**W3Cx-1of5-CSS Basics**

Welcome to the CSS Basics course!



VK7JG-NPHTM-C97JM-9MPGT-3V66T (default product key w10pro)

FMQ78-8TNG8-MVV3P-M6F34-F6YP6

There are 3 methods;

1) HTML & CSS in the same file   
file name : index.html -------------------  
<html>  
<head>  
<title>Home</title>  
<style>  
p  
{color:green;}  
</style>  
</head>  
<body>  
<p>Peter is a good man</p>  
</body>  
</html>

2) CSS code inserted directly in HTML tags   
file name : index.html -------------------  
<html>  
<head>  
<title>Home</title>  
</head>  
<body>  
<p style="color:green;">Peter is a good man</p>  
</body>  
</html>

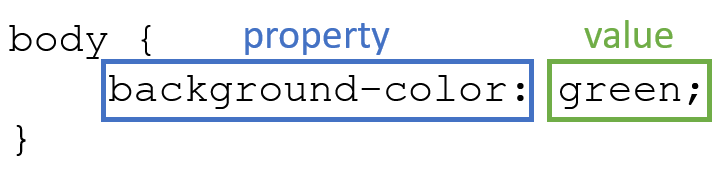
3) CSS & HTML separated but in the same folder  
Html file: index.html -------------------  
<html>  
<head>  
<title>Home</title>  
<link href="style.css" rel="stylesheet" type="text/css" >  
</head>  
<body>  
<p>Peter is a good man</p>  
</body>  
</html>  
-----------------------------------------------   
CSS file: style.css --------------------------  
p  
{color:green;}

Hello,

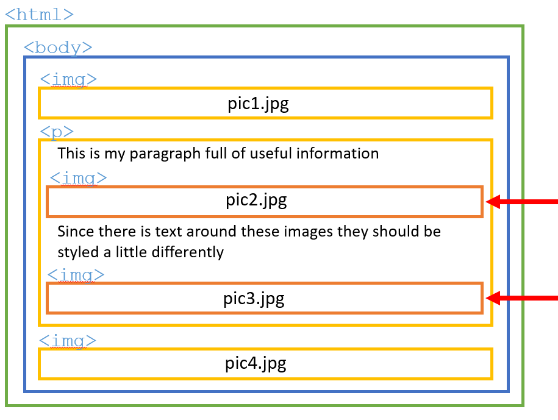
Thank you for joining this course, where you will learn how to take your Web pages from bland to bold with the power of CSS styling. You will review historical and current design trends, and see how to apply the concepts in a beautiful way, use relative positioning, and much more. Bring your HTML knowledge, and plan to spend 5 to 7 hours on each of five modules, including lots of hands-on practice.

### Course outline

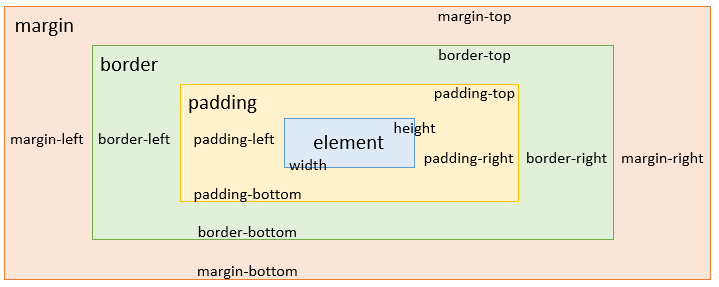
* The first set of lessons are available in **"Module 1: Getting started with CSS"**, where we show different examples of the amazing things you can do with CSS. This module includes a cameo of Bert Bos, co-inventor of CSS.



* In **Module 2**, we will first review HTML and then learn how to build CSS rules with some basic properties that adjust your HTML document's color, font or spacing.



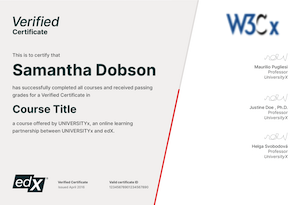
* In **Module 3**, we are going to build slightly more complex Web pages using contextual selectors, IDs, classes and pseudo-classes. The idea is to combine some selectors together into one or more CSS rules. And thanks to the "cascading" part of *Cascading Style Sheets*, it won't be a problem to have multiple CSS rules competing for importance. Even with multiple rules that apply to the same HTML tag, the browser will know how to make it look!



* **Module 4** is about layout and positioning using CSS. You will start by understanding the box model - the foundation of layout on the Web.
* You will then learn about relative positioning and the float property in order to position your HTML elements in relation to one another on a Web page.
* In **Module 5** (final!), we are going to talk about the design aspect of Web pages and how to give your users the best experience. We will discuss lessons learned from historical Web design trends and then meet the newest fashions in Web design, to help you give your Web pages a modern look and feel. And we'll remind you not to forget accessibility and internationalization so as to accommodate an even wider, diverse audience.

### "CSS Basics" Verified Certificate

W3C provides a [W3Cx Verified Certificate for this course](https://ecommerce.edx.org/basket/add/?sku=35F2C9C) that lets you demonstrate to employers, colleagues, and professional organizations that you successfully completed a course that increased your mastery of CSS. This certificate can be used to highlight newly acquired skills on your resume or [LinkedIn profile](https://support.edx.org/hc/en-us/articles/206501938-How-can-I-add-my-certificate-to-my-LinkedIn-profile-).



Once you have a passing score (70% or higher), your certificate will be automatically generated and accessible from your progress page. Please write to [billing@edx.org](mailto:billing@edx.org) if you encounter an issue with the payment of your certificate.

W3Cx, as non-profit, relies on verified certificates to help fund future W3Cx courses for everyone globally.

 "Front-End Web Developer" Professional Certificate



This CSS Basics course is part of the [Front-End Web Developer" (FEWD) Professional Certificate](https://www.edx.org/professional-certificate/w3cx-front-end-web-developer) program. To get this FEWD certificate, you will need to successfully pass all 5 courses that compose that program. Find out more on [w3cx.org](https://w3cx.org/)!

If you already have a verified certificate in one or more of these courses, you *do NOT need to re-take that course*.

### Additional information:

* If you are new to the edX platform, we encourage you to [check out DemoX](https://www.edx.org/course/demox), a quick walk-through of an edX experience. It will help answer basic “how to's” on taking an edX course.
* As an alternative, do not hesitate to browse the [courses' FAQs in edX Help Center](https://support.edx.org/hc/en-us/categories/115002269607-Courses).

We sincerely hope that you will enjoy this W3Cx course.

- The W3Cx team



This course is designed for anyone, no prerequisites required.

### During this course, you will learn:

* Basic selection of CSS properties
* Fundamental CSS selectors
* Designing page layout through relative positioning with CSS
* Best practices in Web design

Have fun!

## Course outline

### Course information

- Welcome to "CSS Basics"  
- Course outline, grading, due dates and verified certificate  
- Course practical information  
- Course tools

### Module 1: Getting started with CSS

1.1 Introduction to Module 1  
1.2 What is CSS?  
1.3 Why CSS is important  
1.4 Project 1 - Your first CSS  
1.5 Conclusion and exercises

### Module 2: Building CSS rules

2.1 Introduction to Module 2  
2.2 HTML review  
2.3 Building a CSS rule  
2.4 Attaching CSS to HTML using selectors  
2.5 Applying styles using CSS properties  
2.6 Style studies  
2.7 Project 2 - About me page  
2.8 Conclusion and exercises

### Module 3: Specific HTML element targeting with CSS selectors

3.1 Introduction to Module 3  
3.2 Using HTML classes and IDs  
3.3 CSS pseudo-classes  
3.4 Combining selectors  
3.5 Style studies  
3.6 Project 3 - My profile  
3.7 Conclusion and exercises

### Module 4: Layout and positioning

4.1 Introduction to Module 4  
4.2 The basics of layout  
4.3 Floating elements  
4.4 Relative positioning  
4.5 Style studies  
4.6 Project 4 - My resume  
4.7 Conclusion and exercises

### Module 5: Designing your Web site for your audience

5.1 Introduction to Module 5  
5.2 The basics of design  
5.3 Designing for your audience  
5.4 Historical Web design trends  
5.5 Current Web design trends  
5.6 Conclusion and final exam

## W3C "Front-End Web Developer" professional certificate

Professional Certificate programs are series of courses designed by industry leaders and/or top universities to build and enhance critical professional skills needed to succeed in today's most in-demand fields.



W3C has designed a ["Front-End Web Developer" (FEWD) Professional Certificate](https://www.edx.org/professional-certificate/w3cx-front-end-web-developer) where you will learn all of the necessary skills needed to build interactive and responsive user experiences on the Web. This program will deepen your knowledge of the 3 foundational languages that power the Web: HTML5, CSS and JavaScript. The W3C FEWD program is composed of 5 courses:

* 1. This course: CSS Basics (W3Cx-1of5-CSS Basics)
  2. HTML5 and CSS Fundamentals (W3Cx-2of5-HTML5 and CSS3 Fundamentals)
  3. HTML5 Coding Essentials and Best Practices (W3Cx-3of5-HTML5 Coding Essentials and Best Practices)
  4. HTML5 Apps and Games W3Cx-4of5-HTML5 Apps and Games)
  5. JavaScript Introduction (W3Cx-5of5-JavaScript Introduction)

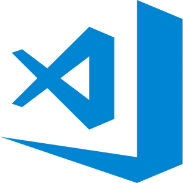
To get a W3C FEWD professional certificate, you must successfully pass and receive a Verified Certificate in the five courses above. Note that the [dashboard link to Front-End Web Developer Program](https://courses.edx.org/dashboard/programs/98b7344e-cd44-4a99-9542-09dfdb11d31b/) displays upgrade links to all of the courses in the program AND offer the option to "upgrade all remaining courses" in bulk.

***IMPORTANT:*** If you already have a verified certificate in one or more of these courses, you do NOT need to re-take that course.

## Web editors

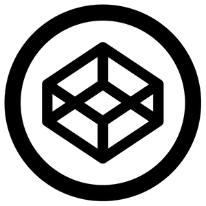
While any text editor, like NotePad or TextEdit, can be used to create Web pages, they don't necessarily offer a lot of help towards that end. Other options offer more facilities for error checking, syntax coloring and saving some typing by filling things out for you.

One caveat, you need to make sure that whatever editor you choose will save to a "plain text" format (".txt"). Some editors will save to other formats by default (.rtf, .doc, etc.) so you need to make sure you save your source code as plain text.



For my demos, I will be using a software program called [Visual Studio Code](https://code.visualstudio.com/) (VS Code). You can download it for free, and for any operating system.

VS Code supports a large array of different Web technologies, but for this course we will be using it to develop our HTML and CSS. It also includes a feature called intellisense which you might have used in other code editors. It will suggest code for you to use, highlight keywords and a lot of other cool features you'll see me use throughout the course.



In this course, we'll also use [Codepen](https://www.codepen.io/" \t "_blank).  
Codepen incorporates editors and a preview service into a single Web application. This is a great service to get you started quickly as it doesn't require you to download anything and you can access the service, along with your saved projects from any Web browser.

Other than these two there are plenty of different programs you can use.

Here are a few possibilities for Web editors:

* [Notepad](https://notepad-plus-plus.org/) - on Windows PC's
* [Visual Studio Code](https://code.visualstudio.com/) - a free open source editor that can run on any operating system. Many developers are already familiar with Visual Studio Code. Many coding videos in this course are demonstrated using Visual Studio Code.
* [XCode](https://developer.apple.com/xcode/) - Mac developers may be familiar with XCode
* TextEdit - this is available on Macs, but be sure you're [saving as plain text](https://discussions.apple.com/thread/1058381?answerId=5014514022#5014514022), not as a ".rtf" or ".doc" file.
* [Sublime Text](https://www.sublimetext.com/) - it is quite popular with developers, though there can be a bit of a learning curve to use its many features.
* [BlueGriffon](http://bluegriffon.org/) - a WYSIWYG ("What You See Is What You Get") content editor for the World Wide Web. Powered by Gecko, the rendering engine of Firefox, it's a modern and robust solution to edit Web pages in conformance to the latest Web Standards.
* [Atom](https://atom.io/) - another cross platform editor, created by [GitHub](https://github.com/).
* [Vim](https://www.vim.org/) or [Emacs](https://www.gnu.org/software/emacs/) - great editors on which the Internet was built, but if you're not already familiar with these, this isn't the time to try.

## W3C validators

The W3C develops and hosts [**free** and **open source** tools](https://w3c.github.io/developers/) used every day by **millions of Web developers and Web designers**. All the tools listed below are Web-based, and are available as downloadable sources or as free services on the [W3C Developers tools](https://w3c.github.io/developers/tools/)site.

### W3C Validator

The [W3C validator](https://validator.w3.org/) checks the [markup validity](https://validator.w3.org/docs/help.html#validation_basics) of various Web document formats, such as HTML. Note that you are automatically directed to the [Nu Html Checker](https://validator.w3.org/nu/) when validating an HTML5 document.

### CSS Validator

The [CSS validator](https://jigsaw.w3.org/css-validator/) checks Cascading Style Sheets (CSS) and (X)HTML documents that use CSS stylesheets.

### Unicorn

[Unicorn](https://validator.w3.org/unicorn/) is W3C's unified validator, which helps people improve the quality of their Web pages by performing a variety of checks. Unicorn gathers the results of the popular HTML and CSS validators, as well as other useful services, such as Internationalization, RSS/Atom feeds and http headers.

### Internationalization Checker

The [W3C Internationalization Checker](https://validator.w3.org/i18n-checker/) provides information about various internationalization-related aspects of your page, including the HTTP headers that affect it. It will also report a number of issues and offer advice about how to resolve them.

### Link Checker

The [W3C Link Checker](https://validator.w3.org/checklink) looks for issues in links, anchors and referenced objects in a Web page, CSS style sheet, or recursively on a whole Web site. For best results, it is recommended to first ensure that the documents checked use valid [(X)HTML Markup](https://validator.w3.org/) and [CSS](https://jigsaw.w3.org/css-validator/).

## W3C cheatsheet

The [W3C cheatsheet](https://www.w3.org/2009/cheatsheet/) provides quick access to useful information from a variety of specifications published by W3C. It aims at giving in a very compact and mobile-friendly format a compilation of useful knowledge extracted from W3C specifications, completed by summaries of guidelines developed at W3C, in particular Web accessibility guidelines, the Mobile Web Best Practices, and a number of internationalization tips.

[](https://www.w3.org/2009/cheatsheet/)

Its main feature is a lookup search box, where one can start typing a keyword and get a list of matching properties/elements/attributes/functions in the above-mentioned specifications, and further details on those when selecting the one of interest.

The W3C cheat sheet is only available as a [pure Web application](https://dev.w3.org/2009/cheatsheet/doc/).

## Browser compatibility

The term browser compatibility refers to the ability of a given Web site to appear fully functional on the browsers available in the market.

The most powerful aspect of the Web is what makes it so challenging to build for: its universality. When you create a Web site, you’re writing code that needs to be understood by many different browsers on different devices and operating systems!

To make the Web evolve in a sane and sustainable way for both users and developers, browser vendors work together to standardize new features, whether it’s a new [HTML element](https://developer.mozilla.org/en-US/docs/Web/HTML/Element), [CSS property](https://developer.mozilla.org/en-US/docs/Web/CSS/Reference#Keyword_index), or [JavaScript API](https://developer.mozilla.org/en-US/docs/Web/API). But different vendors have different priorities, resources, and release cycles — so it’s very unlikely that a new feature will land on all the major browsers at once. As a Web developer, this is something you must consider if you’re relying on a feature to build your site.

We are then providing references to the browser support of HTML5 features presented in this course using 2 resources: [Can I Use](https://caniuse.com/) and [Mozilla Developer Network (MDN) Web Docs](https://developer.mozilla.org/en-US/).

### Can I use

[Can I Use](https://caniuse.com/) provides up-to-date tables for support of front-end Web technologies on desktop and mobile Web browsers. Below is a snapshot of what information is given by CanIUse when searching for "CSS3 colors".



### MDN Web Docs



To help developers make these decisions consciously rather than accidentally, [MDN Web Docs](https://developer.mozilla.org/) provides browser compatibility tables in its documentation pages, so that when looking up a feature you’re considering for your project, you know exactly which browsers will support it.

### External resources

* [MDN browser compatibility data: Taking the guesswork out of web compatibility](https://hacks.mozilla.org/2018/02/mdn-browser-compatibility-data/)
* [Caniuse and MDN compatibility data collaboration](https://hacks.mozilla.org/2019/09/caniuse-and-mdn-compat-data-collaboration/)

## 1.1.1 Welcome to Module 1

Hi! Welcome to the world of CSS or Cascading Style Sheets.

In this module, we're going go over exactly what CSS is, and introduce you to how important it is and making effective Web sites.

We're going to cover the Web before CSS, and how it came about. Show you awesome examples of all the powerful things you can do with CSS. Walk you through setting up your machine so that you can write and test your own HTML CSS Web sites.

And then finally by the end of the module you write your own first piece of CSS.

## 1.1.2 Module 1 outline

### In this module, we will:

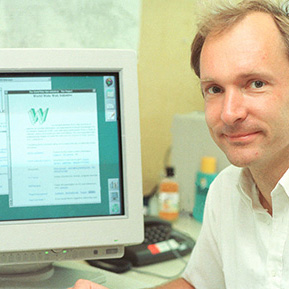
1. Answer the question "What is CSS?"
2. Review the state of the Web before CSS, and how it came about
3. Show different examples of the amazing things you can do with CSS
4. Help you write your first CSS styles

## 1.2.1 What is W3C?

### W3C logo

### As steward of global Web standards, W3C's mission is to safeguard the openness, accessibility, and freedom of the World Wide Web from a technical perspective.

W3C's primary activity is to develop protocols and guidelines that ensure long-term growth for the Web. The widely adopted Web standards define key parts of what actually makes the World Wide Web work.



Tim Berners-Lee at his desk in CERN, 1994

### A few history bits

[Tim Berners-Lee](https://www.w3.org/People/Berners-Lee/) wrote a [proposal](https://www.w3.org/History/1989/proposal.html) in 1989 for a system called the World Wide Web. He then created the first Web browser, server, and Web page. He wrote the first specifications for URLs, HTTP, and HTML.

In October 1994, Tim Berners-Lee founded the World Wide Web Consortium (W3C) at the Massachusetts Institute of Technology, Laboratory for Computer Science [MIT/LCS] in collaboration with [CERN](https://home.cern/), where the Web originated (see information on the [original CERN Server](https://www.w3.org/Daemon/)), with support from DARPA and the [European Commission](https://ec.europa.eu/index_en.htm).

In April 1995, [Inria](https://www.inria.fr/" \t "_blank) became the first European W3C host, followed by [Keio University of Japan](https://www.keio.ac.jp/) (Shonan Fujisawa Campus) in Asia in 1996. In 2003, [ERCIM](https://www.ercim.eu/) took over the role of European W3C Host from Inria. In 2013, W3C announced [Beihang University](https://ev.buaa.edu.cn/" \t "_blank) as the fourth Host.

### Sir Tim Berners-Lee named recipient of the ACM A.M. Turing Award



On 4 April 2017, the ACM, the Association for Computing Machinery, named Sir Tim Berners-Lee, inventor of the Web and Director of the World Wide Web Consortium, as the recipient of the 2016 [ACM A.M. Turing Award](https://amturing.acm.org/). The Turing award is recognized as the highest distinction in Computer Science and is often referred to as the “Nobel Prize of Computing”.

Sir Tim is being given this award for inventing the World Wide Web, the first Web browser, and the fundamental protocols and algorithms allowing the Web to scale. The Web is considered one of the most influential computing innovations in history.

## 1.2.2 The Web is amazing!

The Web is an application of the Internet. The Web is the most popular way of accessing the Internet, but other applications of the Internet are [e-mail](https://en.wikipedia.org/wiki/Email)and [ftp](https://en.wikipedia.org/wiki/File_Transfer_Protocol) for example. One analogy equates the Internet to a road network where the Web is a car, the email is a bicycle, etc.  Read [this article](https://www.lifewire.com/difference-between-the-internet-and-the-web-2483335) for more details about the difference between Internet and the Web.

The W3C community is passionate about creating free and open Web standards. This course discusses one of these Web standards, CSS, in detail. The next video, created in partnership with Microsoft, explains why standards are important to maintain a royalty-free, Open Web Platform, as well as to help shape the Web of the future.

### Video: The Web is amazing!

The Web is amazing.

You can play games with friends on different continents, find answers to any question in seconds, see a live feed of our planet from space, translate languages with a single click, share a recent photo from a fun trip, or tell everyone you know about your cool, new job.

But it wasn’t always like this.

It wasn’t always a beautiful, infinite world of knowledge and activity.

It’s easy to forget how disconnected we were before the Web.

It was a lot harder to find and share information.

So many of our online experiences today are powered by standards under the hood, fruits of years of collaboration.

Without developers working together, we would need different browsers to view different sites, and search engines would be much less powerful, if the Web were split into silos.

In fact, it’s safe to assume that without today’s connected tech landscape, we’d probably be in the middle of a zombie apocalypse, fending off maniacal desert gangs, and watching our civilization crumble into a seemingly endless vacuum of nothing.

A word of advice:

Nobody ever beat zombies alone.

This glorious Web is our collective responsibility, and we need to treat it as such.

It’s a rich, valuable digital world, but it wouldn’t hold together without the standards we use every day.

How can we take care of it?

By taking action.

By coding, writing, designing, discussing, and defending the open Web.

Thank you for supporting Web standards.

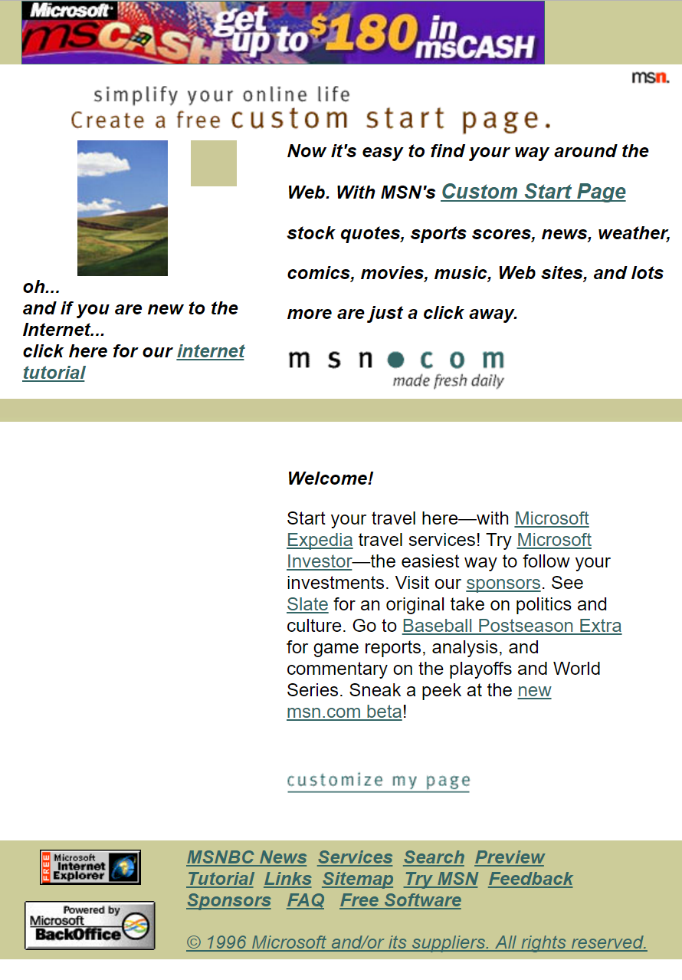
Please help us shape the next era.

Join our efforts to build an open web platform for the future.

## 1.2.3 The invention of CSS

Before CSS, the appearance of a Web page was dictated by HTML, which had very few visual styling tools, meaning most Web pages looked simple.  
Let's take the example of the MSN Web site. MSN is the web portal and related collection of Internet services and apps for Windows and mobile devices, provided by Microsoft and launched on August 24 1995, the same release date as Windows 95.

In 1996, see how it looked like (or check it [online](https://web.archive.org/web/19961022175327/http:/msn.com/)):



For a fun time, warp check out an [archived version of that link above](https://web.archive.org/web/19961026005907/http:/msn.com/tutorial/default.html) for anyone "new to the internet". See how this [MSN Web site](https://www.msn.com/)  (made with lots of CSS) looks like now!

### Before and after CSS

#### Before CSS:

* all documents looked very similar - it was difficult for different companies to express their brand identities in documents
* possibilities for styling were very limited and style was difficult to control and maintain - style had to be applied to content directly, so you couldn't update style without having to touch content and vice versa.

#### After CSS:

* Content authors didn't have to worry about style, they could just focus on content
* Content authors didn't have to worry about what device users would view their document on, those considerations could be handled by the CSS
* Style became much more efficient- a single rule could apply to multiple elements and a single style sheet could apply to multiple documents.

In the video below, you will hear from Bert Bos, the co-creator of CSS, about how he and Håkon Lie developed CSS.

### Video: The history of CSS, by Bert Bos, co-inventor of CSS

In 1994, it was not possible for an author to style HTML documents. Several browsers had style sheets, but only for the user.

On the one hand, that was an advantage: authors didn't have to worry about the style, unlike when they were writing in a word processor.

They could concentrate on the contents.

HTML viewers automatically laid out each document in a way adapted to the user's preferences.

The author didn't have to know anything about screen size, or if the screen had colors, or even whether the device has a screen at all.

But then designers came to the Web, and they found it was also a disadvantage.

A document from company A looked exactly the same as one from company B.

They had to be a way for the author to add some style if he wanted to.

The new Netscape browser, which was quickly becoming the most popular browser on the Web in 1994, added some proprietary HTML elements for style and spacing.

But most people agreed that wasn't the best solution.

When Håkon Lie and I decided to combine our two proposals, in October 1994, it quickly became the most popular.

We called it CSS.

A style sheet language for the Web had to be simple to write: everybody who could write HTML should also be able to make style for it.

It had to be efficient.

Users have to download Web documents over a network, CSS allows to style many elements with one rule, and many documents with one style sheet.

CSS also had to be efficient in the sense that it doesn't take noticeably more time to display a document with a style sheet.

We had to try and make CSS easy to maintain.

Style may change, so they should not just be easy to write, but also easy to read and to modify.

We wanted the language to be extensible: at the time HTML certainly wasn't finished yet and would probably not to be the only language on the Web, CSS itself also would not be complete right away.

Maybe the most important characteristic of CSS is that it preserves the ability for the user to override or ignore the style of a document.

We didn't see CSS as the final solution, certainly not the final solution for complex documents.

Although, already in 1996 we also wrote down some ideas for how to add layout templates and columns to CSS.

Personally, I thought that CSS was a good start but would probably not be enough in the long term.

I expected that after a few years of gaining experience with CSS, we would start writing a better, more powerful style sheet language.

So, that maybe, in ten years time, CSS would be more or less forgotten.

Indeed, in 1997 within W3C we started the XSL language for advanced layout.

But CSS proved to be so extensible and so popular that now in 2016, it is still the primary style language of the Web and we are still adding new features to it.

We must have done something right.

I'm especially happy that despite all the things we added to CSS over the years, the style sheets that that people made with CSS in 1996 are still valid and still work the same.

## 1.2.4 Definitions

CSS, or **C**ascading **S**tyle Sheets, is a style sheet language used to describe the way an HTML or XML document should look to a user. CSS is where you specify the color, size, spacing, font and other visual aspects of the content that you create in your markup language document.

Most often you will see CSS used alongside HTML to describe the way a Web page looks and feels. You can have a Web page without CSS, but it would be very difficult to make it look the way you want with just HTML. This is why almost every Web page is a combination of HTML and CSS.

**CSS** • /si-ɛs-ɛs/ • *noun*

Stands for "**C**ascading **S**tyle **S**heets". A style sheet language for describing how to display an HTML document.

### Sample CSS document:

1. body {
2. background-color: #d0e4fe;
3. }
4. h1 {
5. color: orange;
6. text-align: center;
7. }
8. p {
9. font-family: "Times New Roman";
10. font-size: 20px;
11. }

**HTML** • /eɪʧ-ti-ɛm-ɛl/ • *noun*

Stands for "**H**yper**T**ext **M**arkup **L**anguage", and it is the primary document format on the Web. It is a standardized system for tagging content on a web page so that a web browser knows how to present it properly to the viewer. It is a standardized way to describe a document's structure and the roles of the different parts of that document.

### Sample HTML document:

### <!DOCTYPE html>

<html lang="en">

     <head>

        <meta charset="utf-8">

       <title>My HTML page</title>

    </head>

    <body>

        <p> This is an HTML document </p>

    </body>

</html>

**Web page** • /wɛb peɪʤ/ • *noun*

A hypertext document connected to the World Wide Web.

synonyms: website, home page, landing page

### Internet

A global system of computer networks that connect to one another so that billions of different devices all over the world can share data. The internet is a collection of smaller networks that all combined create one large, all-inclusive global network.

### World Wide Web

A collection of documents linked together by hypertext links, addressed using Uniform Resource Locators (URLs) accessible on the Internet. The World Wide Web is an application of the internet.

abbreviated as WWW or "the Web"

**Web browser** • /wɛb ˈbraʊzər/ • *noun*

A software application for retrieving, presenting and traversing information resources on the World Wide Web.

examples: edge, chrome, Firefox, internet explorer, safari, opera

#### HTTP

Stands for "**H**yper**t**ext **T**ransfer **P**rotocol". It is a protocol managed by the W3C to dictate the manner in which Web pages share data on the World Wide Web. You might recognize this from the start of many Web addresses.

#### HTTPS

Stands for "**H**yper**t**ext **T**ransfer **P**rotocol **S**ecure". It  is the secure version of **HTTP**, the protocol over which data is sent between your browser and the Web site that you are connected to. It means all communications between your browser and the Web site are encrypted. A vast majority of Web site are now served by [https](https://en.wikipedia.org/wiki/HTTPS).

## 1.2.5 Activity - The Web before CSS

Now it's your turn to do some exploration! For this activity, your job is to find examples of Web sites before and after CSS.

A great place to start is at [archive.org](https://archive.org/) (aka, the "WayBack machine") which stores copies of web pages throughout history. You can search for some of your favorite websites and see if they have stored copies older than 1996. You should find that any Web page made before 1996 will look very different than Web sites we typically see today. When you find a real retro gem, please share it in this week's discussion (see below).

## 1.3.1 Separating content from presentation

Up until now, we have been discussing CSS's role within a Web site as the "presentation" component, but what is that and why is it so important?

From the history of CSS, we learned why CSS came about, but the short answer is simply because HTML was never designed to describe the way a Web page was supposed to look. When we use HTML for what it was intended to do, describe content, it leaves space for CSS to properly control a page's visuals. This makes it very easy to update or add content without having to even touch the style.

### Some benefits of CSS:

* CSS has a host of specialized tools to give you powerful control over the look and feel of your Web site, much more powerful than the tools provided by HTML.
* Designers can style many HTML pages with a single CSS document for a consistent look and feel across an entire Web page and less code to maintain.
* Separation of content and presentation makes Web site maintenance much simpler as you can address updates in isolation.
* Over time more and more devices have become internet-capable, and now there are so many different orientations in which your user can view your content. With CSS, you can specifically cater the style to each device to ensure an optimal experience.
* Some users have specific presentation needs based on personal or technological limitations or preferences. Separating content from presentation allows these users the option to control how they view content.
* Before CSS visual elements were almost always achieved with static images, which can have a big affect on network performance. CSS provides an optimized way to style your page so it can load complex visuals quickly.

### External resources:

* [CSS design principles](https://www.w3.org/TR/CSS22/intro.html#design-principles)
* [Effective Use of Style Sheets](https://www.nngroup.com/articles/effective-use-of-style-sheets/) (updated regularly since 1997!)
* [Repurposing of content](https://www.w3.org/People/Bos/DesignGuide/repurposing)

## 1.3.2 Meet CSS Zen Garden

In 2003, [Dave Shea](http://daveshea.com/projects/zen/) began a legendary project called [CSS Zen Garden](http://www.csszengarden.com/) that provided a demonstration of "what can be accomplished through CSS-based design", until submissions stopped in 2013.

In July 2020, Stephanie Eckles ([ModernCSS](https://moderncss.dev/" \t "_blank)) launched [Style Stage](https://stylestage.dev/), a modern CSS showcase styled by community contributions. Meaning that CSS practitioners of any skill level (you!) are welcome to join as a [contributor](https://stylestage.dev/#contribute) and submit a stylesheet! Visit the ["All Styles"](https://stylestage.dev/styles/) directory page and re-style by submitting an alternate style sheet.

The videos below introduce you to the "[CSS Zen Garden](http://www.csszengarden.com/)" Web project.

### Video: Meet CSS Zen Garden

Now that we know exactly how CSS came about, let's talk about how people are using it today.

CSS has empowered people to separate the content, the text, image, videos, all of the stuff that you've created from the way it's presented: the fonts, the colors, the spacing, the backgrounds, so that you can make as many design decisions in your style sheets and keep them separate from all of your content.

There's a particular online community, called CSS ZEN GARDEN, that really likes to play up this advantage.

What they do is, they take a single HTML page and they'll create many different CSS interpretations how to present it.

Let's take a look.

### Live coding video: CSS Zen Garden Demo

This is the homepage for CSS Zen Garden.

This design was submitted by someone just like you, somewhere out on the Internet, and they chose it to be their homepage.

In fact, the homepage rotates between different designs.

Let's take a closer look. If I right-click, I can view the source and I can actually look at the HTML behind this Web page.

I can even comment here and inspect all of the different items, and if I click through, I can find this h1 tag.

That's where the text CSS Zen garden is coming from.

The HTML is here on the left, and here on the right, there's a ton of style associated with it.

In fact, if I scroll through, I can even find the color.

Just to be sure that this h1 tag is corresponding to this title,

let's change the color. Let's try purple.

Oh yeah! That's looks alright. Not great.

But, I definitely am sure that this text corresponds to this HTML tag corresponds to this CSS.

In fact, let's take a look at what it’s like without any CSS.

If I come here to the head, I can actually delete the link to the CSS, and now I have a pure HTML Web site. In fact, look at how my style is deleted.

This is what your Web site would look like without any CSS.

Not super-exciting.

Well in fact, let's get back to the CSS.

Much better! Well, except for that purple text.

Let's take a look at some other designs.

These are the designs from all over the Internet.

Here, we see another completely different design, but that same text!

In fact, if I inspect this element, I can see that it's in that same h1 type.

But look at all the style is different.

If I scroll down, I can even find the color.

Purple doesn't look much better on this page either.

In fact, if I get rid of it, all the HTML, or if I give up all the CSS again, it takes me back to that same core text.

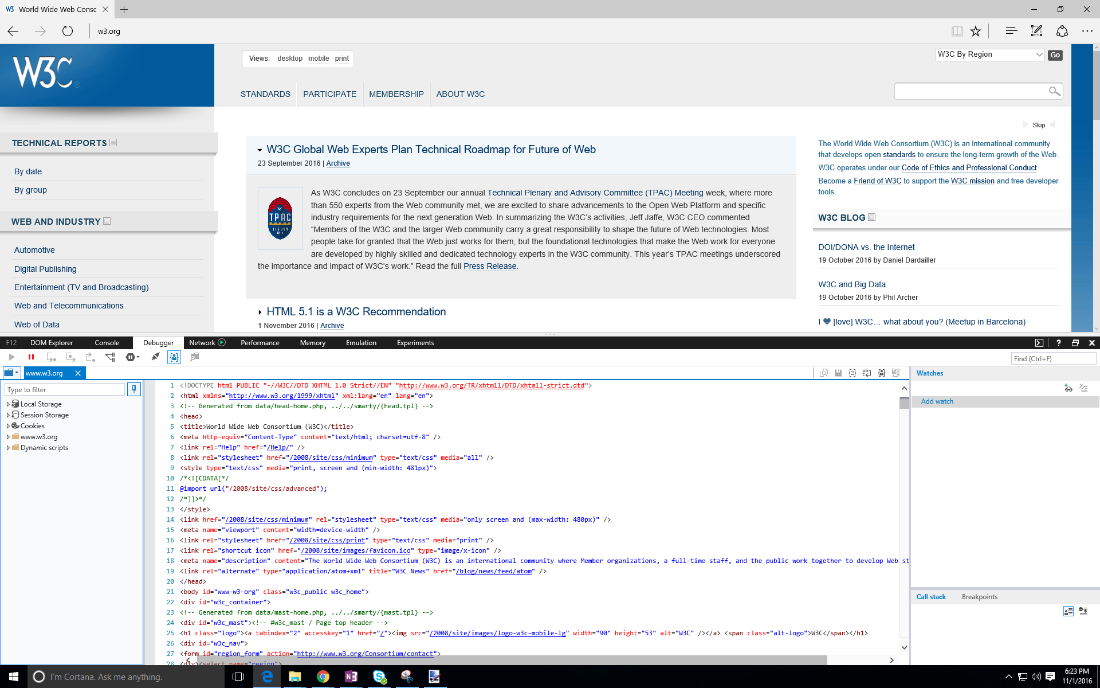
Now you can have a better idea between what part is HTML and what part is CSS.

In fact, now you should look through a few of the different designs, so you can understand what CSS is and get inspired!

### View source and browser tools

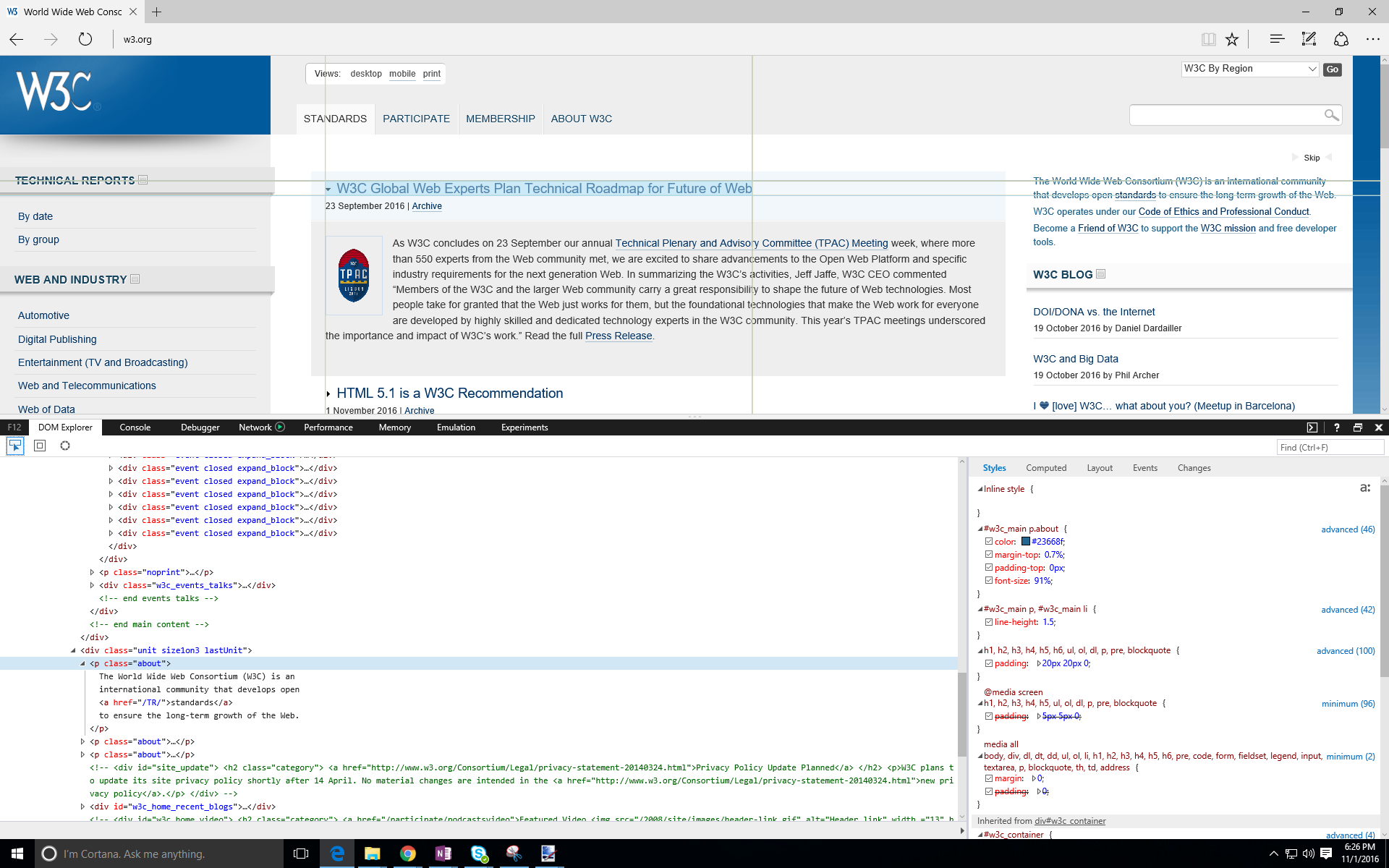
In the above demo, you saw me using what is called the "developer tool" within my Edge Web browser to inspect and real-time change the style of a page's CSS. You can actually right click on any site and choose to look at the code that creates it. This [feature](https://en.wikipedia.org/wiki/Web_development_tools#Web_developer_tools_support) exists in most modern browsers.

Here is what it looks like when I view the source of [W3C's Web site](https://www.w3.org/) (snapshot taken in Sept. 2016):



You can see a window that popped up from the bottom with all the HTML code for that site. Other Web browsers might pop this up in a separate window.

You can also get more specific and look at individual HTML elements with the "Inspect element" option. Here is what it looks like in Edge when I inspect a specific title:



As you can see, not only is the element highlighted on the page, but this also highlights the HTML code and shows you the CSS for that element on the right-hand side. In the video above, you can see me use this view to change the CSS and HTML real-time, which can be a very convenient way to play around with your designs.

As you work in your own sites, you might want to use both of these features of your browser to understand what is happening in your own code, or in Web pages you find on the Web.

## 1.3.3 Activity - CSS Zen Garden critique

Now that you’ve gotten a good idea of what CSS Zen Garden is, take a closer look. Please browse some of the designs, starting from "[Garments](http://www.csszengarden.com/220/) (note that the link to "all designs" is now broken...). Look through the different CSS Zen Garden designs for inspiration. Which is your favorite design? Pick one design and share your critiques with the discussion.

For your chosen design, please answer the following questions:

* What made this design stand out to you?
* What do you like best about this design?
* What is one thing you don't like about this design?

## 1.4.1 Hello beautiful world

Now that you have plenty of context, it's time to dive in.

We're going to start with a Hello World.

The traditional first program you would write when learning a new technology.

Since you already have the text from HTML, which would say Hello World, we're going to write Hello Beautiful World and make that text a different color and a different font.

In this demo, I'm going to walk you through how to set up your environment, and writing the HTML and the CSS that goes with it and then seeing it appear in a Web browser.

Let's get started.

### Live coding video: "Hello beautiful world" Demo

This is Visual Studio code. This is the program I'm going to use to type out all of my HTML and CSS.

When you open the program, it's going to give you a blank file that's untitled and of type ‘plain text’.

But as you add things and save them, it can understand a lot of different file formats.

I'm going to start with some HTML… this is really basic HTML, it just put those puts the words ‘Hello World’ on the page.

As I added this text you can see that a little white dot showed up to tell me that I have unsaved changes.

I'm going to save them!

I have a blank folder here I'm calling Hello World, and this is where I'm going to put all of the things for my single Web site: the CSS, the HTML, any images, etc.

You want to make sure that you have one folder for each of your projects.

I'm going to call this HTML page: ‘index’.

That's a pretty common title for the main page of your Web site.

I'm going to save it, and because I added that ‘.html’ ending, Visual Studio code is smart enough to turn some of the tags blue.

It knows these are tags, and this is text.

Now that we've got that, I can open the Explorer and it will show me which files I'm working with.

In fact, I can right click on this file and I can copy the path.

If I go to my Web browser and I add in the path: that's just the location of the file where this HTML is, on my own personal machine.

It's not on the Internet yet.

But if I go to it, my browser is smart enough to look at that HTML page and render it, or display it the same way it what is it was online.

There are my words: ‘Hello World’.

We have some HTML, but it's not very beautiful.

Let's change some things.

If I change the HTML and I come back, I can just refresh the page and it will show me my changes.

Now that I've got some text, let's actually add some style.

I'm going to open a new file that’s going to store my CSS.

For the purposes of this class, I'm going to keep my CSS in a separate file from my HTML, keeps things cleaner.

I’m going to write something really basic.

I'm going to write a role for the P or paragraph tag. For any rule, I need to set up open and closing curly braces, and then between them, I'm going to add the property.

And then the style I have to type what I'm thinking blue… that's a good color ;)

And then, I'm going to save this as a ‘.css’

You'll often see that style uses a name for the CSS.

See, when I hit save, I told Visual Studio code this is a CSS file.

It made a lot of assumptions about my code.

The P turned yellow to say, ‘hey, I recognize this is attached to HTML’,

the property color changed to a different color, it even gives me this cute little color picker so I know exactly which color I'm choosing.

Now that I've got this I can come to my HTML refresh and see the text turn blue, right?

Ah, nothing happened. That's because my HTML doesn't actually know that my CSS exists yet.

You have to tell the HTML where the CSS file is. So, because I'm a terrible typist, I already have this copied out, I'm going to add the special tag to the header of my HTML.

This tells my HTML where to find my CSS.

And really, what's important, is you need to make sure that you set the name for the link.

And because my CSS is in the same folder as my HTML, I can just give it ‘style.css’ and we'll know where to find it.

Now if I hit save (and make my white dot go away), if I come over here and I refresh, hey magic!

Now my text is blue… in fact, now I've added a permanent link between my HTML my CSS.

I can start adding more styles.

Let's say I want to make the font something sans Serif.

If I add in this extra rule, I can hit save, and now when I refresh, I can see the changes.

Here is the [code](https://codepen.io/techie4good/pen/oxQaVN) we wrote in this demo.

