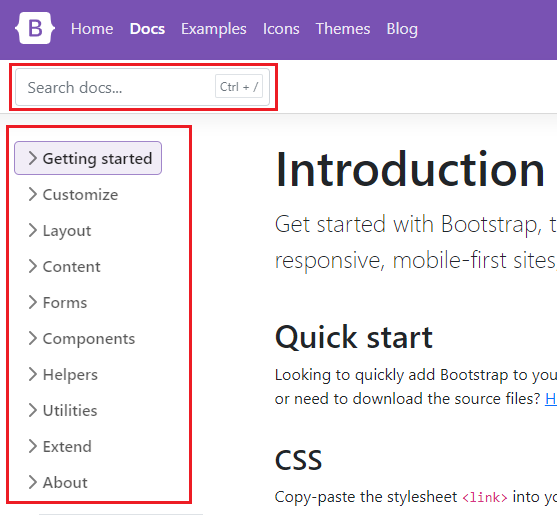
In this reading, you'll explore the frequently used documentation sections.

The documentation for Bootstrap is currently available at the following link.

<https://getbootstrap.com/docs>

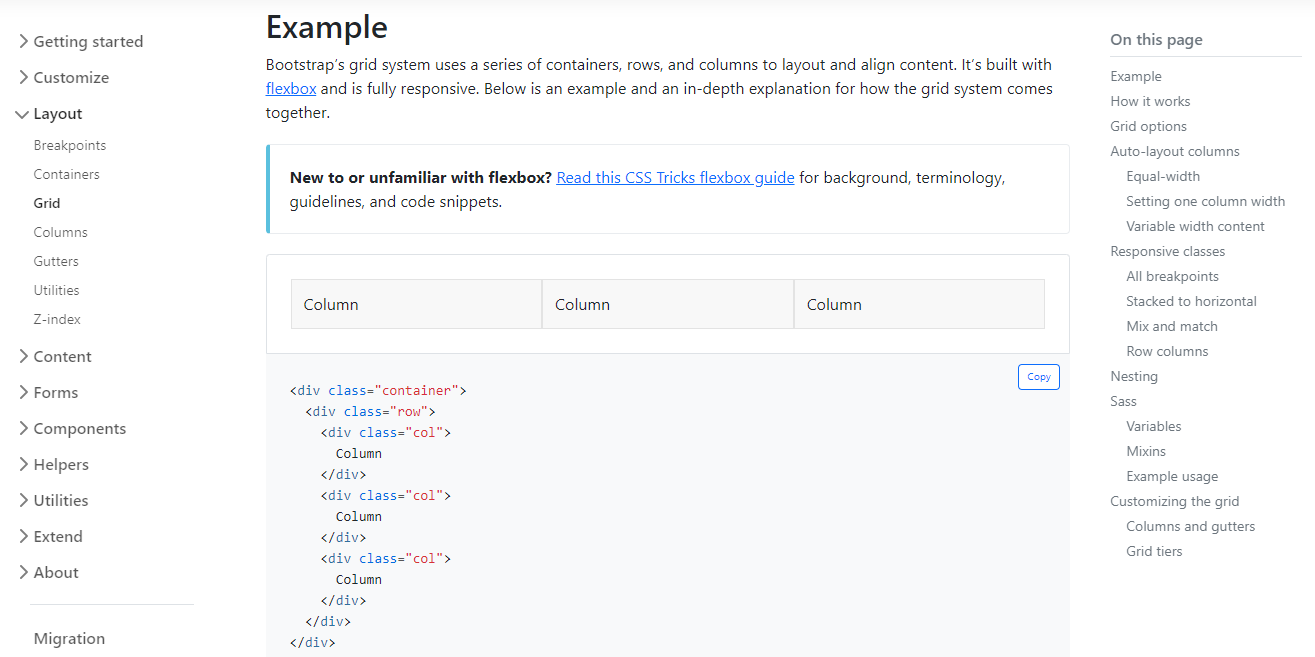
**Navigating the documentation**

The sidebar on the webpage allows you to navigate through the different sections of the documentation. There is also a search box if you need to search for a specific piece of information.



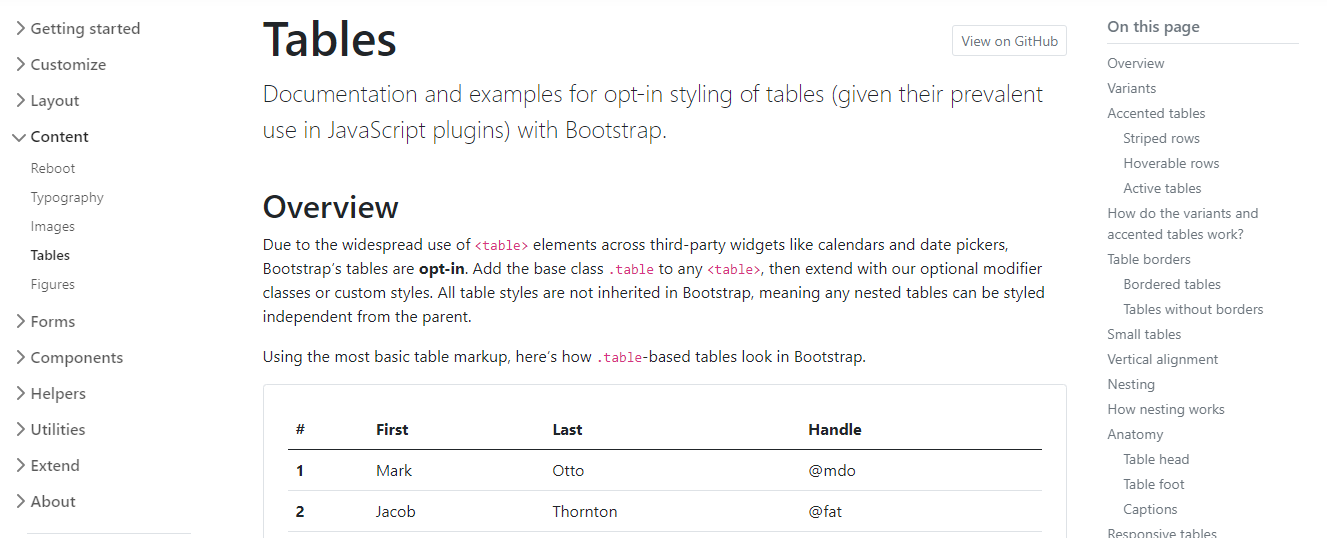
**Layout**

The layout section of the documentation describes how to use the grid system of Bootstrap. This covers what you've learned so far and includes more advanced usage such as offsets, column alignment, auto-layout and variable width columns.



