Chapter 1: Getting Started in Web Design

The Web has been around for more than **25** years
Info Architect-organizes content logically (site diagrams)
Content Strategist - ensures content supports brand identity and goals
UX Designer - makes the site easier to use by concerning themselves with visual design, UI, content quality, and site performance. Plan users "flow" through the site
Graphic Designer - creates the "look and feel" of the site including logos, graphics, type, color, layout, etc
Frontend Dev - creates code that directly relates to the browser like HTML, CSS, and JavaScript
Backend Dev - creates code and manages aspects of the server including the apps and databases that run on it
Full Stack Dev/Unicorn - rare breed of web designer who can manage all tasks design and development wise
Product Manager - guides the design and development in a way that meets business goals. They deal with the overall strategy for the site from a marketing perspective and understand the processes involved in site creation.
Project Manager - coordinates everyone working on the site to make things delivered on time and on budget
SEO Specialist - makes sure the site can be easily found by search engines by fixing structure/code to improve it
Multimedia Producer - artists or technicians in the field of sound, video, animation, and interactivity that can add cool elements to a site
User research/testing results - provides insight into how the site can solve problems or how it will be used
Wireframe diagrams - gives an idea of where elements will be put on the screen
Site diagrams - shows visually how the site is structured and how individual pages relate to one another
Storyboards and user flow charts - includes a script and "scenes" describing how a task will be accomplished.
Graphic designers **should** learn to code because having some coding knowledge helps with communication, improve workflow, and enable better designs.
Frontend Technology - JavaScript, CSS, HTML, Ajax
The **World Wide Web Consortium** or **W3C** oversees the development of web technologies. The group was founded in 1994 by **Tim Berners-Lee**
Server software- Apache, Microsoft IIS
Web App Langs - PHP, Python, .NET, Node.js
Database software - MySQL, Oracle
Equipment for Web Dev - solid up to date computer, large monitor, second computer (generally a different platform), mobile devices, scanner/camera
Coding Tool - Brackets, VS Code, Notepad++, BBEdit
UI and Layout Tools - Sketch
Web Graphic Creation Tools - Photoshop, Gimp
Web Browsers - Chrome, Edge, Safari, Firefox
File Management and Transfer Tools - Filezilla
Terminal Apps - Terminal (built into macOS), PuTTY

Chapter 2: How The Web Works

Internet - international network of connected computers, devices connected to it share info using standardized methods called protocols, no company owns it (it's a cooperative effort governed by a system of standards and rules).
POP3/IMAP/SMTP - used for transferring and storing email on a server and client
FTP - used to transfer files from machine to machine
SSH - provides a secure shell for remote access
HTTP - powers the web allowing for documents to be linked together and served to requesting clients
Web - popularity boosted in 1992 when the first graphical browser (NCSA Mosaic) was introduced, it is a subset of the internet, it was born in a particle physics lab in Geneva, Switzerland 1989, it uses HTTP (HyperText Transfer Protocol) as it's main protocol for transferring data.
Computers that make up the internet and provide documents upon request are referred to as **servers**. More accurately, the name applies to the **software** (not the computer itself). But it's common to use the same term for both. There are many server software options out there, but the two most popular are **Apache** (open source software) and Microsoft **Internet Information Services** (ISS).
IP Address - a unique series of numbers assigned to every individual computer/device attached to the internet
Domain Name System (DNS) - a system developed to allow us to refer to internet-connected computers and devices by name
Domain Name - a more accessible identifier for humans to read and remember (ex: rit.edu)
DNS Server - A separate server that works like a phone book for looking up computer names and numbers.
More than half of all web traffic comes from **mobile browsers**.
Parts of a url http://www.example.com/2018/samples/first.html
Protocol - http://
Name of Site - www.example.com
Absolute Path - /2018/samples/first.html
Host Name - www
Domain Name - example.com
Directory Path - /2018/samples/
Document Name - first.html
Rendering engine - the program responsible for converting HTML and CSS into what you see on your screen
The **most challenging aspect of design and development** is pages may look and perform differently from browser to browser due to varying support for web technologies, varying device capabilities, and user preferences.
The **s in https** means secure.
index.html - the default file that allows you to create URLs that leaf to a directory name rather than to a specific file
An HTML page contains plus special **tags** (indicated with < and >) that describe each element on the page. The process of adding tags to a test document is known as "**marking up**" the document. Web pages use HyperText Markup Language (**HTML**) to define different pieces of content.