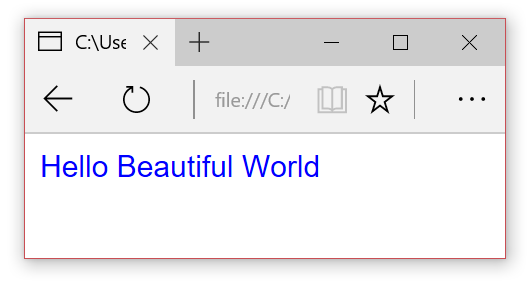
Here is the HTML part:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <link rel="stylesheet" href="style.css">
6. </head>
7. <body>
8. <p>
9. Hello Beautiful World
10. </p>
11. </body>
12. </html>

... and the CSS file (style.css) is below:

1. p {
2. color: blue;
3. font-family:Helvetica;
4. }

... and here is the "output":



## 1.4.2 Comments

As you write your CSS, you might end up with a pretty large document that can be hard to manage, or you might find yourself working on a team and having to read CSS someone else has written. In these cases, it helps to leave "notes" for the humans that read the file.

There is a way to leave notes that the Web browser will ignore when it is reading your CSS code, they're called "comments". In fact, leaving comments in your code is considered a best practice by developers and is a habit we highly recommend you develop now.

To add comments to your CSS file, you need to surround any text you want the computer to ignore with a set of slashes and asterisks like so:

**/\*** those two symbols start my comment block

 I can have more comment text here

and the following two symbols end my comment **\*/**

As you can see, you can put as much text between the open and close symbols as you need, you can even have multiple lines. If you are working in an editor like Visual Studio code, you will notice that when you turn text into a comment, it turns green to indicate that the computer ignores that code.



Generally, it is a good idea to put a comment at the top of each CSS rule, or at the very least at the top of sets of rules that apply to a single category or section of your Web page.

## 1.4.3 Module 1 project - Hello your world

It's finally time to write your own CSS! Open your code editor of choice and save the following code as a new HTML document.

Remember: to do this, you will need to give it a .html file extension when you are saving it. For example, you could call it index.html

<!DOCTYPE html>

<html lang="en">

   <head>

      <meta charset="utf-8">

   </head>

   <body>

      <p>

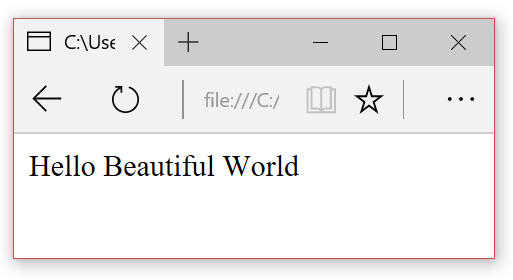
         Hello Beautiful World

      </p>

   </body>

</html>

Once you have your HTML document view it in a Web browser. It should look like this:



Now it’s time to add some CSS. Here is the CSS we wrote in the "Hello Beautiful World" demo. Make a new file with this css and save it with a .css file extension. For example, you can call it styles.css

p {

    color: blue;

}

This won’t change the look of your HTML until you link the two files with this HTML tag.

To do this:

* remember it should be placed in the header, that is between the <head> and </head> tags in the HTML file.
* place the HTML and CSS files in the same folder on your computer
* add the linking code to the HTML header (that means after the <head> tag and before the </head> tag). If your css is called "styles.css", here is what it would look like:

<link rel="stylesheet" href="styles.css">

Now change the HTML and CSS files so that it says “Hello <your name>, welcome to my first CSS Web page” in your favorite color! Here’s what mine looks like:



**HINT**: Is your favorite color not working? Not all color names are recognized by CSS. Sometimes the best way is to use HEX. We'll talk in more detail about colors in the next module, but here is a list of colors you can use: [extended color keywords](https://www.w3.org/TR/css-color/#color-keywords) (from W3C's specification) or [color keywords](https://developer.mozilla.org/en-US/docs/Web/CSS/color_value) (from MDN's documentation).

## Module 1 learnings

After this module, you should feel comfortable…

* Explaining what CSS is, and why is it important
* Opening HTML and CSS files in your chosen code editor
* Using browser tools to inspect the source of a Web page you wrote

In next module, you will:

* Review the basics of HTML
* Learn the anatomy of a CSS "rule"
* Discover the concept of a property
* Get to know selectors and how you can directly attach them to HTML tags
* Finally, for your module project, you'll get a get a chance to build the CSS for an HTML page from scratch

## 2.1.1 Welcome to Module 2

In this module, we're going to focus on building out your CSS toolbox.

First, we'll start with a simple review of HTML.

Don't worry! In this course, we're always going to give you the HTML, but we want you to know what's going on.

Then, we're going to meet the CSS rule and break down the pieces so you can start writing around.

We'll start with the first part of the CSS rule, the property.

The property is the part that helps control the look and feel: the font, the color,… things like that and then we'll meet the selector, the piece that attaches your CSS rule to a specific HTML element.

And then finally, at the end of the module, you get to write your first CSS style specifically for a given HTML Web site.

## 2.1.2 Module 2 outline

### In this module, we will:

1. Review the basics of HTML
2. Introduce you to the anatomy of a CSS "rule"
3. Introduce you to the concept of a property and give you a set of properties to get started
4. Introduce you to selectors and how you can directly attach them to HTML tags
5. Finally, for your module project, you'll get a get a chance to build the CSS for an HTML page from scratch

## 2.2.1 HTML to get you started

In this course, we're talking about CSS.

But CSS is only one aspect of a Web page.

And actually, you can't have CSS unless you have something to attach it to you.

In this case, that's HTML. HTML stands for Hyper Text Markup Language, and really the key is that M for markup.

What this means is that HTML is a way of tagging your existing content, so that the browser or screen readers or search engines know exactly what that content is.

For example, you might want the browser to know, this is a title or this is the button or this is an image, that way it knows how to display it.

But the problem is that HTML does a really bad job with style.

That's where CSS comes in. Once you've tagged all of your content as a specific type, you can use CSS to attach to that HTML.

In this course, we're always going give you the HTML but let's take a look at some sample code, so you know what it's looks like.

Here is just some basic text content.

I've got some titles and I've got a couple paragraphs.

I want the browser to know: this text is a little more important than the rest of the text, say a title, so that I can display it differently using my CSS.

I'm going to tell the browser this is the title by using something called the tag.

It starts with an angle bracket and then it has a name, and then another angle bracket on the other side.

And then, I put that in front of the content, and then I also have to add a closing tag at the end of the content.

Here, I've used the h1 tag for my title that stands for header 1. It means this is a very important title.

I can also use the p, or paragraph tag, to tell the browser: this is a block of body text.

Not only can use HTML to categorize existing text content, but you can use it to insert new elements into your page.

Elements like images or videos, buttons, drop-downs, all of that can be inserted with special HTML tags.

For example, here's the text so that you can add an image.

Now that you've seen what HTML looks like, it will make more sense when we start writing the CSS that corresponds to it.

Because the HTML has to come first.

In this course, we're just talking about CSS, but if you're not familiar with HTML I really encourage you to look online, and learn a bit more because a well-structured HTML document is going to make your CSS a lot nicer.

### HTML 101

HTML (**H**yper **T**ext **M**arkup **L**anguage) documents are made up of content and tags. These tags describe the content so that the web browser understands the structure of the page. HTML tags typically come in pairs, an opening tag before and a closing tag after content like so:

1. <tagname>
2. My content
3. </tagname>

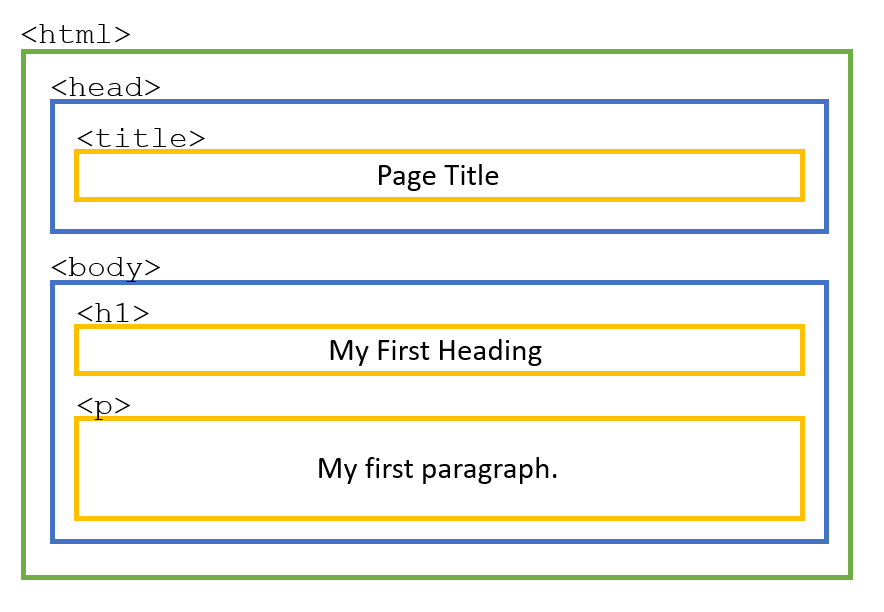
When these three pieces are combined (start tag, content, and end tag), you have what is called an HTML **element**.

Here is a sample HTML doc:

* 1. <!DOCTYPE html> <!-- Doctype declares the document to be HTML 5 type-->
  2. <html lang="en"> <!--All your HTML content must be within this tag-->
  3. <head> <!--Anything in the header provides information about the document, no content here-->
  4. <meta charset="utf-8">
  5. <title>Page Title</title> <!--This text will show up on the tab of the browser for this page-->
  6. </head> <!--end for the header section-->
  7. <body> <!--start tag for the body section, this is where to put all your content to be displayed-->
  8. <h1>My First Heading</h1> <!--content in a h1 tag is a “heading” of the top level-->
  9. <p>My first paragraph.</p> <!--content in a p tag is normal or “paragraph” level text-->
  10. </body>
  11. </html>

\*NOTE: In the code above, the red text contained within the <!-- and --> start and end sequences are comments. Each of them is explaining each tag.

Tags can be nested inside of other tags. This creates a parent/child relationship between HTML elements and forms the overall structure of your HTML document into a tree. This structure has a big affect on your CSS as styles are typically inherited from parent to child. We will take a closer look at style inheritance later in this unit.



There are other types of tags that are called "self-closing", meaning they don't come in an open/close pair. Typically self-closing tags insert content into your page as opposed to surround content. They look like this:

<img src="images/pic1.png" alt="pic1" />

These types of tags rely on "attributes", these are added modifiers on the tag that have their own values. In the above example, we use the src attribute to set the source for the image.

## 2.2.2 Common HTML tags

There are many HTML tags to choose from depending on what elements you want to structure on your page. You can always check what are the [current defined HTML tags](https://www.w3.org/TR/html52/).

Here is a short list of some of the most common HTML tags, ones you'll see us use throughout this course.

### <html>

[[*Documentation*](https://www.w3.org/TR/html52/semantics.html#the-html-element)]

The root element of a document is <html>, and this is the first tag you'll need in your document (after the DOCTYPE, of course!). All your other HTML tags should go inside this one, meaning all HTML documents should start with <html> at the top and end with </html> at the bottom.

You'll notice in the below code that we set the language to English (<html lang="en">) . You can [set another language](https://www.w3.org/International/tutorials/language-decl/#Slide0140) of the text in your page using language attributes (see also [this resource](https://www.w3.org/International/questions/qa-html-language-declarations)).

It is important that you take care to use the lang attribute to indicate the actual language of text in your page because many CSS features will function differently, depending on what language is declared here.

1. <!DOCTYPE html>
2. <html lang="en">
3. <body>
4. <p> Hello World</p>
5. </body>
6. </html>

### <head>

[[*Documentation*](https://www.w3.org/TR/html52/document-metadata.html#the-head-element)]

This is the element that contains all the metadata for your site, such as your link to your CSS, the page's title and links to other files. This should be the first tag in your document, and there should only be one per document.

Note that this is where you will also set the charset to "utf-8" (<meta charset="utf-8">). This shows that you saved the markup using the UTF-8 character encoding, which has many characters outside English, so it should be able to display characters not in the English alphabet.

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>My First Page</title>
6. </head>
7. <body>
8. <p> Hello World</p>
9. </body>
10. </html>

### <body>

[[*Documentation*](https://www.w3.org/TR/html52/sections.html#the-body-element)]

The section element that contains all the visible content for your site like your text, images, links etc. There should only be one body tag per document and it should come after the head tag.

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>My First Page</title>
6. <link rel="stylesheet" href="styles.css">
7. </head>
8. <body>
9. <p> Hello World</p>
10. </body>
11. </html>

### <p>

[[*Documentation*](https://www.w3.org/TR/html52/grouping-content.html#the-p-element)]

"p" stands for "paragraph" which is a block of text that is physically separated from adjacent blocks through blank lines. This is the most basic way to group text content.

1. <p>
2. This is my introductory paragraph to my Web page! This text will wrap around in a single block and then after the paragraph is done there will be a line of white space.
3. </p>

### <a>

[[*Documentation*](https://www.w3.org/TR/html52/textlevel-semantics.html#the-a-element)]

By surrounding text with an <a> tag you turn it into a hyperlink. You will want to use the "href" attribute to indicate to which target the link should take the user when clicked. The default style of the a tag is to turn the text blue and underlined, and then change the color to purple after you have clicked the link. You can adjust all these styles with CSS.

1. <a href="https://www.microsoft.com">Microsoft Main Page</a>

### <img />

[[*Documentation*](https://www.w3.org/TR/html52/semantics-embedded-content.html#the-img-element)]

This tag will insert an image based on the source you provide via the "src" attribute. If the source is inaccessible, you can also specify "fall back" options via the "alt" attribute. You will **always** want to specify the "alt" attribute with a short phrase describing the image. This text is what will be read aloud if your user is using a screen reader, or will be displayed if the user's browser will not load images. Note that this is an example of a "self-closing" tag meaning there is no closing tag, you just end the opening tag with a forward slash.

1. <img src="images/proPic.jpg" alt="a headshot of the instructor" />

### <ul>

[[*Documentation*](https://www.w3.org/TR/html52/grouping-content.html#the-ul-element)]

The UL tag creates an "**u**nordered **l**ist" element, meaning a collection of elements in which the order is meaningless. This is a tag that sets the framework for you to add list elements inside it. You will want to add your elements within the ul tag each surrounded your content with **l**ist **i**tem or "<li>" tags like in the below example.

1. <ul>
2. <li>This is one element in the list</li>
3. <li>One of the elements</li>
4. <li>Another element</li>
5. *</ul>*

### <ol>

[[*Documentation*](https://www.w3.org/TR/html52/grouping-content.html#the-ol-element)]

The OL tag works exactly like the UL tag, except that the list element order matters. OL stands for "**o**rdered **l**ist" and by default, the list element items are displayed with a number preceding them.

1. <ol>
2. <li>This is the first element</li>
3. <li>The second element</li>
4. <li>Finally, this is the third element</li>
5. </ol>

### <br />

[[*Documentation*](https://www.w3.org/TR/html52/textlevel-semantics.html#the-br-element)]

The br element is a self-closing tag that inserts a line break. This is most evident when placed in a block of text as it essentially represents a carriage return or hitting the "enter" key.

1. <p>
2. this is my text.
3. <br />
4. this text will appear on the next line down.
5. </p>

### <header>

[[*Documentation*](https://www.w3.org/TR/html52/sections.html#the-header-element)]

The header tag is one of the section elements, it's role is to group other HTML elements according to their role on their page. The header element contains all the introductory content on the page typically a title and tagline or navigational elements.

1. <body>
2. <header>
3. <h1> Welcome to my page!</h1>
4. <h2> My very first web page</h2>
5. </header>
6. </body>

### <section>

[[*Documentation*](https://www.w3.org/TR/html52/sections.html#the-section-element)]

Another sectioning element, the "section" tag is a general-purpose grouping element. It most often should include a header tag at the top. This typically will come after a header tag and before a footer tag.

1. <body>
2. <header>
3. <h1> My Page </h1>
4. </header>
5. <section>
6. <h2> My Blog </h2>
7. </section>
8. </body>

### <footer>

[[*Documentation*](https://www.w3.org/TR/html52/sections.html#the-footer-element)]

Another sectioning element, the "footer" tag is supposed to organize the final content on the page such as the credits or contact info.

1. <body>
2. <header>
3. <h1>My Page</h1>
4. </header>
5. <section>
6. <h2>My Blog</h2>
7. </section>
8. <footer>
9. <p>
10. copyright 2016
11. </p>
12. </footer>
13. </body>

### <div>

[[*Documentation*](https://www.w3.org/TR/html52/grouping-content.html#the-div-element)]

The div element is a generic element to hold content. It is considered a last resort, for when no other element is suitable but is often used to collect together large portions of a site that contain multiple different types of content.

* 1. <div>
  2. <h1> Title for Content </h1>
  3. <img src="images/contentImage.jpg" />
  4. <p> This is a paragraph explaining this section of content associated with the above image and title </p>
  5. </div>

## 2.2.3 Next steps - learn more HTML

Note that, as this CSS Introduction course focuses on CSS, we will always provide you with the complete HTML for whatever content you will be asked to style. However, to become proficient in Web development, you are going to need a good handle on HTML. You can start by looking into some of these links:

* [A quick introduction to HTML](https://www.w3.org/TR/html52/introduction.html#a-quick-introduction-to-html) (from W3C)
* [Structuring the Web with HTML](https://developer.mozilla.org/en-US/docs/Learn/HTML) (from MDN)

Or of you are looking for more in-depth training, we suggest you check out one of these other W3Cx courses to better understand how to structure your pages with HTML and more:

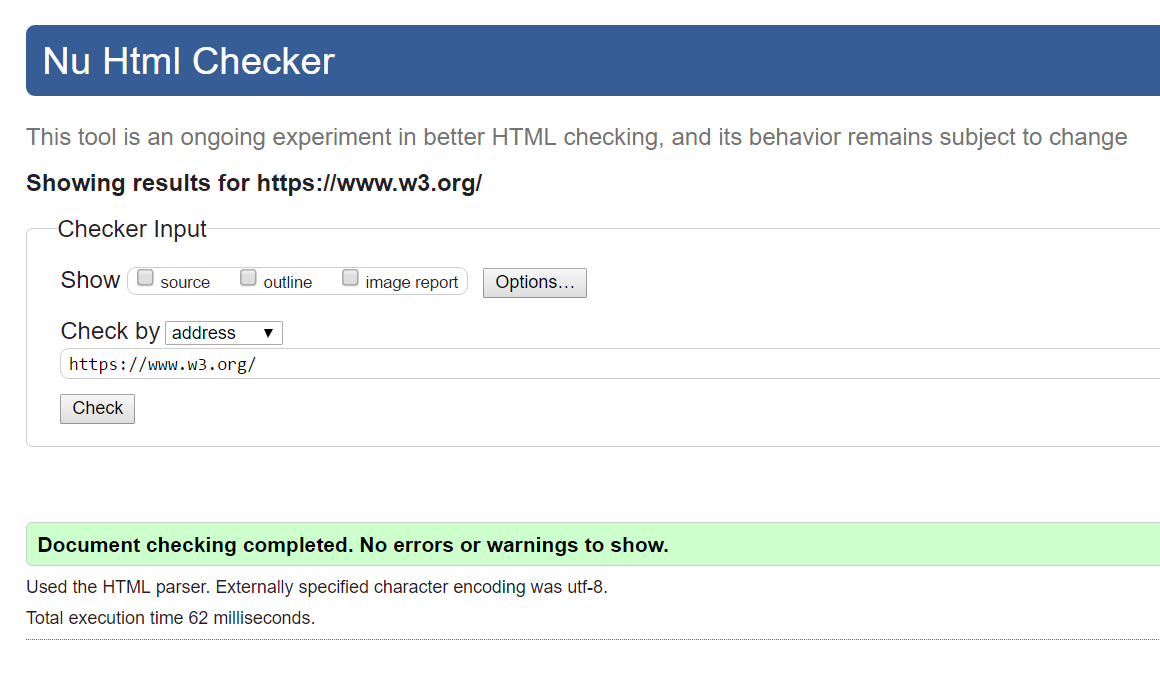
* [HTML5 & CSS Fundamentals](https://www.edx.org/course/html5-and-css-fundamentals) (beginner level)
* [HTML5 Coding Essentials and Best Practices](https://www.edx.org/course/html5-coding-essentials-and-best-practices) (intermediate level)
* [HTML5 Apps and Games](https://www.edx.org/course/html5-apps-and-games) (advanced level)

## 2.2.4 Activity - Practice with HTML validator

HTML has been available to the public since 1991, but since then a lot has changed. One of the ways to make sure your HTML is well structured and up to date is to use the W3C HTML Validator. As you are developing your pages, it's a good idea to regularly check if your HTML is written according to W3C standards.

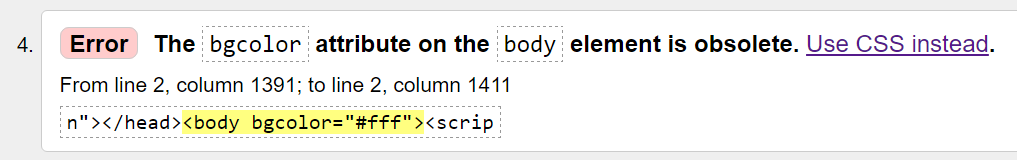
You can find the validator here: <https://validator.w3.org/>

You can pass any URL on the Web into the validator, and it will tell you how the HTML for that page stacks up against Web Standards. If you pass in https://www.w3.org (the W3C's homepage), you see the following:



If you start to try out other URLs, you might find this is a very rare result ;)  
Try passing in your favorite Web address and see what comes up. For example, If you pass in https://www.microsoft.com/en-us/, you get 567 warnings and errors!

One of the more common errors is using an HTML tag that is considered obsolete. Often the error points you to this wiki page "[Use CSS instead](https://wiki.whatwg.org/wiki/Presentational_elements_and_attributes)".



For this activity, please try out some of your favorite Web addresses in this validator and see what happens. Find a page that has one of these types of errors and answer the following questions in the discussion board:

* What URL gave you errors?
* How many warnings and errors does this site have?
* What HTML attribute does it use when it should use CSS instead?

## 2.3.1 The anatomy of a CSS rule

In this module, we're going to focus on building out your CSS toolbox.

First, we'll start with a simple review of HTML.

Don't worry! In this course, we're always going to give you the HTML, but we want you to know what's going on.

Then, we're going to meet the CSS rule and break down the pieces so you can start writing around.

We'll start with the first part of the CSS rule, the property.

The property is the part that helps control the look and feel:

the font, the color,… things like that and then we'll meet the selector, the piece that attaches your CSS rule to a specific HTML element.

And then finally, at the end of the module, you get to write your first CSS style specifically for a given HTML Web site.

## 2.1.2 Module 2 outline

### In this module, we will:

1. Review the basics of HTML
2. Introduce you to the anatomy of a CSS "rule"
3. Introduce you to the concept of a property and give you a set of properties to get started
4. Introduce you to selectors and how you can directly attach them to HTML tags
5. Finally, for your module project, you'll get a get a chance to build the CSS for an HTML page from scratch

## 2.2.1 HTML to get you started

In this course, we're talking about CSS.

But CSS is only one aspect of a Web page.

And actually, you can't have CSS unless you have something to attach it to you.

In this case, that's HTML. HTML stands for Hyper Text Markup Language, and really the key is that M for markup.

What this means is that HTML is a way of tagging your existing content, so that the browser or screen readers or search engines know exactly what that content is.

For example, you might want the browser to know, this is a title or this is the button or this is an image, that way it knows how to display it.

But the problem is that HTML does a really bad job with style.

That's where CSS comes in. Once you've tagged all of your content as a specific type, you can use CSS to attach to that HTML.

In this course, we're always going give you the HTML but let's take a look at some sample code, so you know what it's looks like.

Here is just some basic text content.

I've got some titles and I've got a couple paragraphs.

I want the browser to know: this text is a little more important than the rest of the text, say a title, so that I can display it differently using my CSS.

I'm going to tell the browser this is the title by using something called the tag.

It starts with an angle bracket and then it has a name, and then another angle bracket on the other side.

And then, I put that in front of the content, and then I also have to add a closing tag at the end of the content.

Here, I've used the h1 tag for my title that stands for header 1. It means this is a very important title.

I can also use the p, or paragraph tag, to tell the browser: this is a block of body text.

Not only can use HTML to categorize existing text content, but you can use it to insert new elements into your page.

Elements like images or videos, buttons, drop-downs, all of that can be inserted with special HTML tags.

For example, here's the text so that you can add an image.

Now that you've seen what HTML looks like, it will make more sense when we start writing the CSS that corresponds to it.

Because the HTML has to come first.

In this course, we're just talking about CSS, but if you're not familiar with HTML I really encourage you to look online, and learn a bit more because a well-structured HTML document is going to make your CSS a lot nicer.

### HTML 101

HTML (**H**yper **T**ext **M**arkup **L**anguage) documents are made up of content and tags. These tags describe the content so that the web browser understands the structure of the page. HTML tags typically come in pairs, an opening tag before and a closing tag after content like so:

1. <tagname>
2. My content
3. </tagname>

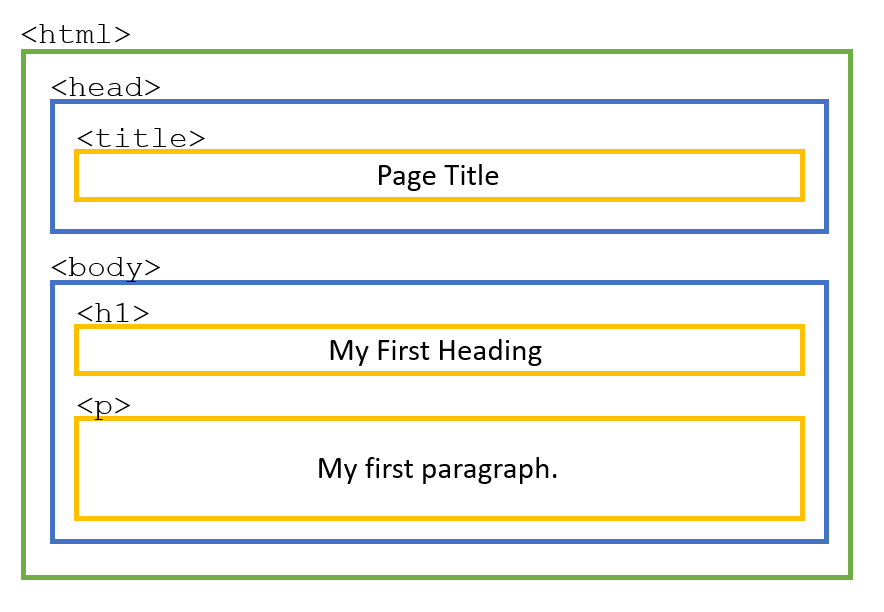
When these three pieces are combined (start tag, content, and end tag), you have what is called an HTML **element**.

* 1. Here is a s<!DOCTYPE html> <!-- Doctype declares the document to be HTML 5 type-->
  2. <html lang="en"> <!--All your HTML content must be within this tag-->
  3. <head> <!--Anything in the header provides information about the document, no content here-->
  4. <meta charset="utf-8">
  5. <title>Page Title</title> <!--This text will show up on the tab of the browser for this page-->
  6. </head> <!--end for the header section-->
  7. <body> <!--start tag for the body section, this is where to put all your content to be displayed-->
  8. <h1>My First Heading</h1> <!--content in a h1 tag is a “heading” of the top level-->
  9. <p>My first paragraph.</p> <!--content in a p tag is normal or “paragraph” level text-->
  10. </body>
  11. </html>

ample HTML doc:

\*NOTE: In the code above, the red text contained within the <!-- and --> start and end sequences are comments. Each of them is explaining each tag.

Tags can be nested inside of other tags. This creates a parent/child relationship between HTML elements and forms the overall structure of your HTML document into a tree. This structure has a big affect on your CSS as styles are typically inherited from parent to child. We will take a closer look at style inheritance later in this unit.



There are other types of tags that are called "self-closing", meaning they don't come in an open/close pair. Typically self-closing tags insert content into your page as opposed to surround content. They look like this:

1. <img src="images/pic1.png" alt="pic1" />

These types of tags rely on "attributes", these are added modifiers on the tag that have their own values. In the above example, we use the src attribute to set the source for the image.

## 2.2.2 Common HTML tags

There are many HTML tags to choose from depending on what elements you want to structure on your page. You can always check what are the [current defined HTML tags](https://www.w3.org/TR/html52/).

Here is a short list of some of the most common HTML tags, ones you'll see us use throughout this course.

### <html>

[[*Documentation*](https://www.w3.org/TR/html52/semantics.html#the-html-element)]

The root element of a document is <html>, and this is the first tag you'll need in your document (after the DOCTYPE, of course!). All your other HTML tags should go inside this one, meaning all HTML documents should start with <html> at the top and end with </html> at the bottom.

You'll notice in the below code that we set the language to English (<html lang="en">) . You can [set another language](https://www.w3.org/International/tutorials/language-decl/#Slide0140) of the text in your page using language attributes (see also [this resource](https://www.w3.org/International/questions/qa-html-language-declarations)).

It is important that you take care to use the lang attribute to indicate the actual language of text in your page because many CSS features will function differently, depending on what language is declared here.

1. <!DOCTYPE html>
2. <html lang="en">
3. <body>
4. <p> Hello World</p>
5. </body>
6. </html>

### <head>

[[*Documentation*](https://www.w3.org/TR/html52/document-metadata.html#the-head-element)]

This is the element that contains all the metadata for your site, such as your link to your CSS, the page's title and links to other files. This should be the first tag in your document, and there should only be one per document.

Note that this is where you will also set the charset to "utf-8" (<meta charset="utf-8">). This shows that you saved the markup using the UTF-8 character encoding, which has many characters outside English, so it should be able to display characters not in the English alphabet.

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>My First Page</title>
6. </head>
7. <body>
8. <p> Hello World</p>
9. </body>
10. </html>

### <body>

[[*Documentation*](https://www.w3.org/TR/html52/sections.html#the-body-element)]

The section element that contains all the visible content for your site like your text, images, links etc. There should only be one body tag per document and it should come after the head tag.

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>My First Page</title>
6. <link rel="stylesheet" href="styles.css">
7. </head>
8. <body>
9. <p> Hello World</p>
10. </body>
11. </html>

### <p>

[[*Documentation*](https://www.w3.org/TR/html52/grouping-content.html#the-p-element)]

"p" stands for "paragraph" which is a block of text that is physically separated from adjacent blocks through blank lines. This is the most basic way to group text content.

1. <p>
2. This is my introductory paragraph to my Web page! This text will wrap around in a single block and then after the paragraph is done there will be a line of white space.
3. </p>

### <a>

[[*Documentation*](https://www.w3.org/TR/html52/textlevel-semantics.html#the-a-element)]

By surrounding text with an <a> tag you turn it into a hyperlink. You will want to use the "href" attribute to indicate to which target the link should take the user when clicked. The default style of the a tag is to turn the text blue and underlined, and then change the color to purple after you have clicked the link. You can adjust all these styles with CSS.

1. <a href="https://www.microsoft.com">Microsoft Main Page</a>

### <img />

[[*Documentation*](https://www.w3.org/TR/html52/semantics-embedded-content.html#the-img-element)]

This tag will insert an image based on the source you provide via the "src" attribute. If the source is inaccessible, you can also specify "fall back" options via the "alt" attribute. You will **always** want to specify the "alt" attribute with a short phrase describing the image. This text is what will be read aloud if your user is using a screen reader, or will be displayed if the user's browser will not load images. Note that this is an example of a "self-closing" tag meaning there is no closing tag, you just end the opening tag with a forward slash.

1. <img src="images/proPic.jpg" alt="a headshot of the instructor" />

### <ul>

[[*Documentation*](https://www.w3.org/TR/html52/grouping-content.html#the-ul-element)]

The UL tag creates an "**u**nordered **l**ist" element, meaning a collection of elements in which the order is meaningless. This is a tag that sets the framework for you to add list elements inside it. You will want to add your elements within the ul tag each surrounded your content with **l**ist **i**tem or "<li>" tags like in the below example.

1. <ul>
2. <li>This is one element in the list</li>
3. <li>One of the elements</li>
4. <li>Another element</li>
5. *</ul>*

### <ol>

[[*Documentation*](https://www.w3.org/TR/html52/grouping-content.html#the-ol-element)]

The OL tag works exactly like the UL tag, except that the list element order matters. OL stands for "**o**rdered **l**ist" and by default, the list element items are displayed with a number preceding them.

1. <ol>
2. <li>This is the first element</li>
3. <li>The second element</li>
4. <li>Finally, this is the third element</li>
5. </ol>

### <br />

[[*Documentation*](https://www.w3.org/TR/html52/textlevel-semantics.html#the-br-element)]

The br element is a self-closing tag that inserts a line break. This is most evident when placed in a block of text as it essentially represents a carriage return or hitting the "enter" key.

1. <p>
2. this is my text.
3. <br />
4. this text will appear on the next line down.
5. </p>

### <header>

[[*Documentation*](https://www.w3.org/TR/html52/sections.html#the-header-element)]

The header tag is one of the section elements, it's role is to group other HTML elements according to their role on their page. The header element contains all the introductory content on the page typically a title and tagline or navigational elements.

1. <body>
2. <header>
3. <h1> Welcome to my page!</h1>
4. <h2> My very first web page</h2>
5. </header>
6. </body>

### <section>

[[*Documentation*](https://www.w3.org/TR/html52/sections.html#the-section-element)]

Another sectioning element, the "section" tag is a general-purpose grouping element. It most often should include a header tag at the top. This typically will come after a header tag and before a footer tag.

1. <body>
2. <header>
3. <h1> My Page </h1>
4. </header>
5. <section>
6. <h2> My Blog </h2>
7. </section>
8. </body>

### <footer>

[[*Documentation*](https://www.w3.org/TR/html52/sections.html#the-footer-element)]

Another sectioning element, the "footer" tag is supposed to organize the final content on the page such as the credits or contact info.

1. <body>
2. <header>
3. <h1>My Page</h1>
4. </header>
5. <section>
6. <h2>My Blog</h2>
7. </section>
8. <footer>
9. <p>
10. copyright 2016
11. </p>
12. </footer>
13. </body>

### <div>

[[*Documentation*](https://www.w3.org/TR/html52/grouping-content.html#the-div-element)]

The div element is a generic element to hold content. It is considered a last resort, for when no other element is suitable but is often used to collect together large portions of a site that contain multiple different types of content.

* 1. <div>
  2. <h1> Title for Content </h1>
  3. <img src="images/contentImage.jpg" />
  4. <p> This is a paragraph explaining this section of content associated with the above image and title </p>
  5. </div>

## 2.2.3 Next steps - learn more HTML

Note that, as this CSS Introduction course focuses on CSS, we will always provide you with the complete HTML for whatever content you will be asked to style. However, to become proficient in Web development, you are going to need a good handle on HTML. You can start by looking into some of these links:

* [A quick introduction to HTML](https://www.w3.org/TR/html52/introduction.html#a-quick-introduction-to-html) (from W3C)
* [Structuring the Web with HTML](https://developer.mozilla.org/en-US/docs/Learn/HTML) (from MDN)

Or of you are looking for more in-depth training, we suggest you check out one of these other W3Cx courses to better understand how to structure your pages with HTML and more:

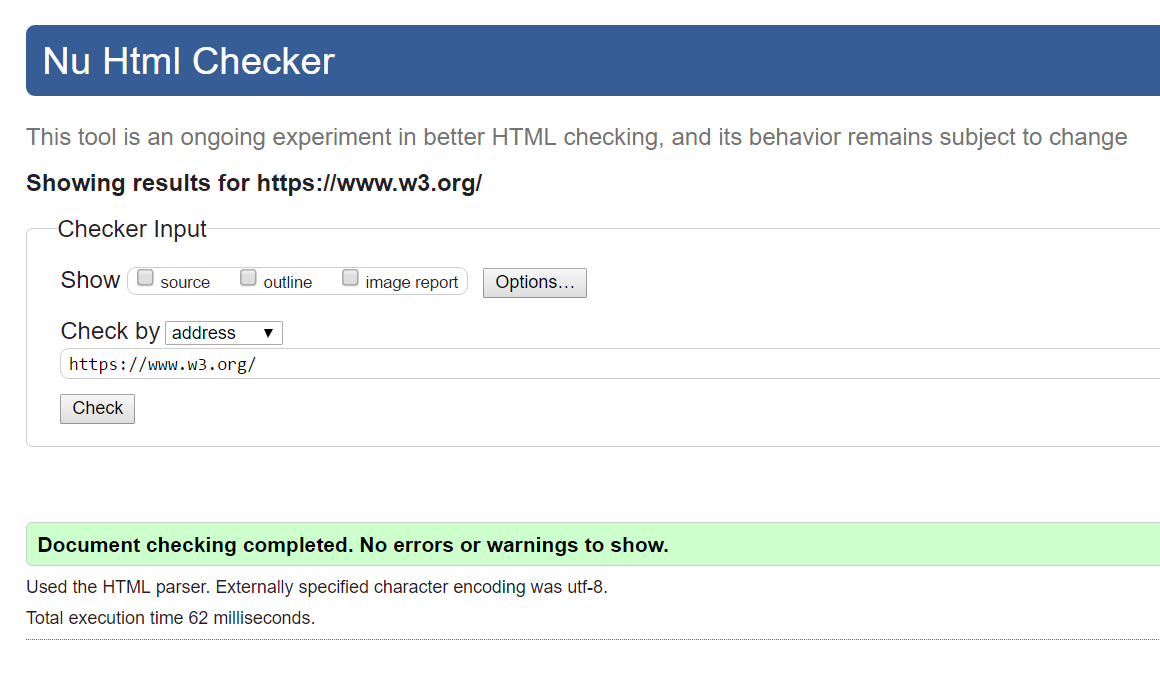
* [HTML5 & CSS Fundamentals](https://www.edx.org/course/html5-and-css-fundamentals) (beginner level)
* [HTML5 Coding Essentials and Best Practices](https://www.edx.org/course/html5-coding-essentials-and-best-practices) (intermediate level)
* [HTML5 Apps and Games](https://www.edx.org/course/html5-apps-and-games) (advanced level)

## 2.2.4 Activity - Practice with HTML validator

HTML has been available to the public since 1991, but since then a lot has changed. One of the ways to make sure your HTML is well structured and up to date is to use the W3C HTML Validator. As you are developing your pages, it's a good idea to regularly check if your HTML is written according to W3C standards.

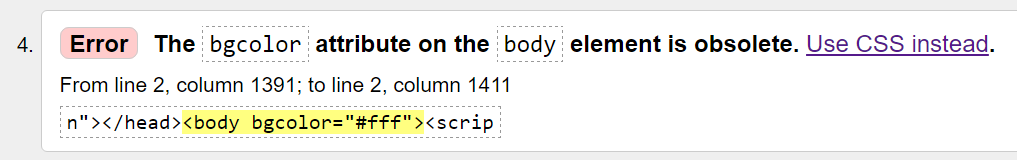
You can find the validator here: <https://validator.w3.org/>

You can pass any URL on the Web into the validator, and it will tell you how the HTML for that page stacks up against Web Standards. If you pass in https://www.w3.org (the W3C's homepage), you see the following:



If you start to try out other URLs, you might find this is a very rare result ;)  
Try passing in your favorite Web address and see what comes up. For example, If you pass in https://www.microsoft.com/en-us/, you get 567 warnings and errors!

One of the more common errors is using an HTML tag that is considered obsolete. Often the error points you to this wiki page "[Use CSS instead](https://wiki.whatwg.org/wiki/Presentational_elements_and_attributes)".



For this activity, please try out some of your favorite Web addresses in this validator and see what happens. Find a page that has one of these types of errors and answer the following questions in the discussion board:

* What URL gave you errors?
* How many warnings and errors does this site have?
* What HTML attribute does it use when it should use CSS instead?

## 2.3.1 The anatomy of a CSS rule

Now that you have HTML, let's talk about how to build CSS rules that are attached to it.

This is a CSS rule. As you can see, it's broken into two pieces.

The property and the selector, and they're separated by these things called curly braces.

The selector is the part that sits above the curly brace.

It's the piece that tells the browser what HTML element to attach this style to.

The property is the part inside the curly braces,

and that's where you define the style of your HTML element.

Things like fonts, colors, spacing, etc.

Now that we know what a CSS rule looks like, let's take a look at different examples

and how you can start building your own.

Here's a basic HTML page I wrote.

If we take a closer look by viewing source,

we can see that it's actually only 3 HTML tags.

A header one tag, which is this text right here; an image tag;

and a paragraph tag, that holds this text down here.

But it's got a lot more style than we've seen before.

Let's take a closer look at this h1 tag.

If I click here, you'll see the style that's attached to it over here in this panel.

See how the selector h1 matches the tag h1.

That's why I'm telling the style: I wanted to apply this text.

I can add a background color -- color or some sizing some space around it, and I

even get some extra style that I added to the body tag,

to this tag, because then it applies to all of my other tags.

This is the code in Visual Studio Code.

Here is my full HTML file and here's my full CSS style. You can start to see the pattern of CSS rules.

A selector followed by an open curly brace,

there's a closed curly brace and then a bunch of stuff in the middle.

The stuff in the middle is the properties. For example, the body tag

right now, has a background color property set and it set to the value of

this nice seafoam green color. It's also got a text align property with a value

center, a font family property with the value Arial, so on and so forth.

Now, let's write our first CSS tag… but we got to have some HTML to attach it to.

Let's add an h2 tag. After I welcome them to my Web site, I'm going to tell them

how glad I am there. I'm going to save this HTML and see

how it looks without any style. As you can see, it's already in

the center and it's already got the right font, because, if I inspect it,

it's already inheriting Arial and the text align center from the body.

But, if I go back to Visual Studio, I can add some more specific style.

Let's add it right here under h1, and we're going to follow that pattern.

I'm going to hit Enter, so it comes after

the closing curly brace. I'm going to add the selector,

which is h2, because I want the style to apply to h2. I'm going to

open curly brace and then Visual Studio will automatically add a closing curly brace for me.

I'll hit Enter so I have some space, and now let's change the

color of the text. How about I want to make this the opposite

of the h1 tag. See how when I hit color and hit space,

Visual Studio Code shows me all the

colors that I have already used. Which is perfect,

because yeah I think I want this to be the same color as the h1 background color.

I'm going to hit a semicolon after the value, and I'm going to save it, and now my text is that color.

But it looks really harsh on that seafoam green.

So, let's add a background color.

I can add as many properties within the CSS rule as I want.

They just all have to be between the curly braces so that it knows to apply them to the h2 tag.

I'm going to hit Enter. I'm going to add background color as a

new property, give me some quotation marks, but I don't need them,

and then I hit Enter again and it's going to give me the selection of colors.

I'm going to make this the light color for this background.

I'm going to hit Enter, it gives me a little preview of the color, I'm going to hit a semicolon.

Now, I have a new rule that follows the same pattern as before.

Selector, open curly brace, property, value, semicolon,

repeat until you hit a closing curly brace. Hit Save.

Let's see how it looks. Much better!

[Here's the code from the video in a Code Pen for you to play around with](https://codepen.io/techie4good/pen/JXwWvr)

The HTML code is shown below:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>My HTML page</title>
6. <link rel="stylesheet" href="style.css">
7. </head>
8. <body>
9. <h1> Welcome to my first CSS Website</h1>
10. <h2> I'm glad you're here</h2>
11. <img src="designer.png" alt="designer"/>
12. <p>
13. This is my first site using CSS, or Cascading Style Sheets. I'm still learning and I have a long way to go, but doesn't it still look nice?
14. It's amazing what a difference just some colors and fonts can make!
15. </p>
16. </body>
17. </html>

And the attached CSS file (style.css):

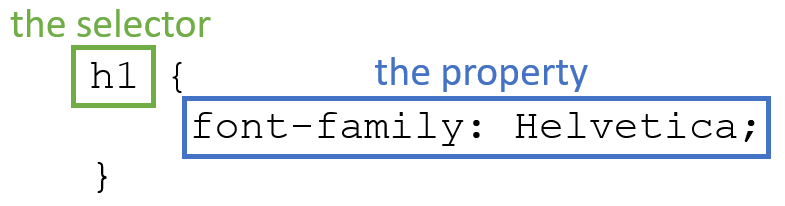
1. body {
2. background-color: #83AF9B;
3. text-align: center;
4. font-family: Arial;
5. padding: 0;
6. margin: 0;
7. }
8. h1 {
9. background-color: #FE4365;
10. color: #F9CDAD;
11. font-size: 8em;
12. padding: 50px;
13. }
14. h2 {
15. color: #FE4365;
16. background-color: #F9CDAD;
17. }
18. img {
19. height: 250px;
20. }
21. p {
22. background-color: #FC9D9A;
23. color: white;
24. padding: 50px;
25. font-size: 2em;
26. }

## 2.3.2 Constructing your CSS rules

Here is an example CSS "rule":

1. p {
2. color: blue;
3. }

This rule tells the browser to make all text within a paragraph tag blue. A CSS rule is broken into two parts: the selector and the property

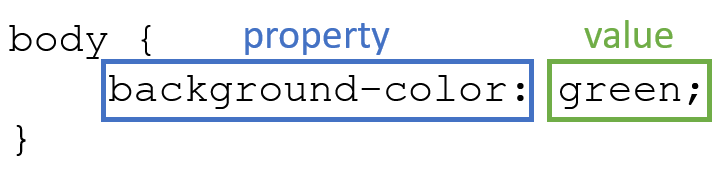


### Selector

This is the portion of the rule before the first open curly brace ( "{" character). This is what tells the browser what HTML tags this rule applies to. Often, you'll just see a selector that matches an HTML tag, like in this instance- our selector is just "p". However, as we get further into this course, you'll find that there are many ways to target specific HTML elements and many different ways to structure selectors so that you are targeting exactly the part of your site you want to style.

### Property

This is the portion of the rule between the two curly braces. This is what tells the browser how to style the HTML tag that has been selected. This can be as many lines of code as you choose, each of which has two parts- the property and the value you want that property to be. For our example, "color" is the property and "blue" is the value, but we could also have had a value of "black" or "#FFFFFF" (which is HEX code for white). Each property line is constructed so:



The style for your page will consist of a list of many CSS rules put together. As we move through this course we will help you build up these rules to style your entire page.