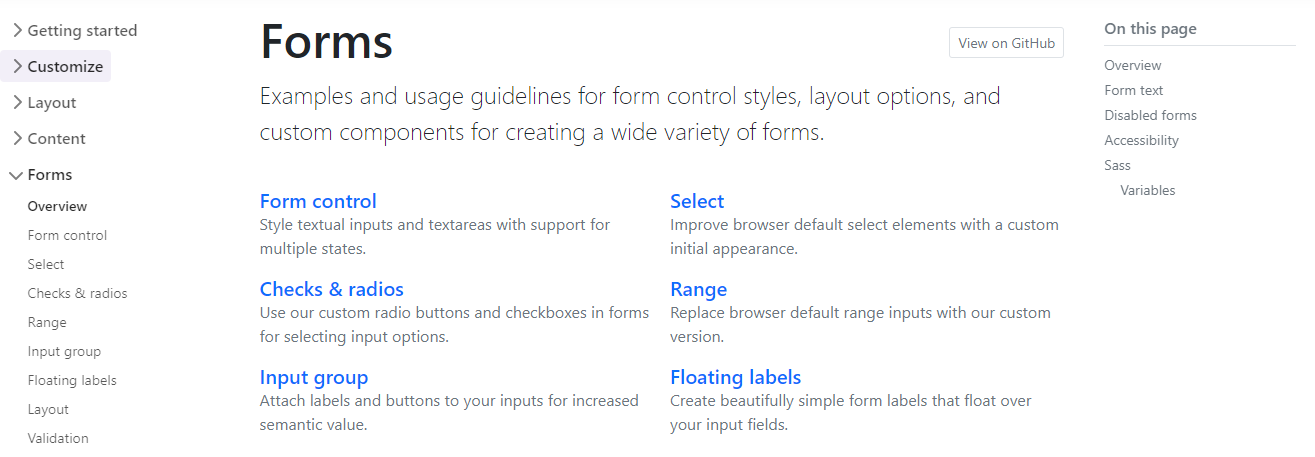
**Content**

The content section of the documentation describes Bootstrap's default text styling and how to use responsive images and tables. You've learned the basics of these earlier on and this section goes into further detail.



**Forms**

The forms section of the documentation describes how to build forms using Bootstrap's styles. The library has many CSS rules to improve your form's user interface and experience. Below are some features you'll frequently use as a developer:



**Form Styling**

Bootstrap includes CSS rules to improve the visual style of input elements.

For example:



This table outlines the different HTML form elements and which Bootstrap CSS class should be used for them.

| **Form Element** | **CSS class** |
| --- | --- |
| *input* | *form-control* |
| *input type="checkbox"* | *form-check-input* |
| *input type="radio"* | *form-check-input* |
| *input type="range"* | *form-range* |
| *select* | *form-select* |

Using these CSS classes will style the elements appropriately for different input types, sizings and states. More information is available on the [Forms documentation page](https://getbootstrap.com/docs/5.0/forms/overview/).

**Switches**

If you've used an app on your mobile device, you're probably familiar with the switch input type.

Bootstrap Doc Switches

Bootstrap includes CSS rules to style checkbox input elements as switches.

To do this:

1. Add the *input* to a *div* element.
2. On the *div* element, apply the *form-check* and *form-switch* CSS classes.
3. On the *input* element, add the *form-check-input* CSS class.

<div class="form-check form-switch">

  <input class="form-check-input" type="checkbox">

</div>

More information is available in the [Switches section of the documentation](https://getbootstrap.com/docs/5.0/forms/checks-radios/#switches).

**Input Groups**

Input groups are useful for providing additional content to the input field. For example, if you wanted to request the user to input a US dollar amount, you can use an input group to show the dollar symbol and cents amount.

Bootstrap Input Groups 

To do this:

1. Add the *input* to a *div* element.
2. Apply the *input-group* CSS classes on the *div* element.
3. Add a *span* element before and/or after the *input* element and apply the *input-group-text* CSS class to it. The text content is then added inside the *span* element.

<div class="input-group">

  <span class="input-group-text">$</span>

  <input type="text" class="form-control">

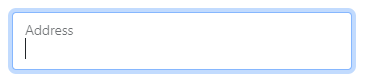
  <span class="input-group-text">.00</span>

</div>

More information is available on the [Input Groups documentation page](https://getbootstrap.com/docs/5.0/forms/input-group/).

**Floating Labels**

Floating labels help provide form information to the user as part of the input itself. These are different from regular form placeholders. The information stays visible if the user is interacting with the element or if the element has content.



To do this, add the *input* to a *div* element. On the *div* element, apply the *form-floating* CSS classes.

<div class="form-floating">

  <input type="email" class="form-control" id="addressInput" placeholder="Address">

  <label for="addressInput">Address</label>

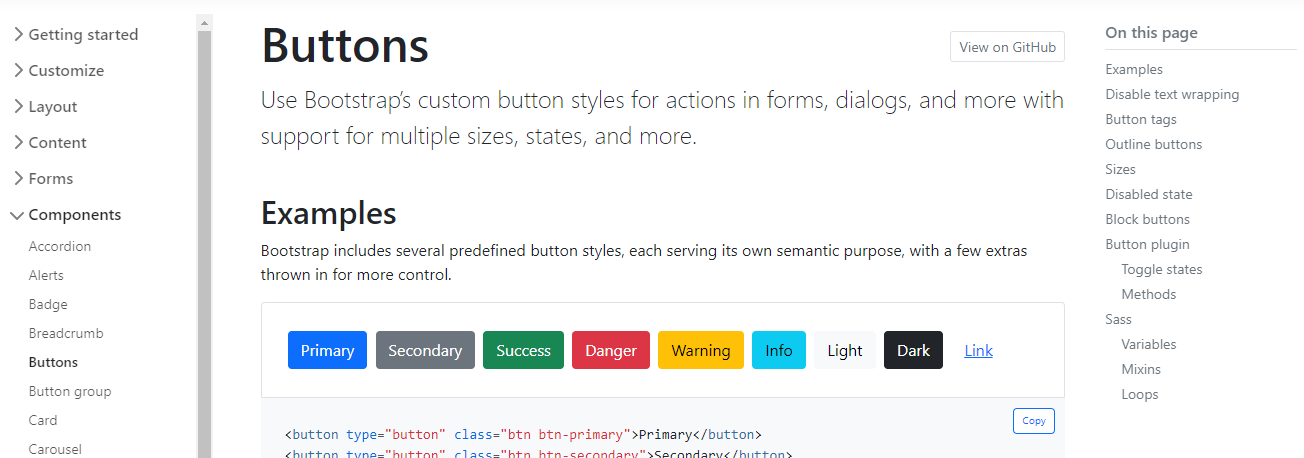
</div>

More information is available on the [Floating Labels documentation page](https://getbootstrap.com/docs/5.0/forms/floating-labels/)

**Components**

As you have learned, Bootstrap comes with many pre-made UI elements and styles to help speed up your development.