ViewSource - a function of most browsers that allow you to see code that was originally downloaded from the server (HTML, CSS, or JavaScript)
img - HTML tag used to display images
HTML element - <h1>Hello</h1>
Opening Tag - <strong>
Closing Tag - </strong>
Empty Element/Tag - <hr>
Style instructions are written according to the rules of **Cascading Style Sheets (CSS)**
HTTP Status Codes - 200 (OK), 301 (Moved Permanently), 404 (Not Found), 500 (Internal Server Error)
HTML request order

1. Type in a URL or click a link in the browser
2. The browser sends and HTTP request
3. The server looks for or assembles the requested file and issues a response header. It sends the file along if it's found
4. The browser parses the document and, if it has images, style sheets, or scripts, contacts the server again for each resource
5. The browser assembles all of the images, styles and scripts into a web page for your viewing and pleasure

Chapter 3: Some Big Concepts You Need to Know

As a web designer we don't know what fonts will be installed, which browser the user will be using, whether the user is on desktop or mobile, how large the browser window will be, whether the pages are being read by a screen reader, whether or not JavaScript is enabled, how fast the internet connection is.
Users might visit our pages on multiple devices so we must consider devices being all different dimensions including mobile and desktop, more people access the web via mobile devices, the way you see your design working on a desktop is not how it will be experienced by everyone.
Web standards should be followed because it ensures that your site is consistent on about 99% of browsers in current use, helps make your content forward-compatible as web technologies and browser capabilities evolve, and your client will appreciate that you create "standard-complaint" sites.
Graceful Degradation - design a fully-enhanced experience first, and then create fall-backs for non-supporting browsers
Progressive Enhancement - start with a baseline experience that makes content or core functionality available even on simple browser and assistive devices. Then, layer on more advanced features for browsers that can handle it.
Authoring strategy - Add these elements in logical order and in meaningful ways so that the simplest browsers will present things clearly and completely.
Styling strategy - create simple versions of these to handle older browsers and then add cutting-edge 'selectors' to deliver more specialized ones. (Older browsers will just ignore ones they don't understand).
Scripting strategy - add these in such a way that the basic functionality of your site (like links and submitting forms) still works even when the feature is potentially disabled.
Responsive Web Design (RDW) - is helpful with matters of layout, but it is not the solution to all mobile web design challenges (sometimes requires the server to detect the device and send something completely different back), phrase was coined in 2010 by Ethan Marcotte, the point is to serve up one single page (at the same URL) to all users but have different style sheet rules be applied based on the devices screen size.
Web Accessibility - more than just pointers and fingertips are used to access websites (ex: voice commands, screen readers, magnifiers joysticks, foot pedals, etc), by making your site accessible they are much easier for search engines to more effectively index, there are four broad categories of disability to consider (vision, mobility, auditory, cognitive), the W3C started several efforts to make the web mor reusable for everyone including the WAI and WAI-ARIA.
You can **improve the site performance** by avoiding unnecessary nesting of elements, removing extra character spaces/line returns in HTML files, and reducing the number of times that a browser makes requests of the server.

Chapter 4: Creating a Simple Page

The **best way to learn** HTML is by coding it by hand and then viewing it in the browser.
When **setting up your text editor** make sure you are editing a "plain text" file and have visible file extensions.
Naming conventions - always end HTML filenames with .htm or .html and images with .jpg or .gif or .png or .svg, consistently using lowercase makes your files more compatible across different servers, no spaces in file names a hyphen or underscore is acceptable.
Browsers ignore - extra spaces between items in HTML code (beyond just one space), HTML comments (surrounded by <!-- and → tags), returns and tabs (they are displayed as a single space)
Forward slash / - used in tags or urls
Backslash \ - used in PC file paths or as an escape character in a text string
An HTML **element** consists of both content and the HTML markup that surrounds it. A typical **container** element consists of an element name appearing in an **opening** (or start) tag and a **closing** (or end) tag where the same element is preceded by a slash. Elements that do not have content, are usually referred to as **empty** (or standalone) elements.
Basic minimal HTML and elements

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset=”utf-8”>
        <title>Title</title>
    </head>
    <body>
    Page content goes here.