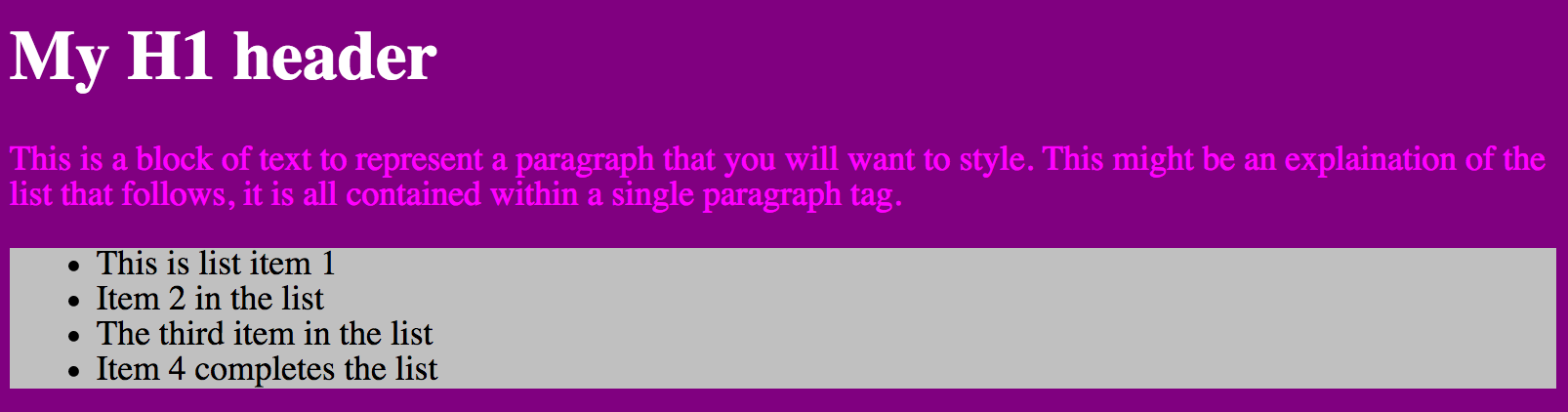
## 2.3.3 Activity - Building your first CSS rule set

Now that you have a basic understanding of how to put the pieces of a CSS rule together, let's do some practice. Here is some HTML for a page you will style:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>My HTML page</title>
6. <link rel="stylesheet" href="style.css">
7. </head>
8. <body>
9. <h1>My H1 header</h1>
10. <p> This is a block of text to represent a paragraph that you will want to style. This might be an explanation of the list that follows, it is all contained within a single paragraph tag.
11. </p>
12. <ul>
13. <li>This is list item 1</li>
14. <li>Item 2 in the list</li>
15. <li>The third item in the list</li>
16. <li>Item 4 completes the list</li>
17. </ul>
18. </body>
19. </html>

[HTML in Code Pen](https://codepen.io/w3devcampus/pen/zovzwE)

Your goal is to get this HTML to look like the following image in the browser:



To do so, you will need to write 4 CSS Rules. You will need to use the following 4 selectors:

* body
* h1
* p
* ul

And you will need the following properties:

* background-color: silver;
* background-color: purple;
* color: white;
* color: fuchsia;

Now it's up to you to combine these selectors and properties into 4 rules to achieve the final style.

## 2.4.1 What is a selector?

In unit 2.3, we defined a CSS selector as the portion of the CSS rule that tells the browser on which HTML element to apply the defined style.

When your HTML is simple, the selectors can be simple as well. The most basic selectors simply mirror the HTML tag. For example "p" attaches to all <p> tags, "img" will attach to all <img> tags and so on. As you can imagine, there will often be times when you don't want every single HTML element of a particular type to have identical style. In Module 3, we'll discuss a variety of ways to use selectors to attach to specific HTML elements.

In unit 2.2, we briefly mentioned the fact that properties apply to the entire hierarchy of HTML elements to which they are attached. This means that you will have to be very careful which selectors you choose to use in combination with your chosen style. When choosing your selector you might want to keep the following aspects of an HTML element in mind:

* How many of these HTML elements are on my page? Do I want this style to apply to every one of these elements?
* What are this HTML element's children, and do I want this style to apply to them as well?
* Is this element a block element or an inline element, and does this style make sense in that context?

It is possible to independently target every HTML element on the page using selectors, but for this module we are going to stick to basics and only use selectors that match the HTML tag name. For example, here are some example selectors we'll use in this module:

1. a {
2. /\* style for a tags \*/
3. }

This would affect the style of all link tags on the page.

1. p {
2. /\* style for p tags \*/
3. }

This would affect the style of all paragraph tags on the page and the style of elements contained within the paragraph tag.

1. body {
2. /\* style for all elements in the body \*/
3. }

This would apply style to the body tag as well as allow the elements inside the body tag to inherit certain styles applied here (check the [related CodePen online](https://codepen.io/techie4good/pen/pEzLrd)).



HTML

<html lang="en">

<head>

<meta charset="utf-8">

<link rel="stylesheet" href="styles.css">

</head>

<body>

<h1>Title</h1>

<p>In unit 2.3, we defined a CSS selector as the portion of the CSS rule that tells the browser on which HTML element to apply the defined style.

<a href="https://www.microsoft.com">Click Here!</a>

</p>

<ul>

<li>When your HTML is simple, the selectors can be simple as well</li>

<li>The most basic selectors simply mirror the HTML tag</li>

<li>For example "p" attaches to all tags, "img" will attach to all tags and so on</li>

<li>As you can imagine, there will often be times</li>

<li>when you don't want every single HTML element of a particular type to have identical style</li>

</ul>

<p>

In Module 3, we'll discuss a variety of ways to use selectors to attach to specific HTML elements.

In unit 2.2, we briefly mentioned the fact that properties apply to the entire hierarchy of HTML elements to which they are attached. This means that you will have to be very careful which selectors you choose to use in combination with your chosen style <br />

<a href="http://www.w3.org">Check this out</a>

It is possible to independently target every HTML element on the page using selectors, but for this module we are going to stick to basics and only use selectors that match the HTML tag name. For example, here are some example selectors we'll use in this module:

</p>

<ol>

<li>This would affect the style of all link tags on the page</li>

<li>This would affect the style of all paragraph tags on the page</li>

<li>and the style of elements contained within the paragraph tag</li>

<li>This would apply style to the body tag</li>

<li>as well as allow the elements inside the body tag to inherit certain styles applied here. </li>

</ol>

</body>

</html>

CSS

body {

color: red; /\* every element inherits this except a tags, which don't inhert by default \*/

}

ul {

color: blue; /\* li elements inherit this color \*/

}

p {

font-style: italic; /\*this even the a tags inherit within the paragraphs \*/

}

li {

text-decoration: line-through; /\* applies to all li elements, in both ul and ol tags \*/

}

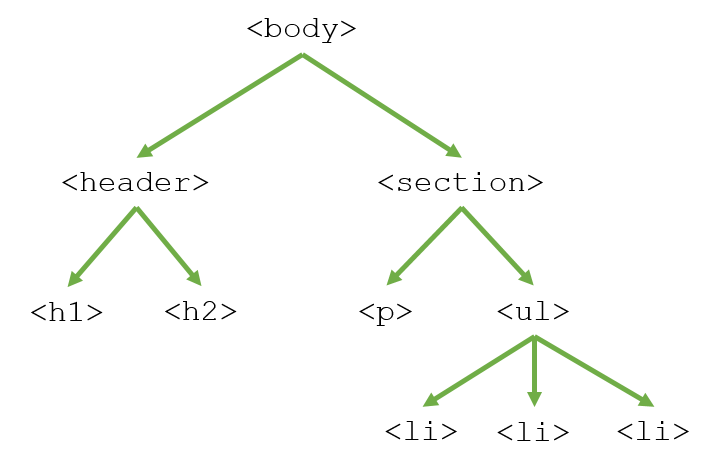
## 2.4.2 Inheriting style

Part of the reason a well structured HTML document is so important is because HTML elements inherit stylistic properties.

Let's say we have an HTML document (see the corresponding [Code Pen](https://codepen.io/techie4good/pen/mAVVkY?editors=1100#0)):

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>My HTML page</title>
6. <link rel="stylesheet" href="style.css">
7. </head>
8. <body>
9. <header>
10. <h1> Title </h1>
11. <h2> sub title </h2>
12. </header>
13. <section>
14. <p> This is my paragraph text </p>
15. <ul>
16. <li> list item 1 </li>
17. <li> list item 2 </li>
18. <li> list item 3 </li>
19. </ul>
20. </section>
21. </body>
22. </html>

This is basic structure from the way I have formatted the tags with tabbing, but here is a more visual representation of the hierarchy of tags. Tags that contain other tags are parents, and the tags inside of them are their children in the following tree representation:



Through inheritance, CSS property values set on one element will be transferred down the tree to that element's children. In this example, every element gets the same font because we applied it to the body tag. Since the body element is a common parent for all visible elements is a convenient selector for when you want to set stylistic rules for the entire document.

Then, we applied different styles at different levels of the tree so that the "li" or list element tag ends up with three different styles (font, underline and green) without us actually applying any style directly to that tag.

body {

   font-family: "Century Gothic", sans-serif;

}

header {

   font-style: italic;

}

section {

   text-decoration: underline;

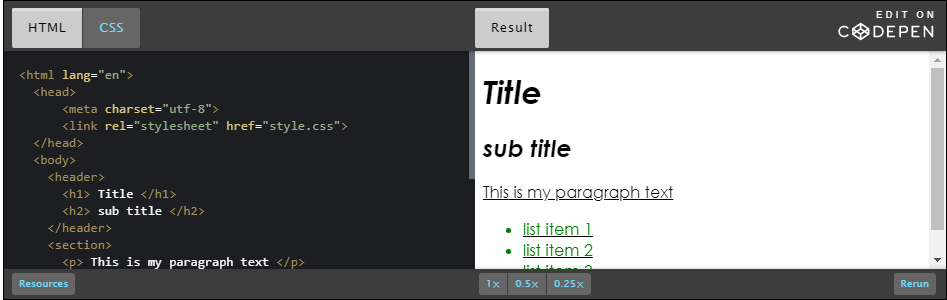
}

ul {

   color: green;

}

Not every property is inherited, but many are. The CSS specification tell you, for each property, whether it is inheritable. It's a good idea to keep in mind the structure of your HTML document when choosing your selectors so you can use inheritance to your advantage by applying styles to the top most element and save yourself extra CSS code.



HTML

<html lang="en">

<head>

<meta charset="utf-8">

<link rel="stylesheet" href="style.css">

</head>

<body>

<header>

<h1> Title </h1>

<h2> sub title </h2>

</header>

<section>

<p> This is my paragraph text </p>

<ul>

<li> list item 1 </li>

<li> list item 2 </li>

<li> list item 3 </li>

</ul>

</section>

</body>

</html>

CSS

body {

font-family: "Century Gothic", sans-serif;

}

header {

font-style: italic;

}

section {

text-decoration: underline;

}

ul {

color: green;

}

## 2.4.3 Combining multiple selectors

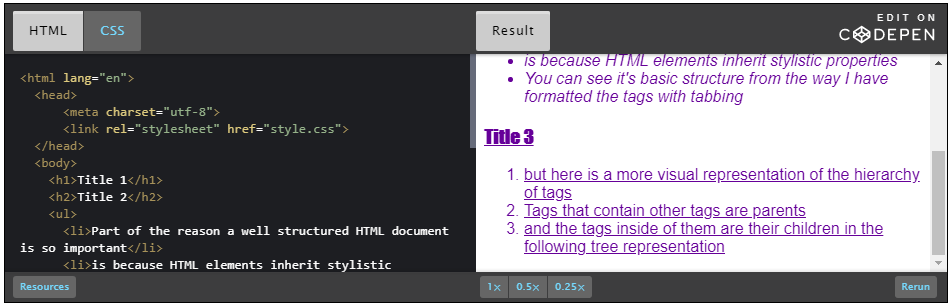
You can imagine that multiple HTML elements on your page will have similar style. If you write a separate CSS rule with the same properties for each of these elements, your CSS file can get very large and hard to manage. When designing CSS, the authors wanted to help make it as easy as possible to write and edit style sheets "by hand", so there are a number of features that help keep your styles succinct.

For example, what if you want to change the font that is consistent across many elements? You would have to change it in many places. Instead, you can combine multiple selectors on the same rule like so:

1. p, ul, ol {
2. color: blue;
3. background-color: pink;
4. }

The comma means that each of these elements should have the same, duplicated style. No need to have repeated style! Of course, you could simply apply this style to an element that contains all of these, say the body element, but not all properties are inherited so using the comma is a direct way to apply consistent style across multiple categories of HTML elements.

Here is a [CodePen](https://codepen.io/w3devcampus/pen/PVJLEm) that explores using the comma to combine selectors.



CSS code:

body {

    color: #660099;

}

h1,h2,h3 {

    font-family: Impact, sans-serif;

}

ul,ol {

    font-family: helvetica, sans-serif;

}

h2,ul {

    font-style: italic;

}

h3,ol {

    text-decoration: underline;

}

HTML

<html lang="en">

<head>

<meta charset="utf-8">

<link rel="stylesheet" href="style.css">

</head>

<body>

<h1>Title 1</h1>

<h2>Title 2</h2>

<ul>

<li>Part of the reason a well structured HTML document is so important</li>

<li>is because HTML elements inherit stylistic properties</li>

<li>You can see it's basic structure from the way I have formatted the tags with tabbing</li>

</ul>

<h3>Title 3</h3>

<ol>

<li>but here is a more visual representation of the hierarchy of tags</li>

<li>Tags that contain other tags are parents</li>

<li>and the tags inside of them are their children in the following tree representation</li>

</ol>

</body>

</html>

## 2.4.4 Activity - Apply your own selectors

Now it's your turn to practice with some selectors of your own.

Let's say you have the following [CodePen](http://codepen.io/w3devcampus/pen/RoKLKE/):

HTML code:

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="utf-8">

        <title>My HTML page</title>

        <link rel="stylesheet" href="style.css">

    </head>

    <body>

        <h1>Main Title</h1>

        <p>

In unit 2.3, we defined a CSS selector as the portion of the CSS rule that tells the browser on which HTML element to apply the defined style.

When your HTML is simple, the selectors can be simple as well. The most basic selectors simply mirror the HTML tag. For example "p" attaches to all &lt;p&gt; tags, "img" will attach to all &lt;img&gt; tags and so on. As you can imagine, there will often be times when you don't want every single HTML element of a particular type to have identical style. In Module 3, we'll discuss a variety of ways to use selectors to attach to specific HTML elements.

In unit 2.2, we briefly mentioned the fact that properties apply to the entire hierarchy of HTML elements to which they are attached. This means that you will have to be very careful which selectors you choose to use in combination with your chosen style. When choosing your selector you might want to keep the following aspects of an HTML element in mind

            </p>

            <h2>Sub Title 1</h2>

            <ul>

                <li>How many of these HTML elements are on my page?</li>

                <li>Do I want this style to apply to every one of these elements?</li>

                <li>What are this HTML element's children</li>

            </ul>

            <h2>Sub Title 2</h2>

            <ol>

                <li>and do I want this style to apply to them as well?</li>

                <li>Is this element a block element or an inline element</li>

                <li>and does this style make sense in that context?</li>

            </ol>

    </body>

</html>

CSS code:

/\*selector here\*/  {

    background-color: #ccffcc;

    color: #336600;

}

/\*selector here\*/  {

    background-color: #336600;

    color: #ccffcc;

}

/\*selector here\*/  {

    border: 3px solid;

}

/\*selector here\*/  {

    background-color: #ffff99;

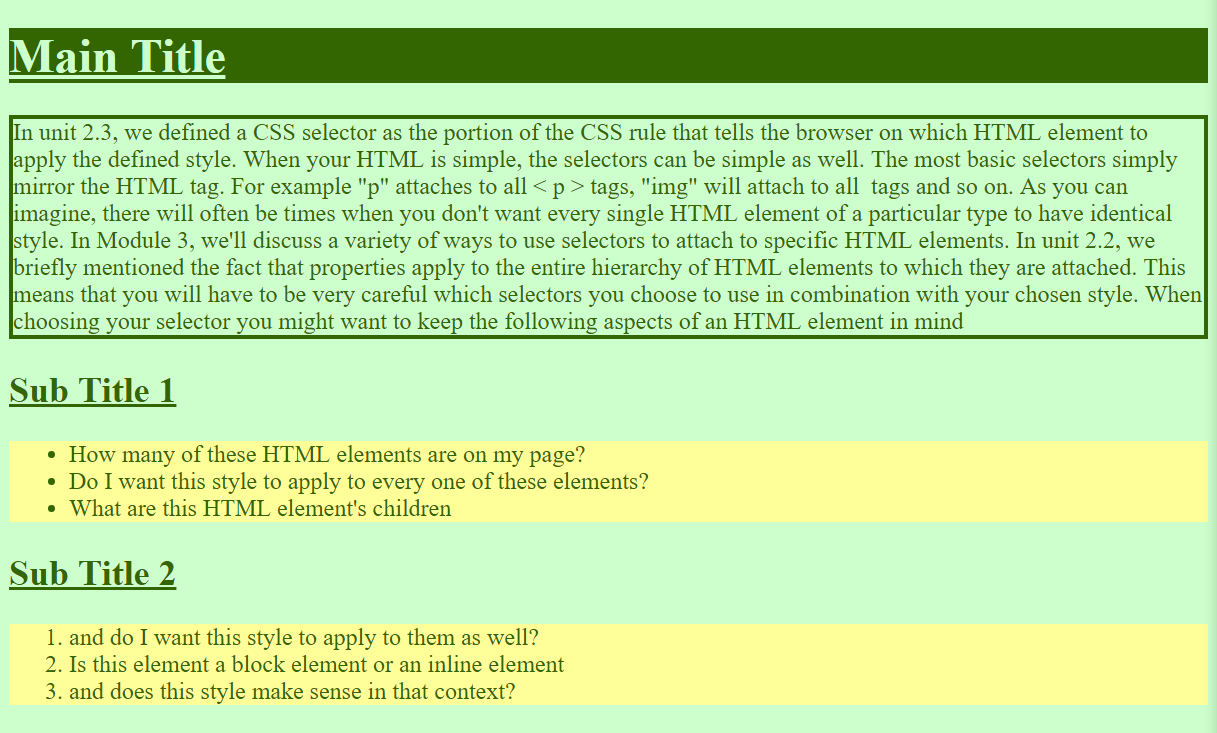
}

/\*selector here\*/  {

    text-decoration: underline;

}

Note that there are HTML and some CSS rules, but the rules are missing their selectors.  
You will need to figure out which selectors belong on top of each of the 5 rules so that your final site looks exactly like this:



**HINT**: Some of the rules require multiple comma separated selectors.

## 2.5.1 What is a property?

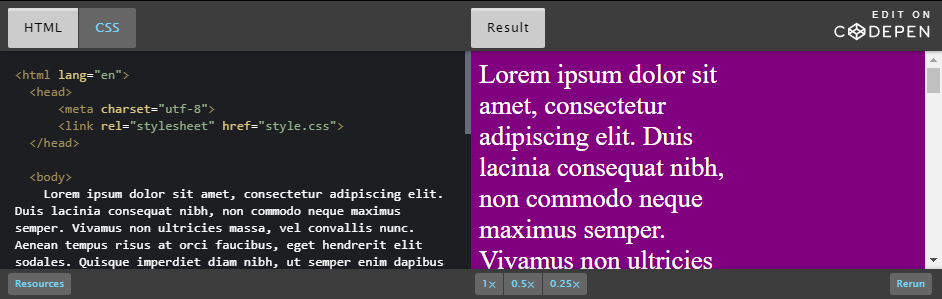
In unit 2.3, we briefly introduced you to "properties", the part of the CSS rule that tells the browser how to style specific aspects of the selected HTML element.

There is a huge array of different aspects you can style: color, font, size, spacing and much more! You can find the complete [list of CSS properties](https://www.w3.org/Style/CSS/all-properties.en.html) (from W3C's Web site). Every property has its own collection of possible values. Some require text input, some specific keywords, some numerical input, etc.

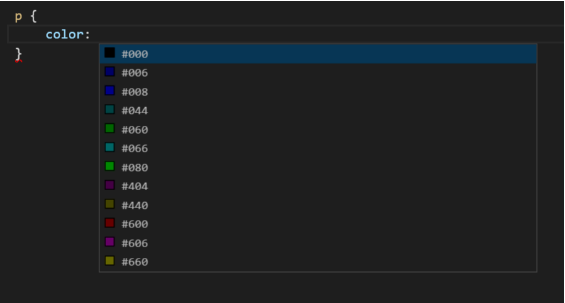
Here are some examples of properties that each accept a different style of value:

1. body {
2. background-color: purple; /\* key word \*/
3. color: #FFFFFF; /\* HEX code \*/
4. width: 60%; /\* percentage \*/
5. font-size: 20pt; /\* numerical value \*/
6. }

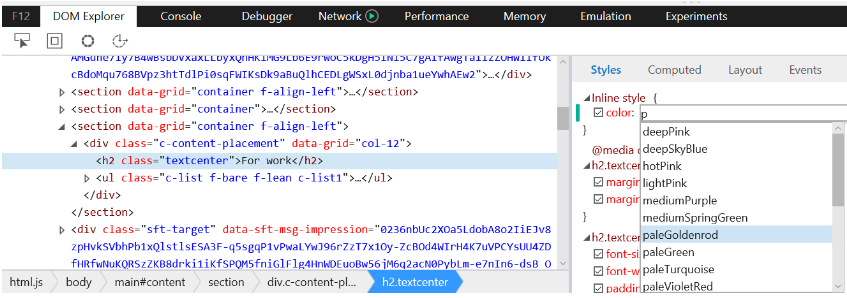
The above style is put into action in the [following CodePen](https://codepen.io/techie4good/pen/bwbLJP):



Make sure to look up what the available values are before using a property, because if the browser doesn't understand your value it will typically just skip applying any style. This is where programs like Visual Studio Code come in handy because as you type out the property, the program will automatically suggest possible values.



Sometimes the easiest way to learn about different properties is to explore the style on one of your favorite Web sites. You can use your browser tool to inspect an HTML element. Here is an example of inspecting a title:



If you are not sure how to do this, please refer to the demo in unit 1.3 to see this in action while inspecting CSS Zen Garden designs.

As you can see, the browser tools display the value of the properties, and if you click into that space it will even give you some of the different value options and you can even change them and watch the Web site update dynamically.

## 2.5.2 Color properties

Color is one of the first things you'll want to explore when designing your Web site. Thankfully, CSS provides a wide array of tools for you to control the color of different HTML elements. There are basic color properties:

### color

[[*Documentation*](https://www.w3.org/TR/css3-color/#foreground)]

This property sets the foreground color of an element's text content. By default, all text content will be set to black. If you set the color on one HTML element it will be inherited by all HTML elements within that. For example, if you set the color property on the body tag to blue, all text on your web page will be changed to blue, unless that text has a more specific color property that will override it.

### background-color

[[*Documentation*](https://www.w3.org/TR/css3-background/#the-background-color)]

This property sets the background color of an element. This color then represents the exact space the element takes up, which is always a rectangular area. The default value is 'transparent' which means whatever is behind the element will shine through. Note that background-color is one example of a property that is not inherited, so you will have to directly set the background-color on each element. To set the overall color of your page, apply a background color to the body tag, and since all other element's background colors will be transparent by default, it will appear as if everything has that same background color.

### color as a value

[[*Documentation*](https://www.w3.org/TR/CSS22/syndata.html#value-def-color)]

These color properties take in a color as their value, and there are three different ways you can define that color: keyword, a HEX code or an rgb value.

#### keyword

Probably the simplest and least flexible way to set colors is using a keyword. A keyword is one of the predefined colors like "blue" or "green".

1. body {
2. background-color: teal;
3. }

The list of color keywords is: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow. You can read more about these keyword colors [here.](https://www.w3.org/TR/CSS22/syndata.html#value-def-color)

#### HEX

A HEX code is a 6 character code to represent the color, giving you a lot more options. The 6 characters of the code are broken into 3 sets of 2, where each set of 2 represents the amount of either red, green or blue that makes up the color. These sets are hexadecimal numbers, which means that each ranges between 00 to FF where 00 means no color and FF means all of that color. Thus #000000 represents pure black and #FFFFFF represents pure white.

When using a hex code in CSS you must put a hash character in front of the 6 characters like so:

1. body {
2. background-color: #00CC00; /\* green \*/
3. }

#### decimal

You can also specify colors using rgb in decimal form like so:

1. body {
2. background-color: rgb(0,204,0); /\*same green as above\*/
3. }

This will give you the same range as HEX values. This method is a less common, but it's up to you which method of specifying colors you prefer.

Here are some of the colors you can use, and the three different ways you can set their value in the color or color-background property:

**maroon** #800000**red** #ff0000**orange** #ffA500**yellow** #ffff00**olive** #808000

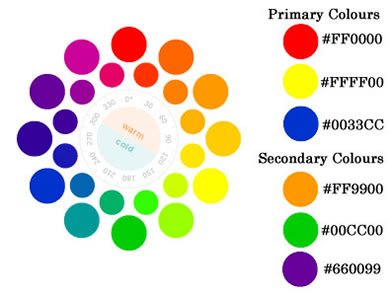
**purple** #800080**fuchsia** #ff00ff**white** #ffffff**lime** #00ff00**green** #008000

**navy** #000080**blue** #0000ff**aqua** #00ffff**teal** #008080

**black** #000000**silver** #c0c0c0**gray** #808080

You can see these color properties in action using all three approaches to setting the value [in this CodePen.](https://codepen.io/techie4good/pen/PGYRYL)

### Other resources:



* We will discuss how to pick a good color palette for your site in Module 5, but in the meantime, [here is a good wiki article from the W3C](https://www.w3.org/wiki/Colour_theory) discussing the general use of colors on the Web.
* More color units are described in the [CSS Color Module Level 3 specification.](https://www.w3.org/TR/css3-color/#colorunits)

## 2.5.3 Font properties

Font is an extremely important part of how you communicate content to your user.

### font-family

[[*Documentation*](https://www.w3.org/TR/css-fonts-4/#font-family-prop)]

Since not all fonts are available on all computers (there are thousands of fonts, and most are not free), CSS provides a system of fallbacks. You list the font that you want first, then any fonts that might fill in for the first if it is unavailable, and you should end the list with a generic font, of which are: [**serif**](https://www.w3.org/TR/css-fonts/#serif-def)**,**[**sans-serif**](https://www.w3.org/TR/css-fonts/#sans-serif-def)**,**[**monospace**](https://www.w3.org/TR/css-fonts/#monospace-def)**,**[**cursive**](https://www.w3.org/TR/css-fonts/#cursive-def) and [**fantasy**](https://www.w3.org/TR/css-fonts/#fantasy-def)**.**

1. p {
2. font-family: Helvetica, Verdana, sans-serif;
3. }

This property sets the font face. There is a collection of Web safe fonts that generally each browser has agreed to support, but there are an unlimited number of different fonts. The problem is they might not all look the way you want them to on different browsers.

That is why this property "font-**family**" allows a list of fonts in the order of your preference. This comma-separated list orders your font preference from left to right. In our above example, our first choice is Helvetica, if the browser doesn't support that it will move to the next on the list, Verdana, and if it still doesn't support that it will just pick any sans-serif font it does support. You should always end your font family with fonts that are likely to be supported by the browser, this way you are guaranteed to have control over the font-family.

See this [quick reference](https://www.w3.org/Style/Examples/007/fonts) listing what the fonts look like with different properties.

Something to keep in mind: some fonts have names with multiple words like "Times New Roman" or "Century Gothic". When using these fonts you'll need to surround the entire name with quotes so the browser understands that is a single font name like so:

1. p {
2. font-family: "Times New Roman", "New Century Schoolbook", serif;
3. }

### font-size

[[*Documentation*](https://www.w3.org/TR/css-fonts-4/#font-size-prop)]

1. h1 {
2. font-size: 2.5em;
3. }

CSS offers a number of different units for expressing length. Font-size sets the overall scale of your text. You can use a lot of different units to set the font size. Some of these units you are probably familiar with if you have used text editors before such as pt size or you can use px size. However, these methods are not advised because they are static and will not adapt based on screen size. It's better to use relative units, such as em, instead.

Use ems to make scalable style sheets! This is especially important for users who have special font preferences due to accessibility requirements. To use em, do not set font-size on the body tag, but instead set the size for each element in relation to the user's default. For example, 1em is the default, 2em is twice as big, 0.5em is half as big etc.

For curious people (i.e. non mandatory material), here is an excerpt from ["The amazing em unit and other best practices"](https://www.w3.org/Style/LieBos3e/em.en.html) (Chapter 3 of the book "Cascading Style Sheet - designing for the Web" 3rd edition, by Håkon Wium Lie & Bert Bos):

Named after the letter "M", the em unit has a long-standing tradition in typography where it has been used to measure horizontal widths. For example, the long dash (—) often found in American texts is known as an "em dash" because historically, it has had the same width as the letter "M". Its narrower cousin (–), often found in European texts, is similarly referred to as "en dash".  
  
The meaning of em has changed over the years. Not all fonts have the letter "M" in them (for example, Chinese), but all fonts have a height. The term has therefore come to mean the height of the font – not the width of the letter "M".  
  
In CSS, the em unit is a general unit for measuring lengths (for example, page margins and padding around elements). You can use it both horizontally and vertically, and this shocks traditional typographers who have always used the em exclusively for horizontal measurements. By extending the em unit to also work vertically, it has become a very powerful unit – so powerful that you seldom have to use other units of length.

Another great resource is this article on [font sizing tips and tricks](https://www.w3.org/Style/Examples/007/units.en.html).

### font-weight

[[*Documentation*](https://www.w3.org/TR/css-fonts-4/#font-weight-prop)]

1. p {
2. font-weight: bold;
3. }

The weight of a font is the thickness of the letters. You can set this property using keywords with which you might be familiar: bold, normal or lighter. You can also set this property more specifically using numerical values 100, 200, 300, 400, 500, 600, 700, 800 or 900. Normal is represented as 400, whereas bold is 700.

Note that few fonts have settings for all values. If the value is not available, the browser will use the nearest available one. For example, if 800 is not available but 700 is, then the browser will display 700. Please try different fonts in [this Codepen](https://codepen.io/techie4good/pen/XNdrMw) to see how they look at each weight setting.

### font-style

[[*Documentation*](https://www.w3.org/TR/css-fonts-4/#font-style-prop)]

1. p {
2. font-style: italic;
3. }

The font style property adjusts the angle of the letters in relation to the horizontal plane. Italic forms are generally cursive in nature while oblique faces are typically sloped versions of the regular face.

### text-decoration

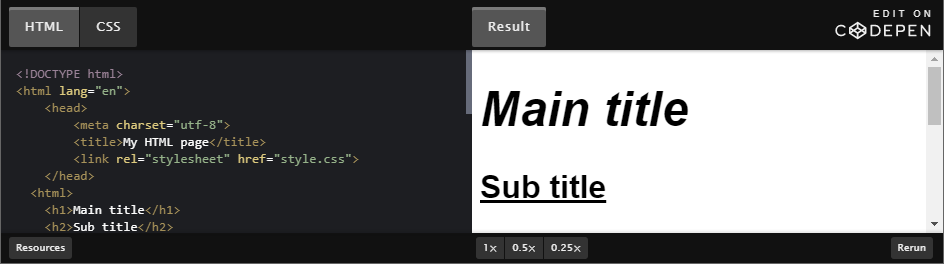
[[*Documentation*](https://www.w3.org/TR/css-text-decor-3/#text-decoration-property)]

1. p {
2. text-decoration: underline;
3. }

Text-decoration adds a line across your text. You can set this line to be underneath your text, underline, through your text, line-through, or on top, overline.

### Example:

Here is a [CodePen exploring each of these styles](https://codepen.io/w3devcampus/pen/ExNExxq).



### HTML code:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>My HTML page</title>
6. <link rel="stylesheet" href="style.css">
7. </head>
8. <body>
9. <h1>Main title</h1>
10. <h2>Sub title</h2>
11. <p>
12. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis lacinia consequat nibh, non commodo neque maximus semper. Vivamus non ultricies massa, vel convallis nunc. Aenean tempus risus at orci faucibus, eget hendrerit elit sodales. Quisque imperdiet diam nibh, ut semper enim dapibus et. Interdum et malesuada fames ac ante ipsum primis in faucibus. Aenean in feugiat neque. Nunc eget libero mauris. Maecenas condimentum luctus nulla.
13. Nulla a sem orci. Cras eget neque viverra, condimentum nulla et, tincidunt libero. In sit amet quam purus. Aliquam erat volutpat. Sed hendrerit urna quis sapien mattis dictum. Etiam vehicula tortor eu libero finibus dapibus. Mauris nunc neque, sodales nec est sed, gravida convallis sem. Nam vulputate sed est sed eleifend. Quisque sodales elit at ornare vulputate.
14. </p>
15. </body>
16. </html>

### CSS code:

1. body {
2. font-family: Helvetica, Verdana, sans-serif;
3. font-size: 12pt;
4. }
5. h1 {
6. font-size: 3em;
7. font-style: italic;
8. }
9. h2 {
10. font-size: 2em;
11. text-decoration: underline;
12. }
13. p {
14. font-weight: bold;
15. }

### Resources

There are even more ways to adjust text appearance and you can read more about them here:

* [CSS Fonts Module Level 4](https://www.w3.org/TR/css-fonts/)
* [CSS Text Decoration Module Level 4](https://www.w3.org/TR/css-text-decor-4/)

## 2.5.4 Spacing properties

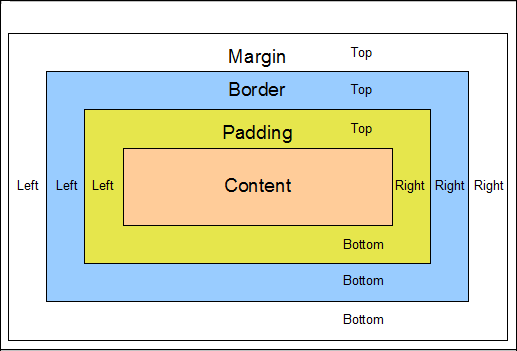
CSS provides a great set of tools to help you position the HTML elements on your page, and we will cover that in depth in Module 4. For now we will talk about how to apply white space around individual HTML elements.

There are two different ways you can define white space:

* in absolute terms: using an exact number of pixels,
* and in relative terms: using percentages or ems.

For now, we'll use pixels because that is easier to learn. However, ultimately you will want to use percentages and ems so your content adapts to different screens. We will discuss how to use percentages in Module 4.

When you view an element in your browser tools you can see the white space around it represented like so:



The above image is called the "box model", which we will get into more detail about in Module 4. For now, you can see that the space around the content is broken into three distinct regions.

### Padding

[[*Documentation*](https://www.w3.org/TR/CSS2/box.html#propdef-padding-top)]

1. p {
2. padding: 20px;
3. }

"Padding" is the white space that sits closest to an HTML element. Many elements already have a default padding defined. For example, ul elements by default are indented to the left a bit because they have a left padding.

You can set the padding on an element's four sides independently using padding-top, padding-right, padding-bottom and padding-left. Or you can use the more compact padding: 10px 15px 20px 25px. In this case, the order of the numbers sets the top, right, bottom and left paddings. In the example below, I collapsed all of these and just set the padding on all four sides to be 20px. Here is a [CodePen that demonstrates all these different ways to set padding](https://codepen.io/w3devcampus/pen/zeEXao).

### Border

[[*Documentation*](https://www.w3.org/TR/css3-background/#borders)]

1. p {
2. border: 1px black solid;
3. }

The "border" is the area outside the padding of an HTML element. By default, borders are set to be empty, but you can set their width, color, pattern, even an image! Like padding, you can even adjust the four sides of a border independent of one another using border-top, border-right, border-bottom or border-left. You can also adjust the different aspects of a border with border-width, border-color, and border-style. In the above example, I collapsed all of these properties into a single simple property and value set.

### Margin

[[*Documentation*](https://www.w3.org/TR/CSS2/box.html#propdef-margin)]

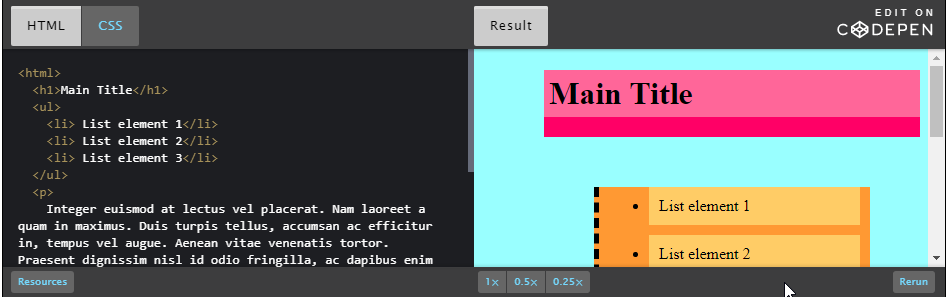
1. p {
2. margin-bottom: 50px;
3. }

An HTML element's "margin" is the white space that sits outside the border. Margins of HTML elements interact with one another on the page to determine how they are arranged on the page. A lot of elements have default margins applied.

For example, the body tag typically has a margin that causes any content to not extend all the way to the extreme edge of the page. Be careful, margins can be tricky. When two margins touch they "collapse" such that the space between the elements is equivalent to the larger of the two margins. Like the above properties, you can also set the margins on each side independently using margin-top, margin-left, margin-bottom and margin-right.

### Example:

Here is a [CodePen exploring padding, border and margin](https://codepen.io/techie4good/pen/vXBRJP).



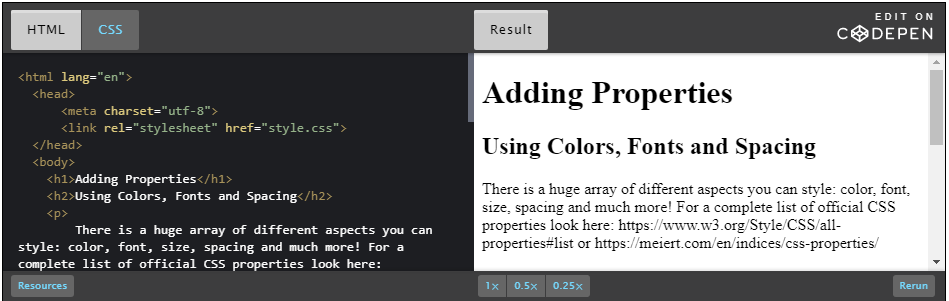
### CSS code:

1. body {
2. background-color: #99ffff;
3. margin-top: 20px;
4. margin-left: 70px;
5. }
6. h1 {
7. background-color: #ff6699;
8. border-bottom: 20px #ff0066 solid;
9. margin-bottom: 10px;
10. padding: 5px;
11. }
12. ul {
13. background-color: #ff9933;
14. border-left: 5px black dashed;
15. margin: 50px;
16. }
17. li {
18. background-color: #ffcc66;
19. margin: 10px;
20. padding: 10px;
21. }
22. p {
23. background-color: #ccff99;
24. border: 10px white double;
25. padding: 0px;
26. margin: 0px;
27. }

## 2.5.5 Activity - Adding your own properties

Now that you have a few properties in your CSS toolbox, let's practice using them.

Here is some HTML and CSS, but as you can see the CSS rules have selectors but no properties. Check also the corresponding [Codepen](https://codepen.io/w3devcampus/pen/WNozNxK" \t "_blank):



HTML code:

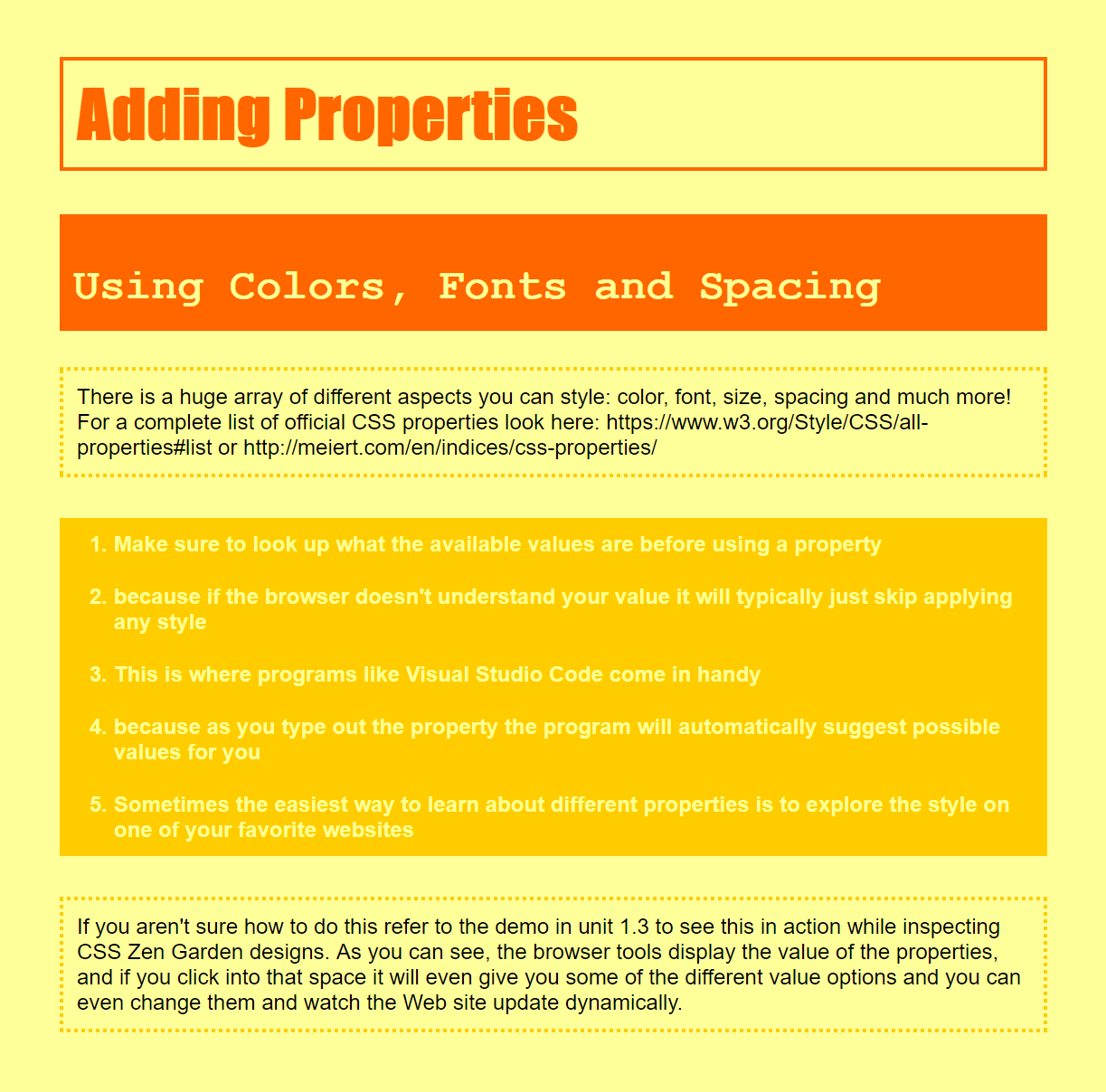
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <title>My HTML page</title>
6. <link rel="stylesheet" href="style.css">
7. </head>
8. <body>
9. <h1>Adding Properties</h1>
10. <h2>Using Colors, Fonts and Spacing</h2>
11. <p>
12. There is a huge array of different aspects you can style: color, font, size, spacing and much more! For a complete list of official CSS properties, please check: https://www.w3.org/Style/CSS/all-properties#list or https://meiert.com/en/indices/css-properties/
13. </p>
14. <ol>
15. <li>Make sure to look up what the available values are before using a property</li>
16. <li>because if the browser doesn't understand your value it will typically just skip applying any style</li>
17. <li>This is where programs like Visual Studio Code come in handy</li>
18. <li>because as you type out the property the program will automatically suggest possible values for you </li>
19. <li>Sometimes the easiest way to learn about different properties is to explore the style on one of your favorite websites</li>
20. </ol>
21. <p>
22. If you aren't sure how to do this refer to the demo in unit 1.3 to see this in action while inspecting CSS Zen Garden designs. As you can see, the browser tools display the value of the properties, and if you click into that space it will even give you some of the different value options and you can even change them and watch the Web site update dynamically.
23. </p>
24. </body>
25. </html>

CSS code:

1. body {
3. }
5. h1 {
7. }

10. h2 {
12. }
14. ol {
16. }
18. li {
20. }
22. p {
24. }

These CSS rules have selectors, but no properties. For this activity, it is up to you to add the correct properties and set their values appropriately to achieve this final style:



Here are the color HEX code used above:

* light yellow: #ffff99
* medium yellow: #ffcc00
* orange: #ff6600

Here are the fonts used above:

* Helvetica
* Impact
* Courier New

**Note**: This exercise uses three common fonts, Helvetica, Impact and Courier New. If you are on Windows or MacOS, you almost certainly have all three of them. But Linux, Android and others usually do not offer Impact and Courier New. Microsoft made free versions available and on most Linux distributions there is an easy way to install them: look in your package manager for a package called *ttf-mscorefonts-installer* (or similar). If you don't want to install fonts and you are on Linux, you can try Courier instead of Courier New. Availability of other fonts differs a lot. Maybe you can look in the system settings to see if there is an interesting font available to use instead of Impact.  
  
But this points to a fundamental limitation of font-family in CSS: you cannot be sure that somebody who views your pages has all the fonts that you chose. The solution is to embed the fonts in the style sheet (a technique known as Web Fonts). But that is beyond the scope of this course.

## 2.6.1 Style studies

In each unit, we will have a section like this one where we profile specific aspects of Web sites and the various ways you can use CSS to style them. This is intended to give you practical examples of how to apply the CSS you are developing along the way.

Each style study will discuss the different concerns of styling a given element and three sample styles.

## 2.6.2 Titles

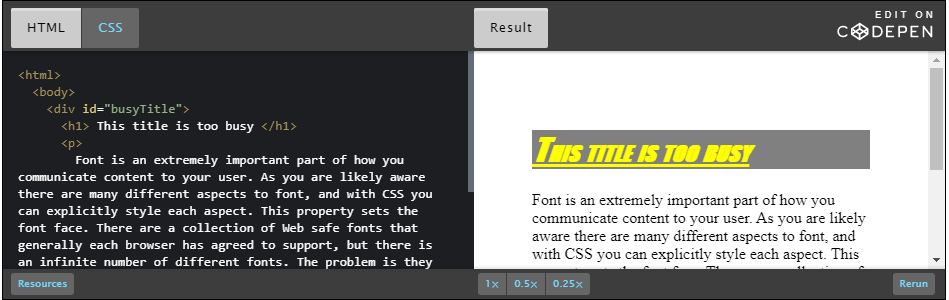
There are a couple different categories of text on a Web page: titles, body text, links, captions, etc.