Some of these components require Javascript to work, while others only require CSS classes applied to HTML elements. The Components section of the documentation explains these requirements on each component page and provides many code examples.



**Conclusion**

Now that you are familiar with how to use the Bootstrap documentation, maybe try some new components and styles on a webpage that you've previously built.

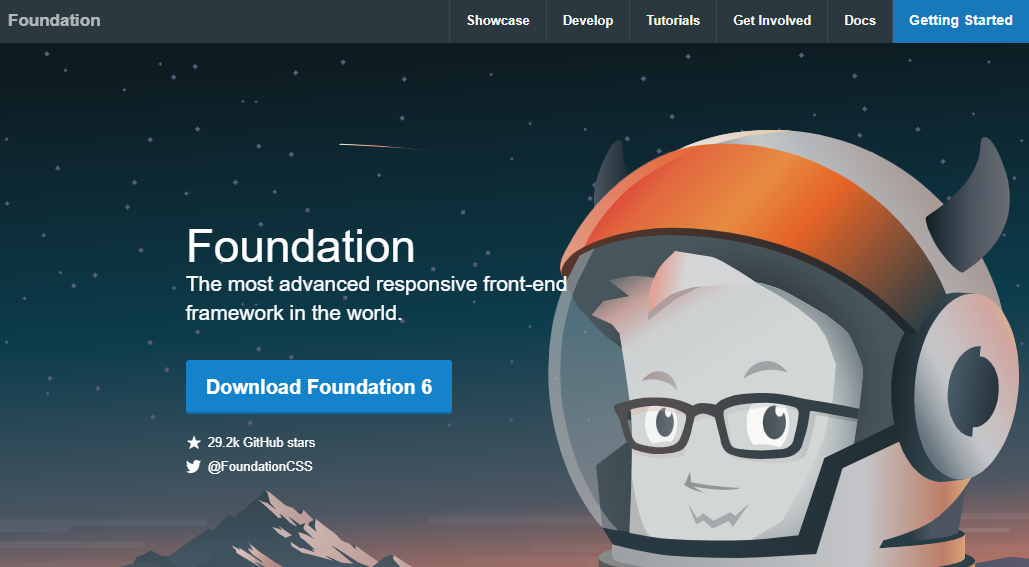
**Other CSS frameworks and libraries**

As a developer, you'll use many CSS libraries and frameworks throughout your career. As you move on to different projects and as technologies advance, knowing what solutions are available is critical. While Bootstrap is one of the most popular CSS libraries, many others are available, each with different purposes, designs and technical approaches. This reading will introduce you to other popular CSS libraries and frameworks.

**Foundation**

[Official Website](https://get.foundation/)

Foundation is a framework for building user interfaces similar to Bootstrap. It is used by many large companies such as Pixar, Polar and Sonos. One prominent feature of Foundation is that it can be used to style content for sending via email.



**Pure.css**

[Official Website](https://purecss.io/)

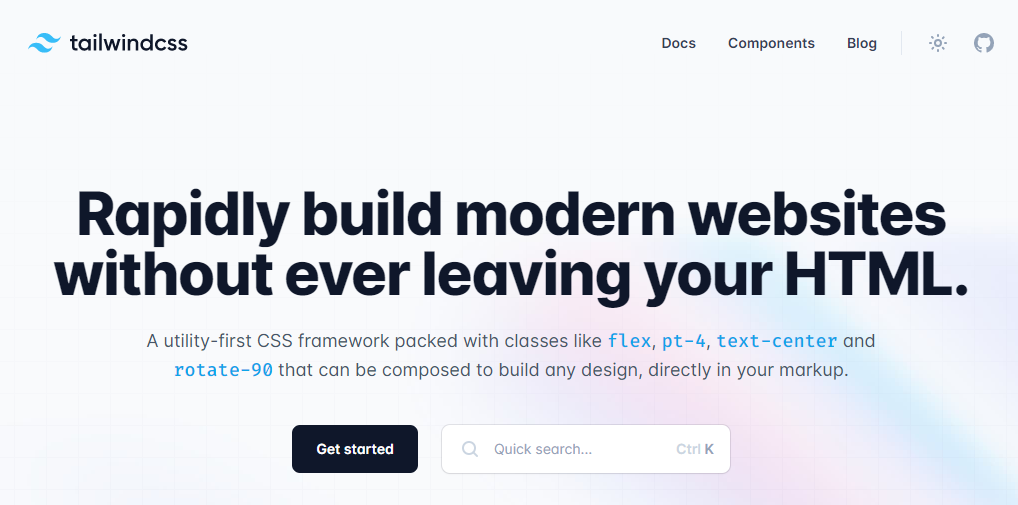
Pure.css is another library for building user interfaces. While it doesn't have as many features as Bootstrap, it is designed to be minimal in file size. Smaller file sizes improve loading times for web pages as there is less data to transfer from the web server. If your next project is focused on minimal loading time, this library is worth considering.



**Tailwind CSS**

[Official Website](https://tailwindcss.com/)

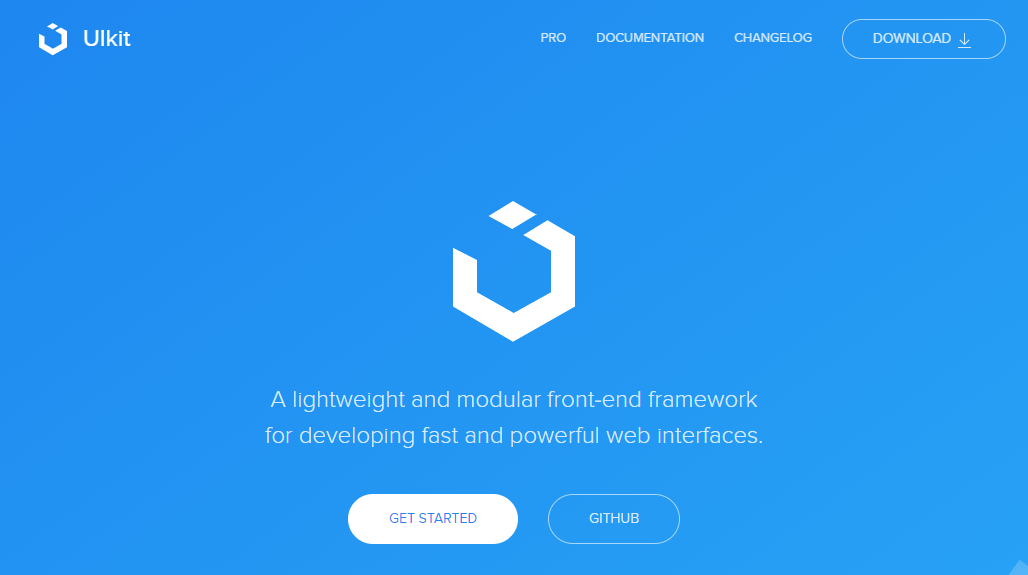
Tailwind CSS is a CSS framework that uses a utility-based approach for its CSS rules. This means that the framework provides many CSS classes with a single purpose. For example, the CSS class pt-6 sets the padding-top CSS property to 6 pixels. This means that you can be precise in applying styling to your HTML without writing CSS. The advantage to this is that it is more flexible for customizing your webpage's design using the framework. However, the disadvantage is that if multiple developers are working on a project, it could lead to inconsistent design if the team is not strict on its design rules.