    </body>
</html>
```

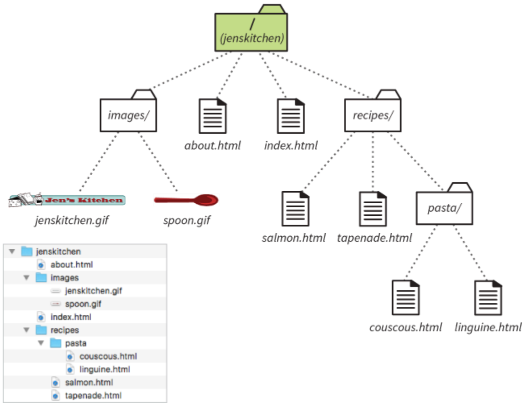
The document type declaration
A root element (specifically the "html" element)
A head element (one of two main parts of the document)
A body element (one of two main parts of the document) which contains everything we want to show up in the browser window
A character encoding specification
The mandatory 'title' element which, according to specification, should contain a descriptive document title
The **purpose of HTML** is to add meaning and structure to the content. It is **not** intended to describe how the content should look (its presentation).
Markup that provides a meaningful description of the content at hand is called **semantic** markup. In addition markup gives the document **structure** which can be thought of as an outline. This outline has a technical name: The **Document Object Model** (or the **DOM** for short). This provides the foundation upon which we'll add presentation instructions with style sheets (CSS) and behaviors with **JavaScript**.
Block Elements - they start on new lines, stack on top of each other as part of the regular document flow, are created by heading tags, paragraph tags and many others.
Inline Elements - they do not start new lines they just go with the flow, <em> for emphasis is an example of an inline tag
User agent style sheets - built in feature that all browsers have that make them display elements in a default style, this is what makes an h1 tag be big and bold
You **should always** start with h1 and then continue your way down.
Image Tags - an example of an empty element, image elements and other tags like 'br' and 'hr' are sometimes written as <br /> <hr /> because of the stricter rules of XHTML that required all elements to be terminated, needs attributes to clarify and modify how it works, needs the src attribute which provided the name of the img file that should be inserted into the doc, the alt attribute provides text that should be displayed if the image is not available, both src and alt are required.
Validation - some browsers and code editors have validators built-in, valid docs are more consistent on a variety of browsers, display more quickly, and are more accessible, validators generally require that you provide a DOCTYPE declaration so it knows which version of HTML to validate, documents that are error free are said to be valid.
Correct markup examples:
<p>First line<br>Second line</p>
<a href = "filename.html">link to other page</a>
<strong>correct</strong>

Chapter 5: Marking Up Text

As we mark up our text it is not important to get it looking right first, instead it is important to choose our elements **semantically** (based on it's meaning rather than appearance).
Different Markup Tags:
Paragraphs - <p>...</p>
Headings (1 is the largest 6 is the smallest) - <h1>...</h1>
Horizontal rule (a break that inserts a horizontal line) - <hr>
Unordered list - <ul>...</ul>
Ordered list - <ol>...</ol>
List item - <li>...</li>
Description list - <dl>...</dl>
Term/label for description list - <dt>...</dt>
Description for description list - <dd>...</dd>
Long quotes - <blockquote>...</blockquote>
Preformatted text - <pre>...</pre>
Related image or resource - <figure>...</figure>
If no semantic elements accurately describe your content, HTML provides two **generic** elements that can be customized to describe your content. There is a **div** which is a block-level element and a **span** which is an inline element. In order to refer to these elements more exactly in style sheets and in JavaScript, there are two important attributes that should be added to them (and can be added to many other tags for that matter). The first is the **id** attribute which must have as its value a unique identifier used only once in the document. The other is the class attribute which provides one or more classification names and can be applied to multiple elements on the page.
ARIA - a standardized set of attributes for making pages easier to navigate and interactive features easier to use.
Escaping characters - you can refer to it by its numeric or named character entity reference, all character references begin with an & (ampersand) and end with a ; (semicolon), &copy; adds a copyright symbol to the page.

Chapter 6: Adding Links

There is one HTML element that makes links work: the <a> tag also called the **anchor** tag. It can be simply wrapped around a selection of **text** to make it clickable or (quite commonly) an **img** element to make it into a clickable button or thumbnail. In order to tell the browser where to go when you click or tap, the **href** attribute must be specified. Between quotes, in the value of this attribute, you must specify a URL in one of two ways: Either an **absolute** URL which provided the full URL for the document (including the protocol, domain name, and pathname) or a **relative** URL which describe the location of the file as it relates to current document.



jenskitchen
about.html
images
jenskitchen.gif
spoon.gif
index.html
recipes
pasta
couscous.html
linguine.html
salmon.html
tapenade.html

Relative link from one site to another in the same folder
Ex: tapenade.html to salmon.html Answer: salmon.html
Relative link from one site to another in a folder below
Ex: tapenade.html to couscous.html Answer: pasta/couscous.html
Relative link from one site to another in a folder above
Ex: tapenade.html to about.html Answer: ../../about.html
Site root relative - a path name that allows you to make a link that starts from the top-level of your directory structure, it begins with a forward slash.