You'll want to style each of these differently to help your user understand the proper context for your text. One of the most important categories of text to stand out are your titles.

There are different aspects of text you can alter to make it stand out.

* size
* font
* capitalization
* color
* emphasis
* weight

However, you should only alter a few of these following aspects at a time to prevent your titles from being too distracting. The below is an example of using too many different aspects of font for emphasis:

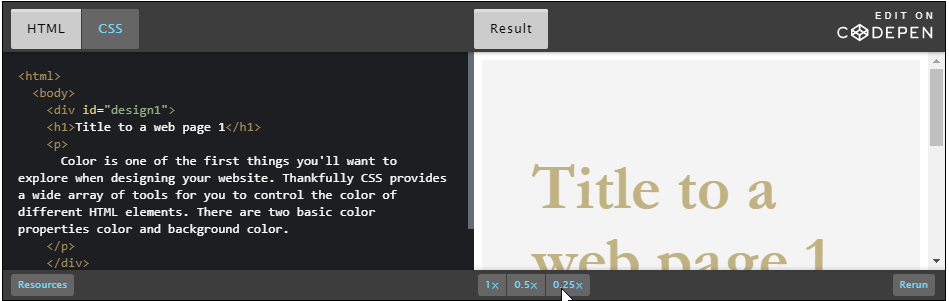


### CSS code:

1. #busyTitle h1{
2. font-size: 2em;
3. font-family: Impact;
4. color: yellow;
5. background-color: gray;
6. font-style: italic;
7. font-variant: small-caps;
8. font-weight: bold;
9. text-decoration: underline;
10. }

### Title 1

This title uses soft clean colors based on print media, so we chose a serif font. We also increased the size and color to help the title appear more prominent than the body text.

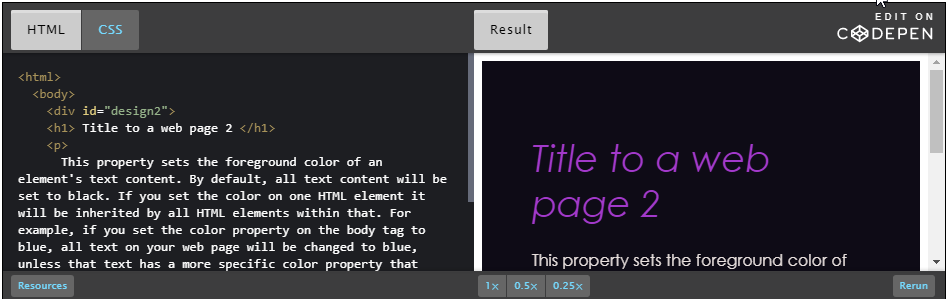


### CSS code:

1. #design1 {
2. background-color: #F4F4F4;
3. font-family: "Lucida Sans Unicode", sans-serif;
4. }
6. #design1 h1 {
7. color: #C0B283;
8. font-size: 4em;
9. font-weight: 700;
10. font-family: Garamond;
11. width: 300px;
12. }
14. #design1 p {
15. color: #373737;
16. font-size: 1.2em;
17. }

### Title 2

This design is intended to look futuristic, so it only uses sans-serif, thin font with high contrast colors.

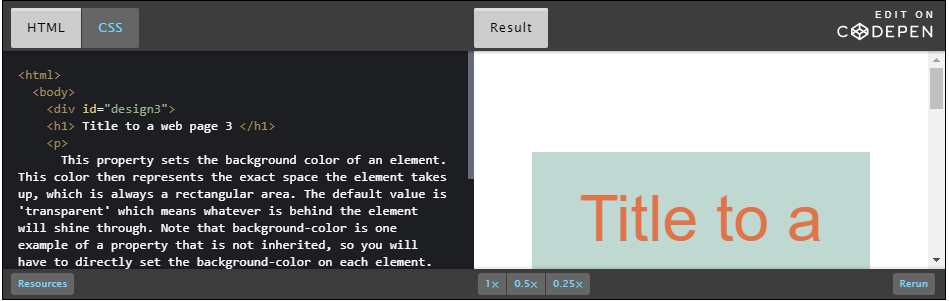


### CSS code:

1. #design2 {
2. font-family: Century Gothic, sans-serif;
3. background-color: #0E0B16;
4. }
6. #design2 h1 {
7. font-weight: 400;
8. font-size: 2.3em;
9. color: #A239CA;
10. font-style: italic;
11. }
13. #design2 p {
14. color: #E7DFDD;
15. }

### Title 3

This design is based on pastel primary colors and uses color as a highlight against the default white background. We have achieved the separation between title and body text by setting its background color separately and giving it a bottom border.



### CSS code:

1. #design3 {
2. color: #DF744A;
3. font-family: Arial, sans-serif;
4. }
5. #design3 h1 {
6. background-color: #BFD8D2;
7. text-align: center;
8. font-size: 4em;
9. font-weight: 100;
10. padding: 30px;
11. border-bottom: 5px #DCB239 solid;
12. font-family: Helvetica, sans-serif;
13. }
14. #design3 p {
15. background-color: #FEDCD2;
16. padding: 50px;
17. }

[Here is a CodePen of all the above examples of different title designs for you to play around with](https://codepen.io/w3devcampus/pen/LYWRVvP).

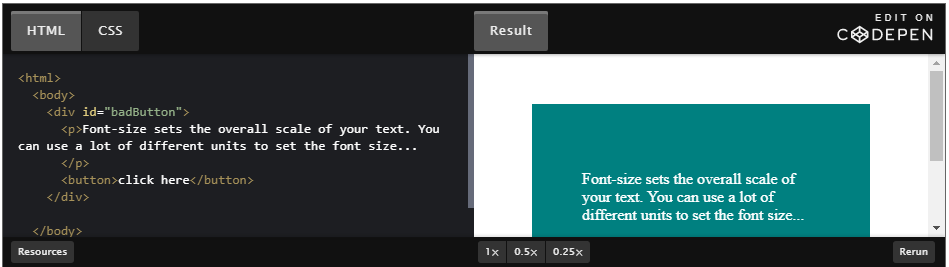
## 2.6.3 Buttons

Buttons are a key way that your users will interact with your page. Often buttons are a call to action for your user- so you will want them to stand out!

When styling a button you aren't just styling the text, but also the area around it, including the border. By giving them a distinct background color or border you make it clear that this whole area is "clickable".

Typically your user will expect your button to be wider than it is tall, to accomplish this you'll want to set the left and right padding to be greater than the top and bottom paddings.

Here is an example of a button where it's not immediately clear that it is actually a button ("click here").



More examples of buttons are described below.

### Button 1

This design is a pretty traditional button where it has a distinct background-color and border-color to help it stand out from both the background and the body text.



### CSS code:

#design1 {

    background-color: #94618E;

    color: #F8EEE7;

    font-family: Corbel;

}

#design1 button {

    background-color: #F4DECB;

    padding: 10px 25px;

    border: 5px solid #F8EEE7;

    color: #49274A;

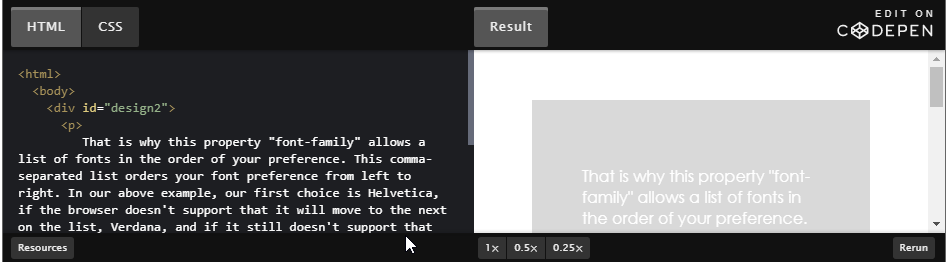
    font-weight: bold;

    font-size: 1.2em;

 }

### Button 2

This is a more modern button design where there is no distinct border, but just a flat color background. You'll note the poor accessibility of the text and button (for example, no sufficient contrast between foreground and background - find out more in these [tips for designing](https://www.w3.org/WAI/tips/designing/)).



### CSS Code:

#design2 {

    background-color: #D9D9D9;

    color: white;

    font-family: "Century Gothic", sans-serif;

}

#design2 button {

    background-color: #4484CE;

    border: 0px;

    color: white;

    padding: 10px 50px;

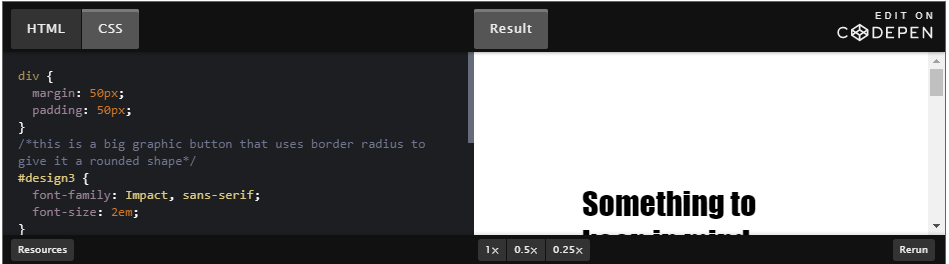
    font-family: Impact, sans-serif;

    font-size: 1.3em;

}

### Button 3

This is a big graphic button that uses border-radius to give the button rounded corners.



CSS code:

1. #design3 {
2. font-family: Impact, sans-serif;
3. font-size: 2em;
4. }
6. #design3 button {
7. font-family: Impact, sans-serif;
8. font-weight: 100;
9. background-color: white;
10. border: 7px #EC576B solid;
11. border-radius: 20px;
12. font-size: 0.8em;
13. }

In summary, here are [three different ways to style buttons you can play around with](https://codepen.io/techie4good/pen/ORXzoo?editors=1100#0).

## 2.6.4 Activity - Titles and buttons

Now that you know a bit more about how CSS is written, you will start to understand and notice more of the design elements as you navigate around the Web.

In this module's style studies section, you learned some effective ways to style titles and buttons. You also learned about some design choices that make your titles and buttons not as effective...

* Titles shouldn't be overly styled. They should only have a few properties set to help them stand out from body text, but not too many otherwise they will be too busy.
* Buttons should send a clear message that they need to be interacted with. If a button doesn't show up as a rectangle, a user typically won't understand that it's a button.

For this activity, please find a Web page that has either titles or buttons and discuss how effectively they are styled. In the discussion below, answer the following questions:

* Which Web page are you evaluating?
* Is their use of titles or buttons as effective as they can be?
* What do you like about their title or button design?
* What would you change about their title or button design to be more effective?

## Module 2 project - About me page

Now that you have played around with applying selectors and properties, you have the tools to make a much more sophisticated Web page than "Hello Your World".

For your module project, you are going to create a page describing yourself using your favorite colors. Your Web page must have the following content:

* an h1 title with your name
* an h2 title with your favorite quote
* a paragraph describing your favorite hobbies
* an ordered list of your favorite foods
* an unordered list of your favorite websites, including links to them

And in addition, you must employ the following styles:

* + 3 different font colors
  + 2 different background colors
  + a serif font
  + a sans-serif font
  + italic text
  + bold text
  + 1 border
  + extra margin between the edge of the screen and your content
  + extra padding around your list items

For example, here is what my page looks like:



Note that you are welcome to look up other styles and apply them however you like as long as you at least have the above requirements met. We've set up a discussion forum below, in case you want to share your work!

Remember that if you want to post an assignment without personal information of yours in, feel free to change the content of the assignment to anything you like, so long as you incorporate the required code features. For example, you could instead create a profile page for an imagined character, or a brief biography of someone you admire.

## Module 2 learnings

### After this module you should be able to...

* understand how HTML and CSS work together to create Web content
* describe what the different parts of a CSS rule are
* employ basic selectors and combine them using the comma
* apply properties that style color, font and spacing

### In module 3, you will then learn:

* classes, IDs
* contextual selectors
* pseudo classes

## 3.1.1 Welcome to Module 3

In this module, we're going to be talking about selectors.

At this point, we've seen basic selectors, but we're starting to have a slightly more complex HTML document,

We’re going to need a lot more tools than just the HTML tag name.

So, in this module, we're going to talk about classes and IDs, pseudo-classes,

contextual selectors, and then how you combine all these selectors together and

understand the Cascading part of Cascading style sheets.

## 3.1.2 Module 3 outline

In this module, we'll learn:

* How to use classes and IDs to independently target HTML elements of the same type
* How to apply different style to the same element based on the way the user interacts with that element using pseudo-classes
* How to scope style rules using contextual selectors and the HTML inheritance structure of your document
* What the "Cascading" part of "Cascading Style Sheets" means

## 3.1.3 The power of selectors

In our last module, we learned about something called Selectors.

This entire module is all about exactly that: Selectors.

Because so far, we've actually seen pretty simple ones.

They probably look something like this, and as you can see,

this portion before the open curly brace, that's what we call Selector.

And so far, we've only used ones that match the name of the HTML tag,

which is totally fine. If you want every single paragraph to look

exactly the same, or say you only have one header on your page.

As our Web pages get a lot more complicated,

we have a lot more of the same HTML element

and chances are we're going to want them to look different from one another.

Thankfully, there's a whole list of different types of Selectors

that we can use to specifically target each one of these HTML elements.

In this module, we're going to

talk about things like classes, IDs, contextual selectors, pseudo-classes.

All those things are going to help us really narrow our CSS.

For example, we've probably written some CSS like this for this HTML page.

But, as you can see, every single one of the paragraphs ends up

looking exactly the same. Because as long as I use the selector P

for paragraph, it applies to every paragraph.

Here is a preview of some of the things we're going to learn in this module.

Now, I have three different rules,

for each of my three paragraphs. And they all look completely different

which makes my Web site a lot more dynamic. So, learning to

properly select your HTML elements through CSS is crucially important

to making sure that you get the most out of CSS.

he CSS code mentioned in above video:

p {

    color: white;

    background-color: midnightblue;

    font-size: large;

}

.middle {

    color: darkviolet;

    background-color: lightgray;

    padding-left: 120px;

    padding-right: 120px;

    font-size: large;

}

#bottom {

    background-color: transparent;

    color: black;

    font-family: 'Franklin Gothic Medium';

}

## 3.2.1 Classes and IDs

Often, a single page will have a whole bunch of the same html tag, but that doesn't necessarily

mean you want each tag of that type to look exactly the same. Say, for example, you have

lots of different links on your page. Things that take you to different pages, things that

take you back, and things that navigate you around the site. You might want to make the

ones that take you back to your home page a little more noticeable. But how do you do that?

They're all link tags. The answer? Classes and IDs!

Classes and IDs are special names that you create and you assign to specific HTML tags.

Once you've got these names attached, you can use those as selectors for your CSS rules.

Assign an element ID when you're only going to assign it to a single thing. Say for example,

you have lots of buttons on your page but you want to make the one that charges the

user's credit card extra noticeable. You can use that by giving it an id of say "pay button",

and then you can make sure you user knows exactly what that button is going to do when they click it.

Assign a class when you want a whole group of elements to all look the same.

Say for example, you want your navigation bar to be filled with links that all look very similar.

If you assign a class, that then say puts them all in the same place and gives

them the same font, your user will know which links are actually navigating them around

the site, and which ones are going to take them away from your site to a different one.

Now, lets take a look at a whole bunch of different examples of how you can use classes and ids

to target exactly what you want.

### Classes and IDs

Classes and IDs are "attribute selectors". This means that you can attach style to HTML elements based on that element's attributes. This empowers you to apply different style to items of the same HTML type.

#### Classes

[[Definition](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/class) and [documentation](https://www.w3.org/TR/html52/dom.html#element-attrdef-global-class)]

A class is an HTML attribute that specifies a name for a group of elements on the page. You can apply the class name to as many elements as you like, even if they are of different HTML tag types. You can use the class name as the selector, like so:

1. <p class="className">The intro paragraph</p>

Class names must be single words, but you can include digits and dashes as long as the name begins with a letter. Note that names are case sensitive.

To apply a CSS rule to a class, you must precede its name by a period ("."), like in the code below:

1. .className {
2. color: blue;
3. }

#### IDs

[[Definition](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/id) and [documentation](https://www.w3.org/TR/html52/dom.html#element-attrdef-global-id)]

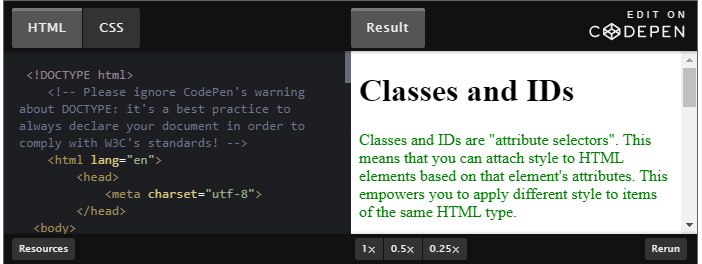
An ID is an HTML attribute that specifies a name or unique identifier for a particular HTML element. They are like classes with a very important distinction: the value of the ID attribute must be unique throughout the document. This lets you target a single HTML element for styling. You set the id of an HTML element like so:

1. <p id="MyFirstId"> This is an extra special paragraph </p>

ID names have the same rules as class names: start with a letter, can include numbers and dashes, no spaces. The way to create a selector for an ID is also similar to how you create a selector for a class, except you replace the period with a hash symbol ("#") like in the code below:

1. #MyFirstId {
2. color: blue;
3. }

### Example



### HTML code:

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="utf-8">

        <title>Classes and IDs</title>

        <link rel="stylesheet" href="style.css">

    </head>

    <body>

        <h1>Classes and IDs</h1>

        <p id="intro">

            Classes and IDs are "attribute selectors". This means that you can attach style to HTML elements based on that element's attributes. This empowers you to apply different style to items of the same HTML type.

        </p>

        <p class="odd">

            Classes are an HTML attribute that specifies a name for a group of elements on the page. You can apply the class name to as many elements as you like, even if they are of different HTML tag types. You can use the class name with a period in front as the selector.

        </p>

        <p class="even">

            Class names must be single words, but you can include digits and dashes as long as the name begins with a letter. Note that names are case sensitive.

        </p>

        <p class="odd">

            An ID is an HTML attribute that specifies a name or unique identifier for a particular HTML element. They are like classes with a very important distinction: the value of the ID attribute must be unique throughout the document. This lets you target a single HTML element for styling.

        </p>

        <p class="even">

            ID names have the same rules as class names: start with a letter, can include numbers and dashes, no spaces. The way to create a selector for an ID is also similar to how you create a selector for a class, except you replace the period with a hash symbol ("#").

        </p>

    </body>

</html>

### CSS code:

#intro {

    color: green;

}

.odd {

    color: blue;

}

.even {

    color: red;

}

## 3.2.2 Activity - Add your own classes and ids

Here is some CSS code:

body {

   background-color: #00ccff;

   color: white;

   font-family: Helvetica, sans-serif;

   margin: 35px 25px 0px 25px;

}

p,h2 {

   padding: 10px;

}

.topSection{

   background-color: #3300cc;

   color: #cccccc;

}

.bottomSection {

   background-color: #cccccc;

   color: #3300cc;

}

#importantItem {

   text-decoration: underline;

   color: #99ff99;

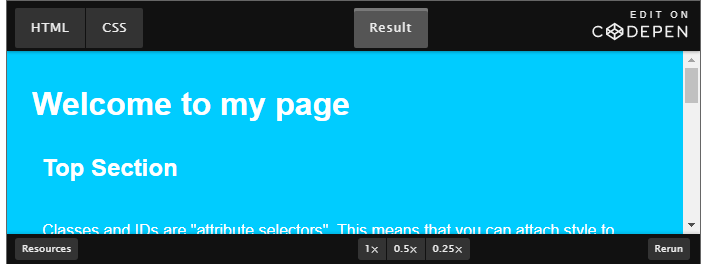
}

#unimportantItem {

   color: gray;

}

As you can note, it's not too interesting because not all of the styles are applied to the HTML as shown in the pen below:



In this activity, your job is to add the HTML id and class attributes to the correct elements so that you get a final result that looks like this:



## 3.3.1 CSS pseudo-classes

Sometimes, you'll want to adjust the style of an element, based on how the user

has interacted with it. The best example of this is links.

Often times, you'll notice a link is blue, and then after you click on it it turns purple.

This is the default style for the pseudo class visited.

There's also a bunch of other pseudo-classes that let you manipulate whether the user is

hovering over it with their cursor, or they're focusing on it and other such activity.

Let’s take a look at the different pseudo-classes and how you can design a really interactive Web site.

Pseudo-classes are a way to select HTML elements based on their state as opposed to their HTML structure.

Pseudo-classes must always be applied to an existing selector. Their "flag character" is the colon (":").  
Here are some of the most popular pseudo-classes.

### :link and :visited

[[Documentation](https://www.w3.org/TR/CSS22/selector.html#link-pseudo-classes)]

1. a:visited {
2. color: gray;
3. font-style: italic;
4. }

These pseudo classes are the ones you are probably most familiar with. Even on this page you've probably noticed that [links have different style](http://www.edx.org/) than paragraph text. The <a> tag by default sets the text color to blue with an underline, but have you ever seen a purple link? This is the "visited" pseudo-class that applies a different style to links that the user has already clicked. The opposite of visited is "link" which is a link a user has not yet clicked. These two states are mutually exclusive, meaning a link cannot be both at the same time.

### :hover, :focus and :active

[[Documentation](https://www.w3.org/TR/CSS22/selector.html#dynamic-pseudo-classes)]

The hover pseudo-class is applied when the user points at an object but doesn't activate it, most commonly when they let their mouse cursor lay on top of an element without clicking. Some form factors don't support this, such as touch devices or pen surfaces. This is a really good way to encourage a user to click a link and you will often see it used in navigation bars.

1. li:hover {
2. background-color: yellow;
3. }

The focus pseudo class applies when a user has chosen to begin interacting with an element, such as when they click into a form input field so that the input field is then ready to accept keyboard input.

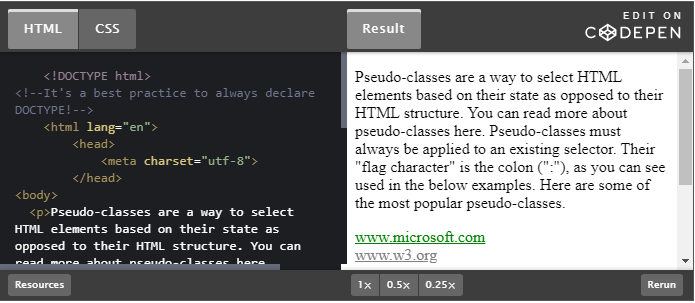
1. input:focus {
2. background-color: blue;
3. }

The active pseudo-class applies when an element is activated. This happens in the time between when the user clicks their mouse and they release it.

1. p:active {
2. color: red;
3. }

You can read more about pseudo-classes in this [MDN "Pseudo-classes"](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes) document.

### Example



## 3.3.2 Activity - Applying pseudo classes

Now it's your turn to try out some pseudo classes. Here is a Web page (see the CodePen [Practice with Pseudoclasses](https://codepen.io/techie4good/pen/rraNJr/)) and the CSS code:

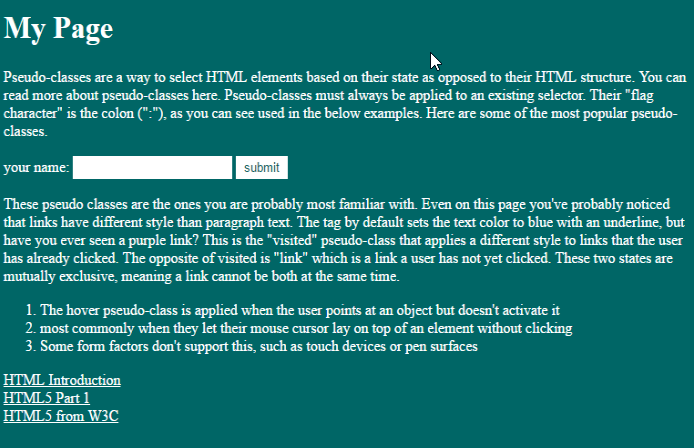
1. body {
2. background-color: #006666;
3. color: white;
4. }
5. h1 {
6. text-decoration: underline;
7. }
8. input {
9. border: 3px white solid;
10. }
11. input {
12. border: 3px yellow solid;
13. }
14. input {
15. background-color: yellow;
16. border: 3px yellow solid;
17. }
18. button {
19. background-color: white;
20. color: #006666;
21. border: 3px white solid;
22. }
23. button {
24. background-color: #006666;
25. color: white;
26. }
27. button {
28. background-color: #33cc99;
29. }
30. li {
31. background-color: white;
32. color: #006666;
33. }
34. li {
35. background-color: #33cc99;
36. color: white;
37. }
38. a {
39. color: white;
40. }
41. a {
42. color: #33cc99;
43. }

If you look at the CSS for this page you'll notice there are multiple CSS rules with the same selectors. That is because some of these rules need to have pseudo classes applied.

Please add pseudo-classes to the existing rules so that:

* The title is underlined when the user hovers their mouse over the text
* The input box gets a yellow border when the user hovers their mouse over the box
* The input box has a yellow background when the user clicks inside the box
* When the user hovers over the button it gets a background color of #006666 (dark green) and a text color of white
* When the user clicks the button it gets a background color of #33cc99 (light green)
* The background color of the list elements turns white and the text turns #006666 (dark green) when the user hover overs them
* When clicked, the list elements get a background color of #33cc99 (light green) while the text stays white
* The links at the bottom of the page start out as white in color and then when they are clicked they turn #33cc99 (light green)

The resulting output should look like this when the user has not interacted with the page in any way:



## 3.4.1 Contextual selectors

In the last section, we actually combined two selectors into one rule.

By adding a pseudo-class onto an existing selector,

we are further targeting a specific HTML element.

There's actually lots of situations in which you want to combine

multiple selectors. You can actually use

all of the selectors, we've seen so far

into a single rule. You do this

by putting them in a specific order

separated by spaces. You would call this approach

contextual selection, because each

selector is referencing the hierarchy

it's been placed in. Now, let's take a look

at how you can order all

the selectors you have in your toolbox

to really target that specific HTML element.

When you use two selectors separated by a space on a rule, you scope the rule to the elements that correspond to the selector on the right that are INSIDE the elements that correspond to the selector on the left. Let's say we have the following HTML:

<!DOCTYPE html>

<html lang="en">

    <head>

        <meta charset="utf-8">

    </head>

    <body>

        <img src="images/pic1.jpg" alt="pic 1" />

        <p>

            This is my paragraph full of useful information

            <img src="images/pic2.jpg" alt="pic 2" />

            Since there is text around these images, they should be styled a little differently.

            <img src="images/pic3.jpg" alt="pic 3" />

        </p>

        <img src="images/pic4.jpg" alt="pic 4" />

    </body>

</html>

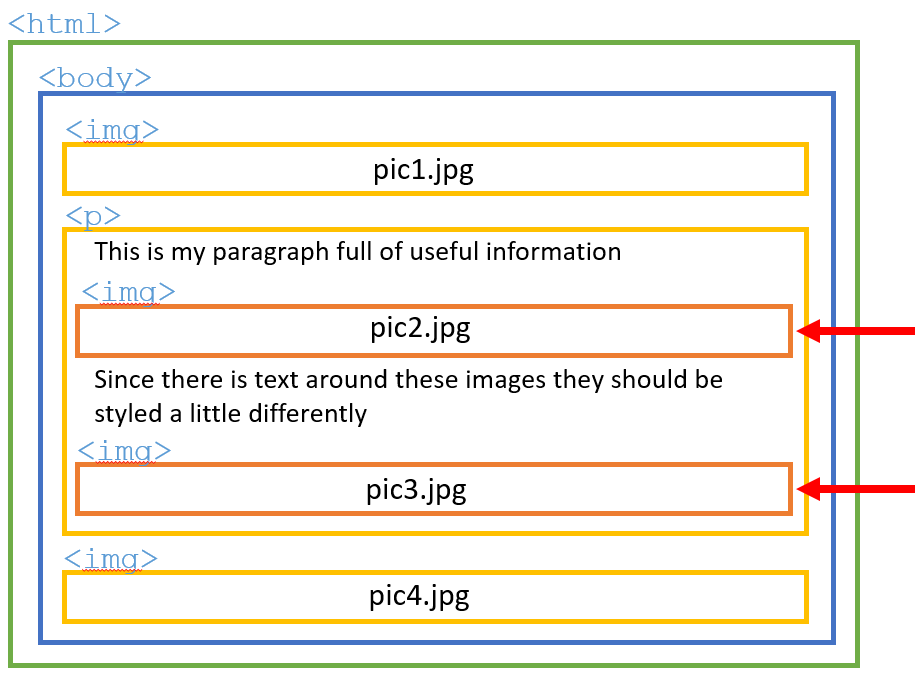
If we applied the following CSS rule then the images INSIDE the paragraph would be set to a width of 100px, but that rule would not apply to the images outside the paragraph.

p img {

width: 100px;

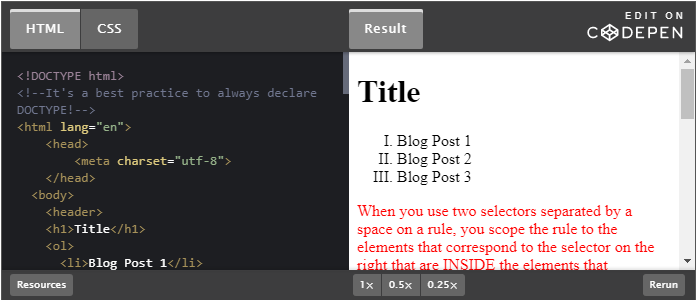
}

Below is a diagram of the given HTML with the two imgs that will be styled by the above rule are indicated by the red arrows.



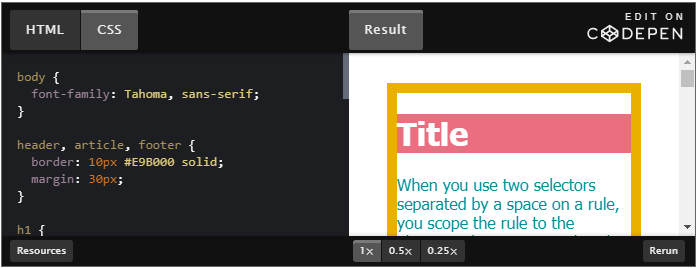
As your web pages get more complex, contextual selectors become more important, because it becomes increasingly unmanageable and inefficient to apply classes and IDs to each individual item, as a page scales up. Contextual selection becomes especially useful when you structure your HTML with section tags like header, section, article and footer.

Pay attention to the styles of the paragraphs and lists in the following example:



## 3.4.2 Activity - Contextual selectors

For practice, we are going to learn how to better scope CSS without the crutch of classes and IDs.

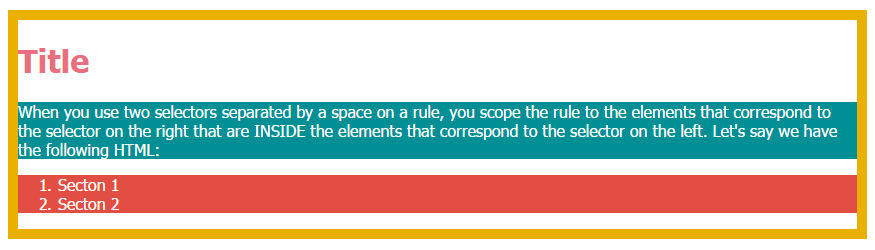


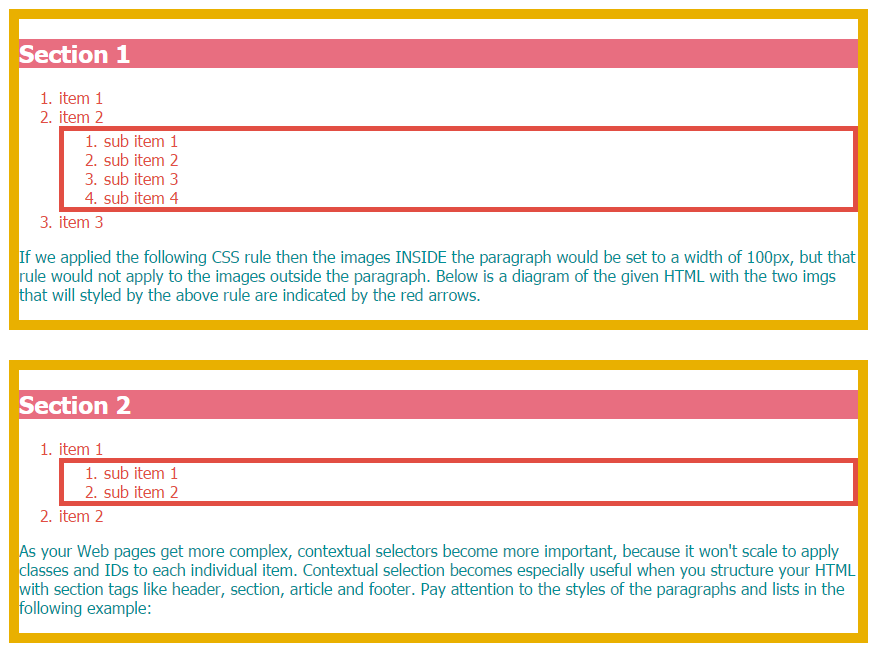
Look carefully at the CSS code:

1. body {
2. font-family: Tahoma, sans-serif;
3. }
4. header, article, footer {
5. border: 10px #E9B000 solid;
6. margin: 30px;
7. }
8. h1 {
9. color: #E86E80;
10. }
11. h1 {
12. background-color: #E86E80;
13. color: #FFFFFF;
14. }
15. p {
16. background-color: #008F95;
17. color: #FFFFFF;
18. }
19. p {
20. background-color: #FFFFFF;
21. color: #008F95;
22. }
24. ol {
25. background-color: #E24E43;
26. color: #FFFFFF;
27. }
28. ol {
29. background-color: #FFFFFF;
30. color: #E24E43;
31. }
32. ol {
33. border: 5px solid #E24E43;
34. }

So, there are CSS rules with duplicate selectors. You'll need to add some extra selectors to the existing CSS rules so that instead of duplicate rules, you have rules that address different instances of the HTML elements based on the structure of the HTML.

Your final site should look like this:







## 3.4.3 Cascading styles

Hopefully, at this point, it's become very clear

why what we're working on is called a style sheet.

But where is the cascading part coming? CSS has a very specific hierarchy of style.

So that you can write multiple rules that apply to the same HTML tag

but the browser will still know how to make it look.

But how does the browser decide when there's multiple rules?

The answer: the most specific rule wins.

But then, why would you want to have multiple rules for the same tag?

A lot of times, you're going to have multiple style sheets.

Imagine you have one parent style sheet that applies to your

whole Web site, but then you have individual smaller style sheets

that apply to the different pieces of it,

like say, your blog or your gallery.

Imagine, you want to highlight a specific piece of text.

You could either write individual rules for each piece of text

in your whole Web site —most of which

will be very similar and then one would be different— or, you can write one style

apply it to the body tag, so that every single piece of text

is affected and then use an ID to target the exceptions.

Because the ID tag is so much more specific than the body that rule wins.

As you can imagine this is going to save you a lot of time and a lot of typing.

And I'm sure you know computer scientists are all about efficiency.

So, now let's take a closer look at

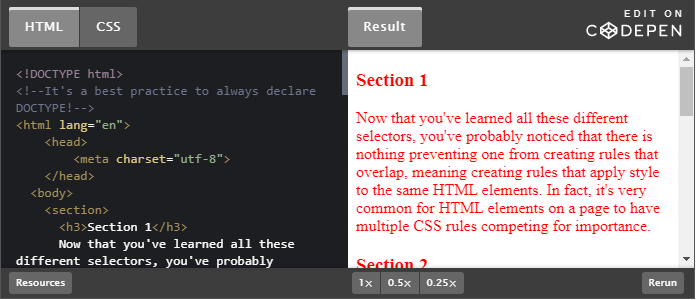
how you can use the cascading nature of CSS to optimize your style.

Now that you've learned all these different selectors, you've probably noticed that there is nothing preventing one from creating rules that overlap, meaning creating rules that apply style to the same HTML elements. In fact, it's very common for HTML elements on a page to have multiple CSS rules competing for importance.

Thanks to the "cascading" part of "Cascading Style Sheets", this isn't a problem. That is because CSS has a way to figure out which rule "wins" when styles are conflicting. CSS actually computes a "weight" for each style rule and the one with the greatest weight wins. If you want all the specifics on how this weight is computed you can read more [here](https://www.w3.org/TR/CSS22/cascade.html). For simplicity's sake, the most specific rule wins!

* A rule is generally more specific if it applies to a fewer number of elements than another rule that encompasses those elements. The more facets a rule has the more specific it is.
  + When pseudo-classes are applied this is more specific than without the pseudo-class. For example, p:hover will win over just p.
  + Contextual selection is more specific because it scopes the rule to elements within a certain subset of those on the page. For example, a rule that applies to all the paragraphs within articles is more specific than a rule that applies to all the paragraphs on the page.
  + IDs are most specific because you are directly applying them to the desired HTML element. The rule based on an ID will always win over other rules
* If two rules have the exact same weight, the one that comes later in the CSS document is what is applied.
* You can use the "!important" modifier on a CSS property so that it will guarantee that style will be applied. This is a way for you to override the calculated weight.

Look at the corresponding HTML and CSS, where code showing how Cascading order applies with many overlapping rules:



1. section {
2. color: red;
3. }
4. section section {
5. color: orange;
6. }
7. section section section {
8. color: green;
9. }
10. #section {
11. color: blue;
12. }
13. section:hover {
14. color: purple;
15. }
16. section:hover section:hover {
17. color: pink;
18. }
19. section:hover section:hover section:hover {
20. color: yellow;
21. }

Things to observe about the above:

* the rule based on an ID isn't even overwritten by the pseudo class
* when you hover over the sections within other sections, multiple hover rules apply!

## 3.4.4 Activity - Cascading order

Cascading order can be difficult to manage.

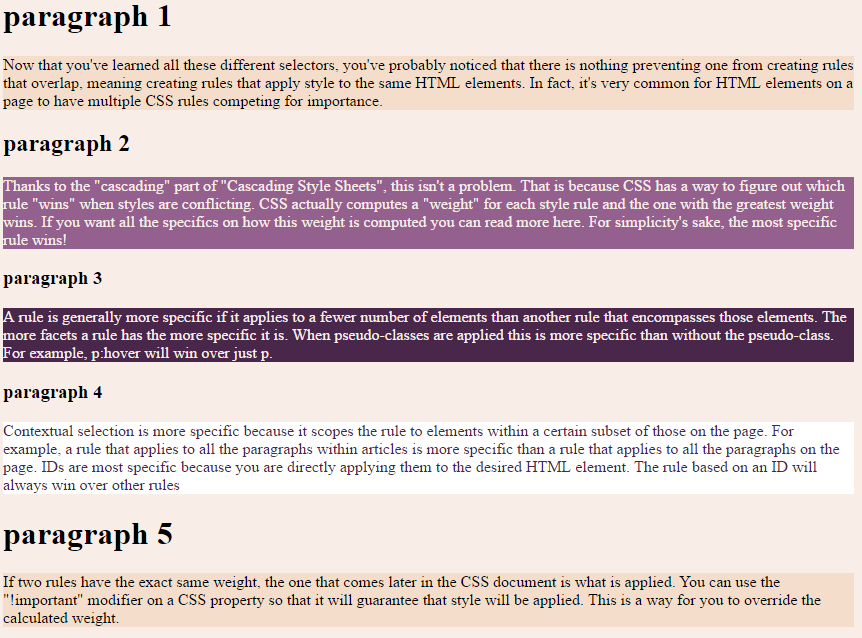
Consider the following pen:



... and specifically, its CSS code:

1. body {
2. background-color: #F8EEE7;
3. }
4. p {
5. background-color: #F4DECB;
6. }
7. p {
8. background-color: #94618E;
9. color: #F8EEE7;
10. }
11. p {
12. background-color: #49274A;
13. }
14. p {
15. background-color: #FFFFFF;
16. color: #49274A;
17. }
18. p {
19. background-color: #B4DBC0 !important;
20. border: 2px dashed black;
21. }

Note that there are a lot of repeat CSS rules. For this activity, you cannot change any of the HTML (meaning you cannot add any IDs), but you'll need to change the CSS to make some of the rules more specific so that you achieve this final result:



The image below shows when you hover "paragraph 4". The same hover should apply to ALL p tags.

body {

background-color: #F8EEE7;

}

p {

background-color: #F4DECB;

}

section p {

background-color: #94618E;

color: #F8EEE7;

}

article p {

background-color: #49274A;

}

#specific {

background-color: #FFFFFF;

color: #49274A;

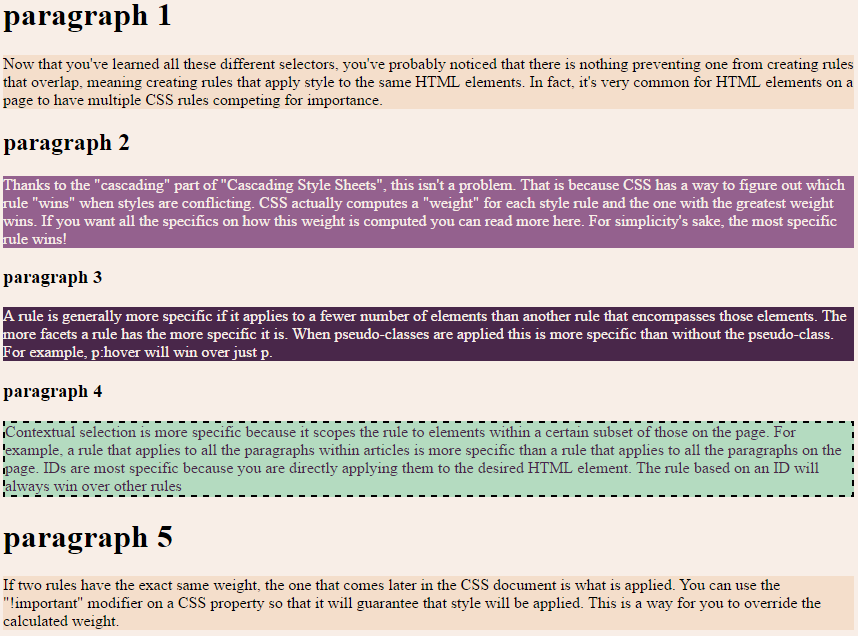
}

p:hover {

background-color: #B4DBC0 !important;

border: 2px dashed black;

}



## 3.5.1 Images

Images are an extremely important part of your page, not only as part of your content but as a key way to help style your page. Before you even get to styling them you need to put considerable thought into what type of images to include in your page. One of the easiest ways to slow down your site's performance is to include lots of large, high-quality images. There are two general things to consider when picking your images: what format and how large are they?

### Formats

* JPEG - Possibly the most common image format used on the web, because it does a good job of compressing colorful, complex images into reasonably sized files. JPEG is often used for photos because otherwise, photos can take a very long time to load.
* PNG - Provides a higher quality image, but you pay for it with a larger file size. PNG also has the ability to have transparent backgrounds, so you'll often see this used for graphics as part of the page's layout.
* GIF - Supports moving images and transparent backgrounds. These can be especially large files so try to use sparingly!

### Sizing

It's important to format and size your image before uploading it to your site, because while you can use CSS to resize it, the browser will download the full image even if you've chosen to display it smaller than it is.

You can resize images using the width or height properties of CSS like so:

1. img {
2. width: 100px;
3. height: 100px;
4. }

In Module 4, we will discuss width and height in detail, but for the purpose of images, know that if you only set one of these two properties the image will scale according to its original dimensions. If you set both width and height, CSS will stretch your image to fit whatever you specify.

### Spacing

Rarely does it look nice when your content images touch your text content. You'll often want to give your images a good amount of white space surrounding them, and you can do this with padding and/or margin. In the examples below, you can see how you can even use padding and margin to achieve some interesting stylistic effects with your images.

### Border Radius

In Module 2, we saw how you can use the "border-radius" property to make a button with rounded corners. You can do the same to images. You might even notice that it has become popular to turn square images into perfect circles, especially for icons or profile photos.

### Images as design elements

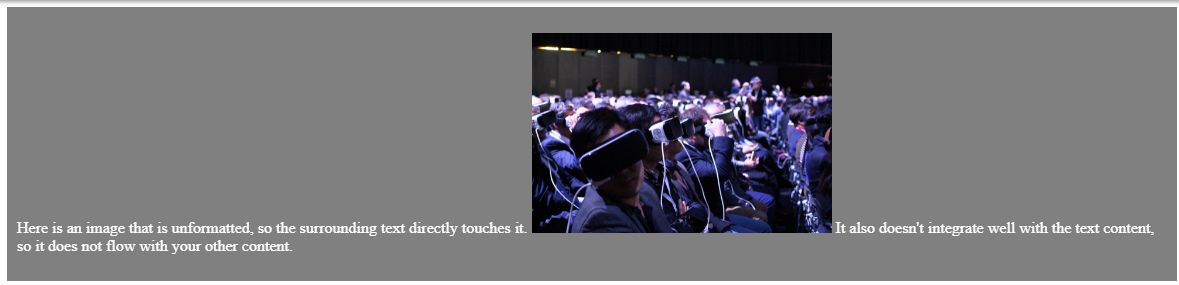
When you see complicated styles as part of a Web page's design, chances are there are images as a foundational element. For example, you will often see images set as backgrounds or as a link. Remember that you can put an <img> tag within an anchor tag to turn it into a link. You can also use the background-image CSS property to set an image as a background instead of just using a solid color. You can [read more about background images here](https://www.w3.org/wiki/CSS_background_images).

### Text in images

It is best to overlay real text rather than use pixelated text. This makes it possible to search, copy and style the text, as well as making translation much easier. It also has [accessibility benefits](https://www.w3.org/WAI/tutorials/images/textual/) and makes translation much easier.