**UIKit**

[Official Website](https://getuikit.com/)

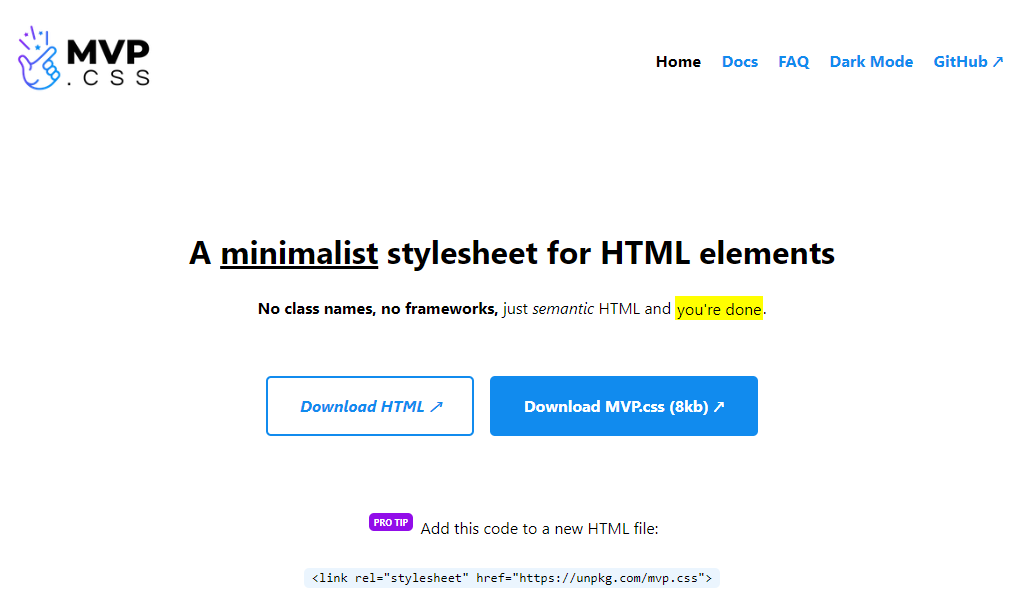
UIKit is a lightweight CSS framework featuring a small set of responsive components. Its simple design allows developers to easily customize the style rules and visuals.



**MVP.css**

[Official Website](https://andybrewer.github.io/mvp/)

MVP.css is a small CSS library that automatically styles HTML elements without needing to apply CSS classes to them. The library aims to allow a developer to quickly prototype a user interface without worrying about the final design, while still being visually appealing. MVP comes from the technical term Minimal Viable Product, a product with sufficient features to demo to customers or other business stakeholders.



**Conclusion**

If you're curious to learn more about these frameworks, their websites feature set up guides, tutorials and documentation to get started. It is a good exercise to compare and contrast different libraries and frameworks to understand different workflows available to you as a developer.

**Additional Resources**

**Bootstrap Official Website**

<https://getbootstrap.com/>

**Bootstrap 5 Foundations by Daniel Foreman**

<https://www.amazon.com/Bootstrap-Foundations-Mr-Daniel-Foreman/dp/B0948GRS8W/>

**Responsive Web Design with HTML5 and CSS by Ben Frain**

<https://www.amazon.com/Responsive-Web-Design-HTML5-CSS/dp/1839211563/>

**Bootstrap Themes**

<https://themes.getbootstrap.com/>

**Static and dynamic content**

I'm sure you've noticed that when you shop online,

websites will often provide

you with personalized recommendations.

But when your friends login to those same websites,

the recommendations will differ.

But did you ever wonder how websites

adapt to different user profiles?

In this video, you'll learn

the difference between static and dynamic content.

You will also be able to differentiate between

the role of a web server and an application server.

By now, you know that when you open a website,

a web server sends the website's content to your browser.

The content can be static or dynamic.

Static content is files that

the server transfers just

as they are stored on the web server,

such as videos or images.

Dynamic content, on the other hand,

is generated when the HTTP request is made.

For example, the content may be

generated based on input from a user,

or when you visit a news website,

it would be based on the current date.

What actually happens,

is that the web server

communicates with another kind of server,

called an application server or a back-end.

The application server generates

the dynamic content that

the web server sends back to the user's browser.

Now you know the difference.

Let's look at examples of how websites

update static and dynamic content.

Say for instance, you want to watch a video on a website,

you click on the Play button and

a request is sent to the web server.

The web server responds by

sending the file to your browser.

Now, how does this compare to dynamic content?

Because dynamic content is

generated while you use a website,

it typically takes longer to generate than

it takes to send back static content.

For example, when you log into this course,

the web server communicates with

an application server to check

that you are in fact enrolled.

The application server confirms your enrollment and

specifies what content should

show for your profile specifically.

Application servers perform

more complex processing than web servers.

For instance, they have to run the application logic,

communicate with the database, and check permissions.

To improve performance,

different application servers have specific purposes.

There are application servers for

every type of content out there,

from music streaming to creating your own blog.

You can even create your own application servers,

such as building a back-end for a web application.

But hang on, If dynamic content is slower to generate,

isn't that a problem for big websites? Yes, it is.

Application servers typically have

a limited capacity on

how many requests they can process per second.

But fortunately, this is

where the web server can help out.

Web servers use a process called caching

instead of generating content

dynamically for every request.

Caching means the web server

keeps a copy of dynamic content.

If the content is requested again,

the web server can immediately send

this cached version instead of passing

the request again to the application server.

On the first request for dynamic content,

the web server checks if the content exists in the cache.