You can link to a specific point in a page by linking to a document **fragment**. It's a two part process, first you need to "plant a flag" in the document at the destination. You do this by adding the id attribute to the target element and giving it a unique identifier. Then you make the link by adding an **octothorpe** symbol (#) followed by the name you chose for the destination to the URL in your anchor tag's href attribute.

Chapter 7: Adding Images

Images can appear in web pages in two ways: Either embedded in the inline **content** (intermingled with the rest of the HTML) or as **background** images (which gets covered in a later chapter). If the image is part of the editorial content, it should be placed right in the flow of the HTML document, however if it's purely decorative then it should be added through **Cascading Style Sheets**. We typically embed images using the img element. There are additional methods including techniques for including SVG (Scalable Vector Graphics) images and methods to make images work well on mobile devices and screens of many sizes. This is referred to as making the images "**responsive**', serving images that are appropriate for their browsing environments.
<img src="pic.png" alt="description of pic">can include a width and height
Image formats that work on the web: WebP, JPEG, GIF, PNG, SVG
Disk Cache - a browser feature that stores a downloaded image so that if it needs to display the image again it can just pull up a local copy of the image without making a new server request.
Alternative Text - it is required in an img element (at least, if you want to validate, it is), it is used by screen reader software so that a visually challenged user can understand what the image is for, it is useful to search engines since a search engine is not able to easily understand what content is in an image file, it will usually display for a user when they either have images 'turned off' or if there was a network problem preventing the image from downloading, you should avoid using 'image of' or 'graphic of' in the alt attribute's text value.
There **was** a time when web developers would always try to include height and width attributes in their img tags (resulting in faster page display), but today with modern web design, they should be omitted if you're defining the image size through CSS instead or if you're using a responsive image technique.
SVG - scalable vector graphics, they have nearly ubiquitous browser support, they can be resized without loss of quality, they can be animated.

Chapter 23: Web Image Basics

JPEG - a widely popular first best for photographs, works especially well with gradients and blended colors, has a lossy compression format meaning that some of the image info is thrown out to reduce file size.
PNG - can have transparency that is either multiple-level (alpha) or simple on/off (binary), has an 8-bit and 24-bit version that are especially relevant to web production, can have embedded color profile info as well as embedded text stored right in the image file.
GIF - is widely supported and can display up to 256 colors, can be animated, it was the first format supported by web browsers for a while
WebP - The newest of these formats and has a sparse browser support so far, an open source image format created by google.
Favicons - the favicon for a site or page shows up to the top left of the page title in the browser tab and often in bookmark lists, the standard favicon file is in ICO (Windows Icon) format, the simplest solution is to just name the ICO file favicon.ico and put it in the root directory of your site, favicons in PNG format can also be linked to from your individual pages through the use of a <link> element

Chapter 11: Introducing Cascading Style Sheets

CSS is the standard for defining the **presentation** of HTML (and other) documents, but is a separate language with its own syntax.
Advantages of CSS - you can achieve precise type and layout controls, you can change the appearance of an entire site just by editing one style sheet making for less work when you want to make big changes, it separates presentation from the rest of the content markup (which can help to make your site more accessible).
The 1-2-3 step sequence to make style sheets work

1. Mark up the document in HTML
2. Write style rules for how you'd like elements to look
3. Attach the style rules to the document

Parts of CSS strong { color:red; font-style: italic; }
Style rule - strong { color:red; font-style: italic; }
Selector - strong
Declaration - color: red;
Declaration Block - { color:red; font-style: italic; }
Property - color
Value - red
There are three main ways to apply style information to an HTML document. One method is to put the collection of style rules into a separate text document with a .css extension. You can then link to (or import) these rules into multiple pages. This is the most powerful and preferred method and is called an **external** style sheet. Another way to apply styles is to put the list of style rules directly into the HTML of the page by putting it between <style> tags in the <head> section of your document. This is called an **embedded** style sheet. And lastly, you can apply styles directly to individual elements by writing the rules into the style **attribute** of the tag. This is called adding an **inline** style. It should be avoided unless absolutely necessary because you don't want to intersperse presentation information into the **structural** markup. Oh, one other thing. You can add **comments** into your CSS by wrapping them with /* and */. You can wrap entire blocks of CSS in these as well which is very helpful for temporarily turning styles on and off while testing.