### Examples

As you can see, there are a lot of things to consider, and if you do not plan your images carefully, they can end up looking very disruptive, like this "Bad Image" example:



##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<div id="badImg">

<p>

Here is an image that is unformatted, so the surrounding text directly touches it.

<img src="https://www.w3c.fr/wp-content/uploads/2016/12/26420235490\_7ae150d467\_b.jpg" width="300" alt="rows of people sitting in a theatre and wearing virtual reality devices" />

It also doesn't integrate well with the text content, so it does not flow with your other content.

</p>

</div>

</body>

</html>

##### CSS

div {

margin-bottom: 50px;

}

#badImg {

background-color: gray;

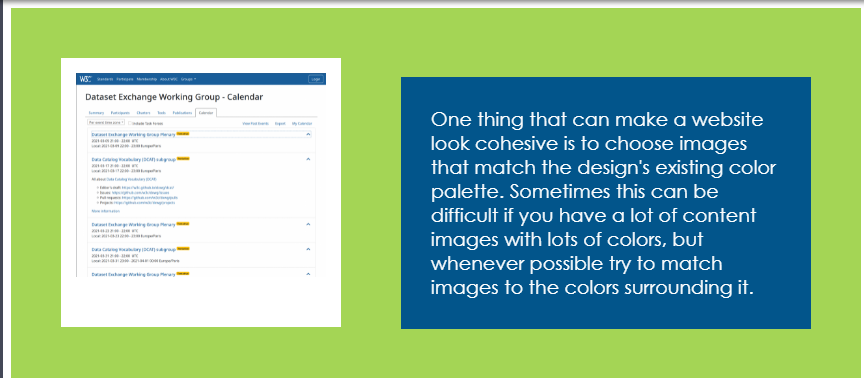
padding: 10px;

color: white;

}

### Images 1

This design shows some of the basics in properly incorporating images. It uses the colors from the image in the design of the overall page to help the image look as if it belongs. It also uses padding to give the image a "polaroid-style" border.



### CSS

div {

margin-bottom: 50px;

}

#images1 {

background-color: #A4D555;

padding: 50px;

width: 750px;

}

#images1 p {

background-color: #02558B;

font-family: "Century Gothic";

font-size: 1.2em;

color: white;

width: 350px;

padding: 30px;

float: right;

}

#images1 img {

width: 250px;

padding: 15px 15px 50px 15px;

background-color: white;

}

### HTML

!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<div id="images1">

<p>

One thing that can make a website look cohesive is to choose images that match the design's existing color palette. Sometimes this can be difficult if you have a lot of content images with lots of colors, but whenever possible try to match images to the colors surrounding it.

</p>

<img src="https://www.w3.org/blog/wp-content/uploads/2021/03/group\_calendar-768x627.png" alt="snapshot of the Dataset Exchange Working Group Calendar" />

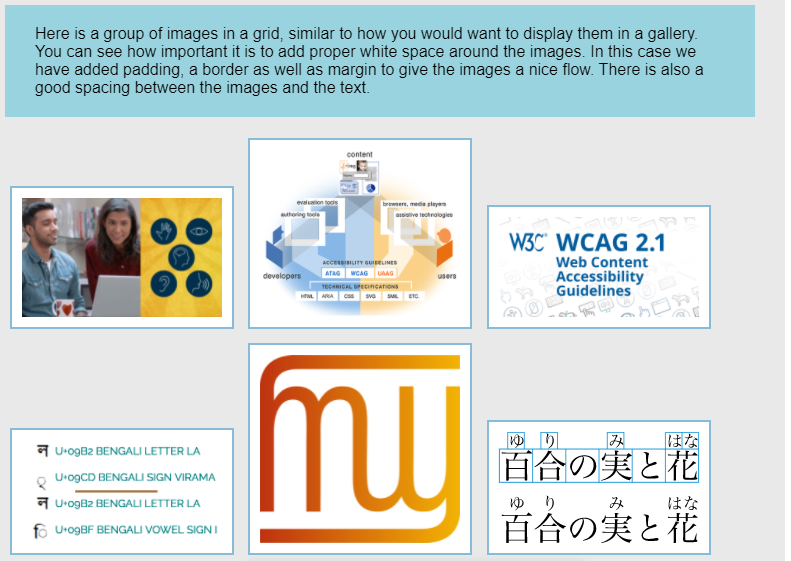
</div>

</body>

</html>

### Images 2

This design is an example of an image gallery design. This gives all the images the same size and alignment while including plenty of white space to help the page not look too overwhelming despite having multiple photos.



### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<div id="images2">

<p>

Here is a group of images in a grid, similar to how you would want to display them in a gallery. You can see how important it is to add proper white space around the images. In this case we have added padding, a border as well as margin to give the images a nice flow. There is also a good spacing between the images and the text.

</p>

<img src="https://www.w3.org/comm/assets/graphics/w3cx-accessibility-intro.jpg" alt="man and woman looking at a computer screen" />

<img src="https://www.w3.org/WAI/content-images/wai-std-gl-overview/specs.png" alt="illustration showing the WAI guidelines for the different components, detailed description at https://www.w3.org/WAI/intro/components-desc.html#guide" />

<img src="https://www.w3.org/WAI/images/WCAG21/small-1024x512@1x.png" alt="WCAG 2.1" />

<img src="https://www.w3.org/International/questions/qa-indic-graphemes-data/jhilli\_codepoints.png" alt="Code points that make up the Bengali word 'jhilli'." />

<img src="https://www.w3.org/International/logos/mlw-notext-500-white.png" alt="logo of the W3C Internationalization Activity" />

<img src="https://www.w3.org/TR/2020/WD-simple-ruby-20200609/img/fig06.svg" alt="Example of horizontal ruby" />

</div>

</body>

</html>

### CSS

div {

margin-bottom: 50px;

}

#images2 {

width: 750px;

background-color: #E9E9E9;

padding: 30px;

font-family: Arial;

}

#images2 p {

background-color: #99D3DF;

padding: 20px 30px;

}

#images2 img {

width: 200px;

padding: 10px;

border: 2px #88BBD6 solid;

margin: 5px;

background-color: white;

}

##### Images 3

This design demonstrates the use of an image both as a background and as a link. When using images as design elements, it is best to use simple images to let your content still be easily consumable.



##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<div id="images3">

<p> Here is some body text that is now on top of an image because it is set as the background. When using an image as a background try to pick an image that is not overly busy, this can make your text or other images difficult to read when placed on top of.

</p>

<div id="navBox">

<div id="clickHere"> <a href="https://www.w3.org">Click The Image</a> </div>

<a href="www.w3.org"><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/c/ce/Font\_Awesome\_5\_solid\_arrow-circle-right.svg/512px-Font\_Awesome\_5\_solid\_arrow-circle-right.svg.png" width="80" alt="A solid-weight icon from Font Awesome, a free web icon font" /></a>

</div>

</div>

</body>

</html>

##### CSS

div {

margin-bottom: 50px;

}

#images3 {

background-image: url('https://upload.wikimedia.org/wikipedia/commons/4/45/Big\_Cumulonimbus.JPG');

background-repeat: no-repeat;

height: 500px;

width: 500px;

}

#images3 p, #clickHere {

color: white;

font-family: Tahoma;

padding: 40px;

}

#navBox {

width: 220px;

margin-left: 30px;

}

#clickHere {

float: right;

padding: 30px 0px;

}

#clickHere a {

color: white;

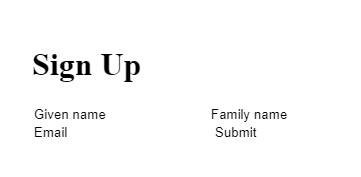
}

## 3.5.2 Forms

Forms can be a surprisingly tricky element to style because you are asking for a lot of interaction from your user, and the burden is on you as the designer to make it clear what exactly you are asking them to do. However, with a few simple design changes you can make it much easier for your user to navigate your form. Here are some key design elements to keep in mind when designing forms:

* **white space** - It is important to separate your form elements from your other content, otherwise it can easily be skipped over. It is also important to use white space surrounding each of your input elements to help draw your user's eye from the start to the end of your form. White space can also give the impression of simplicity and ease, an overcrowded form can seem daunting and tiring to fill out.
* **borders** - Possibly one of the most important design elements of your forms are borders. Borders give your user a visual clue that they should enter text, or select something from a drop down. When you eliminate all borders it is impossible to tell what is a form and what is just static text.
* **labels** - Each input should have a label communicating to the user what information you are asking them to submit. You will want to make sure this text is aligned and flows with the input element so it is clear what label is associated with which input. One of the easiest ways to make sure there is a label is to put the label inside the input text box as demonstrated in some of our sample designs.
* **input sizes** - You can help the user quickly assess how much text you are asking them to provide by how big your input element is. A very small input element will imply that you are only asking for a small number of characters, and a very large box with multiple lines can imply that you are asking a lot more from your user. It is also important to try and gauge the input size accordingly so your user doesn't run out of space as they are entering text.
* **pseudo-classes** - Perhaps one of the best uses of pseudo-classes is with forms. Pseudo-classes can help provide extra feedback as your user interacts with each of your form elements. You'll often see *hover* used to expose more information, *active* used to help a user understand which element they are currently interacting with, and more.
* **submit button** - Don't forget some of the lessons learned in Module 2: it is important to make your button look like a button. You will want to make sure that your submit button is placed so that it is obvious after a user has entered in all the requested info.

Here is an HTML/CSS form:



HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<div id="uglyForm">

<form>

<h1>Sign Up</h1>

<input type="text" value="Given name" />

<input type="text" value="Family name" />

<input type="text" value="Email" />

<button>Submit</button>

</form>

</div>

</body>

</html>

##### CSS

div {

margin-bottom: 50px;

height: 400px;

}

form {

padding: 30px;

width: 500px;

}

/\* Bad Design \*/

#uglyForm input {

border: 0px;

}

#uglyForm button {

background-color: white;

border: 0px;

}

You might never know it, but each of the 4 elements below the title are intended for the user to interact with. By stripping them of their styles you can see how important it is to add visual cues for your user.

### Form 1

The first form design uses basic layout to give each input element a clear label, space and area for input followed by a clear submit button. This design also used pseudo classes to engage the user, which you can see in action in the Code Pen below:



##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<div id="form1">

<form>

<h1>Sign Up</h1>

Given name

<br />

<input type="text" />

<br />

Family name

<br />

<input type="text" />

<br />

Email

<br />

<input type="text" />

<br />

<button>Submit</button>

</form>

</div>

</body>

</html>

##### CSS

div {

margin-bottom: 50px;

height: 400px;

}

form {

padding: 30px;

width: 500px;

}

/\* DESIGN 1 \*/

#form1 {

background-color: #5E0231;

font-family: Arial;

color: #DBC3D0;

}

#form1 h1 {

color: white;

margin: 0px 0px 20px 0px;

}

#form1 input {

background-color: #DBC3D0;

border: 2px white solid;

padding: 10px;

margin-bottom: 15px;

}

#form1 input:hover, #form1 input:focus {

border: 2px #DBC3D0 solid;

background-color: white;

}

#form1 button {

background-color: white;

border: 3px #DBC3D0 solid;

border-radius: 5px;

color: #5E0231;

padding: 10px 30px;

font-size: 1.2em;

}

#form1 button:hover {

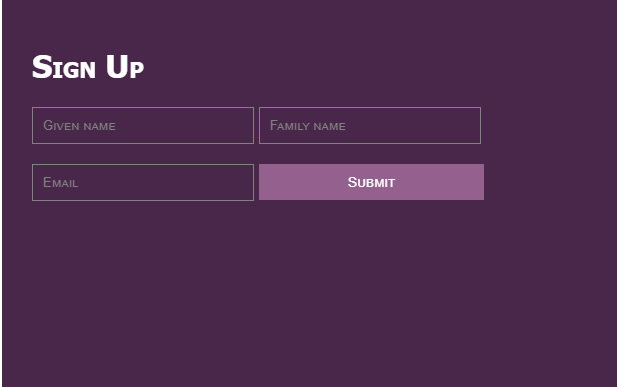
background-color: #DBC3D0;

color: white;

}

Form 2

This form design is more minimalist, using the clear and consistent design and layout to help the user feel like the form is extra short.



HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<div id="form2">

<form>

<h1>Sign Up</h1>

<input type="text" value="Given name" />

<input type="text" value="Family name" />

<br />

<input type="text" value="Email" id="form2email" />

<button>Submit</button>

</form>

</div>

</body>

</html>

CSS

div {

margin-bottom: 50px;

height: 400px;

}

form {

padding: 30px;

width: 500px;

}

/\* DESIGN 2 \*/

#form2 {

background-color: #49274A;

font-family: Tahoma;

font-variant: small-caps;

}

#form2 h1 {

color: white;

}

#form2 input {

background-color: #49274A;

border: 1px #828081 solid;

padding: 10px;

color: #828081;

font-variant: small-caps;

margin-bottom: 20px;

width: 200px;

}

#form2 input:hover, #form1 input:focus {

border: 1px #94618E solid;

color: #94618E;

}

#form2 button {

background-color: #94618E;

border: 0px;

color: white;

font-size: 0.9em;

padding: 10px 20px;

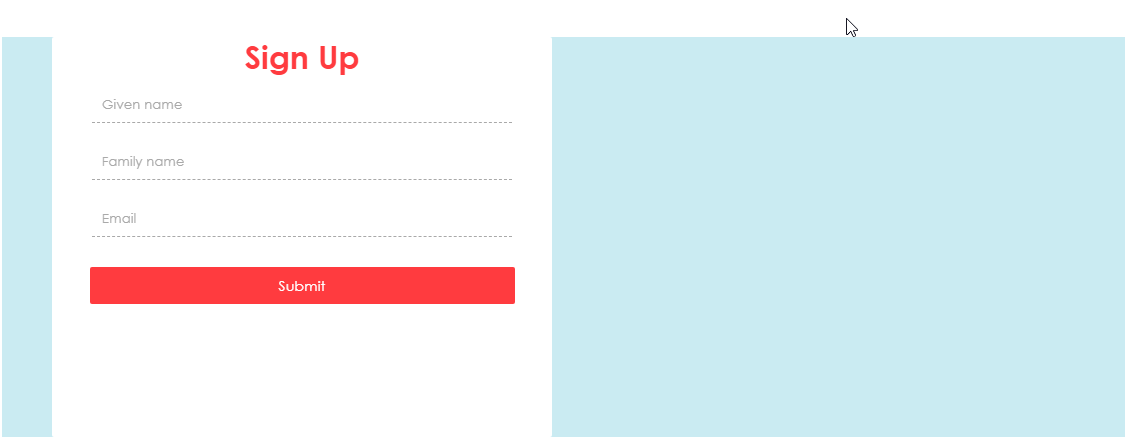
font-variant: small-caps;

width: 225px;

}

### Form 3

This form design leaves lots of white space to help it feel clean and simple.



##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<div id="form3">

<div id="box">

<h1>Sign Up</h1>

<input type="text" value="Given name" />

<input type="text" value="Family name" />

<input type="text" value="Email" />

<button>Submit</button>

</div>

</div>

</body>

</html>

##### CSS

div {

margin-bottom: 50px;

height: 400px;

}

form {

padding: 30px;

width: 500px;

}

/\* DESIGN 3 \*/

#form3 {

background-color: #CAEBF2;

font-family: 'Century Gothic';

}

#box{

background-color: white;

border-radius: 2px;

margin: 50px;

text-align: center;

width: 500px;

}

#form3 h1 {

color: #FF3B3F;

text-align: center;

margin: 0px;

}

#form3 input {

border: 0px;

border-bottom: 1px #A9A9A9 dashed;

padding: 10px;

color: #A9A9A9;

width: 80%;

margin: 10px;

font-family: 'Century Gothic';

}

#form3 input:focus, #form3 input:hover {

border: 1px #FF3B3F solid;

}

#form3 button {

background-color: #FF3B3F;

border: 0px;

border-radius: 2px;

color: white;

font-size: 0.9em;

padding: 10px 20px;

margin: 20px;

width: 85%;

font-family: 'Century Gothic';

}

The pen below is a recap of all forms presented in this page:

##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<div id="uglyForm">

<form>

<h1>Sign Up</h1>

<input type="text" value="Given name" />

<input type="text" value="Family name" />

<input type="text" value="Email" />

<button>Submit</button>

</form>

</div>

<div id="form1">

<form>

<h1>Sign Up</h1>

Given name

<br />

<input type="text" />

<br />

Family name

<br />

<input type="text" />

<br />

Email

<br />

<input type="text" />

<br />

<button>Submit</button>

</form>

</div>

<div id="form2">

<form>

<h1>Sign Up</h1>

<input type="text" value="Given name" />

<input type="text" value="Family name" />

<br />

<input type="text" value="Email" id="form2email" />

<button>Submit</button>

</form>

</div>

<div id="form3">

<div id="box">

<h1>Sign Up</h1>

<input type="text" value="Given name" />

<input type="text" value="Family name" />

<input type="text" value="Email" />

<button>Submit</button>

</div>

</div>

</body>

</html>

##### CSS

div {

margin-bottom: 50px;

height: 400px;

}

form {

padding: 30px;

width: 500px;

}

/\* Bad Design \*/

#uglyForm input {

border: 0px;

}

#uglyForm button {

background-color: white;

border: 0px;

}

/\* DESIGN 1 \*/

#form1 {

background-color: #5E0231;

font-family: Arial;

color: #DBC3D0;

}

#form1 h1 {

color: white;

margin: 0px 0px 20px 0px;

}

#form1 input {

background-color: #DBC3D0;

border: 2px white solid;

padding: 10px;

margin-bottom: 15px;

}

#form1 input:hover, #form1 input:focus {

border: 2px #DBC3D0 solid;

background-color: white;

}

#form1 button {

background-color: white;

border: 3px #DBC3D0 solid;

border-radius: 5px;

color: #5E0231;

padding: 10px 30px;

font-size: 1.2em;

}

#form1 button:hover {

background-color: #DBC3D0;

color: white;

}

/\* DESIGN 2 \*/

#form2 {

background-color: #49274A;

font-family: Tahoma;

font-variant: small-caps;

}

#form2 h1 {

color: white;

}

#form2 input {

background-color: #49274A;

border: 1px #828081 solid;

padding: 10px;

color: #828081;

font-variant: small-caps;

margin-bottom: 20px;

width: 200px;

}

#form2 input:hover, #form1 input:focus {

border: 1px #94618E solid;

color: #94618E;

}

#form2 button {

background-color: #94618E;

border: 0px;

color: white;

font-size: 0.9em;

padding: 10px 20px;

font-variant: small-caps;

width: 225px;

}

/\* DESIGN 3 \*/

#form3 {

background-color: #CAEBF2;

font-family: 'Century Gothic';

}

#box{

background-color: white;

border-radius: 2px;

margin: 50px;

text-align: center;

width: 500px;

}

#form3 h1 {

color: #FF3B3F;

text-align: center;

margin: 0px;

}

#form3 input {

border: 0px;

border-bottom: 1px #A9A9A9 dashed;

padding: 10px;

color: #A9A9A9;

width: 80%;

margin: 10px;

font-family: 'Century Gothic';

}

#form3 input:focus, #form3 input:hover {

border: 1px #FF3B3F solid;

}

#form3 button {

background-color: #FF3B3F;

border: 0px;

border-radius: 2px;

color: white;

font-size: 0.9em;

padding: 10px 20px;

margin: 20px;

width: 85%;

font-family: 'Century Gothic';

}

## Module 3 project - My profile

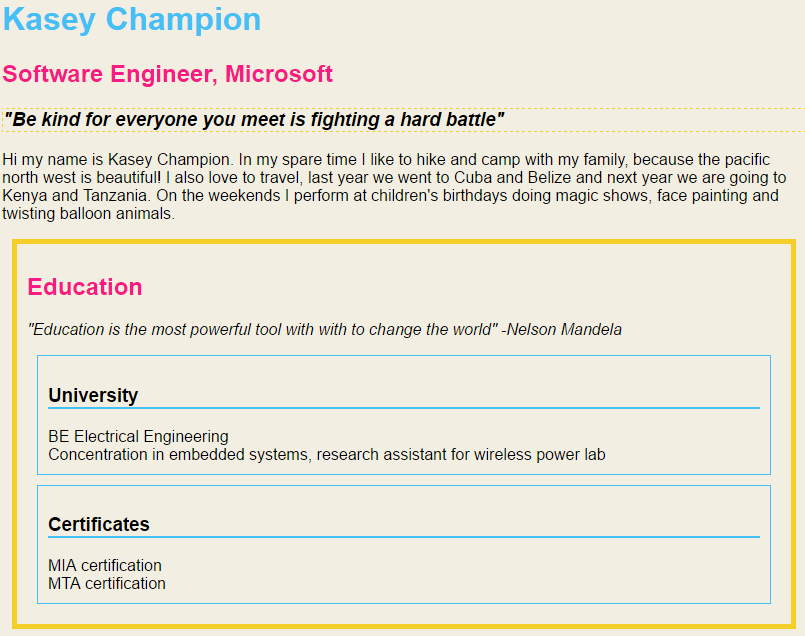
Now that you have a host of new selectors available, you can style much more complicated HTML. Take your "about me page" from Module 2, and add to it so that it becomes more of a "profile". Your page must have the following elements:

* header and footer tags
* an education, work experience and favorite things sections
* at least two subsections within each main section. You could use either the <section> tag, the <article> tag or the <div> tag for this.
  + Each subsection should have its own title and two inner subsections

Once you have all these elements, style them to look nice, but you must have each of the following:

* 1 ID
* 2 different classes, each with at least 2 members
* 1 special style applied when an element is hovered over by the user's cursor
* 1 special style applied when the user clicks an element
* 3 uses of contextual selectors

For reference here is what my solution looks like:



## Module 3 learnings

### After this module you should know how to...

* Specifically style any single element on a page using IDs
* Group HTML elements with similar style using classes
* Employ pseudo classes to make HTML elements more dynamic
* Scope CSS rules using contextual selection
* Build CSS rule sets with intention leveraging the cascading nature of CSS to manage conflicting rules

### In the next module,  you will:

* Learn about layout and positioning with CSS
* Understand the box model
* Explore relative positioning
* Learn about the flow property

## 4.1.1 Welcome to Module 4

In this module, we're going to talk about layout and positioning using CSS.

We're going to start by understanding the ‘Box model’

or the white space around each and every HTML element.

Then we're going to learn about ‘Relative positioning’.

That's how you position your elements on the

page in relation to one another, so that

no matter what screen you're looking at,

it all looks nice. And then finally, we're

going to learn about the ‘Float property’,

a way for you to pick up an HTML element and move it around on the page,

so you get more of a balance between the left and the right.

We're going to combine all these things together.

so you can make a balanced well-designed Web page.

## 4.1.2 Module 4 outline

### In this module, you will learn:

* How to use padding and margin to position elements relative to each other and the window
* How to use alignment to control how your content sits within its HTML element
* How to use the float property to let multiple HTML elements share horizontal space
* How to use relative positioning to design your page so that it maintains its layout regardless of screen size

## 4.1.3 Meet the Box Model

The Box Model is a way for us to distinguish the areas surrounding an

HTML element and directly control each of their white space and HTML element

is to be considered, to be contained within a box with four sides.

The areas around this box are separated into three types

The Padding, The Border and The Margin.

The Padding sits between the HTML element and The Border.

The Border sits between the Padding and Margin.

You can control The Borders color weight and style.

The Margin sits between The Border and the rest of the page.

But be careful

when two Margins meet the bigger one wins.

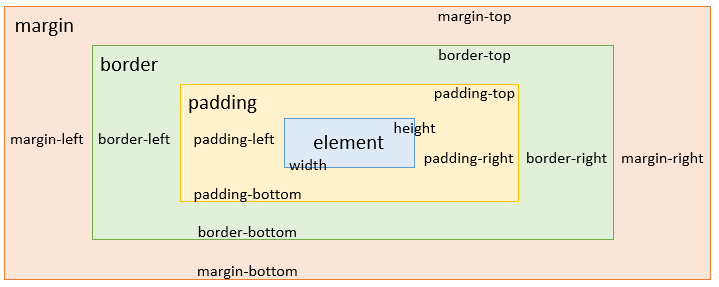
## 4.1.4 The Box Model

### The Box Model

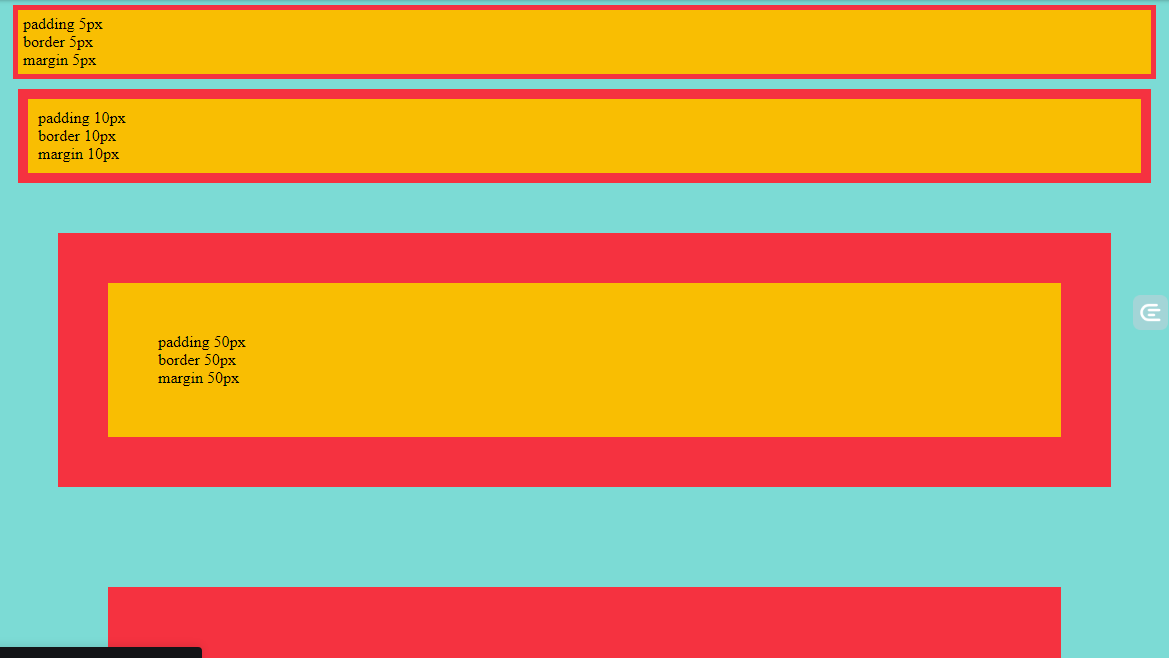
The Box Model is how Web browsers see individual HTML elements. When laying out a document, the browser's rendering engine represents each element as a rectangular box according to the standard **CSS basic box model**. CSS determines the size, position, and properties (color, background, border size, etc.) of these boxes.

Each element is comprised of 4 areas: the **element**, the **padding**, the **border** and the **margin**.

We discussed how to adjust the white space of these areas in Module 2.5, but in this module we will be discussing these areas as a method to position elements on a page.



* **element** - This is always contained within a rectangle, even if it is a text block with jagged edges or a transparent image that isn't rectangular. Web browsers will impose a rectangle around the smallest area the HTML element's content actually occupies. Until now we've allowed the Web browser to determine the size of the element based solely on the content, but later in this section we'll learn how to adjust this sizing.
* **padding** - This is the white space just outside the element's content. You can set each of the four sides independently, and you can set the value to 0. If you set the element's background color, that color will apply to the padding as well.
* **border** - This is the area just outside the padding. Most HTML element's border default width is 0 and thus invisible. You can set each of the four sides independently. You can set a color, a pattern, even an image. This is a great way to add horizontal or vertical lines to an element on your page.
* **margin** - This is the space surrounding an element, outside the border. Margins are the part of HTML elements that interact with one another. When two margins interact the larger of the two wins meaning the smaller margin "collapses", thus the actual space between two elements is the larger of the two, not the sum of the margins.





HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<p id="five">

padding 5px <br />

border 5px <br />

margin 5px <br />

</p>

<p id="ten">

padding 10px <br />

border 10px <br />

margin 10px <br />

</p>

<p id="fifty">

padding 50px <br />

border 50px <br />

margin 50px <br />

</p>

<p id="oneHundred">

padding 100px <br />

border 100px <br />

margin 100px <br />

</p>

</body>

</html>

CSS

body {

background-color: #7CDBD5;

}

p {

background-color: #F9BE02;

}

#five {

padding: 5px;

border: 5px #F53240 solid;

margin: 5px;

}

#ten {

padding: 10px;

border: 10px #F53240 solid;

margin: 10px;

}

#fifty {

padding: 50px;

border: 50px #F53240 solid;

margin: 50px;

}

#oneHundred {

padding: 100px;

border: 100px #F53240 solid;

margin: 100px;

}

### External resource:

* [Box model definition](https://www.w3.org/TR/CSS22/box.html) in the W3C's CSS2.2 specification

## 4.2.1 The alignment property

One of the simplest ways to align content is to use the direct text-align property. This can help you set the alignment of text or inline content within the context of their containing HTML element.

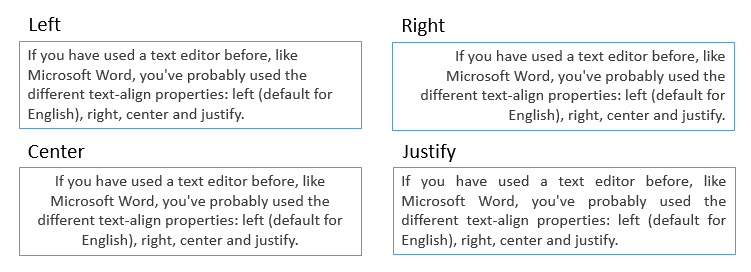
### text-align

[[*Documentation*](https://www.w3.org/TR/CSS22/text.html#alignment-prop)]

1. h1 {
2. text-align: center;
3. }

If you have used a text editor before, like Microsoft Word, you've probably used the different text-align properties: left (default for English), right, center and justify. Text-align in CSS works the same way. Left, center and right specify how the lines of text within the text block are arranged. Justify sets the left and right edges of the text flush with the container's edges, which stretches the white space between words so that the overall block fits in a perfect rectangle.

See below for examples of what each of these values will do to your text:



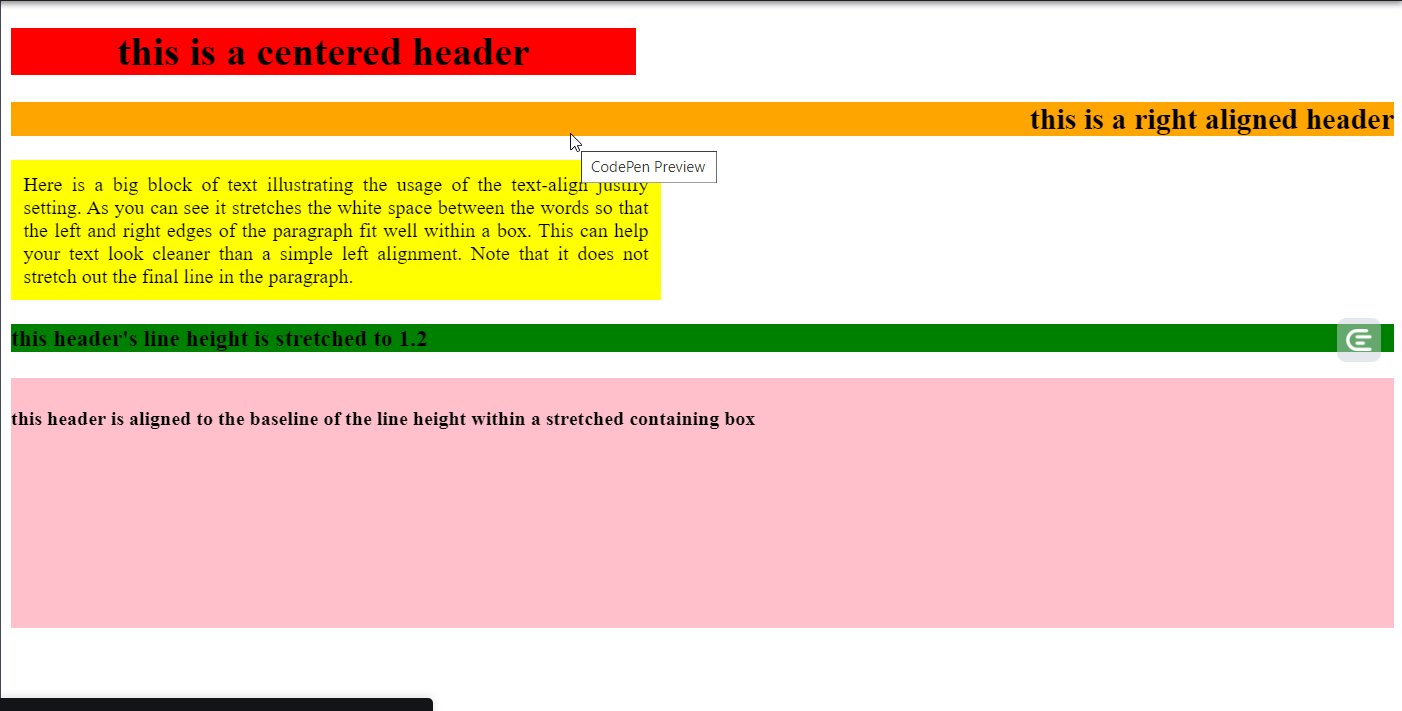
Note that this property can only apply to block elements like paragraphs, divs and headers.

### line-height

[[*Documentation*](https://www.w3.org/TR/CSS22/visudet.html#line-height)]

1. h1 {
2. line-height: 1.2;
3. }

You may have noticed that the text-align property sets the content's alignment horizontally, but it leaves its vertical alignment unchanged. Text lives within a specified vertical space, in which the text is drawn by default in the middle of that vertical space. If you change the height of the containing HTML block, the text will remain at the top of the block. However, if you instead use the "line-height" property, then the block will grow and the text will vertically center within it.



HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Aligning text </title>

<meta charset="utf-8">

</head>

<body>

<h1>this is a centered header</h1>

<h2>this is a right aligned header</h2>

<p>

Here is a big block of text illustrating the usage of the text-align justify setting. As you can see it stretches the white space between the words so that the left and right edges of the paragraph fit well within a box. This can help your text look cleaner than a simple left alignment. Note that it does not stretch out the final line in the paragraph.

</p>

<h3>this header's line height is stretched to 1.2</h3>

<h4>this header is aligned to the baseline of the line height within a stretched containing box</h4>

</body>

</html>

CSS

h1 {

background-color: red;

width: 500px;

text-align: center;

}

h2 {

background-color: orange;

text-align: right;

}

p {

background-color: yellow;

width: 500px;

padding: 10px;

text-align: justify;

}

h3 {

background-color: green;

line-height: 1.2;

}

h4 {

background-color: pink;

height: 200px;

line-height:4;

}

### International considerations

Please do not use text-align indiscriminately. If there's a possibility that your site will need to be translated into a language that uses the Arabic, Hebrew, or Thaana scripts (which read from right to left), it creates difficulties to have to change all the  right values to left and vice versa.

Most, but not yet all, major browsers support the values start and end. The start value aligns text with the side of the line where you start reading, whether that's on the left for English or the right for Urdu. They also make more sense for use with vertical text, such as for Japanese and Mongolian. Once these values are widely supported by browsers, they will often be a better choice than right and left, since there's no need to change the values for pages as the language changes.

Also, note that CSS will in the future provide better support for justification in languages where words are not separated by spaces, such as Chinese and Thai, or languages where words are separated by special marks, such as in Amharic. Check more information about [different approaches to justification](https://www.w3.org/International/articles/typography/justification).

Once you finish this course, look out for these and other international features of CSS as you explore its features further.

## 4.2.2 Element width and height

Until now we've let the browser decide how big the element is, but you can actually adjust its width and height manually.

### width and height

[Documentation: [the width property](https://www.w3.org/TR/CSS22/visudet.html#the-width-property) and [the height property](https://www.w3.org/TR/CSS22/visudet.html#the-height-property)]

1. p {
2. width: 30%;
3. }

You can use pixel values for both width and height, but you'll most often want to use percentages to set these so that your elements grow and shrink as appropriate based on the screen size.

For example, if we set the width of a paragraph to 30% as you resize the browser window, you'll see how that element dynamically resizes. That's because when you use percentages, the size is computed based on the element's "[containing block](https://www.w3.org/TR/CSS22/visuren.html#containing-block)", or the element that contains the one you're styling. If your element is just within the body tag, the width is computed based on the relationship with the screen width.

Things are a bit more complicated with using a percentage to set an element's height. This is because typically the body's height is not specified, so if you use a percentage the size won't adjust.

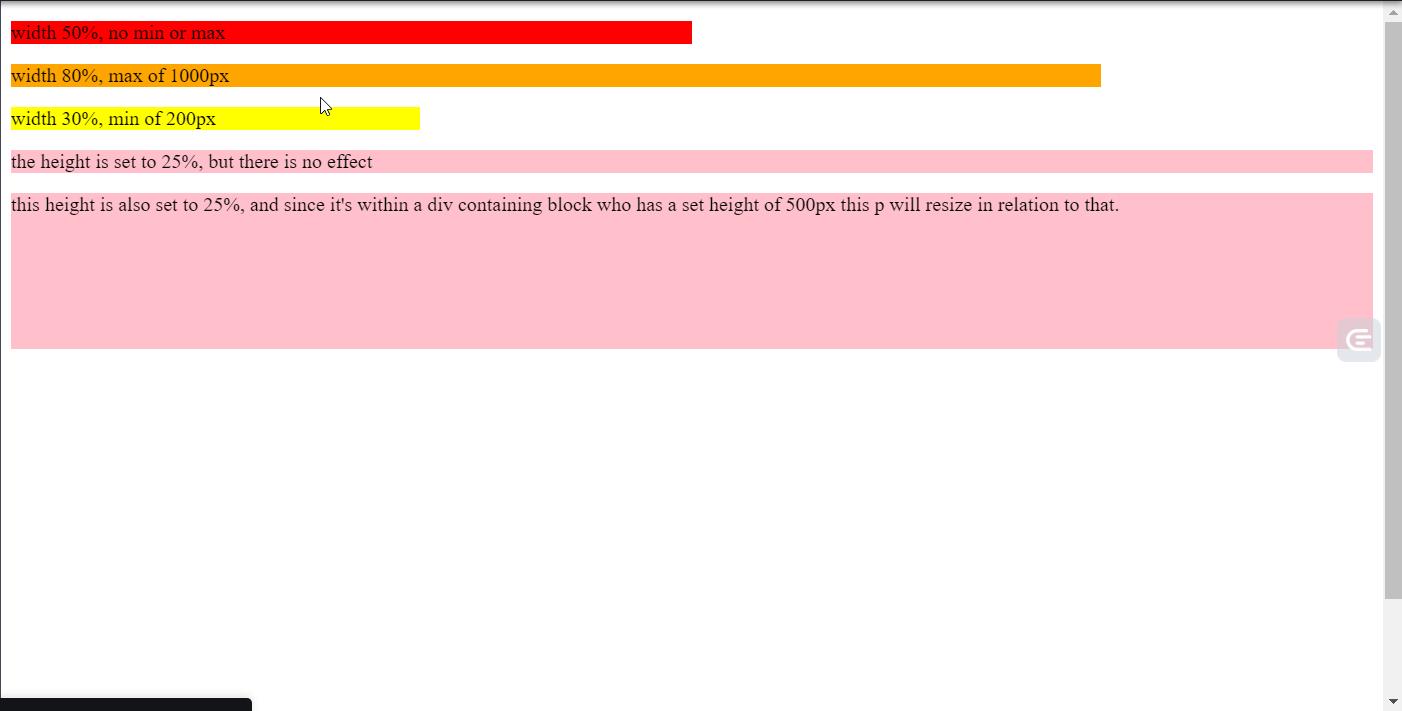
### min-width, max-width, min-height, max-height

[Documentation: [max and min width](https://www.w3.org/TR/CSS22/visudet.html#min-max-widths) and [max and min height](https://www.w3.org/TR/CSS22/visudet.html#min-max-heights)]

Setting width and height with percentages will save you work because your design will automatically optimize for the user's screen size. However, some elements can't grow and shrink as dynamically as text can.

For example, images will get "pixelated" if you let them grow too large, and they can look really distorted. Thankfully, you can set max and min width and heights. This way, you can set a range for your image to grow and shrink where you know it will still look good.

1. img {
2. width: 100%;
3. max-width: 1024px;
4. }



### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Dynamic width and height</title>

<meta charset="utf-8">

</head>

<body>

<p id="width50">

width 50%, no min or max

</p>

<p id="width80max">

width 80%, max of 1000px

</p>

<p id="width30min">

width 30%, min of 200px

</p>

<p class="heightSet">

the height is set to 25%, but there is no effect

</p>

<div id="containingBlock">

<p class="heightSet">

this height is also set to 25%, and since it's within a div containing block who has a set height of 500px this p will resize in relation to that.

</p>

</div>

</body>

</html>

CSS

#width50 {

background-color:red;

width: 50%;

}

#width80max {

background-color: orange;

width: 80%;

max-width: 1000px;

}

#width30min {

background-color: yellow;

width: 30%;

min-width: 200px;

}

.heightSet {

background-color: pink;

height: 25%;

}

#containingBlock {

height: 500px;

}

When you view the above example, the paragraphs will dynamically resize based on the size of your window. Please view the code first in a wide window, and then in a much narrower window.

You will note that the elements have resized accordingly, but have hit the limits of their min and max constraints. This is why using percentages for width and height are so important, it helps you write code that works for all screen sizes.

## 4.2.3 Padding and margin

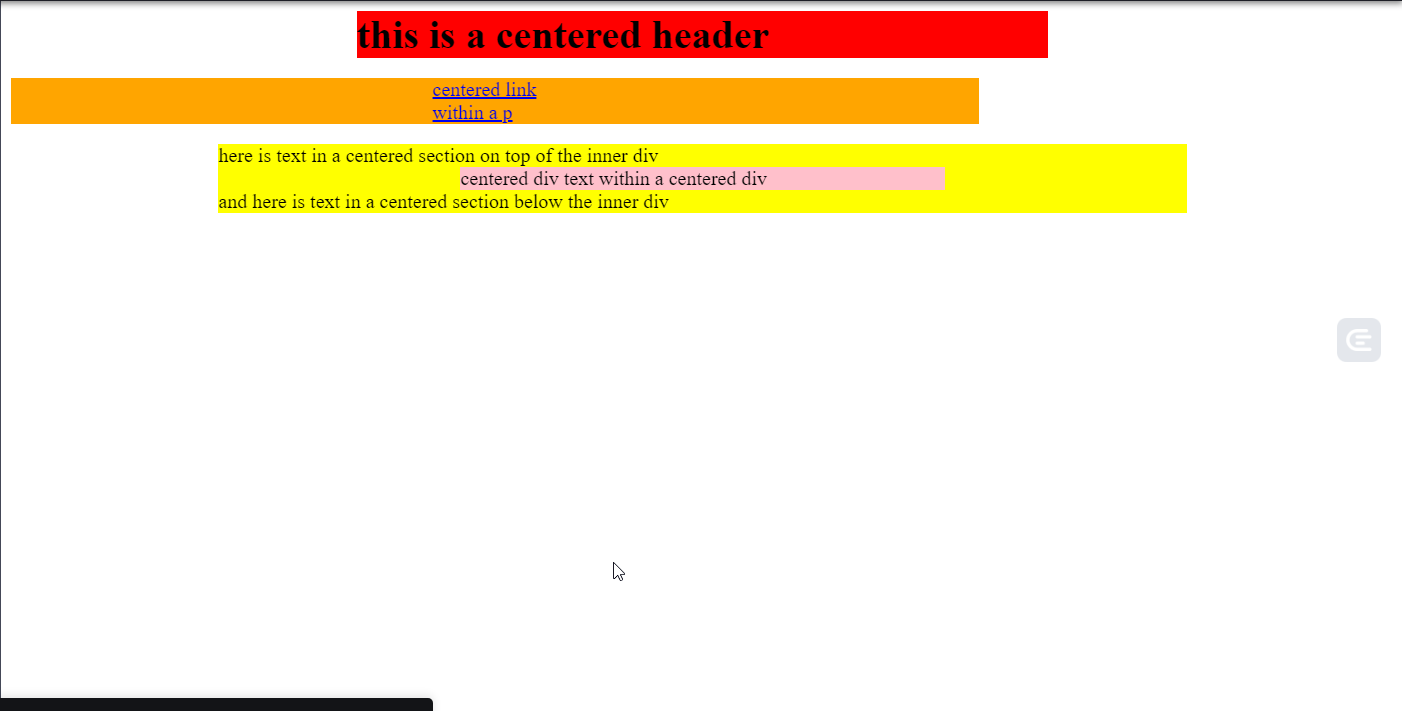
Whenever possible, it is ideal to position your elements by adjusting their padding and margins. Sometimes this isn't enough to get the element exactly where you'd like it to be, so to achieve this we'll learn more tools later in this module. Regardless, you'll almost always want some padding and margin around your element so it's best to adjust these before progressing onto more complicated positioning methods.

Once you have set the width for your element, you can then set margins as a way to position your element relative to others. One of the most commonly used margin settings is "auto". That is because if you set an element's left and right margin to auto it will be dynamically centered within its containing block.