If it does not exist,

the content is requested from

the application server and stored in the cache.

The web server then sends back

the dynamic content to the browser.

On subsequent requests, the web server

immediately sends back the content stored in the cache.

This reduces the amount of

dynamic content that

the application server has to generate.

Then, after a period of time

or with the next user interaction,

the web server updates the cache with the latest content.

Now you know the difference between

static and dynamic content,

as well as the difference between

web and application servers.

With this new information,

try to identify with the content on

the websites you visit is static or dynamic.

**Single page applications**

Let's say that you're thinking of building a web app.

You probably want a user-friendly,

mobile-friendly experience.

Then you should consider using

single-page applications or SPAs to build a unique,

speedy, and engaging experience for your users.

You're probably using many SPAs every day.

Think of your favorite social network or

messaging app or the map application

you use to find local businesses.

The great user experience you get is

driven by single-page applications.

So, what exactly is an SPA? How does it work?

How is it different from a traditional website?

Let's first explore how traditional website works.

Before the advent of modern JavaScript frameworks,

most websites were implemented

as multi-page applications.

But this makes traditional

applications resource intensive to web

servers because sending entire web pages

for every request consumes

excessive bandwidth and uses

CPU time to generate dynamic pages.

If your website is complex,

the site browsing experience may appear slow to users.

It will be even slower if they have

a poor or limited internet connection.

To solve this problem,

many web developers build their web applications as SPAs.

It's called single-page, but that doesn't

mean your website has only one page of content.

What it means is that there is

only one HTML page

that gets sent from the server to the browser,

but that page will update its content

as your users interact with your website.

A SPA allows the user to interact with

the website without the application

needing to download entire new web pages.

Instead, it rewrites

the current web page as the user interacts with it.

The result is a browsing experience that feels

faster and more responsive to user input.

When the user navigates to

the web application in the browser,

the web server returns

the necessary resources to run the application.

A SPA has two approaches to serving code and resources.

The first is called bundling,

and the second approach is known as

lazy loading or code splitting.

With bundling, when

the browser requests the application,

the server returns and loads all necessary HTML,

CSS, and JavaScript immediately.

With lazy loading, the browser requests

the application and the server returns

only the minimum HTML,

CSS, and JavaScript needed to load the application.

Additional resources are downloaded as required.

For example, when a user

navigates a specific section of the application,

both approaches are valid.

The choice depends on the size,

complexity, and bandwidth

requirements of the application.

For instance, with the bundling approach,

if your application is complex

and has a lot of resources,

your bundles will grow quite

large and take a long time to download.

You could end up with a site that is

slower than a traditional web application.

Let's explore an example.

Imagine you have a site dedicated to reviewing movies,

people never know what to watch next so you decide to

install a feature that randomly

suggests the movie for a user.

You create a web page that

has a what to watch next button.

You want to display a random movie name

when the button is clicked.

In a traditional website,

when the button is clicked,

the browser will send a post request to the web server.

The web server returns

a new web page containing the button and movie name.

The web browser then renders the new page.

In an SPA, when the button is clicked,

the browser will send a post request to the web server.

The web server will return a JSON object.

The application reads the object and

updates by displaying the text of the movie name.

That's more efficient because the rest of

the page remains as it was and it's

content does not need to be sent

by the server and rendered by the browser.

But what if you need to update the majority of

the page to display a different form of content?

Well, let's explore that scenario.

Suppose you are building a web application

that has two pages.

One page shows the latest news and

the other page shows the current user's profile page.

Navigation bar at the top of

the site contains a link to each page.

In the traditional websites,

when the user clicks a profile link,

the web browser sends the request to the web server.

The web server generates

the entire HTML page

and sends it back to the web browser.

The web browser then renders a new web page.

In a single-page application,

different pages are broken into templates,

also known as views.

Each view will have HTML code

that can be updated by the application.

For instance, the profile page would have a username,

a first name, and a last name.

The web browser sends the request to

the web server and the web server sends back

a file called a JavaScript Object

Notation or JSON object.

This contains only the data to be displayed,

such as the user's first name and last name,

and the SPA will update the HTML.

This is much smaller than sending an entire web page.

The web browser then updates the web page by inserting

the template with items

replaced by the values in the JSON object.

For your next website,

consider whether its complexity suits

a traditional multi-page application

or would perform better as an SPA.

Remember that a single-page application

has two methods of delivery resources.

Delivering all resources immediately or

delivering resources dynamically as required.

If all resources are

delivered when the application is loaded,

the single-page application must

include the views for every page on initial load.

If resources are loaded dynamically,

the single-page application requests the views as

required and stores views

in the browser for subsequent requests.

Now you can compare some popular SPA websites to

traditional ones and judge

the performance difference for yourself.

**What is React?**

I'm sure you've created something from scratch before.

I bet you realize how challenging it can be as opposed to using pre made

components.

Think about the effort that goes into building a brand new house versus

the convenience of constructing a modular building with pre made components that you

just need to put together.

React is focused on working with components,

which makes it simple to build functional user interfaces on web and mobile.

And, since React is one of the most popular JavaScript libraries,

it's important that you can explain what its purposes and

describe how its components work together.

Available since 2013 React is an open source library with a community of core

contributors and companies that maintain it.

Developers use React to develop single page applications and

you can also develop mobile applications with React native.

When you develop an app, you can choose to use React to develop the user interface.

But building an application requires more than that.

You must also consider the navigation and

how the app will request data from a web server.

Therefore React is used in conjunction with other JavaScript libraries during

development.

But what makes React so

useful is that it allows developers to write less code to create functionality.

This in turn makes it easier to maintain code in the long term and

simplifies testing.

Because it is so central to front end development,

there are many tools that you can use with the React.

These tools can help you understand how React is running your code and

how you can improve performance.

Now that you have a better idea of what React is let me tell you more about Reacts

components.

The key concept behind React is that it allows you

to define components that you can combine to build a web application.

A component is basically a small piece of user interface, such as a music player or

photo gallery.

This component model allows you to do several things, such as developing and

testing parts of their application in isolation and it also allows you to reuse

components across multiple sections of the application as well.

Let's explore an example.

Most Web applications have user profiles with pictures.

Normally several parts of the application use a version of this profile picture,

such as in the navigation bar beside the user's name.

It could also show in a user search result along with several other user pictures and

the user picture might also display an instant message notifications.

Instead of programming the logic for

every instance of a user profile picture React allows you to create a user icon

component that displays a version of a profile picture every time it is used.

You can then reuse this component throughout the application.

Can you imagine how it will improve your efficiency.

There are in fact many open source libraries that provide pre made components

for React projects.

For example, if you want to add a video player to your website,

there's a React component library for that.