CSS inheritance - it is the term given to when styled HTML elements are "passed down" certain properties to the elements they contain, document structure plays a role in understanding the way that properties pass down, the elements can be represented by an upside down tree structure, two elements with the same parent are called siblings, properties related to the

styling of the tet are inherited from the ancestor elements while properties that affect the boxed area around an element are not inherited.

It is possible for the user of the website to apply their own "**user style sheet**" to the pages that they view.

Certain types of CSS selectors are considered more "specific" than others giving them a greater weight when determining which of several conflicting style rules will be applied.

Group Selector - if multiple elements have the same trait they can be written out using a grouped selector

Ex: h1 {color: red;} p {color: red} blockquote {color: red}

Selector - h1, p, blockquote

Whole thing written out - h1, p, blockquote {color: red;}

CSS measurements

px - a pixel (defined in CSS3 to be 1/96 of an inch)

vh - the viewport height, equal to 1/100 of the current browser window height

rem - a root em (equal to the em size of the root element)

em - a unit of measurement equal to the size of the current font

ex - a unit of measure equal to the height of a lowercase 'x' in the current font

vw - equal to 1/100 of the current viewport width

pt - a point, 1/72 of an inch commonly used in print design

Chapter 12: Formatting Text

Font Family Property - all font names (unless they are generic) must be capitalized, if a font name has a space in its name it must be in quotation marks, until the widespread adoption of Web Fonts around 2010 you were limited to the fonts that were installed on your hard drive.

Web Fonts - you can host your own special web font and store it on your server (first need to find a font, save it in multiple formats, upload it write a @font-face rule in your style sheet), a font embedding service such as Google Web Fonts can do the heavy lifting for you and all you need to do is select the font you want and copy and paste the relevant code into your own document.

Generic font types

Serif - fonts with decorative slab-like appendages on the end of certain letter strokes

Sans-serif - fonts with straight letter strokes that don't have slab-like appendages

Monospace - fonts in which all characters take up the same amount of space

Cursive - fonts that emulate a script or handwritten appearance

Fantasy - purely decorative fonts that would be appropriate for headlines and other display types but not for large block of small text

Since you never have absolute control over which font your users will see, **specifying** fonts for the web is more like merely **suggesting** them.

Despite many options that you have for specifying text size, the preferred values for font-size in contemporary web design are the relative length units **em** and **rem**, as well as **percentage** values

Font properties:

font-weight - make the font bold

font-style - make the font italicized

font-variant - make the font appear in small capital letters

font-stretch - make the font pick a condensed version if it has one available

Correct ways to format a style rule containing the font property shortcut

```
h2 { font: 2em serif; }
h3 { font: bold small-caps 1.4em/1.6em Arial, sans-serif; }
h1 { font: bold 1.5em "Arial Black", Verdana, sans-serif; }
```

The **color property** can change the text color, but it is not strictly a text-related property. It's more accurately described as changing the foreground color of the element (as opposed to the background) which can include an item's border, for example.

Selector Types:

Element selector - p

Grouped selectors - p, ul, blockquote

Descendant selector - p em

ID selector - #professor

Child selector - li>a

Next-sibling selector - h1+p

Sequential-sibling selector - h1~h2

Class selector - .group

Universal selector - *

Most to least specific - inline styles with the style attribute, ID selectors, Class selectors, Individual element selectors

Text Properties:

line-height - changes the space between baselines of rows of text

text-indent - changes the offset (to the right or the left) of the first line of a block text

text-align - changes the position of text within a block to the left, center, right, etc

text-decoration - can add an underline (or overline) to a bit of text

word-spacing - cause a certain amount of space to be added between words in a sentence for example

text-shadow - allows for a drop-shadow to be added behind a segment of text

Chapter 13: Colors and Backgrounds

Ways to specify colors in a style sheet:

White - rgb(255,255,255) or #FFF

Black - rgb(0,0,0) or #000000

Gray - rgb(128,128,128) or silver

Blue - #0000FF or rgb(0%,0%,100%)

In **hexadecimal** the colors ordered from left to right are red, green, blue

The **a in RGBa** stands for alpha

A background color fills the canvas behind the element that includes the content area, and any padding (extra space) added around the content, extending behind the border out to its outer edge.