1. div {
2. width: 50%;
3. margin-left: auto;
4. margin-right: auto;
5. }

Note however, that this only works for block HTML elements like paragraphs, divs and headers. If you want to use this to position an inline element, such as img or a, you will need to tell CSS to treat them as block elements by setting display: block;

1. img {
2. display: block;
3. width: 200px;
4. margin-left: auto;
5. margin-right: auto;
6. }



### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<meta charset="utf-8">

</head>

<body>

<h1>this is a centered header</h1>

<p>

<a href="www.microsoft.com">centered link within a p</a>

</p>

<section>

here is text in a centered section on top of the inner div

<div>

centered div text within a centered div

</div>

and here is text in a centered section below the inner div

</section>

</body>

</html>

### CSS

h1 {

background-color: red;

width: 50%;

margin: 0 auto;

}

p {

background-color: orange;

width: 70%;

}

a {

display: block;

width: 100px;

margin: 0 auto;

}

section {

background-color: yellow;

width: 70%;

margin: 0 auto;

}

div {

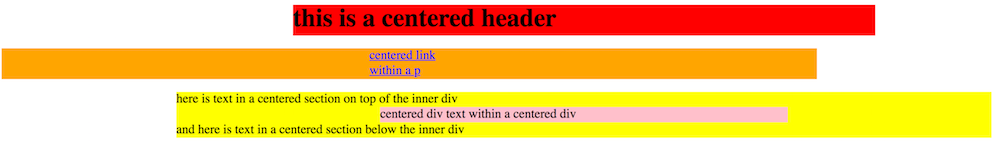
background-color: pink;

width: 50%;

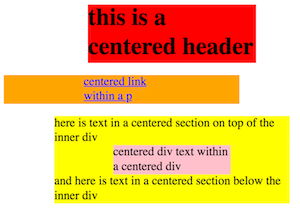
margin: 0 auto;

}

Here is what the above code looks like in a wide window:



Now, if you resize the window, the elements remain centered no matter what. Here is the above code in a narrow window:



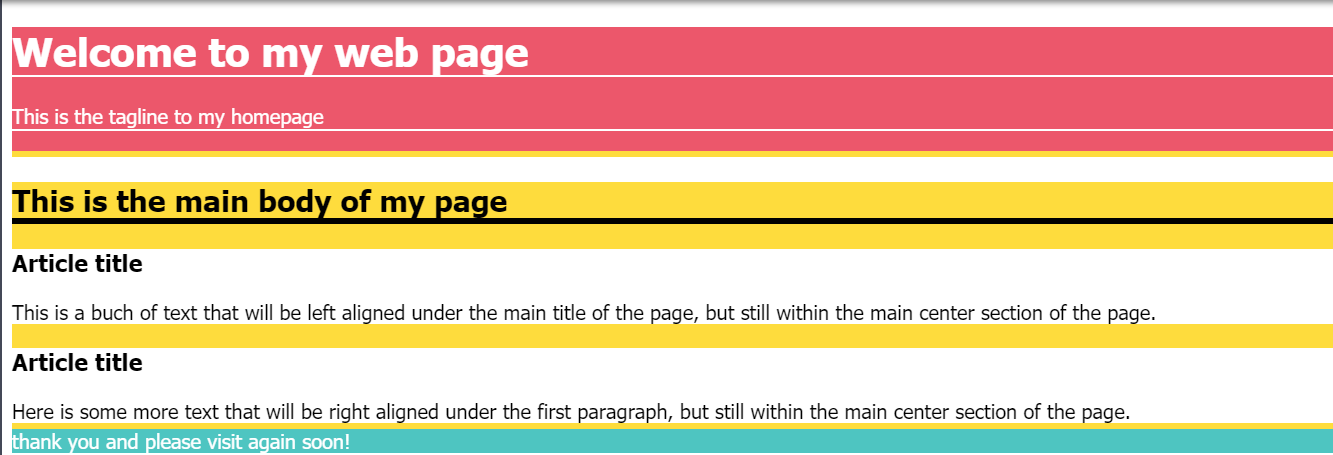
### External resources:

* A W3C CSS tip: [CSS centering things](https://www.w3.org/Style/Examples/007/center), on different ways to center your content.
* A "CSS Tricks" article on [What you should know about collapsing margins](https://css-tricks.com/what-you-should-know-about-collapsing-margins/)

## 4.2.4 Activity - Practice with alignment

For this activity, you are going to practice some of the basic alignment properties you've learned in this unit.

**Here is some HTML and CSS:**



### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Practice with Alignment</title>

<meta charset="utf-8">

</head>

<body>

<header>

<h1>Welcome to my web page</h1>

<p>

This is the tagline to my homepage

</p>

</header>

<section>

<h2>This is the main body of my page</h2>

<article id="leftP">

<h3>Article title</h3>

This is a buch of text that will be left aligned under the main title of the page, but still within the main center section of the page.

</article>

<article id="rightP">

<h3>Article title</h3>

Here is some more text that will be right aligned under the first paragraph, but still within the main center section of the page.

</article>

</section>

<footer>

thank you and please visit again soon!

</footer>

</body>

</html>

##### **CSS**

body {

font-family: Tahoma;

}

header {

background-color: #EC576B;

color: white;

border-bottom: 5px #FEDC3D solid;

}

header h1 {

border-bottom: 2px #FFFFFF solid;

}

header p {

border-bottom: 2px #FFFFFF solid;

}

section {

background-color: #FEDC3D;

}

h2 {

border-bottom: 5px #000000 solid;

}

article {

background-color: #FFFFFF;

}

#leftP {

}

#rightP {

}

h3 {

}

footer {

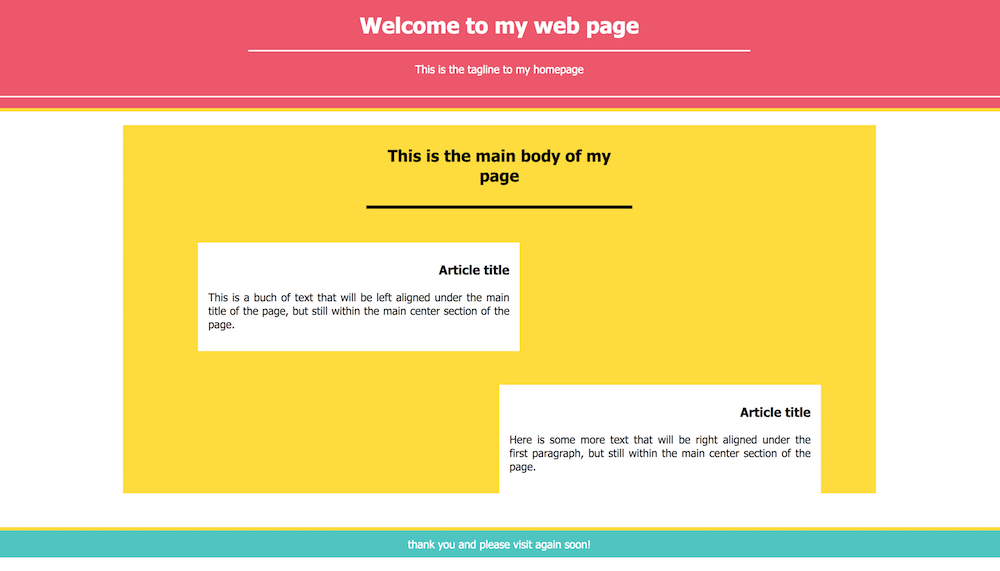
background-color: #4EC5C1;

border-top: 5px #FEDC3D solid;

color: white;

}

The resulted HTML and CSS codes produce a Web page where elements are not very well aligned. Your goal is to add properties to the existing CSS rules so that the final page looks like this:



To do this you will need to:

* remove the margin at the very edges of the page so the header and footer touch the edges of the page.
* set the size of the h1 title to 50% of the page's width, with horizontally centered text and vertically centered text within a height of 75px.
* stretch the height of the header p to 50px, but let the text remain top aligned. The header should be centered.
* set the width of the section to 75% of the page, but never less than 500px. The section should be centered within the window.
* set the width of the h2 to 30% of the section, centered within that container. It should have a padding of 30px all around it.
* set the width of each article to 40% of the width of the section, with justified text. Each should have a margin of 50px all around it, a top padding of 10px, a left and right padding of 15px and a bottom padding of 30px. The h3 within each article should be aligned to the right.
* give the #leftP article a left margin of 10% of the section.
* give the #rightP article a left margin of 50% of the section.
* the footer should have a padding of 10px and a margin between it and the section of 30px. The footer should be centered.

## 4.3.1 The float property

After when you look at other Web pages, you'll see multiple HTML elements

sitting side-by-side. When you look at our Web pages, there's only one element per horizontal space. That's because by default, CSS will place your HTML elements one after another on a vertical line.

But how do you save all your space and prevent your user from scrolling

endlessly? The ‘Float property’.

The float property liberates your elements so they can share horizontal space and you can make the best use of the whole width of your window. Now you too can condense

your content and save your user from endless scrolling.

If your elements are still not exactly where you want them to be after adjusting the padding, margins and alignment, then you can try out the float property. The "float" property is one of the most powerful tools you can master when learning CSS.

### The float property

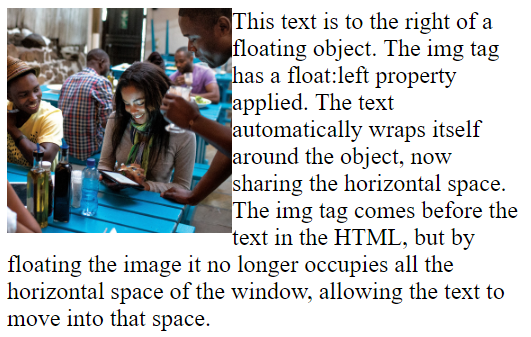
[[*Documentation*](https://www.w3.org/TR/CSS22/visuren.html#float-position)]

1. h1 {
2. width: 20em;
3. float: right;
4. }

Up until now, we haven't moved elements very far from wherever the web browser automatically places them, but as you've probably noticed this has left our page very left side heavy. This is because, by default, elements are stacked one on top of the other, and they don't share horizontal space. With the float property, we can change this.

The float property liberates an element from its automatic position and lifts it up to "float" on top of other elements in the direction you specify.  You can specify float either right, left or the default of none.

Elements underneath a floating object will automatically wrap themselves around the content. For example, if you float an image, the text underneath will wrap around it so that none of it is actually obscured underneath the image, but now both text and an image can share horizontal space.



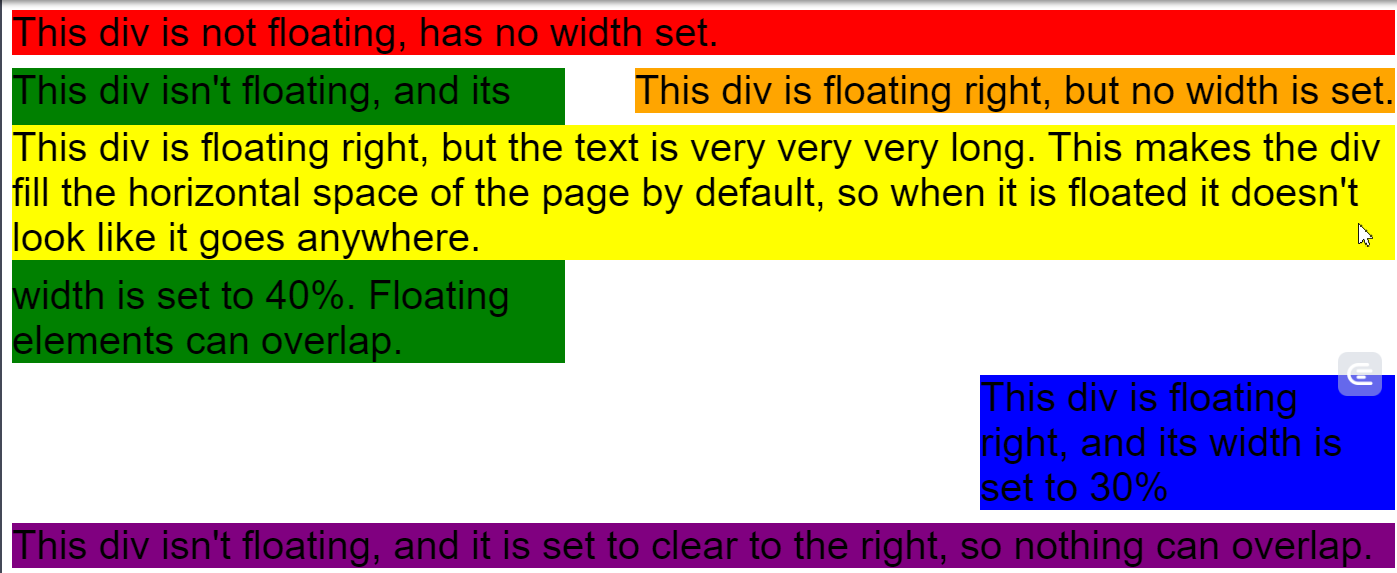
You'll often want to set the width of a floating object so that you have tighter control over the space that object occupies. Remember that, by default, block HTML elements occupy the entire width of the page, even if there isn't actual content that extends that far. In this case, you'll want to set the width so that your element's size more accurately represents its content and you don't have unnecessary white space.

### The clear property

Once you have some elements floating things can get a little messy. Its easy for floating objects to overlap, and to prevent this you can use the "clear" property.

[[*Documentation*](https://www.w3.org/TR/CSS22/visuren.html#propdef-clear)]

1. p {
2. clear: both;
3. }



<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Floating elements</title>

<meta charset="utf-8">

</head>

<body>

<div id="default">

This div is not floating, has no width set.

</div>

<div id="floatRightNoWidth">

This div is floating right, but no width is set.

</div>

<div id="floatRightTooWide">

This div is floating right, but the text is very very very long. This makes the div fill the horizontal space of the page by default, so when it is floated it doesn't look like it goes anywhere.

</div>

<div id="noFloatWidthSet">

This div isn't floating, and its width is set to 40%. Floating elements can overlap.

</div>

<div id="floatRightWidthSet">

This div is floating right, and its width is set to 30%

</div>

<div id="noFloatClearRight">

This div isn't floating, and it is set to clear to the right, so nothing can overlap.

</div>

</body>

</html>

CSS

body {

font-size: 24pt;

font-family: helvetica, sans-serif;

}

div {

margin-bottom: 10px;

}

#default {

background-color: red;

}

#floatRightNoWidth {

background-color: orange;

float: right;

}

#floatRightTooWide {

background-color: yellow;

float: right;

}

#noFloatWidthSet {

background-color: green;

width: 40%;

}

#noFloatClearRight {

background-color: purple;

clear: right;

}

#floatRightWidthSet {

background-color: blue;

width: 30%;

float: right;

}

## 4.3.2 Activity - Practice with float

One of the toughest parts of layouts with CSS is figuring out which elements to apply a float property to.

Here is [some HTML and CSS](https://codepen.io/techie4good/pen/xEbWrp):



##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Practice with Float</title>

<meta charset="utf-8">

</head>

<body>

<header>

<h2>welcome greeting subtitle</h2>

<h1>Homepage Title</h1>

</header>

<section id="navigation">

<ul>

<li>nav link 1</li>

<li>nav link 2</li>

<li>nav link 3</li>

<li>nav link 4</li>

</ul>

</section>

<section id="content">

<div id="topRight">

If your elements are still not exactly where you want them to be after adjusting the padding, margins and alignment, then you can try out the float property. The "float" property is one of the most powerful tools you can master when learning CSS.

</div>

<div id="topLeft">

Up until now, we haven't moved elements very far from wherever the web browser automatically places them, but as you've probably noticed this has left our page very left side heavy.

</div>

<div id="bottomRight">

The float property liberates an element from its automatic position and lifts it up to "float" on top of other elements in the direction you specify. You can specify float either right, left or the default of none.

</div>

<div id="bottomLeft">

Elements underneath a floating object will automatically wrap themselves around the content. For example, if you float an image, the text underneath will wrap around it so that none of it is actually obscured underneath the image.

</div>

</section>

</body>

</html>

##### CSS

body {

background-color: #4ABDAC;

color: #FFFFFF;

font-family: Georgia, serif;

}

header {

background-color: #F7B733;

height: 75px;

}

h1 {

padding: 15px;

}

#navigation {

height: 30px;

width: 30%;

margin-left: auto;

margin-right: auto;

}

#navigation li:hover {

border-bottom: 1px #FC4A1A solid;

}

#content {

background-color: #DFDCE3;

width: 50%;

margin: 0 auto;

}

div {

background-color: #FC4A1A;

width: 250px;

height: 150px;

padding: 10px;

margin: 20px;

}

As you can see the layout is pretty messy. Your job in this activity is to decide which elements deserve a float property (such as the navigation!). The main focus of the task is about understanding floats, but as before, try to ensure your page also works well when the window is resized.  
You might also need to adjust some widths, margins and paddings to get everything looking like the final image below:



**HINT**: Pay close attention to the IDs applied to the HTML elements

##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Practice with Float</title>

<meta charset="utf-8">

</head>

<body>

<header>

<h2>welcome greeting subtitle</h2>

<h1>Homepage Title</h1>

</header>

<section id="navigation">

<ul>

<li>nav link 1</li>

<li>nav link 2</li>

<li>nav link 3</li>

<li>nav link 4</li>

</ul>

</section>

<section id="content">

<div id="topRight">

If your elements are still not exactly where you want them to be after adjusting the padding, margins and alignment, then you can try out the float property. The "float" property is one of the most powerful tools you can master when learning CSS.

</div>

<div id="topLeft">

Up until now, we haven't moved elements very far from wherever the web browser automatically places them, but as you've probably noticed this has left our page very left side heavy.

</div>

<div id="bottomRight">

The float property liberates an element from its automatic position and lifts it up to "float" on top of other elements in the direction you specify. You can specify float either right, left or the default of none.

</div>

<div id="bottomLeft">

Elements underneath a floating object will automatically wrap themselves around the content. For example, if you float an image, the text underneath will wrap around it so that none of it is actually obscured underneath the image.

</div>

</section>

</body>

</html>

##### CSS

body {

background-color: #4ABDAC;

color: #FFFFFF;

font-family: Georgia, serif;

margin: 0px;

padding-bottom: 0px;

}

header {

background-color: #F7B733;

height: 75px;

margin-top: 0px

}

h2 {

float: right;

padding-right: 20px;

}

h1 {

padding: 20px;

margin-top: 0px;

}

#navigation {

height: 50px;

width: 92%;

}

#navigation li:hover {

border-bottom: 1px #FC4A1A solid;

}

ul {

width: 400px;

margin-left: auto;

margin-right: auto;

list-style: none;

}

li {

float: right;

margin-top: 10px;

margin-left: 20px;

}

#content {

background-color: #DFDCE3;

width: 20%;

margin: 0 auto;

min-width: 700px;

}

div {

background-color: #FC4A1A;

width: 250px;

height: 150px;

padding: 10px;

margin: 20px;

}

#topRight {

float: right;

margin-top: 0px;

}

#bottomRight {

float: right;

margin-top: 0px;

margin-bottom: 0px;

}

#topLeft {

margin-top: 0px;

}

#bottomLeft {

margin-bottom: 0px;

}

## 4.4.1 Relative positioning

Up and till now, you've probably noticed

that all the Web sites you've been making:

a lot of content on the left and a lot of white space on the right.

This is because we've allowed all of

your HTML elements to stay in their default positions.

However, when you look at Web sites out in

the world, a lot of them are much better balanced.

This is because those designers

are controlling the relationship of the

elements to each other and to the browser window.

You too can place your elements using

only six properties: top, right, bottom, left, width and height.

By designing the positioning of your Web site through the

relationship of HTML elements to each

other and to the browser window,

as the browser window shrinks and grows, your layout maintains its shape.

The "position" property sets the algorithm for how the Web browser will compute the way the HTML elements are placed on the page. There are four different value options for the position property:

* **relative** - This is the position setting we will be discussing in detail as it is the best way to build on the dynamic layout methods we have learned thus far. This lets you specify where an HTML element should be relative to where it would be by default.
* **static** - This is the default setting and will place an element wherever the web browser computes it should be.
* **fixed** - This places an element in a specific location within the window. You can use this to set an element to remain where it is regardless of scrolling. This was used heavily when "frames" were popular, however now it should be used very sparingly.
* **absolute** - This allows you to position elements relative to their containing box. This removes an element from its normal flow (just like a floating element) so it can be difficult to control and make a layout that is truly dynamic.

1. div {
2. position: relative;
3. }

[[*Documentation*](https://www.w3.org/TR/CSS22/visuren.html#position-props) for the box offsets: 'top', 'right', 'bottom', 'left']

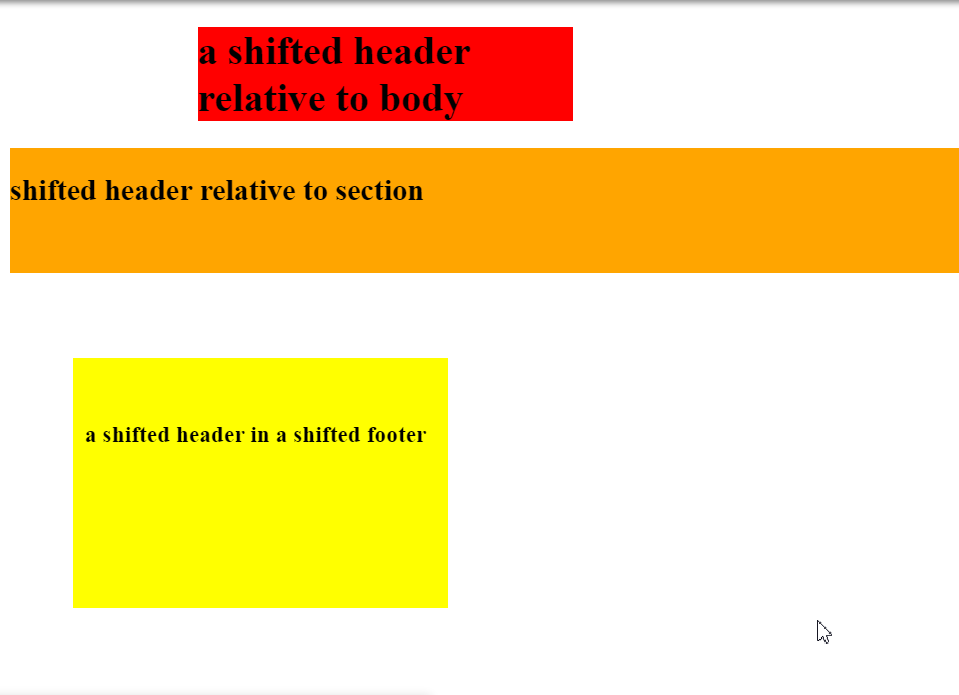
Once you've set the position to "relative" that frees you up to set the **top**, **right**, **bottom** and **left** properties- otherwise known as the "box offsets". These properties specify the distance between this object and its normal static position and the corner of the box that we are specifying.

For example, if we set the "left" to be "30px", it will move the element 30px to the right away from the left of where it was placed by default.

1. p {
2. position: relative;
3. left: 10px;
4. }

Note that position is not an inherited property so you will have to apply it individually to each element. Because of this it is best to use this approach to designing your layout sparingly and should only be used after you cannot achieve your desired layout with alignment or floating.

##### Example 1:



##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Relative Positioning</title>

<meta charset="utf-8">

</head>

<body>

<h1>a shifted header relative to body</h1>

<section>

<h2>shifted header relative to section</h2>

</section>

<footer>

<h3>a shifted header in a shifted footer</h3>

</footer>

</body>

</html>

##### CSS

h1 {

background-color: red;

width: 300px;

position: relative;

left: 150px;

}

section {

background-color: orange;

height: 100px;

}

h2 {

position: relative;

top: 20px;

}

footer {

background-color: yellow;

height: 200px;

width: 300px;

position: relative;

left: 50px;

top: 50px;

}

h3 {

position: relative;

top: 50px;

left: 10px;

}

### Example 2:

### 

Note that relative positioning can make elements overlap - check the following CSS code as a second example:

h1 {

background-color: red;

width: 300px;

position: relative;

left: 150px;

}

section {

background-color: orange;

height: 100px;

}

h2 {

position: relative;

top: 20px;

}

footer {

background-color: yellow;

height: 200px;

width: 300px;

position: relative;

left: 50px;

top: -50px;

}

h3 {

position: relative;

top: 50px;

left: 10px;

}

## 4.4.2 Activity - Practice with relative positioning

For this activity, we are going to focus on using relative positioning to adjust how items sit on the page.

Here is [some HTML and CSS:](https://codepen.io/techie4good/pen/KgwRwg)



##### CSS

body {

background-color: #EFEFEF;

color: #A9A9A9;

font-family: "Book Antiqua", serif;

}

h1 {

color: #FF3B3F;

}

h2 {

}

div {

background-color: #CAEBF2;

width: 80%;

}

h3 {

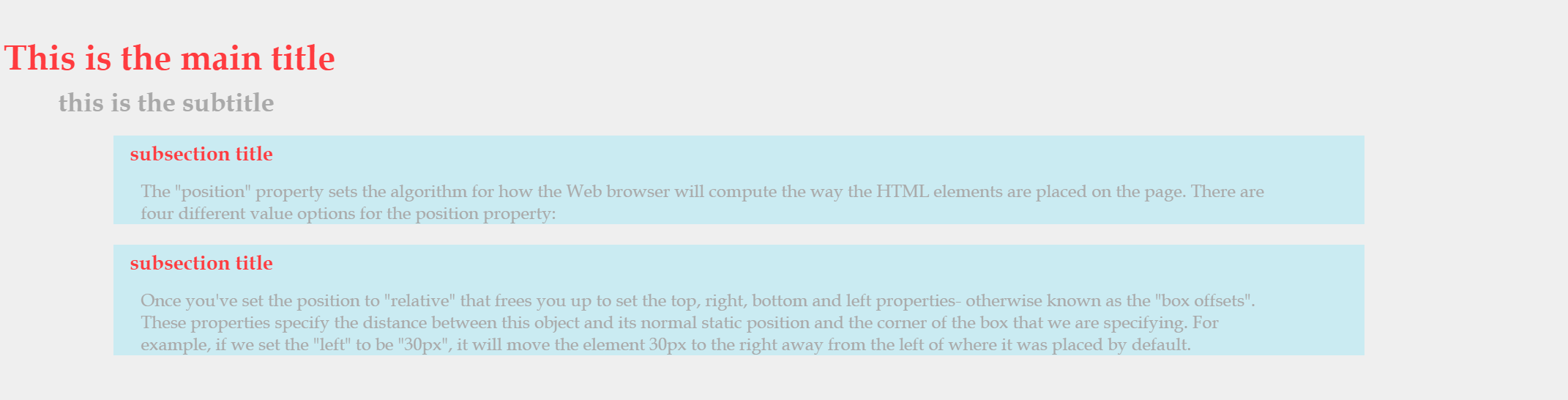
color: #FF3B3F;

}

p {

width: 90%;

}

Your task is to add CSS so that you achieve this final layout:  


You can do this with padding and margins, but limit yourself to only add position, top and left properties. Try resizing your browser window, and if you've implemented everything correctly all the elements will stay in the same position relative to one another.

##### CSS

body {

background-color: #EFEFEF;

color: #A9A9A9;

font-family: "Book Antiqua", serif;

width: 950px

}

h1 {

color: #FF3B3F;

height: 25px

}

h2 {

postiion: relative;

margin-left: 45px;

height: 25px

}

div {

background-color: #CAEBF2;

width: 80%;

position: relative;

left: 80px

}

h3 {

color: #FF3B3F;

padding-left: 10px;

}

p {

width: 90%;

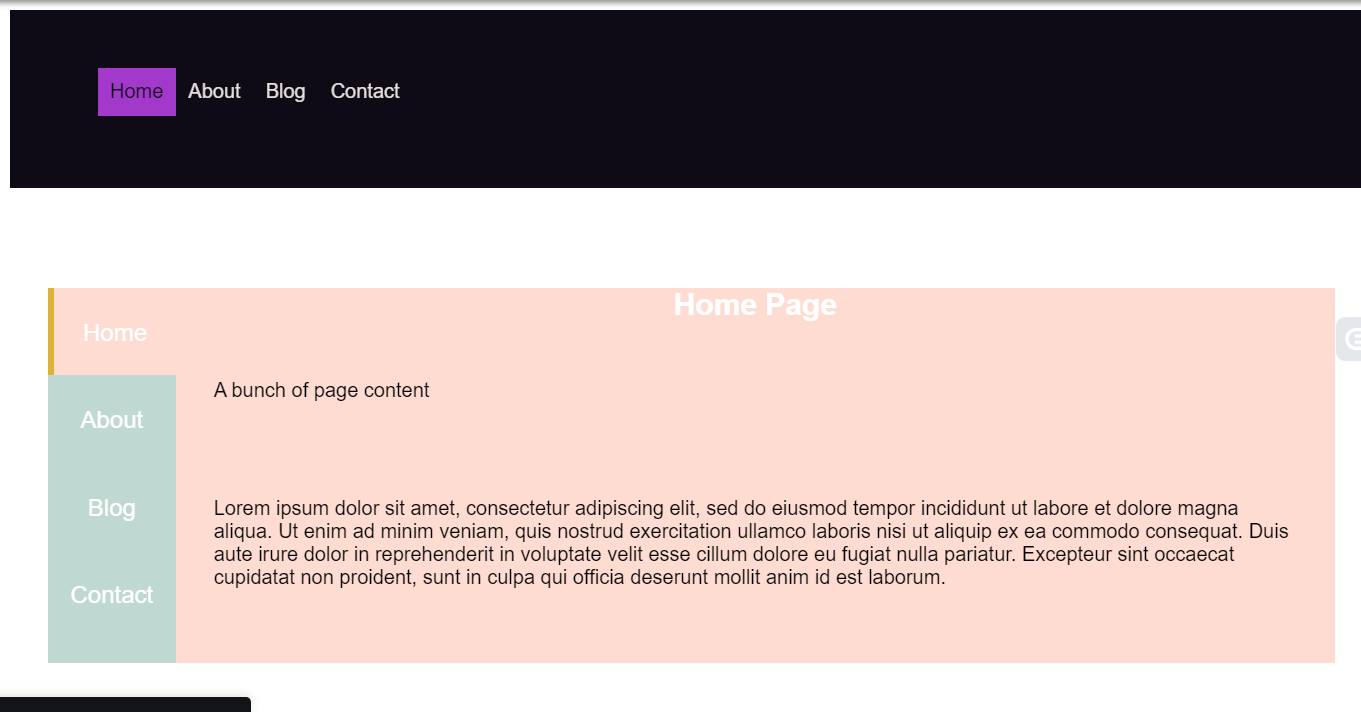
padding-left: 25px;

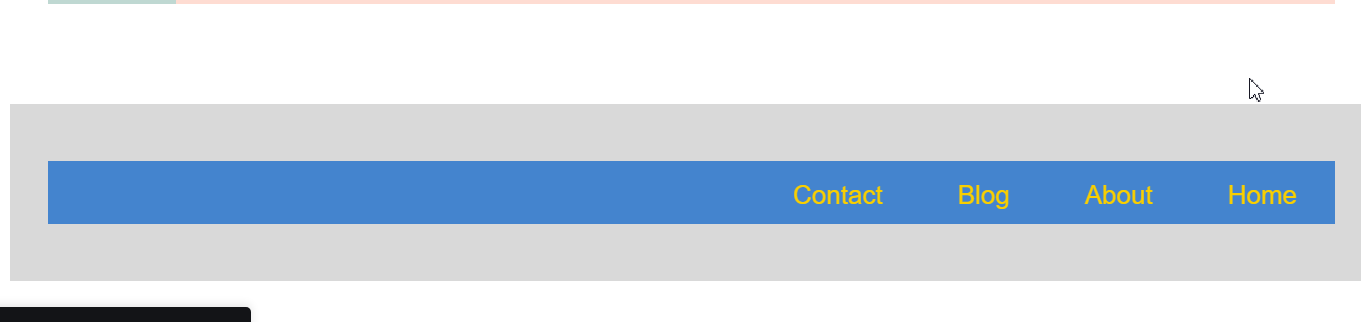
}

## 4.5.1 Menus

One of the most important aspects of any Web site is the navigation menu. Over time, top level navigation has become fairly standardized so your user will be looking for some key elements that help them find their way around your site:

* A small group of descriptive links in a rectangular arrangement, often horizontally
* A hover style to give your user some amount of responsiveness
* A special style indicating the link for the page you are currently on





### Menu 1

This is a very basic menu design. It floats the list elements to the left and gives them each a simple hover property (underline) and a new background color for the link representing the page you are currently viewing.

##### CSS

div {

clear: both;

margin-bottom: 50px;

padding: 30px;

font-family: Arial;

}

ul {

list-style: none;

height: 50px;

}

a {

text-decoration: none;

}

#menu1 {

background-color: #0E0B16;

}

#menu1 a {

color: #E7DFDD;

}

#menu1 li {

padding: 10px;

float: left;

}

#menu1 a:hover {

border-bottom: 2px #A239CA solid;

}

#menu1 .currentPage {

background-color: #A239CA;

}

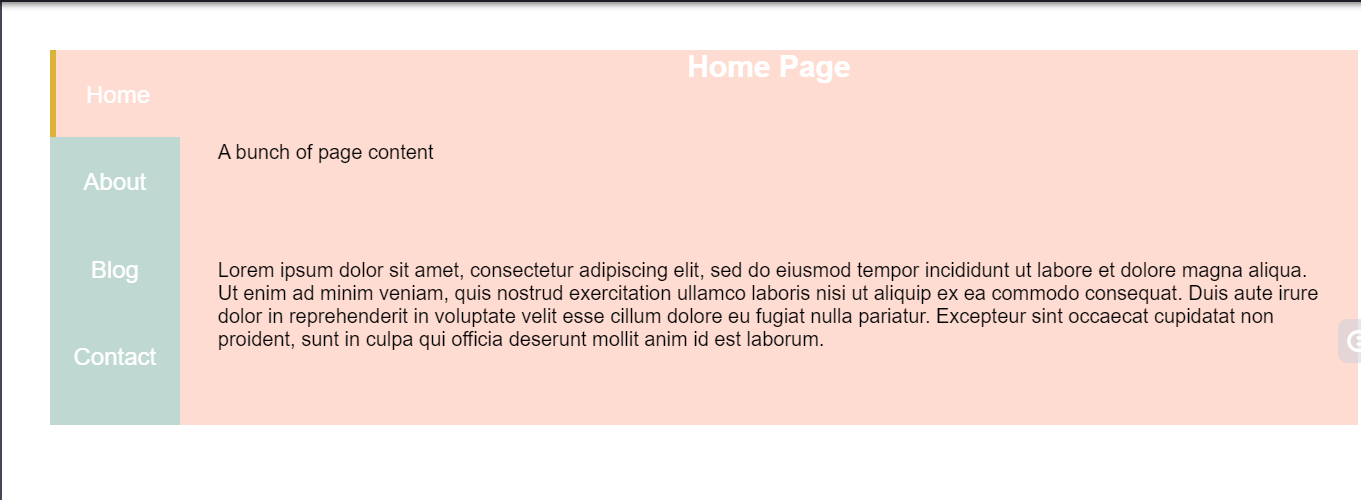
#menu1 .currentPage a {

color: #0E0B16;

}

### Menu 2

This menu design uses a vertical arrangement but still floats the overall menu object so it can sit level with your content. You can also see a tabbed format here where the page you are currently viewing directly connects to the menu item representing it.



##### CSS

div {

clear: both;

margin-bottom: 50px;

padding: 30px;

font-family: Arial;

}

ul {

list-style: none;

height: 50px;

}

a {

text-decoration: none;

}

#menu2 ul {

width: 10%;

height: 300px;

background-color: #BFD8D2;

margin: 0px;

text-align: center;

padding: 0px;

float: left;

}

#menu2 a {

color: #FFFFFF;

}

#menu2 li {

height: 15%;

padding-top: 25px;

}

#menu2 li:hover {

background-color: #FEDCD2;

border-left: 5px #DCB239 solid;

}

#menu2 li a {

font-size: 1.2em;

}

#menu2 .currentPage {

background-color: #FEDCD2;

border-left: 5px #DCB239 solid;

}

#menu2 section {

background-color: #FEDCD2;

margin-left: 10%;

height: 300px;

}

#menu2 h1 {

margin: 0px;

color: #FFFFFF;

text-align: center;

}

#menu2 p {

padding: 30px;

}

##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Style studies: menu 2</title>

<meta charset="utf-8">

</head>

<body>

<div id="menu2">

<ul>

<li class="currentPage"><a href="">Home</a></li>

<li><a href="">About</a></li>

<li><a href="">Blog</a></li>

<li><a href="">Contact</a></li>

</ul>

<section>

<h1>Home Page</h1>

<p>A bunch of page content </p>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</p>

</section>

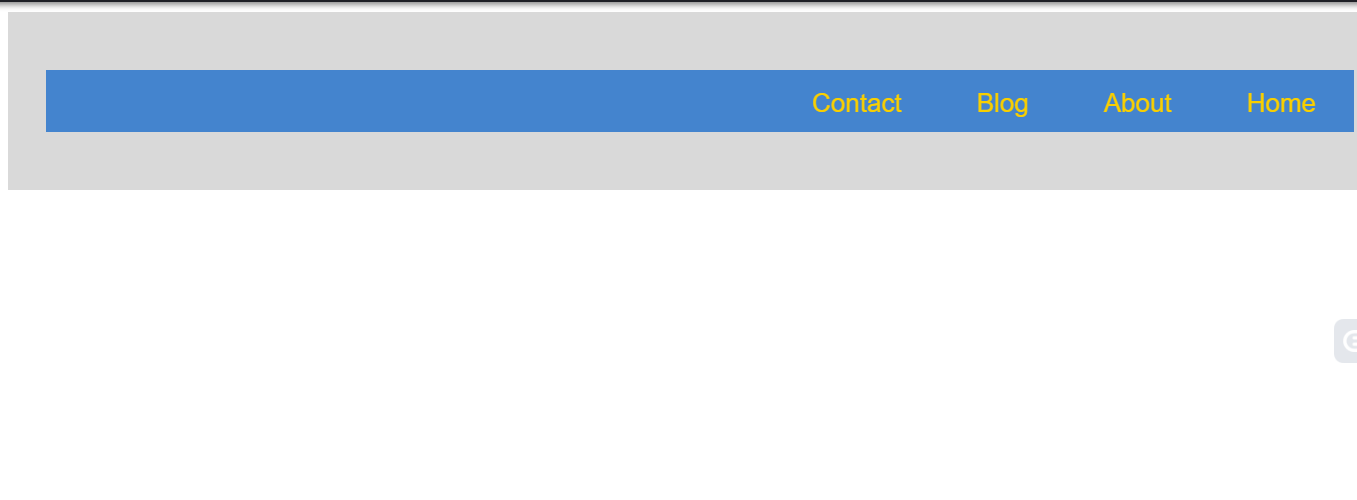
</div>

</body>

</html>

### Menu 3

This third design employs hover as a way to expose secondary links. This lets you leave the top level clean and simple but gives the user the power of more specific options when they interact with your header.



CSS

div {

clear: both;

margin-bottom: 50px;

padding: 30px;

font-family: Arial;

}

ul {

list-style: none;

height: 50px;

}

a {

text-decoration: none;

}

#menu3 {

background-color: #D9D9D9;

}

#menu3 ul {

background-color: #4484CE;

}

#menu3 li {

float: right;

padding: 15px 30px;

}

#menu3 li:hover {

background-color: #F9CF00;

}

#menu3 li:hover a {

color: #4484CE;

}

#menu3 a {

color: #F9CF00;

font-size: 1.3em;

}

#menu3 .subitems {

background-color: #F9CF00;

width: 100%;

padding: 0px;

display: none;

}

#menu3 .subitems li {

width: 100%;

margin: 0px;

padding: 0px;

float: none;

color: #4484CE;

}

#menu3 #about:hover .subitems {

display: block;

}

HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Style studies: menu 3</title>

<meta charset="utf-8">

</head>

<body>

<div id="menu3">

<ul>

<li class="currentPage"><a href="">Home</a></li>

<li id="about">

<a href="">About</a>

<ul class="subitems">

<li>subitem 1</li>

<li>subitem 2</li>

</ul>

</li>

<li><a href="">Blog</a></li>

<li><a href="">Contact</a></li>

</ul>

</div>

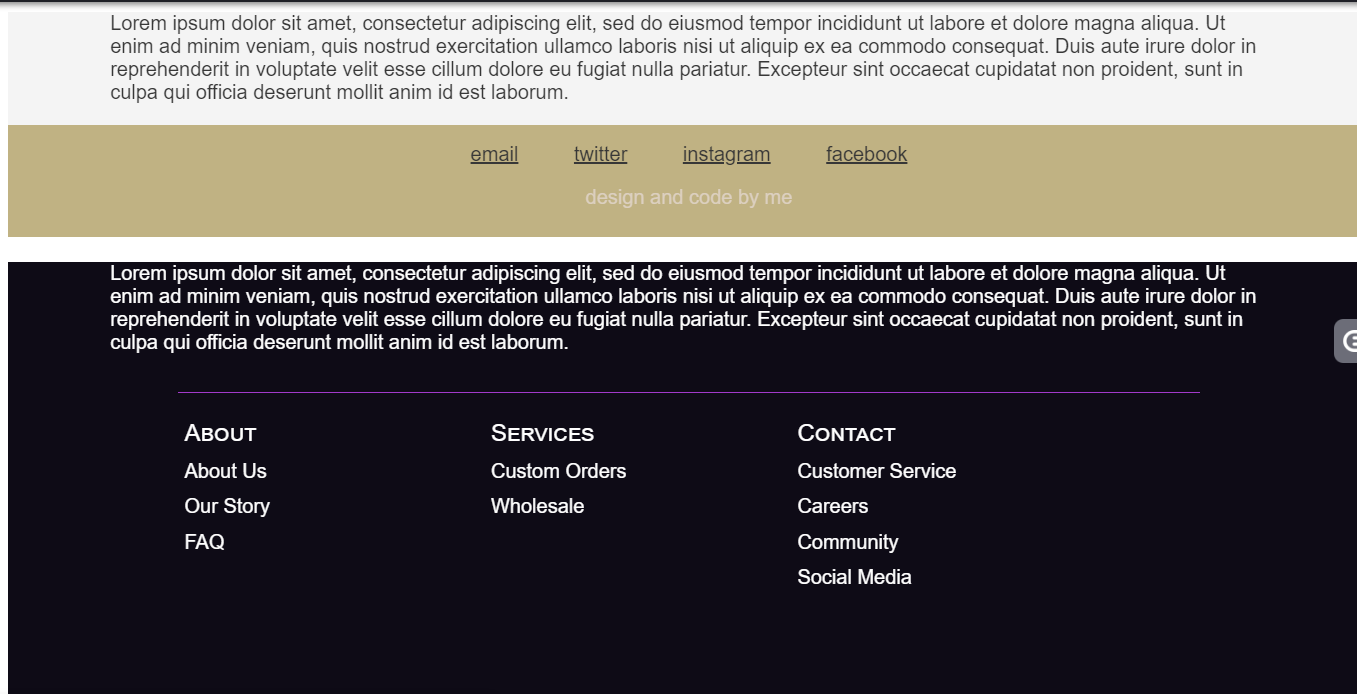
</body>

</html>

## 4.5.2 Footers

The footer of your page is typically the last thing your user will see, so it's important that you provide them any essential information. Typically, you will see footers that just contain contact info, but often they can also include navigation links, search bars or other calls to action.

You will want your footer to flow with your overall page design, but to be distinct from your content.



CSS

div {

font-family: Arial;

}

#footer1 {

background-color: #F4F4F4;

height: 150px;

margin-bottom: 50px;

}

#footer1 section{

height: 60%;

color: #373737;

width: 85%;

margin-left: auto;

margin-right: auto;

}

#footer1 footer {

background-color: #C0B283;

color: #DCD0C0;

height: 40%;

text-align: center;

padding: 15px;

}

#footer1 footer a {

color: #373737;

padding: 5px 20px;

}

#footer1 footer a:hover {

color: #F4F4F4;

}

#footer2 {

background-color: #0E0B16;

color: #FFFFFF;

height: 350px;

margin-bottom: 50px;

}

#footer2 section {

width: 85%;

margin-left: auto;

margin-right: auto;

padding-bottom: 30px;

}

#footer2 footer {

width: 75%;

margin-left: auto;

margin-right: auto;

color: #FFFFFF;

border-top: 1px #A239CA solid;

}

#footer2 ul {

float: left;

width: 30%;

list-style: none;

padding: 0px;

}

#footer2 li {

padding: 5px;

}

#footer2 .topLevel {

font-size: 1.2em;

font-variant: small-caps;

}

#footer3 {

height: 200px;

background-color: #F4DECB;

}

#footer3 section {

width: 70%;

margin-left: auto;

margin-right: auto;

background-color: #94618E;

color: #F8EEE7;

padding: 30px;

margin-bottom: 30px;

}

#footer3 footer {

width: 70%;

margin-left: auto;

margin-right: auto;

background-color: #49274A;

color: #94618E;

padding: 5px 30px;

text-align: center;

}

#footer3 a {

color: #FFFFFF;

}

#footer3 a:hover {

color: #F4DECB;

}

HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Style studies: footers</title>

<meta charset="utf-8">

</head>

<body>

<div id="footer1">

<section>

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

</section>

<footer>

<div class="contacts">

<a href="">email</a>

<a href="">twitter</a>

<a href="">instagram</a>

<a href="">facebook</a>

</div>

<p>design and code by me</p>

</footer>

</div>

<div id="footer2">

<section>

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

</section>

<footer>

<ul class="navCat">

<li class="topLevel">About</li>

<li>About Us</li>

<li>Our Story</li>

<li>FAQ</li>

</ul>

<ul class="navCat">

<li class="topLevel">Services</li>

<li>Custom Orders</li>

<li>Wholesale</li>

</ul>

<ul class="navCat">

<li class="topLevel">Contact</li>

<li>Customer Service</li>

<li>Careers</li>

<li>Community</li>

<li>Social Media</li>

</ul>

</footer>

</div>

<div id="footer3">

<section>

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

</section>

<footer>

Please <a href="#">contact us</a> if you have any questions

</footer>

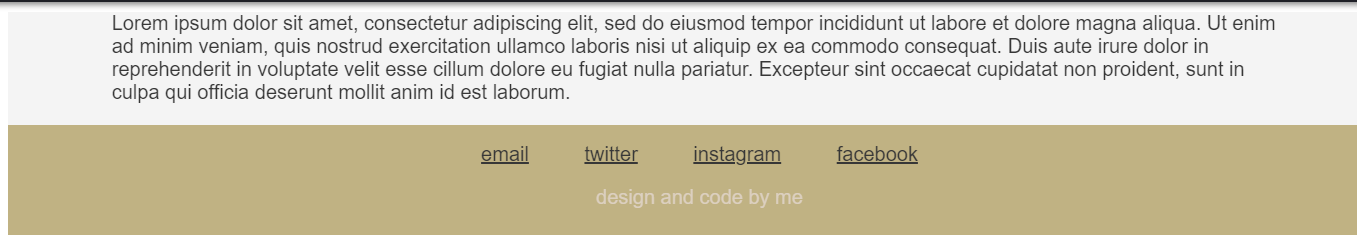
</div>

</body>

</html>

##### Footer 1

This is a basic footer that uses background color to help it stand out from the rest of the content. It contains contact links and a subtle reference to the designer of the page.