Well, maybe you want to embed a map.

There's a React component library for that too.

As the React community grows bigger, it continues to add new features and

improvements continuously.

Annual React conferences, encourage community sharing and

introduce upcoming developments in the library.

I hope that in the near future you might be joining the React community as

an active member and

as a future React community member you now know the React libraries purpose and

how its components work together to make life easier for front end developers.

After all,

why build something from scratch if the components are already there to use?

**Case Study: Why did Facebook engineers create React?**

There are a lot of JavaScript Model-View-Controller (MVC) frameworks out there. Why did we build React and why would you want to use it?

**React isn’t an MVC framework.**

React is a library for building composable user interfaces. It encourages the creation of reusable UI components which present data that changes over time.

**React doesn’t use templates.**

Traditionally, web application UIs are built using templates or HTML directives. These templates dictate the full set of abstractions that you are allowed to use to build your UI.

React approaches building user interfaces differently by breaking them into **components**. This means React uses a real, full-featured programming language to render views, which we see as an advantage over templates for a few reasons:

* **JavaScript is a flexible, powerful programming language** with the ability to build abstractions. This is incredibly important in large applications.
* By unifying your markup with its corresponding view logic, React can actually make views **easier to extend and maintain**.
* By baking an understanding of markup and content into JavaScript, there’s **no manual string concatenation** and therefore less surface area for XSS vulnerabilities.

We’ve also created [JSX](https://reactjs.org/docs/jsx-in-depth.html), an optional syntax extension, in case you prefer the readability of HTML to raw JavaScript.

**React updates are dead simple.**

React really shines when your data changes over time.

In a traditional JavaScript application, you need to look at what data changed and imperatively make changes to the DOM to keep it up-to-date. Even AngularJS, which provides a declarative interface via directives and data binding [requires a linking function to manually update DOM nodes](https://code.angularjs.org/1.0.8/docs/guide/directive#reasonsbehindthecompilelinkseparation).

React takes a different approach.

When your component is first initialized, the *render* method is called, generating a lightweight representation of your view. From that representation, a string of markup is produced and injected into the document. When your data changes, the *render* method is called again. In order to perform updates as efficiently as possible, we diff the return value from the previous call to *render* with the new one and generate a minimal set of changes to be applied to the DOM.

The data returned from *render* is neither a string nor a DOM node — it’s a lightweight description of what the DOM should look like.

We call this process **reconciliation**. Check out [this jsFiddle](http://jsfiddle.net/2h6th4ju/) to see an example of reconciliation in action.

Because this re-render is so fast (around 1ms for TodoMVC), the developer doesn’t need to explicitly specify data bindings. We’ve found this approach makes it easier to build apps.

**HTML is just the beginning.**

Because React has its own lightweight representation of the document, we can do some pretty cool things with it:

* Facebook has dynamic charts that render to *<canvas>* instead of HTML.
* Instagram is a “single page” web app built entirely with React and *Backbone.Router*. Designers regularly contribute React code with JSX.
* We’ve built internal prototypes that run React apps in a web worker and use React to drive **native iOS views** via an Objective-C bridge.
* You can run React on the server for SEO, performance, code sharing and overall flexibility.
* Events behave in a consistent, standards-compliant way in all browsers (including IE8) and automatically use event delegation.

Head on over to [https://reactjs.org](https://reactjs.org/) to check out what we have built.

**How React works**

By now you've learned,

that you can build components in React to develop user interfaces.

But how do React components turn into the elements on the web page that you use.

Remember when your web browser receives an HTML page, it constructs a DOM,

to represent it.

But updating the DOM is considered expensive,

because it is very time intensive for the web browser to do so.

Every time the browser DOM is updated, it causes the browser to re compute the page.

Yet many big and popular websites still load in no time today.

How can it be?

In this video, I will explain how React solves the problem,

by computing its own virtual DOM.

Let's start with how React components relate to the website displayed in

your browser.

A reality component,

has a one to one relationship to an HTML element that is displayed on the webpage.

But how does React keep track of which HTML elements need to be updated.

This is where reacts virtual DOM comes into play.

When React builds out its tree of components,

it builds out its own dome in memory called the virtual DOM.

The virtual DOM is a representation of the browser DOM that is kept in memory.

React uses this virtual DOM to update the browser DOM, only when it needs to.

This ensures that your application is fast and responsive to user input.

Let me explain how it works, React checks to see if the HTML components in

the virtual DOM matches the browser DOM.

If a change is required, the browser dome is updated.

If nothing has changed, then no update is performed.

This process is called reconciliation.

Let's break down what happens when you update a component and React.

Firstly, the virtual DOM is updated.

Then React, compares the virtual DOM to the previous version of the virtual DOM,

and determines which elements have changed.

The changed elements, and only those elements are updated in the browser DOM.

Changes on the browser DOM, cause the displayed web page to change.

In this video, you have learned that, React constructs and

updates a virtual DOM.

So that the browser will only update certain HTML elements,

instead of recreating the entire page.

**The Virtual DOM**

* React builds a representation of the browser Document Object Model or DOM in memory called the virtual DOM. As components are updated, React checks to see if the component’s HTML code in the virtual DOM matches the browser DOM. If a change is required, the browser DOM is updated. If nothing has changed, then no update is performed.
* As you know, this is called the **reconciliation** process and can be broken down into the following steps:
  + **Step 1:** The virtual DOM is updated.
  + **Step 2:** The virtual DOM is compared to the previous version of the virtual DOM and checks which elements have changed.
  + **Step 3:** The changed elements are updated in the browser DOM.
  + **Step 4:** The displayed webpage updates to match the browser DOM.
* As updating the browser DOM can be a slow operation, this process helps to reduce the number of updates to the browser DOM by only updating when it is necessary.
* But even with this process, if a lot of elements are updated by an event, pushing the update to the browser DOM can still be expensive and cause slow performance in the web application.
* The React team invested many years of research into solving this problem. The outcome of that research is what’s known as the React Fiber Architecture.
* The Fiber Architecture allows React to incrementally render the web page. What this means is that instead of immediately updating the browser DOM with all virtual DOM changes, React can spread the update over time. But what does "over time" mean?
* Imagine a really long web page in the web browser. If the user scrolls to the bottom, the top of the web page is no longer visible. The user then clicks a button on the bottom of the web page that updates some text on the top of the web page.
* But the top of the page isn’t visible. Therefore, why update it immediately?
* Perhaps there is text currently displayed on the bottom of the page that also updates when the button is clicked. Wouldn’t that be a higher priority to update than the non-visible text?
* This is the principle of the React Fiber Architecture. React can optimize when and where updates occur to the browser DOM to significantly improve application performance and responsiveness to user input. Think of it as a priority system. The highest priority changes, the elements visible to the user, are updated first. While lower priority changes, the elements not currently displayed, are updated later.
* While you’re unlikely to interact with the virtual DOM and Fiber Architecture yourself, it’s good to know what’s going on if issues occur during the development of your web application.
* There are many tools available to help you investigate how React is processing your webpage. The official React Developer Tools web browser plugin developed by Meta will be one of the key tools in your developer toolbox. So, if you do have to look deeper into the code, you’ll have the right toolbox available to help you. These tools will be explored later on.