Pseudo-class selectors

:link - applies a style to unvisited links

:visited - applies a style top a previously visited links

:focus - applies a style when the element is selected and ready for input

:hover - applies a style when the mouse pointer is over the element

:active - applies a style when the element is in the process of being clicked or touched

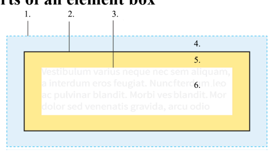
In order for pseudo-class selectors to be properly applies to your links they must be in a certain order **(:link, :visited, :focus, :hover, :active**) otherwise LoVe For Hairy Animals

Correct ways to link an external style sheet

```
<head>
    <title>Title,etc</title>
    <link rel = "stylesheet" href = "external.css">
</head>
<style>
    @import url("external.css")
     p.otherstyles {color: red };
</style>
```

Chapter 14: Thinking Inside the Box

Parts of an element box


1.
2.
3.
4.
5.
6.
Vestibulum varius neque nec sem aliquam, a interdum eros feugiat. Nuncfterdum leo ac pulvinar blandit. Morbi ves blandit. Mor dolor sed venenatis gravida, arcu odio

Outer edge: 1

Border: 2

Inner edge: 3

Margin area: 4

Padding area: 5

Content area: 6

When specifying the dimensions of a box element, you override the browser default sizes. By default the box will be as **wide** as the browser window or other containing element and as **tall** as necessary to fit the content. There are actually two ways to think about the size of an element. And you can choose which one your element will use by changing the **box-sizing** property. There are two property values: **content-box** which applies the height and width to the content area only such that padding and borders will be added on to the determined size. The other alternative is **border-box** which will make the width of the visible element box be ONLY as wide as the measurement of width you specify. No wider.

If you set the height property of a text container to a specific value but there is no space to hold all the text, the overflow will be handled by whatever method your **overflow** property is set to.

Shorthand padding property:

If you have **4** values listed they will apply in clockwise order around the element starting with the top

If you have **3** values the top will be the first, then the left and right will be the second, and the third will be the bottom

If you have **1** value it will apply to all sides of the box

When specifying borders, you can set the style, width, and color independently for all four sides. However, if you use the **border** shorthand, you can set each of those three properties, but they apply to all four sides together.

Features of margins: you can center an element in its container horizontally by setting the left and right margins to auto, you can use negative margins to move the content, padding, and border in the opposite direction than it normally would have, items with margins that bump up against each other top to bottom will "collapse" their margins and only use the larger of the two distances.

Correct Display declarations

ul.navigation li a { display: block; } -- Turns a link (within a bulleted list) into a rectangular area that can have a height and width applied.

div.warning { display: none; } -- Hides a block from view on the page (removing it from the flow entirely). You'd probably use JavaScript or some other CSS to display it later when the situation is right.

ul.navigation li { display: inline; } -- Turns a normal bulleted list into a horizontal navigation bar. (but it would still need some extra styling to look good)

Box-shadow property in order if you use them all: horizontal offset, vertical offset, blur distance, spread distance, color

Chapter 15: Floating and Positioning

In the normal flow, **text** elements are laid out from top to bottom in the order in which they appear in the source, and from left to right. **Block** elements **stack** up on top of one another and fill the available **width** of the browser window or other containing element.**Inline** elements and text **characters** line up next to one another to fill the block elements. When the window or containing element resizes, the block elements **expand** or **contract** to the new width, and the inline content **reflows** to fit it. Objects in the normal flow affect the layout of the objects around them... Elements don't **overlap** or bunch up. They make room for one another.

Float property - moves an element as far as possible to the left or right, allowing the following content to wrap around it, you should always provide a width for floated elements, floated inline elements behave as block elements, margins on floated elements don't collapse.

You should clear the element that you want to **start** below the float.

Flexbox and Grid are gaining browser support but you should still provide a universally supported fallback, like one built with floats.

If you float all elements in a container element, the containing element will collapse so you should use the "**clearfix**" technique in which the :after pseudo-element is used to insert a character after the container, and clear it on both sides.

Positioning Types:

Static - the normal positioning scheme in which elements stay where they'd normally occur in the document flow

Relative - moves the element from its original position to a new location, leaving the space where it used to be located as an empty area.

Absolute - move the element from its original location to a spot that is related to the viewport or containing element. The space it previously occupied is closed up and has no influence on other elements.

Fixed-the element becomes positioned in one spot (even if the doc scrolls) relative to the viewport rather than another element in the doc.

Sticky - a combination of other positioning behaviors that allow the elements to scroll into a specific position relative to the viewport, at which point it remains in a fixed place.