##### CSS

div {

font-family: Arial;

}

#footer1 {

background-color: #F4F4F4;

height: 150px;

margin-bottom: 50px;

}

#footer1 section{

height: 60%;

color: #373737;

width: 85%;

margin-left: auto;

margin-right: auto;

}

#footer1 footer {

background-color: #C0B283;

color: #DCD0C0;

height: 40%;

text-align: center;

padding: 15px;

}

#footer1 footer a {

color: #373737;

padding: 5px 20px;

}

#footer1 footer a:hover {

color: #F4F4F4;

}

##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Style studies: footer 1</title>

<meta charset="utf-8">

</head>

<body>

<div id="footer1">

<section>

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

</section>

<footer>

<div class="contacts">

<a href="">email</a>

<a href="">twitter</a>

<a href="">instagram</a>

<a href="">facebook</a>

</div>

<p>design and code by me</p>

</footer>

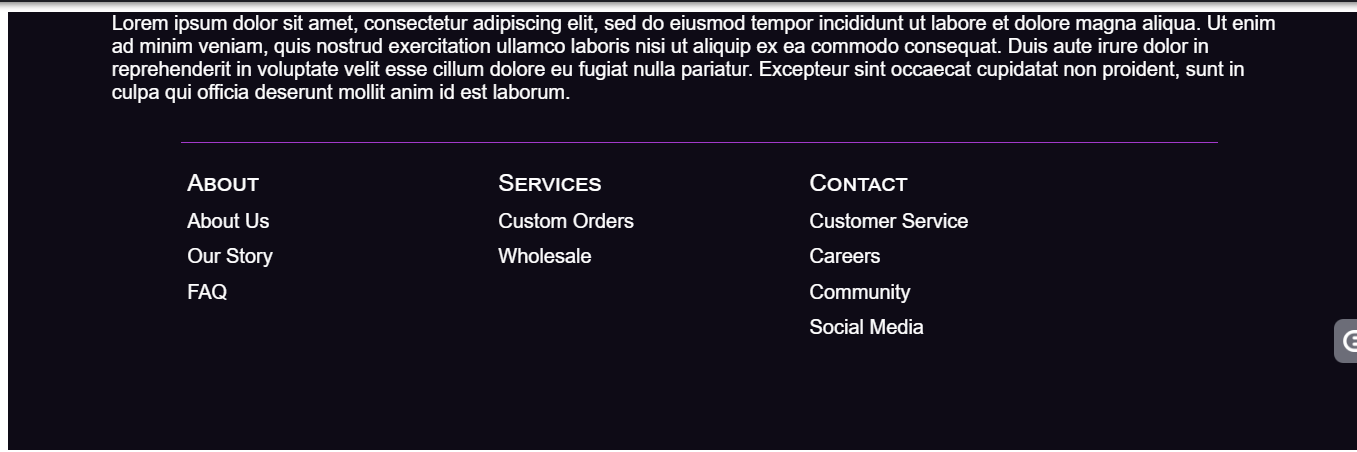
</div>

</body>

</html>

##### Footer 2

This footer provides navigation links. Because the footer is at the bottom of the page, you can get away with more links being exposed because limiting area isn't as important as in the top level header.



CSS

div {

font-family: Arial;

}

#footer2 {

background-color: #0E0B16;

color: #FFFFFF;

height: 350px;

margin-bottom: 50px;

}

#footer2 section {

width: 85%;

margin-left: auto;

margin-right: auto;

padding-bottom: 30px;

}

#footer2 footer {

width: 75%;

margin-left: auto;

margin-right: auto;

color: #FFFFFF;

border-top: 1px #A239CA solid;

}

#footer2 ul {

float: left;

width: 30%;

list-style: none;

padding: 0px;

}

#footer2 li {

padding: 5px;

}

#footer2 .topLevel {

font-size: 1.2em;

font-variant: small-caps;

}

HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Style studies: footer 2</title>

<meta charset="utf-8">

</head>

<body>

<div id="footer2">

<section>

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

</section>

<footer>

<ul class="navCat">

<li class="topLevel">About</li>

<li>About Us</li>

<li>Our Story</li>

<li>FAQ</li>

</ul>

<ul class="navCat">

<li class="topLevel">Services</li>

<li>Custom Orders</li>

<li>Wholesale</li>

</ul>

<ul class="navCat">

<li class="topLevel">Contact</li>

<li>Customer Service</li>

<li>Careers</li>

<li>Community</li>

<li>Social Media</li>

</ul>

</footer>

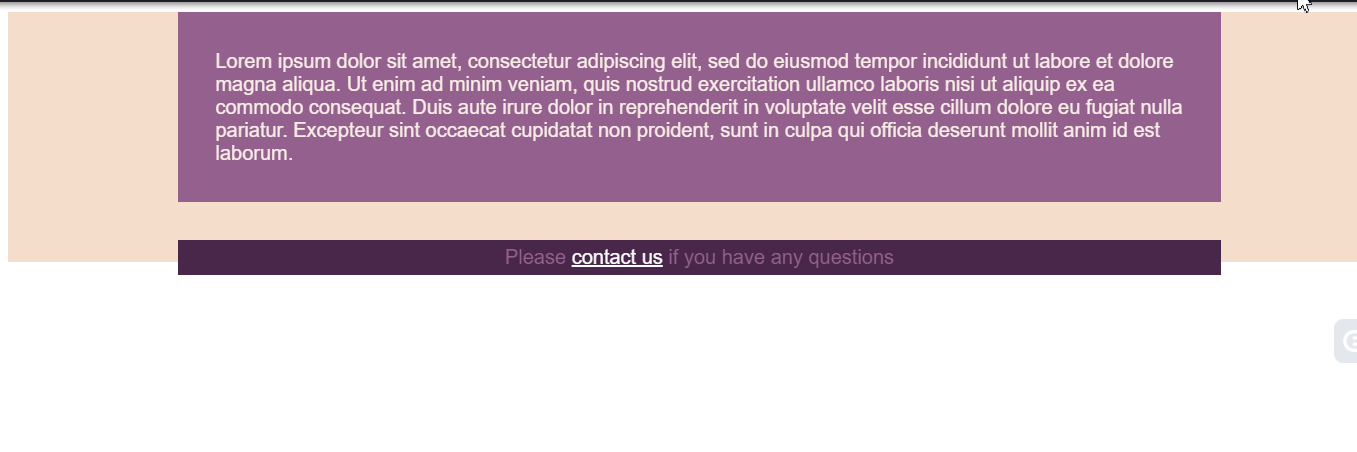
</div>

</body>

</html>

##### Footer 3

This final design flows with the overall structure of the page, but limits the content to a single simple contact link.



##### CSS

div {

font-family: Arial;

}

#footer3 {

height: 200px;

background-color: #F4DECB;

}

#footer3 section {

width: 70%;

margin-left: auto;

margin-right: auto;

background-color: #94618E;

color: #F8EEE7;

padding: 30px;

margin-bottom: 30px;

}

#footer3 footer {

width: 70%;

margin-left: auto;

margin-right: auto;

background-color: #49274A;

color: #94618E;

padding: 5px 30px;

text-align: center;

}

#footer3 a {

color: #FFFFFF;

}

#footer3 a:hover {

color: #F4DECB;

}

##### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Style studies: footer 3</title>

<meta charset="utf-8">

</head>

<body>

<div id="footer3">

<section>

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

</section>

<footer>

Please <a href="#">contact us</a> if you have any questions

</footer>

</div>

</body>

</html>

## 4.6.1 When to use what?

Now that we are at the end of Module 4, you have a long list of different ways to move HTML elements around your page using CSS. With so many tools come choices, as you can now accomplish the same task multiple ways. Here are some guidelines on how to decide when to use which tool, in the order in which you should use them when positioning an element.

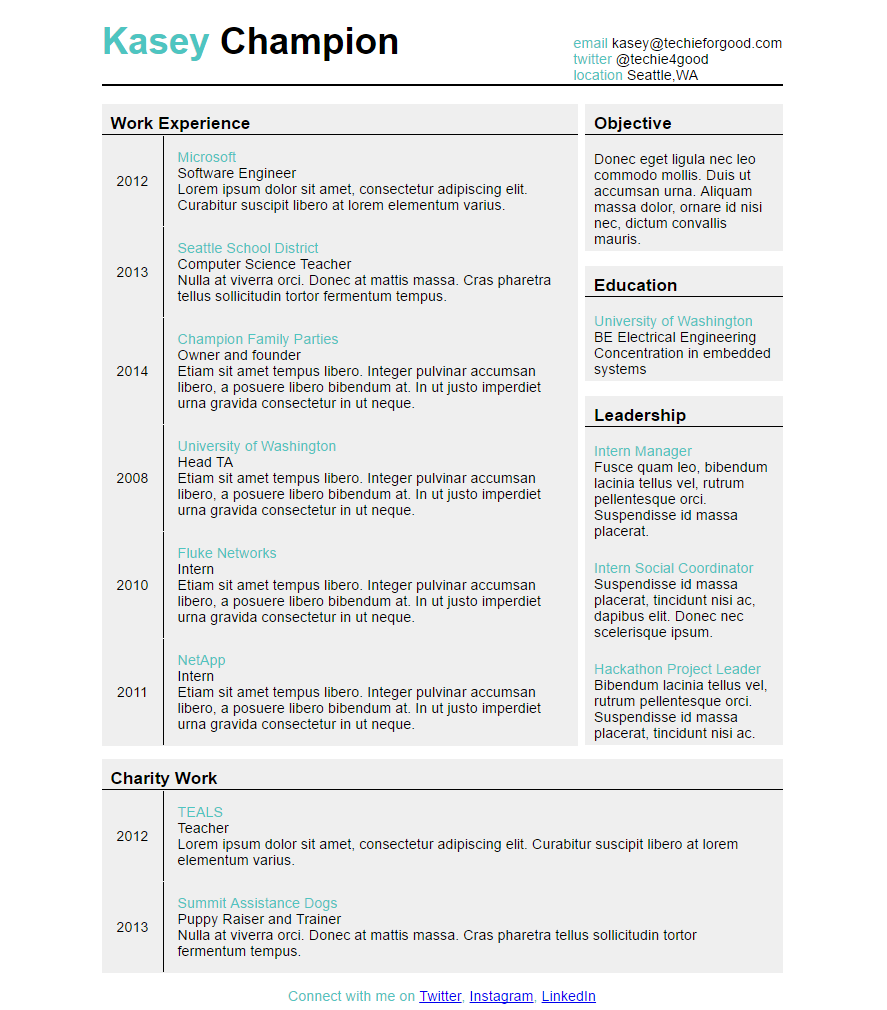
1. Use padding and margin whenever you can. This will keep things simple in relation to the box model and the overall flow of elements in relation to one another.
2. The float property is necessary when you want two block elements to share horizontal space. It can be easy to overuse the float property, keep in mind that if elements aren't floating the way you want them to you might want to adjust their order in the HTML instead of applying float to more and more elements.
3. If the above tools aren't getting your element where it should be, you can use relative positioning and directly set the box offsets (top, bottom, left, right). This makes things a bit more complicated especially when you try to inspect the box model around your element, but sometimes this is unavoidable because of collapsing margins.
4. If all that is not enough, there are advanced CSS features such as flexbox, table layout, absolute positioning and grid layout.

## 4.6.2 Module 4 project - My resume

In Modules 2 and 3, you've been building your profile as a Web page. For this module project, we are going to continue to build on that work and turn the profile into a "resume" or "CV", as a kind of online portfolio.

In some countries, traditional printed resumes typically are required to fit on a single piece of paper (A4 or Letter sizes), which means they need to make very effective use of the space available.  
For this project, you are going to try to reproduce a typical resume layout but with HTML and CSS.

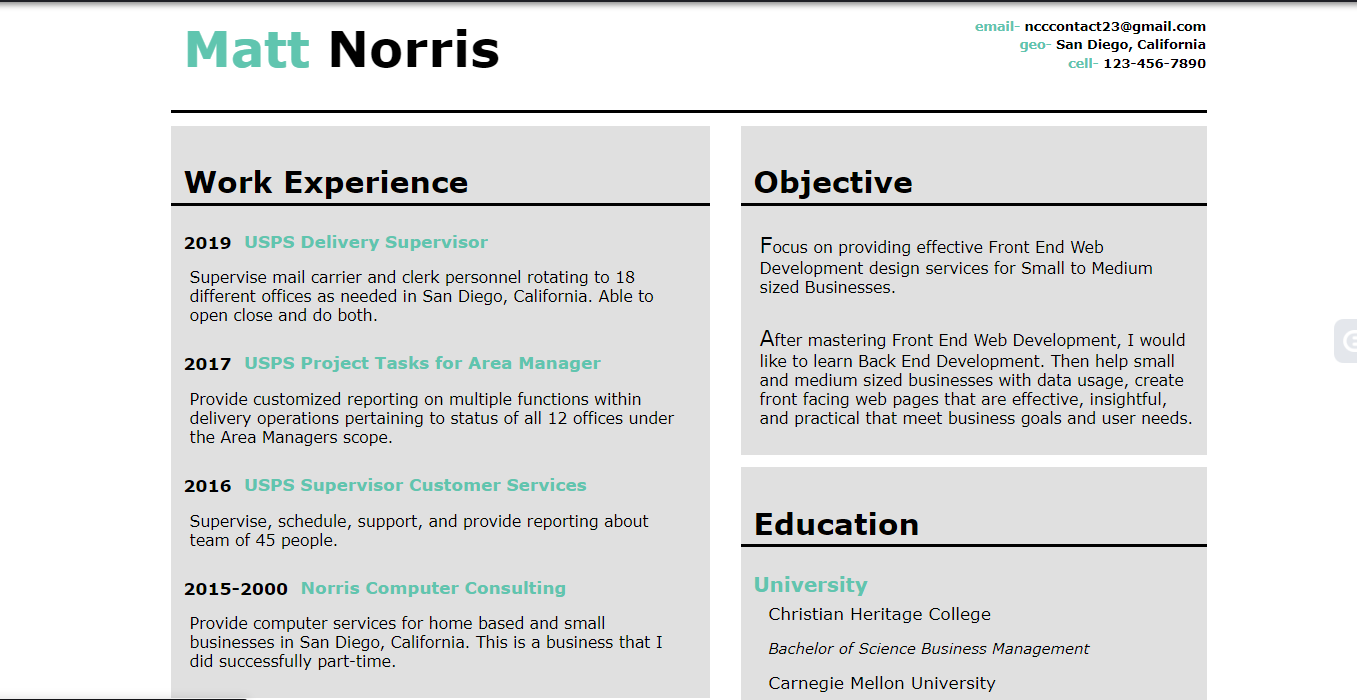
Here is what my Web resume looks like:



To more closely simulate a paper resume, your entire page's content must be centered in the middle 75% of the page. All your content should remain centered, and in the same position relative to the other elements on the page when you resize the browser window.

For this project you must at least:

* Use percentages to set the width of 3 elements
* Change the text-alignment property to something other than "left" on at least 1 element
* Have 4 floating elements
* Use the clear property once
* Use the position property and box offsets to position at least 2 elements



##### CSS

/\*

Project 4 Module 4.6.2

At times I use the number 2 at the beginning of an element to disable it. The number 2 is just random. So when you see that in my code, that is all this is about.

\*/

body {

background-color: white;

margin: auto;

font-family: Verdana, Geneva, Tahoma, sans-serif;

}

#resume {

position: relative;

}

header {

border-bottom: solid;

}

header li {

text-align: right;

}

header, #resume {

width: 75%;

margin: 10px auto;

min-width: 480px;

background-color: white;

}

h1 {

font-weight: bold;

font-size: 2.5em;

position: relative;

width: auto;

}

header ul {

float: right;

clear: right;

margin: -75px 0px 20px 10px;

width: 0 auto;

text-align: right;

}

#contact {

font-size: .75em;

font-weight: bold;

}

#workExperience {

min-width: 200px;

width: 52%;

background-color: rgb(224, 224, 224);

float: left;

clear: both;

margin-bottom: 10px;

}

#workExperience p {

width: 90%;

}

#objective {

width: 45%;

background-color: rgb(224, 224, 224);

float: right;

clear: right;

}

#education {

width: 45%;

background-color: rgb(224, 224, 224);

float: right;

clear: right;

margin-top: 10px;

margin-bottom: 10px;

}

#leadership {

width: 52%;

background-color: rgb(224, 224, 224);

margin-top: 10px;

clear: left;

}

.leadershipParagraph {

width: 90%;

padding-bottom: 10px;

}

#charity {

min-width: 200px;

width: 100%;

background-color: rgb(224, 224, 224);

float: left;

margin-top: px;

clear: both;

}

.charityWorkParagraph {

margin: 0 10px 0 0;

}

.objectiveSizing p {

width: 90%;

margin: 0 10px 0 0;

}

span {

font-size: 1.40em;

}

.textColorOffset {

color: #60c5af;

font-weight: bold;

font-size: 1em;

padding-left: 10px;

}

.year {

float: left;

padding: 1px 10px 8px 10px;

clear: both;

font-size: .85em;

2border-right: solid black;

2height: 25px;

}

.workPosition {

margin-left: 10px;

font-size: .85em;

}

p {

position: relative;

left: 10px;

text-align: left;

font-size: .78em;

padding: 0 20px 10px 5px;

font-weight: light;

}

h2 {

padding: 10px 0 3px 10px;

border-bottom: solid;

}

h3 {

font-size: .65em;

margin: 0;

}

#2favoriteQuote {

2font-family: "Times New Roman", Times, serif;

2padding: 20px;

2margin-top: 0;

2margin-bottom: 0px;

2color: black;

2font-style: italic;

2text-align: center;

2float: left;

2clear: both;

}

header ul li {

list-style: none;

padding: 1px;

}

ul {

margin: 5px 5px 5px -20px ;

2padding: 10px;

list-style: none;

}

li {

margin: 0;

padding: 0 0 12px 2px;

font-size: .85em;

}

.firstChoice {

font-weight: bold;

}

.degree,

.courseWork {

list-style: none;

font-size: 0.75em;

font-style: italic;

}

a:link {

color: blue;

}

a:visited {

color: blueviolet;

}

a:hover {

color: crimson;

text-decoration: underline;

text-decoration-thickness: 0.05em;

}

a:active {

color: #1da185;

}

footer {

background-color: white;

}

footer p {

2position:relative;

2top: 43px;

2left: 30px;

font-size: .75em;

clear: left;

padding-top: 10px;

}

footer a {

padding-left: 10px;

position:relative;

left: 180px;

top: -54px;

font-size: .75em;

text-decoration: none;

}

footer img {

width: 30px;

position: relative;

left: 20px;

top: 10px;

2clear: left;

2float: left;

2display: block;

2margin-left: auto;

2margin-right: auto;

2padding-top: 0px;

}

img:hover {

transform: scale(1.35);

}

a:hover, img:active {

opacity: 50%;

}

##### HTML

<!Doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>My Online Resume</title>

<link rel="stylesheet" href="css/style.css">

</head>

<body>

<!-- Header Section -->

<header>

<h1><span class="textColorOffset">Matt</span> Norris</h1>

<ul id="contact">

<li><span class="textColorOffset">email- </span>ncccontact23@gmail.com</li>

<li><span class="textColorOffset">geo- </span>San Diego, California</li>

<li><span class="textColorOffset">cell- </span>123-456-7890</li>

</ul>

</header>

<section id="resume">

<!-- Work Experience Section -->

<section id="workExperience">

<h2>Work Experience</h2>

<h3 class="year">2019</h3>

<h3 class="textColorOffset workPosition">USPS Delivery Supervisor</h3>

<p>

Supervise mail carrier and clerk personnel rotating to 18 different offices as needed in San Diego,

California. Able to open close and do both.

</p>

<h3 class="year">2017</h3>

<h3 class="textColorOffset workPosition">USPS Project Tasks for Area Manager</h3>

<p>

Provide customized reporting on multiple functions within delivery operations pertaining to status

of all 12 offices under the Area Managers scope.

</p>

<h3 class="year">2016</h3>

<h3 class="textColorOffset workPosition">USPS Supervisor Customer Services</h3>

<p>

Supervise, schedule, support, and provide reporting about team of 45 people.

</p>

<h3 class="year">2015-2000</h3>

<h3 class="textColorOffset workPosition">Norris Computer Consulting</h3>

<p>

Provide computer services for home based and small businesses in San Diego, California.

This is a business that I did successfully part-time.

</p>

</section>

<!-- Ojective Section -->

<section id="objective">

<h2>Objective</h2>

<p class="objectiveSizing">

<span>F</span>ocus on providing effective Front End Web Development design services for Small to Medium sized Businesses.</p>

<p class="objectiveSizing">

<span>A</span>fter mastering Front End Web Development, I would like to learn Back End Development. Then help small and medium sized businesses with data usage, create front facing web pages that are effective, insightful, and practical that meet business goals and user needs.

</p>

</section>

<!-- Education Section -->

<section id="education">

<h2>Education</h2>

<section>

<h3 class="textColorOffset">University</h3>

<ul>

<li>Christian Heritage College</li>

<li class="degree">Bachelor of Science Business Management</li>

<li>Carnegie Mellon University</li>

<li class="courseWork">Course Work in Information Systems</li>

<li>San Diego Community College</li>

<li class="courseWork">Course Work in Math and Information Sustems</li>

</ul>

</section>

<section>

<h3 class="textColorOffset">Certifications</h3>

<ul>

<li>Microsoft Certified Professional</li>

<li>CompTia A+</li>

<li>Front End Web Developer Professional Certificate - W3C,Microsoft (In Progress)</li>

</ul>

</section>

</section>

<!-- Leadership and Charity Section -->

<section id="leadership">

<h2>Leadership</h2>

<section>

<h3 class="textColorOffset">Problem Solve</h3>

<p class="leadershipParagraph">

Provide problem solving and decision making on a daily basis with delivery team and customers.

</p>

</section>

<section>

<h3 class="textColorOffset">Create</h3>

<p class="leadershipParagraph">

Create customized reporting for management team members to help make informed and efficient decisions.

</p>

</section>

</section>

<section id="charity">

<h2>Charity Work</h2>

<section>

<h3 class="textColorOffset charityWork">College Ave Baptist Church</h3>

<p class="charityWorkPargraph">

<span>H</span>elp where needed. Lorem ipsum, dolor sit amet consectetur adipisicing elit. Ducimus asperiores voluptatibus est error fugit accusantium nobis aspernatur, id dolorum veniam debitis hic architecto voluptate ipsum maiores tempore libero distinctio consequatur!

</p>

</section>

<section>

<h3 class="textColorOffset charityWork">Inter-Faith Shelter, San Diego, CA</h3>

<p class="charityWorkPargraph"><span>P</span>rovide designated food, serve meals to single people and families, talk with adults and children, clean kitchen and eatting area afterwards.</p>

<p class="charityWorkPargraph"><span>I</span>nter-Faith Shelter is a community effort to help people get back on their feet with a plan to obtain training or work. Congregations involved from many different faiths as many as 80 join on a rotational basis to house and feed people in need of getting back on their feet to be self sustaining. Their is accountability and rules to abide by out of respect for others to remain in this care situation. All is done on a volunteer basis.

</p>

</section>

</section>

<!-- Footer with Contact Info Section -->

<footer>

<p>

Connect with me on:

</p>

<a href="mailto:ncccontact23@gmail.com" target="\_blank">

<!-- <img

src="https://edxuploads.s3.amazonaws.com/162309585670190256428.png"

alt="picture of a letter for contacting via email"> -->Gmail</a>

<a href="https://facebook.com" target="\_blank">

<!-- <img

src="https://edxuploads.s3.amazonaws.com/1623095892121252315507.png"

alt="picture of a phone to signify contacting via phone"> -->Facebook</a>

<a href="#top"><img class="arrow" src="https://upload.wikimedia.org/wikipedia/commons/thumb/8/87/Arrow\_top.png/640px-Arrow\_top.png" alt="picture of an arrow symbol pointing up for going back up to top of page" title="Back to Top of Page"><a>

</footer>

</body>

</html>

## Module 4 learnings

### After Module 4, you should be able to...

* Control exactly where an HTML element is placed on a page
* Design the layout of your HTML elements such that they stay aligned as the window grows and shrinks
* Employ the alignment, float and position properties when appropriate for optimal page structure

### In the next module, we'll...

* Learn about how to design your Web page to meet your user's needs
* Review historical web design trends and what we can learn from them
* Explore current Web design styles and learn how to best use them

## 5.1.1 Welcome to Module 5

In our final module, we're going to be talking about design. It's one thing to know how to use CSS properties and selectors.

And it's another thing to know how to use them well.

This module is all about the design aspect of Web pages and how you can give your user an optimum experience.

First, we're going to talk about the different aspects of design to keep in mind when you're designing for your chosen audience, then we're going talk about historical design trends and the things we've learned from them.

And then we're going to end with current design trends, and specifically why we're choosing to make Web pages the way we are right now, so you can understand how to keep your Web page looking modern.

## 5.1.2 Module 5 outline

### In this module, we will:

* Learn how to apply basic design principals based on the context of your HTML elements
* Explore CSS and HTML's accessibility features and how you can design your page to accommodate a diverse audience
* Introduce you to features to help internationalize your page and make it easier for those viewing it in different languages
* Discuss the lessons learned from historical Web design trends and give you the new tech we use instead
* Meet the newest fashions in Web design to help you give your Web pages a modern look and feel

## 5.1.3 The importance of design

Have you ever visited a Web site before?

And you left feeling really frustrated?

A lot of times, people don't really put a lot of thought into their audience when they're designing their Web site.

For example, they can put things that you're really looking for in a place that's hard to find them.

Or they can forget that you're going to be looking at this Web site on a mobile phone.

When Web design goes wrong, it's really memorable and it doesn't leave a lot of room for error.

But you can avoid a lot of these mistakes with one simple thing.

Keeping in mind your audience.

Thinking about things like what devices your audience is going to use to access your page, what information is most important for them to find, and making sure that you're always designing for an inclusive audience will keep your user frustration low.

For example, let's say you build a beautiful Web site that looks great on your big monitor.

You're going to want to look at it on a phone. Because a lot of Web traffic these days, comes through a mobile phone.

And it's actually pretty easy to adjust your screen so that it adapts to the mobile phone size.

You're also going to want to keep in mind what information your users really looking for in your page, and use CSS to highlight that information and make it very visible. For example, don't hide your pay button! Make it sure your user knows exactly how to give you money.

And then, with inclusive design you're going to want to keep in mind users that are low visioned or colorblind.

For example, if you're using color to indicate something, you want to combine that with shape.

That it's really clear what you are trying to say in case color isn't a really clear factor for your user.

And with low vision, you're going to want to keep in mind that a lot of people access your Web site with a screen reader.

You're always going to want to fill out the alt tag for your images and follow through and make sure that when your Web site is read aloud, it still makes sense.

Just these simple tweaks can make sure that your user leaves very happy after accessing your Web site.

### Web stats

Numerous studies have been done to determine exactly how important Web design is. Here are a few interesting stats to guide you:

* 38% of people will stop engaging with a website if the content/layout is unattractive
* 2 out of every 3 minutes spent on the Web are via a mobile device

## 5.1.4 CSS best practices (optional)

You will find below an excerpt of CSS best practices (see the [full slide set](https://fantasai.inkedblade.net/style/talks/best-practices/#title)) that were written by  Elika J. Etemad (also known as [fantasai](https://fantasai.inkedblade.net/" \t "_blank)). Elika is an expert on the [W3C CSS Working Group](https://www.w3.org/Style/CSS/) (since 2004!) and a longtime contributor to the Mozilla Project. In addition to editing many of the CSS3 specifications, she’s worked on layout engine testing and development for Gecko and managing the CSS test suites at W3C.

### Executive summary

* **Logical source order:**  
  The order of the HTML content should make sense even without the CSS: for accessibility, mobile optimization, device adaptability, and long-term maintainability.
* **Liquid layouts and relativity:**  
  Use smart relative sizing: to optimize layouts while minimizing media query code forks.
* **Media queries:**  
  Adapt to screen size changes; get font size adaptation free by using ems.
* **Prevent zombie code:**  
  Dead code may come alive as CSS changes. Delete it before it does, and ruins your layout.
* **Test in multiple browsers:**  
  Your favorite browser is not always right.
* **Don't use proprietary features!**  
  Keep the Web open to everyone! Don't rely on the latest -WebKit- invention.
* **Turn off CSS:**  
  A well-coded page will be understandable without it.

### Foundations

* Indent your code for readability ease
* Learn how to code CSS before relying on frameworks (such as Bootstrap, etc.)
* **Separate content and style**
  + Use semantic markup, ie., "classes for meaning, not for show".  
    The following article is helpful to understand this concept: [Meaningful CSS: Style Like You Mean It](https://alistapart.com/article/meaningful-css-style-like-you-mean-it/) (Tim Baxter, May 2016 - A list apart). It is also fully described in the [HTML5&CSS Fundamentals course](https://www.edx.org/course/html5-and-css-fundamentals).
  + Use <table> for tabular data: don't use tables for layout, but if your content is tabular like a catalog, a calendar, or a price list, then the table element is the correct markup.
* **Linearized logical source order**The order of the HTML content should make sense even without the CSS.  
  Benefits are numerous as it works best:
  + for long-term site maintainability
  + for mobile
  + for accessibility
  + as a foundation for device adaptation (media queries)
* **Linguistic variations**: set the language correctly for better typography (see the section later in this module entitled "why Internationalization is important")

### Testing

* **Test without CSS**: turn off CSS, and if the page makes no sense, fix your markup.
* **Test in multiple environments**:
  + Resize the window
  + Zoom the text
  + Try a mobile browser
  + Navigate by keyboard
* **Test in multiple browsers**: remember that just testing in Chrome does not work for everyone!  ;)

### Adaptability

* **Media queries**: set media query breakpoints in em or ch, not always in px.
* **Liquid layouts and relativity**: what is your sizing based on?
* Containing block size? → Use percents.
* Viewport size? → Use viewport units: vw, vh, vmin, vmax
* Font height? → Use em or rem.
* Font pitch? → Use em or ch.
* Content size? → Use auto or min-content/max-content.
* Combination of the above? → Use the appropriate layout formulas: flex, min-width, max-width, etc.

Absolute units are usually the wrong answer.

### Defensive Coding

* !important means never override- to use with caution
* Use !important to define overriding rules, not for fixups
* Duplicate selectors if you need to increase specificity, or
* Simplify selectors if you need to decrease specificity
* **Don't over-escalate**: understand your code, and don't overkill.
* For example, avoid:        . z-index: 9999999999999999999999999999999999999;  
          . position: absolute; left: -10000000000px
* **Drop dead code**: you tried something and it didn't work? Delete it right away!
* Code to Standard
* Don't rely on proprietary extensions
* Don't use experimental features in production or commit to keeping up-to-date.
* Provide fallbacks / use @supports.

## 5.2.1 Applying basic design principals

Often it's hard to pull apart the pieces of a design that are "good" or "bad", most of that is subjective. Instead, it's better to think of individual pieces of a design as "effective" or "ineffective" according to the demographic and the task they are trying to achieve.

Web sites can become pretty complicated if you add many design elements, and it can be intimidating to figure out how to get started when thinking about how to present your Web page. One of the best ways to get started is to think about each of the individual HTML elements we've learned how to style so far, and how to style according to your specific goals.

In this module, we'll discuss how to style the three most fundamental design aspects of a Web page: typography, color, and white space.

## 5.2.2 Typography

A good rule of thumb when designing your Web site is to use no more than two different typefaces per page. Typically, this means that you select 1 bold typeface for titles or other eye-catching pieces of text, and a neutral typeface for large blocks or the body text of your page.

There is no official taxonomy of fonts, and not all browsers support all fonts, so it can be difficult to get your font looking exactly the way you want on all platforms. While you’re starting out, it’s best to stick to the most popular and commonly used fonts. [Here is a good list](https://www.w3.org/Style/Examples/007/fonts.en.html)of fonts for English pages and the appropriate fall-backs to get you started. On the Web, you can find other lists of available system fonts for non-Latin scripts, such as [this one](https://r12a.github.io/scripts/fontlist/).

As you get more comfortable, you can branch out to more exotic fonts. Remember that, when you are building your font-family set, you will want to always include a Web safe font alternative for an exotic font in case the user’s device doesn’t support it.

Body {

   font-family: “Segoe UI”, Helvetica, sans-serif;

}

When choosing your font, probably the biggest choice you’ll make is what category of font to use. There are 5 basic categories of font:



* **sans-serif** – These are the most popular fonts for Web pages. This means the letters do not have added flourishes, so the typefaces are simpler. Their simplicity makes them easier to display on computer screens as their resolution is much lower than a printed document. It is often suggested you choose a sans-serif font for large blocks of digital text.

*Examples*: Helvetica, Verdana, Arial, Tahoma

* **serif** - These fonts are the second most popular typefaces. "Serif" refers to the small flourish lines at the edges of letters and symbols. "Serifs" make each character more distinct, making text easier to read in print. This is why these fonts might remind you of a text from a typewriter, or of the fonts you see in printed books, newspapers or magazines. These typefaces can often be used effectively for titles or emphasis.

Examples: Times New Roman, Book Antiqua, Georgia

* **monospace** - These fonts guarantee that all letters have the same fixed width. This is similar to a manual typewriter, or how computer code appears in editors. These fonts were designed for the ease of the technology, not humans, so they should be used sparingly. The most effective time to use these is when showing snippets of code.

*Example*: Courier New

* **cursive** - These fonts mimic human handwriting often by joining letters or having an italic slant. For some languages, these fonts are extra effective such as Arabic. Other than for specific languages, these fonts in English can be rather complex so they are best use extremely sparingly.

##### *Example*: Comic Sans MS

* **fantasy** - This is the most diverse category of fonts and includes all of those that are particularly decorative. These can make really great top headers as they can give your Web page a very distinct visual identity. Rarely will you want to use these for anything other than titles. It is also good to note that few of these are widely supported, so to use these you'll probably want to download them from a font service to make them available for your user.

##### Example: Impact

##### External resource

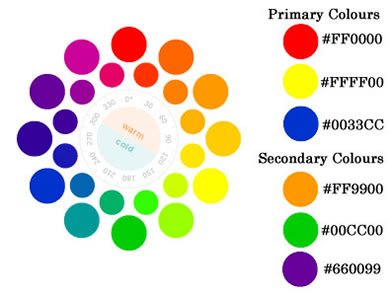
* Here is the W3C documentation for all of CSS's font properties: [CSS Fonts Module Level 3](https://www.w3.org/TR/css-fonts-3/).

## 5.2.3 Color

One of the most important design decisions you can make is your Web site's color palette. You should choose a palette before you begin designing to keep a cohesive visual identity. A common mistake it to use too many colors on a page. As you are starting out, it is best to limit yourself to just a few colors per page.

For a consistent look and feel your users will recognize, you will want to limit yourself to around 4 colors for all non-graphic content like text, backgrounds, borders, etc. There are different color palette tools on the Web you can play around with, but one of the best ways is to find a site you like the look and feel of and see if you can identify what color palette they are using.

When getting into design, it's a good idea to brush up on the basics of color theory, but just in case here's a short refresher. This is the color wheel:



### Colors make other colors

* The historical color wheel is organized around the three **primary** colors: red, yellow and blue.
* The **secondary** colors are the combinations of these primary colors: orange, green and purple.
* You can follow this pattern of varying the amount of each primary color to create infinite intermediary colors.
* However, as you've seen for the Web, we define all colors as combinations of [Red, Green and Blue](https://en.wikipedia.org/wiki/RGB_color_space) (green, not yellow). The short answer for this is because that is how the human eye perceives color, something that was unknown when the historical color wheel was made.

### Build a palette based on the color wheel

* Colors that are across the wheel are called "complementary", blue and orange, red and green, etc.
* Colors that are next to one another are considered  "analogous" like navy, blue and teal or lime, green and hunter.
* When picking a color palette, you should generally pick between one that is comprised mostly of analogous colors, or mostly of complementary colors. Thankfully, there are lots of wonderful tools to help you do this! One we suggest is [Palleton](https://www.paletton.com/" \t "_blank), where you can choose a starting color. It will generate for you a set of other colors that according to color theory will look pleasing together.

In your 4 colors, you'll want to keep a consistent tone so that your colors look good together. You'll want at least 1 very light color and 1 dark color. Avoid having all dark colors or all light, as often having contrast is important for readability. Also keep in mind not all users have full-color vision, so try to avoid too many similar colors. We will discuss how to choose inclusive colors in more detail in a later section.

### External resources

* Here is a good article that goes into detail about [color theory.](https://www.w3.org/wiki/Colour_theory)
* [LCH colors in CSS: what, why and how?](http://lea.verou.me/2020/04/lch-colors-in-css-what-why-and-how/), by Lea Verou (April 2020)

## 5.2.4 White space

It can be difficult to strike a good balance of white space. The most common mistake beginner Web designers make is to not leave enough white space or empty space between HTML elements.

You will want to make liberal use of paddings and margins to make sure that your site has plenty of empty space. Empty space makes it easier for a user's eyes to move around your page. You will want to ensure there is space between your HTML elements as well as between your elements and the edge of the page.

Good balance between content and white space prevents your user from becoming fatigued while looking at your site. For example, when lines of text are very long, it is difficult to read without losing track of where you are. It is also fatiguing when the lines of text are too short because the user has to read vertically too much. There is considerable research on the topic, but a good rule of thumb is to limit lines of text to 50-75 characters wide.

## 5.2.5 Activity - Breaking design guidelines

For this activity, you'll need to find a Web site that violates one of the following design guidelines:

* 1. Uses more than 2 different font-faces within a single page
  2. Uses more than 4 different colors within the same page
  3. Displays text wider than 50-75 characters across

Once you have found a site, please share it in the discussion and answer the following questions:

* 1. Which of the 3 design guidelines are violated? More than 1 guideline?
  2. Do you think that violation is a problem in this design?
  3. How would you improve the design overall?

##### Goldsmith & handcraft websites - do you have similar experiences?

I made a research about crafts and goldsmiths in germany for my bachelor thesis a few months back and already knew that they are mostly violating all the design guidelines because they don’t spend effort on building a decent website. It just hurts my eyes to look at them again and again, especially because their craft is so amazing. Just try out any goldsmith-website at your location and let me know if you have similar experiences. Would be happy if you share them with me :) Here is one example close to my location: <http://1a-trauring-area.de/>

i tried checking those around my area, I found this <https://www.kolanus.de/> and this <https://www.trauringstudio-bayreuth.de/> and I think they are okay, but I think this <https://schmuckwerkstatt-robert.de/> and this <https://juwelier-boehnlein.de/ueber-uns/bayreuth-trend/>.

I gave up searching for the nicht sehr gut ones. Cheers!

## 5.3.1 Web accessibility

The power of the Web is in its universality.  
Access by everyone regardless of disability is an essential aspect.  
**Tim Berners-Lee, W3C Director and inventor of the World Wide Web**

The Web has become an essential aspect of our daily lives, and everyone should have access to this technology. Web accessibility focuses on ensuring equivalent access for people with disabilities. It is increasingly important to many organizations and governments from around the world, and has many business benefits. Access to information, including on the Web, is also recognized by the UN Convention on the Rights of Persons with Disabilities (CRPD).

### Who is impacted?

Web accessibility addresses all disabilities, including hearing, learning and cognitive, neurological, physical, speech, and visual disabilities. Some examples of Web accessibility features include:

* Captions on audio and multimedia content for people who are hard of hearing;
* Clear and consistent layout for people with learning and cognitive disabilities;
* Keyboard support for people who do not use their mouse, for example, people with physical disabilities;
* Text alternatives for people with visual disabilities and using screen readers.

### Web accessibility benefits people with and without disabilities

Web accessibility features also benefit many more users, such as:

* People with temporary situational limitations, such as a broken arm;
* People using mobile devices, televisions, and other access channels;
* People using older computers, with low bandwidth, and other limitations;
* People who are new to computers, to the Web, or to your own website;
* People who are not fluent in the language of your particular website.

The Web is an increasingly important resource in many aspects of life: education, employment, government, commerce, health care, recreation, and more. When Web pages, Web technologies, Web tools, or Web applications are badly designed, they can create barriers that exclude people from using the Web. More information is available in the [W3C Accessibility](https://www.w3.org/standards/webdesign/accessibility) overview.

### First steps in Web accessibility

There are many simple Web accessibility improvements that you can implement and check right away, even when you are new to this topic. Two example excerpts are provided below on this page but you can find more tips and information from W3C/WAI:

* [Tips for Getting Started with Web Accessibility](https://www.w3.org/WAI/gettingstarted/tips/)
* [Easy Checks - A First Review of Web Accessibility](https://www.w3.org/WAI/eval/preliminary)

#### Example 1: page title

Good page titles are particularly important for orientation — to help people know where they are and move between pages open in their browser. The first thing screen readers say when the user goes to a different Web page is the page title. In the Web page markup, they are the <title> within the <head>.

**Check #1: There is a title that adequately and briefly describes the content of a page, and that it distinguishes the page from other Web pages.**

Example:

1. <head>
2. ...
3. <title>Web Accessibility Initiative (WAI) - home page</title>
4. ...
5. </head>

#### Example 2: image text alternatives ("alt text")

Text alternatives ("alt text") are a primary way of making visual information accessible, because they can be rendered through any sensory modality (for example, visual, auditory or tactile) to match the needs of the user. Providing text alternatives allows the information to be rendered in a variety of ways by a variety of user agents. For example, a person who cannot see a picture can have the text alternative read aloud using synthesized speech.

**Check #2: Every image has alt with appropriate alternative text.**

Example: See the W3C logo below. It contains a link that points to the W3C Web site. The text alternative is going to be a brief description of the link target.

1. <a href="https://w3.org">
2. <img src="images/w3c\_home.png" width="72" height="48" alt="W3C Web site">
3. </a>

## 5.3.2 Inclusive design

As you are designing your site, it is critical that you keep in mind the range of users who might be viewing your page. There are some simple design choices you can make to ensure that your web page is as inclusive as possible.

Typically, a good approach is to keep your designs on the simpler side. Not only is this in line with current trends, but when there are fewer elements and styles, it is easier for everyone to consume the information you are trying to convey.

### Color-blind

When a user is color-blind, it typically means there is a category of colors that are difficult for them to distinguish between. So, by picking diverse colors that contrast well, you can ensure that these users can still read your page. To do this:

1. Pick a base color (example: blue).
2. Choose one or two shades of the same color (example: pastel blue, navy blue) as supplementary colors. Make sure these colors contrast with the base color, meaning that if one was placed on top of the other, it is still legible. Also, if you are choosing multiple supplementary colors, try to pick one from the yellows group (warmer colors) and one from the blues group (cooler colors). This way, a color-blind user is likely to only have difficulty seeing one of these, not both.
3. Select an accent color that is a totally different color from the previously chosen colors.

Even with a carefully chosen palette, it might still be difficult for a user to distinguish between colors. In addition to using color to indicate something, you'll also want to use shape. For example, if you have two buttons, one that is green for "agree" and one that is red for "disagree", it can be almost impossible for some users to decide which to click! In this case, you'll want to make sure they are both clearly labeled, or the shapes of the buttons are distinctly different.

Thankfully, The Paciello Group has created a "[Color Contrast Analyser (CCA)](https://www.tpgi.com/color-contrast-checker/)" tool you can download to check if your chosen colors provide enough contrast for color blind users.

### Low-vision

It is extremely common (as many as 20% of users over 45) for a user to have difficulty reading small text. These users often will have their Web browser set to magnify text to a size that is more comfortable for them, which can make all your carefully designed layouts look very different. These users will typically do one of two things:

* zoom in on the entire page: you can test to see what this experience will be like with your own Web browser, typically with the "ctrl+" shortcut.
* use a text-only zoom: to see what it looks like, you might want to test your layout with larger fonts, like 24pt or 32pt, to make sure your layout can grow to accommodate the overflow text. Also, note that long lines of text (over 50 characters wide) can be especially difficult to scroll through to read.

In general, you will want to make sure your font size is at least 1em. By using em you can let the text size grow based on the user's browser settings.

### External resources

On W3C's WAI Web site:

* [Tips on designing for Web accessibility](https://www.w3.org/WAI/tips/designing/)
* [Color contrast](https://www.w3.org/WAI/test-evaluate/preliminary/#contrast)

## 5.3.3 Web internationalization

Access to the Web for all has been a fundamental concern and goal of the W3C since the beginning. It is easy to overlook the needs of people from cultures different to your own, or who use different languages or writing systems, but you have to ensure that any content or application that you design or develop is ready to support the international features that they will need.

'Internationalization' is sometimes abbreviated to 'i18n' in English, because there are 18 characters between the 'i' and the 'n'.

[](https://www.w3.org/International/)

The [W3C Internationalization Activity](https://www.w3.org/International/) works with W3C working groups and liaises with other organizations to make it possible to use Web technologies with different languages, scripts, and cultures.

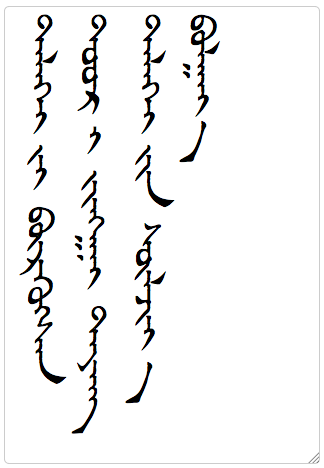
People who use non-Latin writing systems or use the Latin script for certain languages, often have specific typographic needs that differ from text in, say, English. As you learn more about CSS, you will find that it provides many features to support those needs.

Whereas HTML markup provides structure for the content of your page, CSS bring the expressive power to make the page look the way a person from a particular culture would expect.

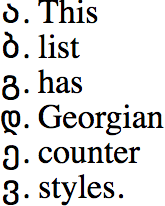
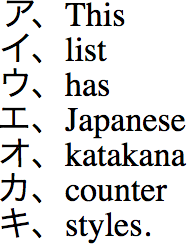
### Examples

Here are some examples of things that can be done with CSS.