**Component hierarchy**

Planning an application as

a series of components can be tricky at first,

but after doing it a few times,

it will become set in nature.

We've already learned about the core Internet

technologies and now you know that

JavaScript libraries in React allow you

to develop user interfaces much faster.

Isn't it cool that you can develop

user interfaces so quickly?

In this video, you will learn how

the component hierarchy works in React.

Every React application contains at least one component,

the root component or app component.

Components are added to the app component to build out

a tree structure of components

that make up the application.

Let's explore an example of how

an application might be structured as components.

Suppose your application is a shopping list of

items that needs to be picked up from the grocery store.

At the root we have the app component.

The app component has two child components,

new item bar and shopping list.

The new item by component

allows us to add new items to the list.

The shopping list component contains a shopping item,

child component for each item in the shopping list.

Even though the item itself might be different,

such as chicken or noodles,

it is displayed in the same manner as other items.

Therefore, you can reuse

the shopping item component to display multiple items.

When a user takes off the items,

the list will update and remove

the corresponding shopping item child component

from the shopping list component.

Now that you understand the component hierarchy,

let's explore how a website can

be broken down into components.

As an example, I will talk you

through a typical blog websites.

Normally there is a heading,

a menu, and a search bar.

You also usually have a featured blog posts that

displays larger than previous blog post below it.

Now, let's unpack the component hierarchy.

At the root of the component hierarchy,

you have the app component.

This is the entire webpage.

The app component has to child components,

the Navbar component for

the navigation bar and the Page component.

The Navbar component contains the title of the blog,

navigation links for the website,

along with the search component.

In the page component, we have three components.

The main feature component and

two instances of a small feature component.

The main feature component displays

a blog summary about a specific blog post.

In the small feature component,

there is a blog summary and a thumbnail component

to display an image associated with the blog post.

Notice that the small feature component is used twice.

It is the same piece of code,

but for the second small feature,

the code uses different properties.

That is exactly the benefit of

building a website with React components,

you can reuse code for multiple sections.

Using a series of components to

build an application can be tricky at first,

but after you do it a few times,

you will appreciate how handy it is.

Good luck with planning your

component hierarchies in React.

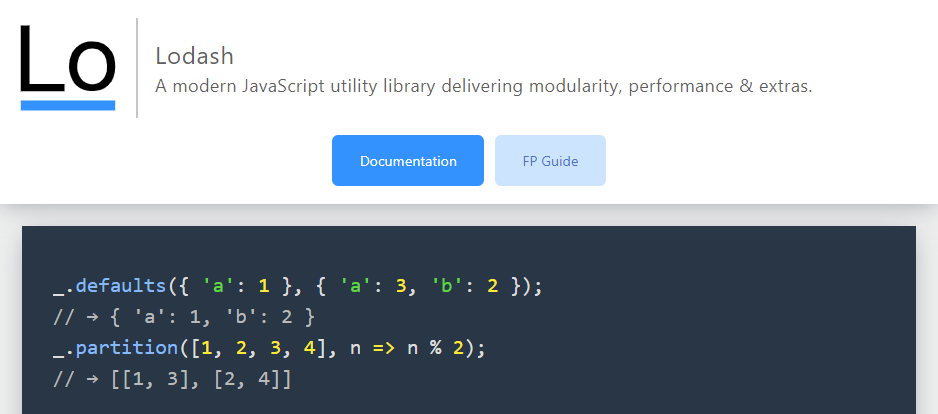
**Alternatives to React**

React is a library and not a framework. This means you'll often use other JavaScript libraries with it to build your application. In this reading, you will be briefly introduced to some JavaScript libraries commonly used with React.

**Lodash**

[Official Website](https://lodash.com/)

As a developer, there's a lot of logic you'll commonly write across applications. For example, you might need to sort a list of items or round a number such as *3.14* to *3*. Lodash provides common logic such as these as a utility library to save you time as a developer.



**Luxon**

[Official Website](https://moment.github.io/luxon/#/)

You'll be working with dates and times often as a developer. Think of viewing a list of orders and when they were placed, or displaying a calendar schedule for an event. Dates and times are everywhere.

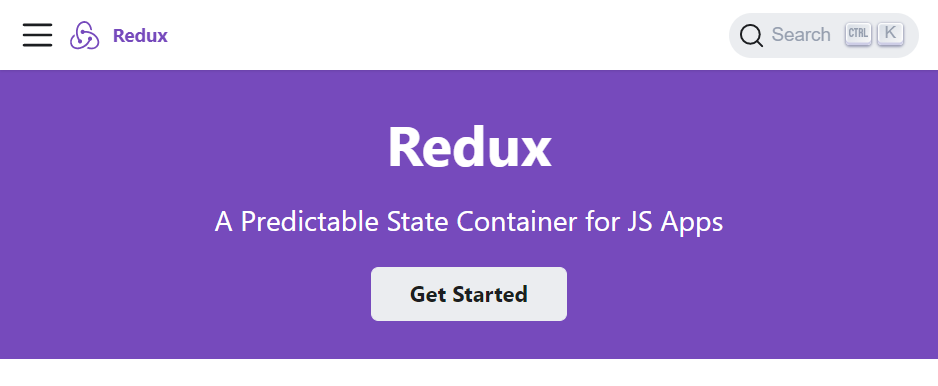
Luxon helps you work with dates and times by providing functions to manipulate and display them. For example, think of how dates are formatted in different countries. In the United States the format is *Month Day Year* but in Europe it is *Day Month Year*. This is one area where Luxon can help you display the date in the user's local format.