The 4 properties that are used to put a positioned element into the proper location is **top, right, bottom, left**

We've seen how relative positioning works. **Absolute** positioning works a bit differently and is more flexible for placing items on the page. The location of elements placed this way can be relative to the **viewport** but not always. What really happens is that elements are placed relative to the nearest "containing block". This containing block is sometimes called the positioning **context**. Which basically boils down to this: If the positioned element is not contained in another positioned element, it will be placed relative to the html element. However, if it has an ancestor (that has its position set to relative, absolute, or fixed), then the element will be positioned relative to the edges of that element instead.

The **stacking order** is determined by the z-index property.

Chapter 16a: CSS Layout and Flexbox

Flexbox - gives you control over the layout of items along a single axis, can easily turn blocks or other elements into many thing (ex: menu bars, product listing, galleries), items in a flexbox layout can expand and shrink or wrap onto multiple lines, you can change the order of the items in a flexbox layout without the need to change the HTML source.

Grid - gives you control over the layout of items in two dimensions along both axes

display: flex - a CSS declaration used to turn on flexbox mode for an element

When you create a **flex container** all of its direct children become **flex items.**

By default the flex items in a flex container automatically **become lined up in a row** (along the main axis)

The **flex-direction** by default is row but can be switched to column when the page is encoded into a vertically oriented language.

When the flex-direction is set to column the main axis is vertical and the cross axis is horizontal.

flex-wrap is important because without setting it, all the items in your flex container would (by default) attempt to squish into a single line.

Correct ways to write a declaration of the flex-flow shorthand property:

```
flex-flow: row-reverse wrap;
flex-flow: column wrap-reverse;
flex-flow: row-row row-reverse;
```

justify-content - effects how the left-over space should be distributed around or between items (that are inflexibly sized) on that axis

stretch is the default value for how items should be aligned on the cross axis.

So, the "flex" in flexbox is mostly about how the individual items can "flex" or resize to fix the available space and there are a variety of ways it can be specified. There is an important shorthand property simply called **flex** that has these three values listed in order: **flex-grow flex-shrink flex-basis**

Flex Declarations:

flex: 1 0 200px; - specifies that items in the main axis start at a specific minimum width and are allowed to grow, but not shrink

flex: 0 1 auto; (or flex:initial;) - specifies that items in the main axis start spread out according to width/height

flex: 1 1 auto; (or flex:auto;) - items on the main axis are fully flexible and can grow or shrink as needed

flex: 0 0 auto; (or flex:none;) - items on the main axis are completely inflexible, not growing or shrinking but set to the width and height properties.

Flexbox order - items with different order values are arranged from the lowest value to highest, in screen reader software the items will still be read in the order that the items are listed in the HTML source not the way CSS reorders it, the value of the order property can be a positive or negative number.

The flexbox specification went through a lot of changes in its development and along the way several big releases of the flexbox spec came out which use a different **display** property value to enable them as well as a bunch of different vendor prefixes.

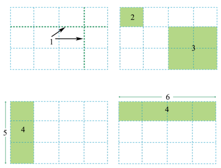
Chapter 16b: CSS Layout with Grid

Grid - after 25 years web designs and developers finally have a CSS module for using an underlying grid to achieve true page layout, CSS Grid can allow for sophisticated layout including typography and whitespace, Grid Layout is one of the more complex specs within CSS

Order for using CSS Grid Layout:

1. Use the **display** property to turn an element into a grid container
2. Set up the columns and rows for the grid
3. Assign each grid item to an area on the grid

Parts of a CSS Grid



Grid lines - 1

Grid cell - 2

Grid area - 3

Grid tracks - 4

Block axis (for languages written horizontally) - 5

Inline axis - 6

display: grid - is used for making a grid container

Since they're hard to figure out in your head, a quick **sketch** of how you'd like your final grid to look is a good first step. You should be able to split the drawing up into multiple **row** tracks and multiple **column** tracks. Some of your content areas will **span** over more than one cell. Next you'll want to go to your CSS and specify heights (for each row) and widths (for each column) by using the template properties, **grid-template-rows** and **grid-template-columns** which get applied to the **container** element.

fr - a unit specific to CSS grid for specifying widths of tracks and columns that depend upon available space.