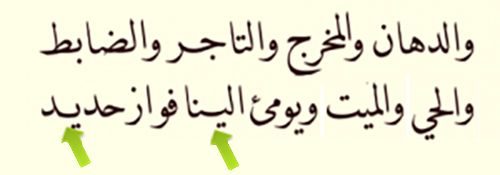
* It is already possible to make text run vertically in CSS for languages such as Chinese, Japanese, Korean and Mongolian. For more information see[Styling vertical Chinese, Japanese, Korean and Mongolian text](https://www.w3.org/International/articles/vertical-text/).



* You can also style counters for lists or chapter headings and such like according to local preferences. Here we see lists using Georgian and Japanese labels.

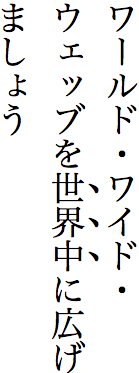
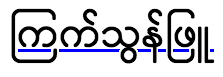
 

* When you want to justify text so that the lines are straight on both sides of your column, different strategies are used for different scripts. Most Western typography puts an emphasis on adjusting inter-word spaces, but Chinese doesn't use spaces between words, so you generally do inter-character spacing. In text written using the arabic script it is common to stretch the baseline that joins letters, or use other techniques to balance the line.



Some scripts allow words to be hyphenated in order to improve the visual effect of a paragraph, but note that the way in which words are hyphenated depends on the language. (And in arabic script, the CSS specification requires that both parts of the word retain their joining line during hyphenation.)

* Text decoration and text style features can vary in applicability from script to script. For example, Japanese characters are fairly complicated so, rather than italicise their text for emphasis, which can make it harder to read at small sizes, they have a tradition of placing special marks alongside the emphasised text (see the middle line of the Japanese example below). Also, it may be important to avoid underlines running over descenders in some scripts, since it can obscure important marks attached to a base character, so CSS allows you to skip 'ink' as shown in the Burmese example below.

These are just a few examples. There are many more.

### CSS & language

**An important point to bear in mind is that for many of these features to work as expected, you need to declare the language of the content.** For example:

* Hyphenation won't work unless the content is labelled for language. This is because the way that hyphenation works, and the dictionaries it uses, are language-specific.
* If you want to convert Turkish or Azeri text to uppercase or vice versa, you will get incorrect results unless the browser knows that the text is in that language, because they have a dotted i and a non-dotted ı which do case conversion differently from European languages.
* If text wraps to a new line, by default it does so differently dependent on whether you are dealing with Chinese or Japanese.
* And we could continue...

Therefore, you should always ensure that the correct language is specified in the lang attribute on the html tag, to indicate the default language of the page. And if you have passages in another language inside the page, you should put a lang attribute on markup that surrounds them, too.

See also this article (19 April 2020) by Chen Hui Jing on "[CSS for internationalisation](https://www.chenhuijing.com/blog/css-for-i18n/)".

### Localization

In addition, CSS provides tremendous help if you have to translate content from one language to another. Being able to change a single line in a style sheet to appy a change to all the pages being translated, rather than having to edit every page, saves a massive amount of time. However, this works best when you keep the distinction between semantics (markup) and presentation (styling) clear.

Don't use CSS to apply direction for bidirectional or right-to-left scripts, such as content in Arabic, Hebrew, Persian, Urdu, Divehi, etc. [Use HTML markup instead](https://www.w3.org/International/questions/qa-bidi-css-markup).

### Internationalization quick tips

1. **Language:** Always declare the default language of your page using the lang attribute on the html tag, and indicate internal language changes.
2. **Localizable styling:** Use CSS styling for the presentational aspects of your page. So that it's easy to adapt content to suit the typographic needs of the audience, keep a clear separation between styling and semantic content, and don't use 'presentational' markup.
3. **Use international features:** Use the international features provided by CSS to make your pages look natural to your audience. The more you use such features, and the more you request them, the better browsers will support them.
4. **Check your colors and styles:** Be sensitive to local preferences of your audience for things such as color, but also use of white space, two-dimensional vs. uni-directional display of information, etc.
5. **Use start and end:** Using these values, where possible, rather than left and right makes it easier to convert content between languages that use right-to-left and left-to-right scripts.

## 5.4.1 The history of Web design

The way Web pages look has changed a lot over the years, mostly because the technology has come a really long way. But as Web sites change, we can look at them as a brief history of the development of technology and also the development of design.

Let's take a look at Web sites in the very early 90’s. There, after CSS was invented, were a lot of crazy graphics a lot of marquees a lot of moving text, mostly because we were all so excited to that we had this new tool and we could do so many fun things with it.

I know I am guilty of having lots of cursors with dragging gifs behind them and marqueed

titles, but slowly after all the designers had too much fun, we realized that when you make your Web page really busy, its really hard to get your user to focus on exactly what you want them focus on. So those things went out of style.

We then moved into the HTML ‘tabled’ era.

That's where people were using tables to structure their Web page.

Which makes sense because an HTML table is essentially just a backbone and you can put all of your information in it.

The problem is, it was a really rigid backbone, and it didn't let you really grow or shrink

your Web page, and so as we got more screens involved and more Web browsers and more users, we needed to grow out of such a rigid backbone and let us just organize our Web site using relative positioning like we've learned in this course.

Then there was the ‘flash’ era, where everything was animated, and we had these huge loading screens and download times and everything looked really nice, and then it really only worked for desktop machines.

And now that we're getting more into a mobile-first world we don't have the data to download these giant flash sort of Web sites.

Thanks to CSS3, we now have animations built into CSS, so we can have a little bit of interactivity, but in a much lower impact on our data usage.

These design trends still pop up every now and then, so it's not necessarily a bad thing to use an HTML table, especially if you want to display a table.

And it's not necessarily a bad thing to use Flash, but you just have a whole lot more tools at your disposal.

It's really fun to go and look back at historical Web sites, and pay attention to what they were using, but why they were using it, and wonder if maybe now there's a better technology.

## 5.4.2 Moving pieces

As Web design tools became more sophisticated, the Web seemed to fill up with lots of fancy features that ultimately were just distracting. Especially in an era before mobile phones, certain items were used to add emphasis and style, but now are considered dated and would be a huge issue for mobile Web browsers.

Here is a list of style elements that should be avoided:

* **marquee and rotating text** - instead, use the different aspects of font to emphasize text like: size, color, weight or typography.
* **cursors with animation** - a very exciting feature at the time, now would be considered very dated, especially since mobile Web browsers don't even show a cursor. If you are going to adjust the cursor's appearance, do so using the [CSS cursor property](https://www.w3.org/wiki/CSS/Properties/cursor) and limit your usage to help your users understand when an item is interactive.
* **lots of GIFs** - GIFs are still very in style, especially with the popularity of services like [giphy.com](https://giphy.com/). Be aware however that GIFs draw a LOT of attention, so try to only use them when necessary.
* **videos that play automatically** - these days, you'll find this tactic used almost exclusively by advertisements, so it is a good idea to avoid it if you don't want your user to dismiss your content.
* **background music** - sometimes still used to set a tone for your site, but often is considered disruptive by users. If you absolutely must have background music, make sure to give your user a way to turn it off that is easily accessible.

## 5.4.3 Rigid layouts

Now, when we design sites, we are always talking about "different screen sizes", "different devices", etc.

This is because there is such a huge array of ways that users will access a Web site.

Historically, this was not the case, and it was a safe assumption that your user would access your site using a device and screen that were very similar to the ones you used to make it.

As our devices evolved, so did our CSS layout tools, so we could move away from these old trends.

### Frames

Used as a crutch for Web site navigation, this was a site within a site that followed your user around as they navigated. It was often used in placement of a menu bar. This feature was removed in HTML5.

Instead, focus on making a user's flow through your site intuitive by building a visually distinct and consistent form of navigation.

### HTML tables for layout

Tables organize data into structured rectangular sections, so it was an easy way to organize your site in a grid-like layout. HTML tables came with lots of problems. They overloaded HTML trees with lots of unnecessary tags that all come with their own set style that had to be adjusted, and they were extremely limited in how much you could actually adjust them. They also can pose problems with accessibility, it was easy for your content to be displayed in the wrong order.

Initially, we replaced HTML tables with <div> tags, a multi-purpose tag that was much more flexible than a table row or column, but while divs were flexible they weren't very descriptive. With HTML5, we gained more content descriptive tags like <header>, <footer>, <section> and <article> that have the same flexibility of div but that better describe your content.

## 5.4.4 Heavy content

These days, it is extremely likely that many of your users will view your page on a mobile device, typically using cell data as their connection. Cell networks are much more limited than the hard-wired connections that were once the only gateway to the Internet.

Back when this was the case, Web designers employed a number of techniques that were very "heavy" (often Adobe Flash) or required a lot of data to download. This was a way for designers to add interactivity and sophisticated animations to sites. However, [Adobe no longer supports Flash](https://www.adobe.com/products/flashplayer/end-of-life.html) after 31 December 2020.

With CSS3, we now have a host of animations that come standard with CSS so there is less need for an external, heavy technology to make dynamic pages. Animations are beyond the scope of this class, however be aware that there are two ways to do animations: using [CSS animations](https://www.w3.org/TR/css3-animations/) or [CSS transitions](https://www.w3.org/TR/css3-transitions/).

## 5.4.5 Activity - Retro design styles

For this activity, please find a retro styled Web site that uses one of the following outdated design elements. You might want to use the [Internet Archive](https://archive.org/web/) to find some older Web pages where these elements were more prevalent.

1. overly animated text
2. animated cursors
3. Frames
4. HTML table layouts

Once you've found an example page, post it in the discussion and answer the following questions:

1. Which historic design element does this page use? More than one?
2. What new technology would you use to modernize this element?

#### Retro PayPal

discussion posted 2 months ago by [**blither**](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/discussion/forum/users/6016861)

[PayPal site from 1999](https://web.archive.org/web/20000125012558/http:/www.paypal.com/cgi-bin/webscr?cmd=index) uses a table for layouts. This would now be achieved using divs or the newer HTML5 tags such as section, article, header, footer etc.

I also found the [ABC News (Australia) in 1997](https://web.archive.org/web/19970412141359/http:/www.abc.net.au/), which uses frames and has some very dated menu buttons on the left hand side, with an animation that highlights the buttons one after the other every 5 or so seconds. I would use a better design, with the navigation links in a horizontal bar that only change colour when hovered over, rather than having the animation run continuously.

#### Old website Library of Congress

discussion posted a day ago by **[AllaPetr](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/discussion/forum/users/2254953)**

[https://web.archive.org/web/20040617105540/http://shopndrop.com/crinfo.php?seqtag=1](https://web.archive.org/web/20040617105540/http:/shopndrop.com/crinfo.php?seqtag=1) This is an old version of the website of the Library of Congress. I think it has a very rigid design with lots of tables.

For improvement, I would try to vary the presentation of the content by adding some color, images and different fonts. A menu bar would be more attractive and better organized as a structure.

#### activity

discussion posted about a month ago by [**Naenae3**](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/discussion/forum/users/24574149)

I suppose it's boring, but this site also uses a table for layout. [https://web.archive.org/web/19961227203847/http://ubl.com/](https://web.archive.org/web/19961227203847/http:/ubl.com/) You could easily fix that problem using divs or flex box.

Just for fun, I went looking for sites with animation. This one has some animation, but not heavily. [https://web.archive.org/web/19961223105317/http://www.feedmag.com/](https://web.archive.org/web/19961223105317/http:/www.feedmag.com/)

Also has some animation, but not much. [https://web.archive.org/web/20131021205646/http://www.wikihow.com/Main-Page](https://web.archive.org/web/20131021205646/http:/www.wikihow.com/Main-Page)

## 5.5.1 The current state of Web design

Now, it's 2016. And I say the year because the design trends I am going to talk about are really specific to this time period now.

Like anything, design grows with time. And something new will come along pretty soon.

But I want to talk about current design trends so you have a snapshot of where we are at now and where we might be going.

These days, the big ticket item is flat design, minimalism. We just came out of the era of a lot of drop shadows, and beveled edges, and now everybody wants very nice clean Web sites.

You'll also see a lot of very simplistic interaction models.

A lot of Web pages are just one long scroll.

That's because, if you think about your Web page on any screen, a long scroll does a really good job adapting, because it optimizes for vertical space, which works on your mobile phone or your desktop, and it doesn't put a lot of pressure on horizontal space, which can be so varied based on the screen.

When you're designing your Web site, you might want to go out and look at some example sites and you're going to see a lot of patterns.

You're going to see a lot of simple clean color schemes, you're not going to see very many drop shadows, and you're going to see a lot of long scrolling Web pages.

You don't have to design your Web pages this way, but that's where things are going, and if you want a nice modern look, it's a good idea to kind of focus on these principals.

The other thing that has come really into popularity is a focus on typography.

These days, we have a lot of options for what fonts you can use on a page, not only are there the Web safe fonts we've been talking about all along, but there's a lot of new services that give you access to even more fonts.

Now, don't go crazy, you're going to only want to pick one, maybe two fonts, for your page, but picking a nice font can have a real impact on how the user consumes your content.

And you'll see that a lot of people sort of pick signature fonts especially for their title, or for their logo.

If you keep these basic ideas in mind, long scrolling, flat designing and typography, you're probably going to end up with a really nice modern page.

## 5.5.2 Minimalism

This Web design trend is part fashion, part function. Since we always keep mobile devices in mind when designing, the "rich design" styles of the past (drop shadows, gradients, lots of images, colors, and fonts) have been replaced with the current trend "flat design".

Flat design emphasizes:

* simple elements - grid layouts, limited number of elements
* clean typography - sans-serif fonts, large sized and lower case
* flat colors - no drop shadows, no attempt to make 2D elements look 3D

##### Benefits of flat design:

* low load times - better for mobile network connections
* responsiveness - when there are a small number of elements, it is easier to manage their relationships so that the overall layout can grow and shrink to accommodate different screen sizes
* intuitive experience - research has shown that the cleaner a site, the easier it is for a user to find the information they are looking for

##### External resource

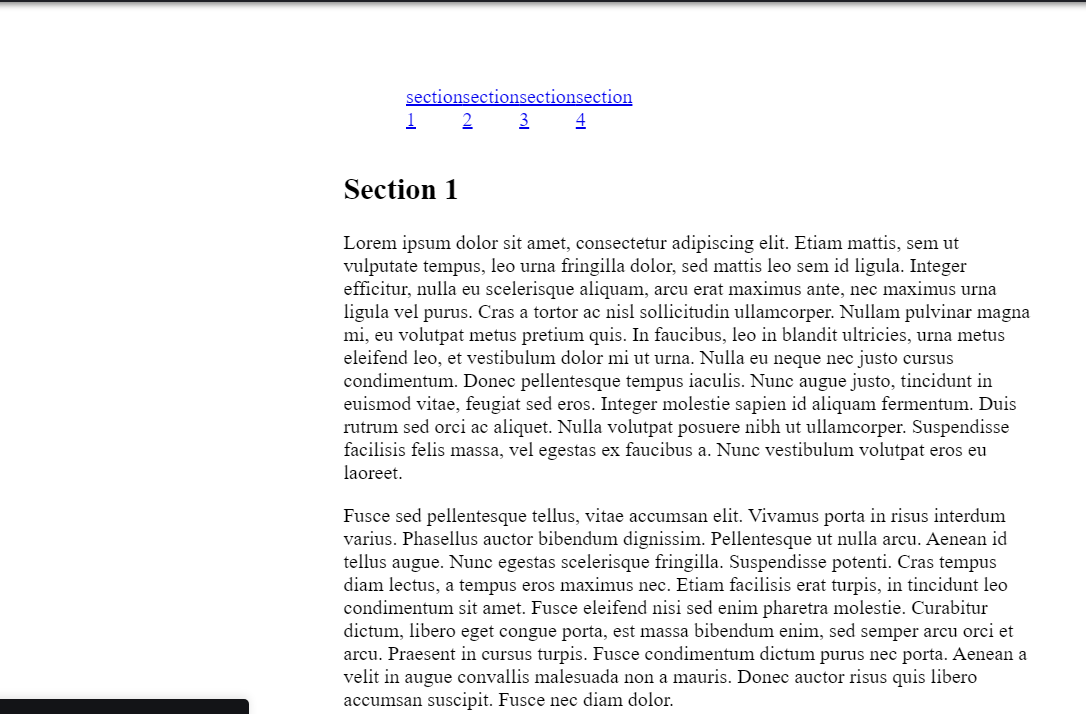
* A 2020 web.dev article: [Ten modern layouts in one line of CSS](https://web.dev/one-line-layouts/)

1 LINE LAYOUTS: <https://youtu.be/qm0IfG1GyZU?t=118>

## 5.5.3 Long scrolling

In the past, Web sites were comprised of many separate pages that you would navigate to, leaving behind the page you came from. These days, you'll often find pages that are comprised of a single long scrolling page with links that just jump to different portions of the page. The reason for this is less about fashion and more about technological developments. Especially on touch devices, scrolling is a more natural human way to interact with a page.

This can be accomplished with something called "internal links". This is where you load the entire page as one long site and the links at the top simply jump to different locations within the site. To do this, you need to give the section you want to jump to an id, and then you use the id selector as the href setting on the anchor tag.



### CSS

body {

width: 50%;

margin-left: auto;

margin-right: auto;

}

ul {

list-style: none;

padding: 50px;

}

ul li {

float: left;

width: 10%;

}

section {

clear: both;

}

### HTML

<!DOCTYPE html>

<!--It's a best practice to always declare DOCTYPE!-->

<html lang="en">

<head>

<title>Long Scrolling with Anchor Links</title>

<meta charset="utf-8">

</head>

<body>

<nav>

<ul>

<li><a href="#section1">section 1</a></li>

<li><a href="#section2">section 2</a></li>

<li><a href="#section3">section 3</a></li>

<li><a href="#section4">section 4</a></li>

</ul>

</nav>

<section id="section1">

<h1>Section 1</h1>

<p>

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam mattis, sem ut vulputate tempus, leo urna fringilla dolor, sed mattis leo sem id ligula. Integer efficitur, nulla eu scelerisque aliquam, arcu erat maximus ante, nec maximus urna ligula vel purus. Cras a tortor ac nisl sollicitudin ullamcorper. Nullam pulvinar magna mi, eu volutpat metus pretium quis. In faucibus, leo in blandit ultricies, urna metus eleifend leo, et vestibulum dolor mi ut urna. Nulla eu neque nec justo cursus condimentum. Donec pellentesque tempus iaculis. Nunc augue justo, tincidunt in euismod vitae, feugiat sed eros. Integer molestie sapien id aliquam fermentum. Duis rutrum sed orci ac aliquet. Nulla volutpat posuere nibh ut ullamcorper. Suspendisse facilisis felis massa, vel egestas ex faucibus a. Nunc vestibulum volutpat eros eu laoreet.

</p>

<p>

Fusce sed pellentesque tellus, vitae accumsan elit. Vivamus porta in risus interdum varius. Phasellus auctor bibendum dignissim. Pellentesque ut nulla arcu. Aenean id tellus augue. Nunc egestas scelerisque fringilla. Suspendisse potenti. Cras tempus diam lectus, a tempus eros maximus nec. Etiam facilisis erat turpis, in tincidunt leo condimentum sit amet. Fusce eleifend nisi sed enim pharetra molestie. Curabitur dictum, libero eget congue porta, est massa bibendum enim, sed semper arcu orci et arcu. Praesent in cursus turpis. Fusce condimentum dictum purus nec porta. Aenean a velit in augue convallis malesuada non a mauris. Donec auctor risus quis libero accumsan suscipit. Fusce nec diam dolor.

</p>

<p>

Proin aliquet, diam ultricies dignissim lacinia, libero lectus semper est, in lacinia eros augue ut risus. Maecenas sit amet metus finibus augue congue sagittis nec mattis leo. Quisque venenatis a erat nec gravida. Nulla sit amet pharetra risus. Nam sit amet nulla tellus. Pellentesque libero ex, ultrices eu tristique ut, mattis eget augue. Cras ac dolor convallis, convallis tortor nec, rutrum eros. Nunc nisl est, efficitur sit amet lorem at, egestas vulputate augue. Nunc sed ornare orci. Suspendisse potenti. Fusce risus dolor, pellentesque ac commodo a, pretium vel enim. Phasellus finibus aliquam purus sit amet ultricies. Vestibulum vitae libero scelerisque, maximus ex nec, pharetra orci. Nulla cursus volutpat nisl in dictum.

</p>

<p>

Phasellus quis lobortis orci. Nunc tincidunt lacus in sapien tristique, dignissim pretium eros varius. Praesent dapibus tincidunt lectus, id tristique arcu iaculis id. Mauris euismod placerat pellentesque. Proin ultricies porttitor porttitor. Suspendisse vehicula arcu a finibus dapibus. Nam non urna nulla. Vivamus aliquet, ipsum et convallis blandit, mauris lacus malesuada dolor, vitae vestibulum velit lectus vel lacus. Phasellus eget ipsum ultrices ipsum sodales interdum. Pellentesque in viverra nisi. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In aliquam convallis lorem vel pellentesque. Curabitur id ante ac risus condimentum accumsan. Aliquam venenatis facilisis fermentum.

</p>

<p>

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nulla in sapien ac dolor venenatis fermentum et porttitor lacus. Curabitur eu venenatis urna. Nam auctor orci id dui condimentum convallis. Nunc ut nisl rhoncus, elementum tortor sit amet, rutrum tellus. Morbi placerat neque id urna maximus interdum. Sed volutpat, nisl ut tempor bibendum, ipsum mi egestas sem, a tempus sem orci et mi. Nullam sit amet consectetur mi.

</p>

</section>

<section id="section2">

<h1>Section 2</h1>

<p>

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam mattis, sem ut vulputate tempus, leo urna fringilla dolor, sed mattis leo sem id ligula. Integer efficitur, nulla eu scelerisque aliquam, arcu erat maximus ante, nec maximus urna ligula vel purus. Cras a tortor ac nisl sollicitudin ullamcorper. Nullam pulvinar magna mi, eu volutpat metus pretium quis. In faucibus, leo in blandit ultricies, urna metus eleifend leo, et vestibulum dolor mi ut urna. Nulla eu neque nec justo cursus condimentum. Donec pellentesque tempus iaculis. Nunc augue justo, tincidunt in euismod vitae, feugiat sed eros. Integer molestie sapien id aliquam fermentum. Duis rutrum sed orci ac aliquet. Nulla volutpat posuere nibh ut ullamcorper. Suspendisse facilisis felis massa, vel egestas ex faucibus a. Nunc vestibulum volutpat eros eu laoreet.

</p>

<p>

Fusce sed pellentesque tellus, vitae accumsan elit. Vivamus porta in risus interdum varius. Phasellus auctor bibendum dignissim. Pellentesque ut nulla arcu. Aenean id tellus augue. Nunc egestas scelerisque fringilla. Suspendisse potenti. Cras tempus diam lectus, a tempus eros maximus nec. Etiam facilisis erat turpis, in tincidunt leo condimentum sit amet. Fusce eleifend nisi sed enim pharetra molestie. Curabitur dictum, libero eget congue porta, est massa bibendum enim, sed semper arcu orci et arcu. Praesent in cursus turpis. Fusce condimentum dictum purus nec porta. Aenean a velit in augue convallis malesuada non a mauris. Donec auctor risus quis libero accumsan suscipit. Fusce nec diam dolor.

</p>

<p>

Proin aliquet, diam ultricies dignissim lacinia, libero lectus semper est, in lacinia eros augue ut risus. Maecenas sit amet metus finibus augue congue sagittis nec mattis leo. Quisque venenatis a erat nec gravida. Nulla sit amet pharetra risus. Nam sit amet nulla tellus. Pellentesque libero ex, ultrices eu tristique ut, mattis eget augue. Cras ac dolor convallis, convallis tortor nec, rutrum eros. Nunc nisl est, efficitur sit amet lorem at, egestas vulputate augue. Nunc sed ornare orci. Suspendisse potenti. Fusce risus dolor, pellentesque ac commodo a, pretium vel enim. Phasellus finibus aliquam purus sit amet ultricies. Vestibulum vitae libero scelerisque, maximus ex nec, pharetra orci. Nulla cursus volutpat nisl in dictum.

</p>

<p>

Phasellus quis lobortis orci. Nunc tincidunt lacus in sapien tristique, dignissim pretium eros varius. Praesent dapibus tincidunt lectus, id tristique arcu iaculis id. Mauris euismod placerat pellentesque. Proin ultricies porttitor porttitor. Suspendisse vehicula arcu a finibus dapibus. Nam non urna nulla. Vivamus aliquet, ipsum et convallis blandit, mauris lacus malesuada dolor, vitae vestibulum velit lectus vel lacus. Phasellus eget ipsum ultrices ipsum sodales interdum. Pellentesque in viverra nisi. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In aliquam convallis lorem vel pellentesque. Curabitur id ante ac risus condimentum accumsan. Aliquam venenatis facilisis fermentum.

</p>

<p>

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nulla in sapien ac dolor venenatis fermentum et porttitor lacus. Curabitur eu venenatis urna. Nam auctor orci id dui condimentum convallis. Nunc ut nisl rhoncus, elementum tortor sit amet, rutrum tellus. Morbi placerat neque id urna maximus interdum. Sed volutpat, nisl ut tempor bibendum, ipsum mi egestas sem, a tempus sem orci et mi. Nullam sit amet consectetur mi.

</p>

</section>

<section id="section3">

<h1>Section 3</h1>

<p>

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam mattis, sem ut vulputate tempus, leo urna fringilla dolor, sed mattis leo sem id ligula. Integer efficitur, nulla eu scelerisque aliquam, arcu erat maximus ante, nec maximus urna ligula vel purus. Cras a tortor ac nisl sollicitudin ullamcorper. Nullam pulvinar magna mi, eu volutpat metus pretium quis. In faucibus, leo in blandit ultricies, urna metus eleifend leo, et vestibulum dolor mi ut urna. Nulla eu neque nec justo cursus condimentum. Donec pellentesque tempus iaculis. Nunc augue justo, tincidunt in euismod vitae, feugiat sed eros. Integer molestie sapien id aliquam fermentum. Duis rutrum sed orci ac aliquet. Nulla volutpat posuere nibh ut ullamcorper. Suspendisse facilisis felis massa, vel egestas ex faucibus a. Nunc vestibulum volutpat eros eu laoreet.

</p>

<p>

Fusce sed pellentesque tellus, vitae accumsan elit. Vivamus porta in risus interdum varius. Phasellus auctor bibendum dignissim. Pellentesque ut nulla arcu. Aenean id tellus augue. Nunc egestas scelerisque fringilla. Suspendisse potenti. Cras tempus diam lectus, a tempus eros maximus nec. Etiam facilisis erat turpis, in tincidunt leo condimentum sit amet. Fusce eleifend nisi sed enim pharetra molestie. Curabitur dictum, libero eget congue porta, est massa bibendum enim, sed semper arcu orci et arcu. Praesent in cursus turpis. Fusce condimentum dictum purus nec porta. Aenean a velit in augue convallis malesuada non a mauris. Donec auctor risus quis libero accumsan suscipit. Fusce nec diam dolor.

</p>

<p>

Proin aliquet, diam ultricies dignissim lacinia, libero lectus semper est, in lacinia eros augue ut risus. Maecenas sit amet metus finibus augue congue sagittis nec mattis leo. Quisque venenatis a erat nec gravida. Nulla sit amet pharetra risus. Nam sit amet nulla tellus. Pellentesque libero ex, ultrices eu tristique ut, mattis eget augue. Cras ac dolor convallis, convallis tortor nec, rutrum eros. Nunc nisl est, efficitur sit amet lorem at, egestas vulputate augue. Nunc sed ornare orci. Suspendisse potenti. Fusce risus dolor, pellentesque ac commodo a, pretium vel enim. Phasellus finibus aliquam purus sit amet ultricies. Vestibulum vitae libero scelerisque, maximus ex nec, pharetra orci. Nulla cursus volutpat nisl in dictum.

</p>

<p>

Phasellus quis lobortis orci. Nunc tincidunt lacus in sapien tristique, dignissim pretium eros varius. Praesent dapibus tincidunt lectus, id tristique arcu iaculis id. Mauris euismod placerat pellentesque. Proin ultricies porttitor porttitor. Suspendisse vehicula arcu a finibus dapibus. Nam non urna nulla. Vivamus aliquet, ipsum et convallis blandit, mauris lacus malesuada dolor, vitae vestibulum velit lectus vel lacus. Phasellus eget ipsum ultrices ipsum sodales interdum. Pellentesque in viverra nisi. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In aliquam convallis lorem vel pellentesque. Curabitur id ante ac risus condimentum accumsan. Aliquam venenatis facilisis fermentum.

</p>

<p>

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nulla in sapien ac dolor venenatis fermentum et porttitor lacus. Curabitur eu venenatis urna. Nam auctor orci id dui condimentum convallis. Nunc ut nisl rhoncus, elementum tortor sit amet, rutrum tellus. Morbi placerat neque id urna maximus interdum. Sed volutpat, nisl ut tempor bibendum, ipsum mi egestas sem, a tempus sem orci et mi. Nullam sit amet consectetur mi.

</p>

</section>

<section id="section4">

<h1>Section 4</h1>

<p>

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam mattis, sem ut vulputate tempus, leo urna fringilla dolor, sed mattis leo sem id ligula. Integer efficitur, nulla eu scelerisque aliquam, arcu erat maximus ante, nec maximus urna ligula vel purus. Cras a tortor ac nisl sollicitudin ullamcorper. Nullam pulvinar magna mi, eu volutpat metus pretium quis. In faucibus, leo in blandit ultricies, urna metus eleifend leo, et vestibulum dolor mi ut urna. Nulla eu neque nec justo cursus condimentum. Donec pellentesque tempus iaculis. Nunc augue justo, tincidunt in euismod vitae, feugiat sed eros. Integer molestie sapien id aliquam fermentum. Duis rutrum sed orci ac aliquet. Nulla volutpat posuere nibh ut ullamcorper. Suspendisse facilisis felis massa, vel egestas ex faucibus a. Nunc vestibulum volutpat eros eu laoreet.

</p>

<p>

Fusce sed pellentesque tellus, vitae accumsan elit. Vivamus porta in risus interdum varius. Phasellus auctor bibendum dignissim. Pellentesque ut nulla arcu. Aenean id tellus augue. Nunc egestas scelerisque fringilla. Suspendisse potenti. Cras tempus diam lectus, a tempus eros maximus nec. Etiam facilisis erat turpis, in tincidunt leo condimentum sit amet. Fusce eleifend nisi sed enim pharetra molestie. Curabitur dictum, libero eget congue porta, est massa bibendum enim, sed semper arcu orci et arcu. Praesent in cursus turpis. Fusce condimentum dictum purus nec porta. Aenean a velit in augue convallis malesuada non a mauris. Donec auctor risus quis libero accumsan suscipit. Fusce nec diam dolor.

</p>

<p>

Proin aliquet, diam ultricies dignissim lacinia, libero lectus semper est, in lacinia eros augue ut risus. Maecenas sit amet metus finibus augue congue sagittis nec mattis leo. Quisque venenatis a erat nec gravida. Nulla sit amet pharetra risus. Nam sit amet nulla tellus. Pellentesque libero ex, ultrices eu tristique ut, mattis eget augue. Cras ac dolor convallis, convallis tortor nec, rutrum eros. Nunc nisl est, efficitur sit amet lorem at, egestas vulputate augue. Nunc sed ornare orci. Suspendisse potenti. Fusce risus dolor, pellentesque ac commodo a, pretium vel enim. Phasellus finibus aliquam purus sit amet ultricies. Vestibulum vitae libero scelerisque, maximus ex nec, pharetra orci. Nulla cursus volutpat nisl in dictum.

</p>

<p>

Phasellus quis lobortis orci. Nunc tincidunt lacus in sapien tristique, dignissim pretium eros varius. Praesent dapibus tincidunt lectus, id tristique arcu iaculis id. Mauris euismod placerat pellentesque. Proin ultricies porttitor porttitor. Suspendisse vehicula arcu a finibus dapibus. Nam non urna nulla. Vivamus aliquet, ipsum et convallis blandit, mauris lacus malesuada dolor, vitae vestibulum velit lectus vel lacus. Phasellus eget ipsum ultrices ipsum sodales interdum. Pellentesque in viverra nisi. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In aliquam convallis lorem vel pellentesque. Curabitur id ante ac risus condimentum accumsan. Aliquam venenatis facilisis fermentum.

</p>

<p>

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nulla in sapien ac dolor venenatis fermentum et porttitor lacus. Curabitur eu venenatis urna. Nam auctor orci id dui condimentum convallis. Nunc ut nisl rhoncus, elementum tortor sit amet, rutrum tellus. Morbi placerat neque id urna maximus interdum. Sed volutpat, nisl ut tempor bibendum, ipsum mi egestas sem, a tempus sem orci et mi. Nullam sit amet consectetur mi.

</p>

</section>

</body>

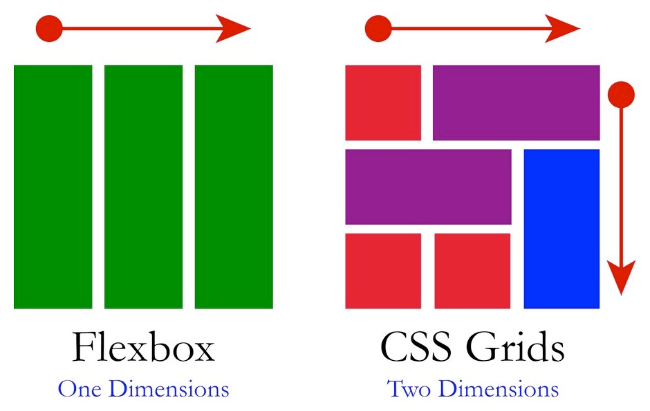
</html>

There are other methods that are more advanced that save you from loading the entire page. One technique under development to make scrolling easier and more accurate is a module called "[CSS Scroll Snap](https://www.w3.org/TR/css-scroll-snap-1/)".

## 5.5.4 New layout techniques (optional)

Web layout is always constrained by the limitations of CSS, but future trends will be able to make use of new tools, such as CSS Flexbox (officially: CSS Flexible Box Layout) and CSS Grid.

Since the summer of 2017, the major browsers support this module of CSS, which provides the beginnings of something professional designers have long been waiting for: [a design grid](https://en.wikipedia.org/wiki/Grid_(graphic_design)).



It's been a few years now that the CSS Flexbox module gives advanced control over alignment and spacing of elements on a line or in a stack. The Grid module now allows alignment in two dimensions: to elements on the same line and to elements in the same column.

Grid doesn't replace Flexbox. There is some flexibility when placing elements in one line that you lose when you add a second dimension. At the same time there are layouts that were difficult or impossible even with Flexbox and are now easy with Grid.  
  
The Grid module is only a first step. Level 1 has a number of limitations (just like Flexbox), due to issues around complexity and performance. For example, you can currently only align elements that have the same parent. That said, for laying out graphical user interfaces (GUIs), Flexbox and Grid work very well.

We briefly present them below, and give a few resources at the bottom of the page so that you can explore more.

### CSS Flexbox

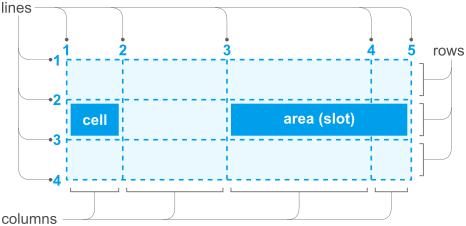
Since its introduction in 2012, CSS Flexbox has become widely used and its [browser support](https://caniuse.com/flexbox) is really good. Flexbox made a lot of complex layout tasks easier, like equidistant spacing between elements, top-to-bottom layouts or the [holy grail](https://en.wikipedia.org/wiki/Holy_Grail_(web_design)). Find out about Flexbox and how to use it in the last section of the [W3C HTML5 and CSS Fundamentals course](https://www.edx.org/course/html5-and-css-fundamentals)!

Despite the similarities in concept and syntax, Flexbox and Grid are not competing layout techniques. Grid arranges in two dimensions, while Flexbox lays out in one. There is synergy when using the two together.

### CSS Grid

CSS Grid is a CSS module that defines a two-dimensional grid-based layout system, optimized for user interface design. In the grid layout model, the children of an element (the ‘grid container’) can be positioned into arbitrary slots in a predefined flexible or fixed-size layout grid.

If that sounds a bit too abstract, here is another way of looking at it. The idea behind the Grid module is that you split the [box](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/jump_to_id/61920cda43ca49cca2fcf26e763bbe16) that makes up an element into many individual ‘slots’, arranged in a matrix, and separated from each other by (invisible) horizontal and vertical lines. You do that with a property called 'grid', which contains the desired number of rows and columns and/or their sizes. Each child element goes into a slot, so that they end up aligned as in a table. But you have full control over which slot they go into, you can change their order, they can span more than one row or column, and you can leave some slots empty.



The Grid module provides several different ways to define such a grid and to place the child elements. Too many, in fact, to present here.

Because they can refer to a previously defined grid of horizontal and vertical lines, the properties from the CSS Grid module provide more control over the alignment of elements than most other properties in CSS, such as the table-related properties or the 'float' and 'clear' properties, while also allowing elements to be displayed out of order. As such they are especially appreciated for (Web) applications with user interfaces that are made with HTML and CSS. The Grid module is not yet the ‘[design grid](https://en.wikipedia.org/wiki/Grid_(graphic_design))’ that typographers want for the layout of magazines and books, but it is a first step. (E.g., one obvious thing to do, applying grid properties to an HTML table, doesn't work, because the properties do not handle nested elements yet.) Even though this is only level 1 of the module, it is well worth trying out.

The properties from the Grid module have only been available in major browsers since mid 2017 (see the [status of browser support](https://caniuse.com/css-grid)). But the ideas behind the Grid module aren't new. From the start of CSS, there have been proposals to use CSS properties to define a template or matrix to guide the layout of elements, e.g.: [Frame-based layout](http://www.w3.org/TR/WD-layout), [Advanced Layout](http://www.w3.org/TR/2005/WD-css3-layout-20051215/) (later called [Template Layout](http://www.w3.org/TR/css-template-3/)), [Grid Style Sheets](https://github.com/gss) and [Constraint CSS](https://constraints.cs.washington.edu/web/ccss-uwtr.pdf). But only recently has technology become good enough to support some (not all!) of those ideas.

### A few resources

If you look at the CSS Grid module, you may notice that it has rather a large number of properties: 18. That is because it tries to allow different manners of writing style sheets. There are many shorthand properties and many alternative ways to define the same grid. In practice, most style sheet writers will select a set of just three or four properties that suits their way of working.

When considering the CSS Grid module, also look at the CSS Flexible Box module. It only provides for alignment of elements in a single row or column, but has some features that Grid doesn't have (and it has been around longer and works in older browsers). On the other hand, even for a single row or column, the Grid properties may turn out easier in some cases.

### CSS Layout

* [CSS Layout News](https://csslayout.news/) - A weekly collection of tutorials, news and information on all things CSS Layout, by [Rachel Andrew](https://rachelandrew.co.uk/).

### CSS grid resources

* [A complete guide to Grid](https://css-tricks.com/snippets/css/complete-guide-grid/) (CSS Tricks)
* [CSS Grid](https://www.w3.org/TR/css-grid-1/), the W3C specification
* [Grid by example](https://gridbyexample.com/): this site is a collection of examples, video and other information to help you learn CSS Grid Layout.

### CSS flexbox resources

* [A complete guide to Flexbox](https://css-tricks.com/snippets/css/a-guide-to-flexbox/) (CSS Tricks)
* [CSS Flexible Box Layout](https://www.w3.org/TR/css-flexbox/), the W3C specification
* [Flexbox froggy](https://flexboxfroggy.com/) (game to practice CSS flexbox code)

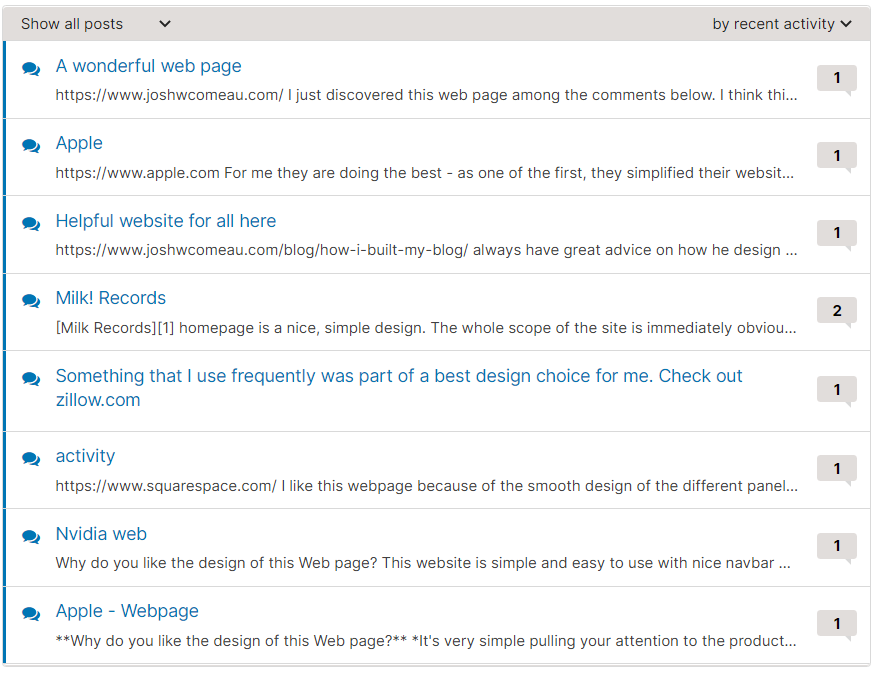
## 5.5.5 CSS Grid layout (optional)

Self-gratification . ie, busting their arms patting themselves on the back. Fucking MS.

## 5.5.6 Activity - Best design

For this activity, go out and find what you think is the best-designed Web page! Post a link to it in the discussion along with the answers to the following questions:

1. Why do you like the design of this Web page?
2. What is one aspect of this Web page you would like to use in your own design?
3. Is this Web page accessible?



#### A wonderful web page

discussion posted about 20 hours ago by **[AllaPetr](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/discussion/forum/users/2254953)**

<https://www.joshwcomeau.com/> I just discovered this web page among the comments below. I think this is one of the most attractive and balanced websites I have seen. I like the choice of colors because it is both simple and warm. Sometimes web pages can feel cold and not welcoming. This one's doesn't. I also love the attention to details shown by the use of animations in the logo. It adds the extra little something that makes it very appealing. The one thing I would use in my own web page is the animated logo combined with the neat and minimalist design of the page. Yes, I think it is accessible.

#### Apple

discussion posted a day ago by **[LauraLucie](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/discussion/forum/users/42009473)**

[https://www.apple.com](https://www.apple.com/) For me they are doing the best - as one of the first, they simplified their website, as much as their products. I would love to have my website as simply and clean designed

#### Helpful website for all here

discussion posted 23 days ago by **[therockitscientist](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/discussion/forum/users/9451919)**

<https://www.joshwcomeau.com/blog/how-i-built-my-blog/>

always have great advice on how he design his own site.

#### Milk! Records

discussion posted 2 months ago by [**blither**](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/discussion/forum/users/6016861)

[Milk Records](https://milk.milkrecords.com.au/) homepage is a nice, simple design. The whole scope of the site is immediately obvious from the navigation bar, and links to social media are artfully displayed. Each artist has a photo to click on to find out more, and are presented in a nice spaced out grid. However, I looked at the HTML and there are no alt texts for the artist photos. I opened up the Windows Narrator to see if the site could be screen read and it only read out the text in the header.

#### Something that I use frequently was part of a best design choice for me. Check out zillow.com

discussion posted about a month ago by [**MattN77**](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/discussion/forum/users/39571304)

**Site Choosen** -> www.zillow.com

* Zillow is a residential real estate information provider that covers the United States, both for sale and for rent. For buyers and sellers there is information such as area statistics, pricing, finance snapshots to purchase, school ratings, walkable ratings, data about past sales, mapping, street views, you name it, there is a lot of useful stuff here. They also provide tools for prospective owners that need help with renting their property, short of providing property management services. They provide what one would need to list, screen tenants, set up and sign a lease, as well as many tips on getting important things done.

**1. Why do you like the design of this Web page?**

* I chose this as my favorite site based on the fact I use it frequently. The overall design of the site is easy to use, and presents information that I am looking for in a well laid out format, as Kasey Champion put it, information that is easy to consume or take in. There are many impressive facets to this site that are practical, relevant, intuitive, and all are managed well.

**2. What is one aspect of this Web page you would like to use in your own design?**

* The ease and intuitive use are number one for me, and it is facilitated by a simple uncluttered well organized layout and menu's that make finding information easy, and quickly understand the presentation of it. Another important aspect of this site I noticed behind the scenes are flexbox is used extensively, not grid. Zillow seems to work flawlessly on any device.

**3. Is this Web page accessible?**

* Yes, this site seemed to address many accessibility needs or points.
* There are many keyboard focusable points so the keyboard can be used without a mouse, color contrast is rated in many spots that are good, there is enough white space so information is not cluttered. The site seems to be organized very well, so it is easy to navigate logically. The screen reader I used seemed to work well with this site. The site seemed to adjust easily to screen zoom. Images also have alt text for screen readers.

### activity

discussion posted about a month ago by [**Naenae3**](https://courses.edx.org/courses/course-v1:W3Cx+CSS.0x+3T2020/discussion/forum/users/24574149)

<https://www.squarespace.com/>

I like this webpage because of the smooth design of the different panels. I like the way that the videos/animations are activated by a scroll. I think that I would use that in my own websites. Although they do have a language declaration, good contrast in colors and labled buttons, they do not have alt tags for many of their images, which could detract from the accessibility. It makes sense that this would be the best website I could find, because their business is designing websites!

## Continue your studies



Hopefully at this point you feel comfortable with all the basics of CSS, but, as mentioned throughout this course, there is still a lot more to learn. Here are some courses we recommend you check out next as part of the [W3C "Front-End Web Developer" Professional Certificate](https://www.edx.org/professional-certificate/w3cx-front-end-web-developer) program:

* HTML5 & CSS Fundamentals
* HTML5 Coding Essentials and Best Practices
* HTML5 Apps and Games
* JavaScript Introduction