**Redux**

[Official Website](https://redux.js.org/)

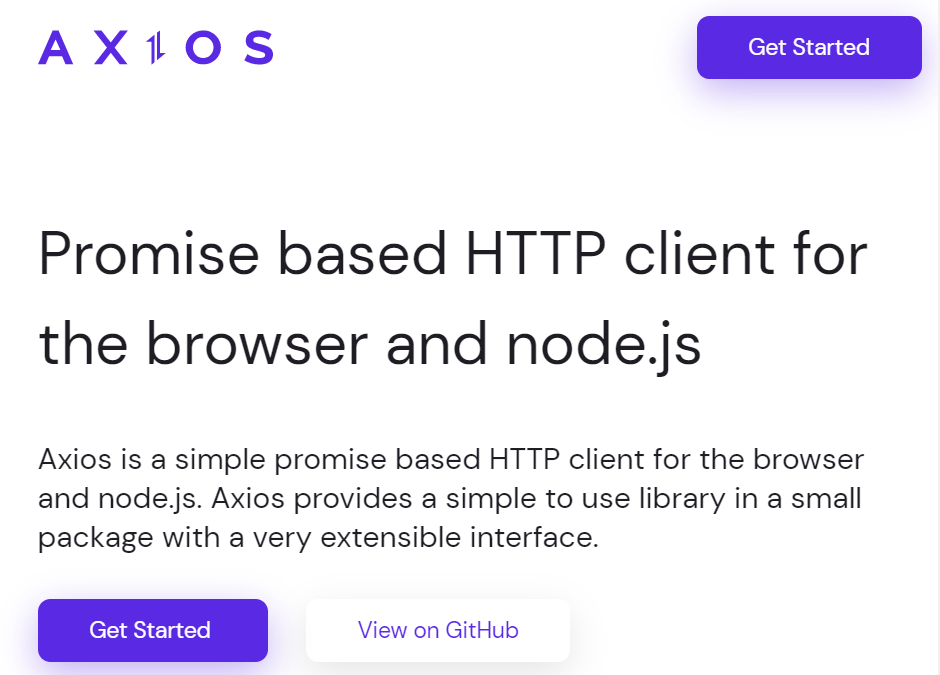
When building a web application, you'll need to keep track of its state. Think of when you shop online. The web application tracks items currently in your shopping cart. When you remove an item from the cart, the application needs to update what displays on the screen. This is where Redux comes in. It helps you manage your application state and even has advanced features such as undo and redo.



**Axios**

[Official Website](https://axios-http.com/)

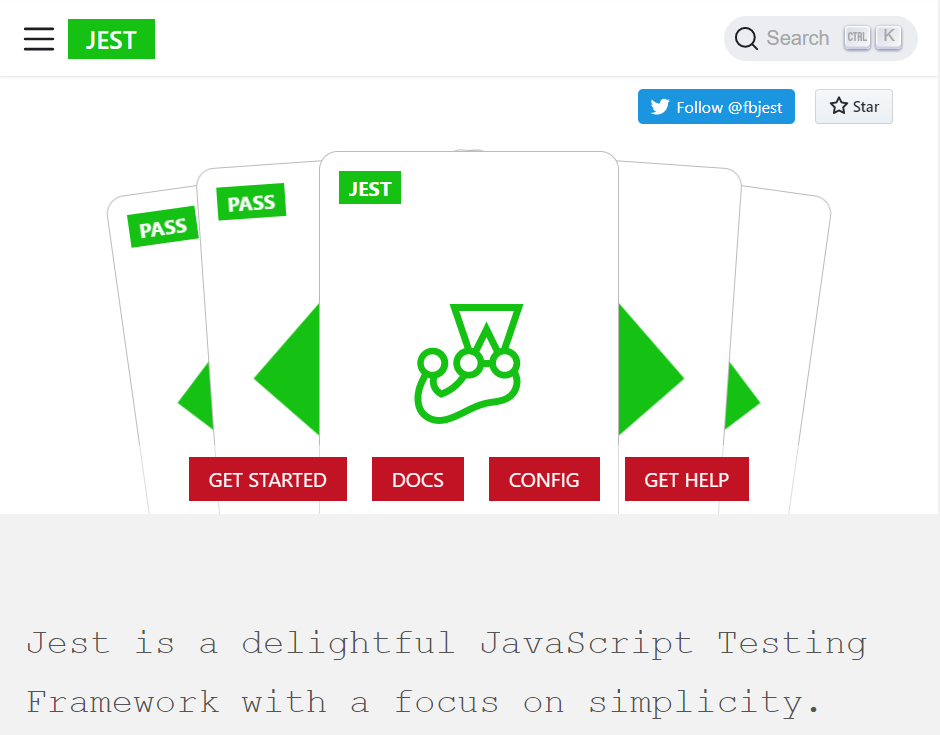
As a developer you'll be communicating with APIs over HTTP frequently. The Axios library helps to simplify sending HTTP requests and processing the response. It also provides advanced features allowing you to cancel requests and to change data received from the web server before your application uses the data.



**Jest**

[Official Website](https://jestjs.io/)

It is good practice to write automated tests for your code as a professional developer. The jest library helps you to do this and works with many libraries and frameworks. It also provides reporting utilities such as providing information on how much of your code is tested by your automated tests.



**Conclusion**

If you're curious to learn more about these libraries, their websites feature setup guides, tutorials and documentation to get started. These libraries will be covered later on.

**Additional Resources**

**Learn more​** Here is a list of resources that may be helpful as you continue your learning journey.

**React Official Website**

<https://reactjs.org/>

**Choosing between Traditional Web Apps and Single Page Apps (Microsoft)**

<https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps>

**React Source Code (Github)**

<https://github.com/facebook/react>

**Introduction to React.js**

*The original video recorded at Facebook in 2013.*

[*https://youtu.be/XxVg\_s8xAms*](https://youtu.be/XxVg_s8xAms)

**Improve your Bio page with Bootstrap Lab**

<!DOCTYPE html>

<html>

<head>

    <title>My Bio Page</title>

    <link href="bootstrap.min.css" rel="stylesheet">

</head>

<body>

    <div class="container">

        <div class="row">

            <div id="bio" class="col-12 col-lg-6 text-center">

                <h1>Petar</h1>

                <img src="photo.jpg" class="img-fluid">

            </div>

            <div id="more" class="col-12 col-lg-6">

                <h2>Favorite Music Artists</h2>

                <ul>

                    <li>Metallica</li>

                    <li>ACDC</li>

                    <li>Slash, Myles Kennedy and The Conspirators</li>

                    <li>Disturbed</li>

                    <li>Slipknot</li>

                </ul>

                <h2>Favorite Films</h2>

                <ol>

                    <li>Gladiator</li>

                    <li>Saving Private Ryan</li>

                    <li>Enemy at the Gates</li>

                    <li>The Lord of the Rings Trilogy</li>

                    <li>Iron Man</li>

                </ol>

                <a href="https://www.meta.com/user/12" class="btn btn-primary">My Meta Profile</a>

            </div>

        </div>

    </div>

    <script src="bootstrap.bundle.min.js"></script>

</body>

</html>