minmax(10em, 30em) - is an in a grid-template-column property that specifies a certain column should be as much as 30em wide but no less than 10em wide

grid-template-columns: 30px repeat(2, 1fr 10px 2fr 10px) 30px - is a repeat pattern for specifying track sizes and is equivalent to **grid-template-columns: 30px 1fr 10px 2fr 10px 1fr 10px 2fr 10px 30px**

grid-template-areas - allows you to assign names to various parts of the grid by creating a list of names for every cell in the grid

grid-area - an excellent way to position your items (after having given names to all of the areas of your grid)

When you deliberately place items in specifically defined areas of the grid you are defining **explicit** grid behaviors. However, the grid system has a number of automatic or **implicit** behaviors that occur. One example is the way that grid items **flow** into the grid sequentially. Another is how the grid will create new rows and/or columns as needed to accommodate items that don't fit into the defined grid. There are a set of properties that all include the four letter word, **auto** that give the grid more guidance about how to handle automatically generated or placed parts.

Properties that add space between tracks: grid-row-gap, grid-column-gap, grid-gap

Declarations:

justify-self: end; - position an item on the right edge of its grid area (in a left to right reading language)

align-items: end; - push all the images in a grid to the bottom of their respective cells

align-content: center; - center the whole grid vertically in its container

align-self: stretch; - make a particular item stretch to fill its container

justify-items: center; - center all items in their areas horizontally

Chapter 17: Responsive Web Design

The core principle of responsive web design is that all devices get the same **HTML** source, located at the same **URL**, but different **sizes** are applied based on the **viewport** size to the rearrange components and optimize **usability**.

Responsive design is becoming the default way to build a website that meets the demands of our current multi-device environment.

The core components of responsive web design are **CSS media queries, a flexible grid, flexible images, and the viewport meta element**

Content - an attribute of the viewport meta element that would supply with the value of "**width=device-width**" to make the width of the viewport equal to the width of the device screen

A downside of **fluid layouts** aside from the additional knowledge and care that it takes to implement them well are text line lengths can become uncomfortably long.

max-width: 100% - a declaration for a style rule that can be applied to an image to make it scale down to fit the size of its container, but never grow wider than its original size, is also the go to feature to test when creating mobile-first responsive design

A **breakpoint** is the point at which we use a **media query** to introduce a style change. A common practice is to create the design for **narrow** screens first, and then resize the browser **wider** and pay attention to the point at which each part of the page starts to become unacceptable.

Tips for responsive web design:

Content hierarchy - Consider and organize this carefully before any code gets written. Pare it down to what is most important and useful, but keep in mind that you should try to make sure that all of it should be accessible regardless of device.

Layout - Designers typically start with a simple one-column one in mind that fits well on small screens and then as more space is available, add additional columns, etc. A good way to tell when to make an adjustment is to look at text line lengths. Optimal line lengths are between 45 and 75 characters.

Typography - You'll want to make choices here that keep your words legible and pleasant to read. For example saving stylized fonts for wider screens & using narrower margins on smaller screens.

Navigation - It's critical to get this right, especially at smaller widths and the book contains 7 or 8 successful "patterns" that can help users get to where they need to go.

Images - The key issues surrounding these tend to deal with keeping file sizes down as appropriate and making sure that users are able to see what they are supposed to see in them.

Special content (like tables, forms, interactive features) - Certain ones of these can have serious troubles with small screens compared to the way that they appear larger and easier to use on bigger screens. Many approaches deal with making your elements simpler, scrollable, and/or accessible by other means like external applications.

Chapter 18: Transitions, Transformations, and Animation

Tweening - the smoothing out of the otherwise abrupt changes between two animation states over time by filling in the frames in between

If a transition had keyframes it would have **2.**

transition-duration: 0.5s; - a transition declaration that would make a transition last .5 seconds

You can animate: **padding, word-spacing, and width**

transition: color 2s linear 0.5s; - the 0.5s describes the transition delay property (how long the start of the animation is delayed).

Correct transform declarations:

```
transform: skewY(15deg);
transform: translate(25px, -30px);
Transform: rotate(30deg);
```

Chapter 19: More CSS Techniques

CSS reset - is used to override browser defaults, make presentation more predictable across browsers, and to prevent elements from inheriting unexpected styles.

Image replacement technique - doing this allows you to use a decorative graphic in place of text, to maintain the semantic content of the document.

CSS sprite - is used to improve site performance and to reduce the number of HTTP requests.

Additional Notes:

CRAP (contrast, repetition, alignment, proximity)

"**Above the fold**" means any content you see before scrolling