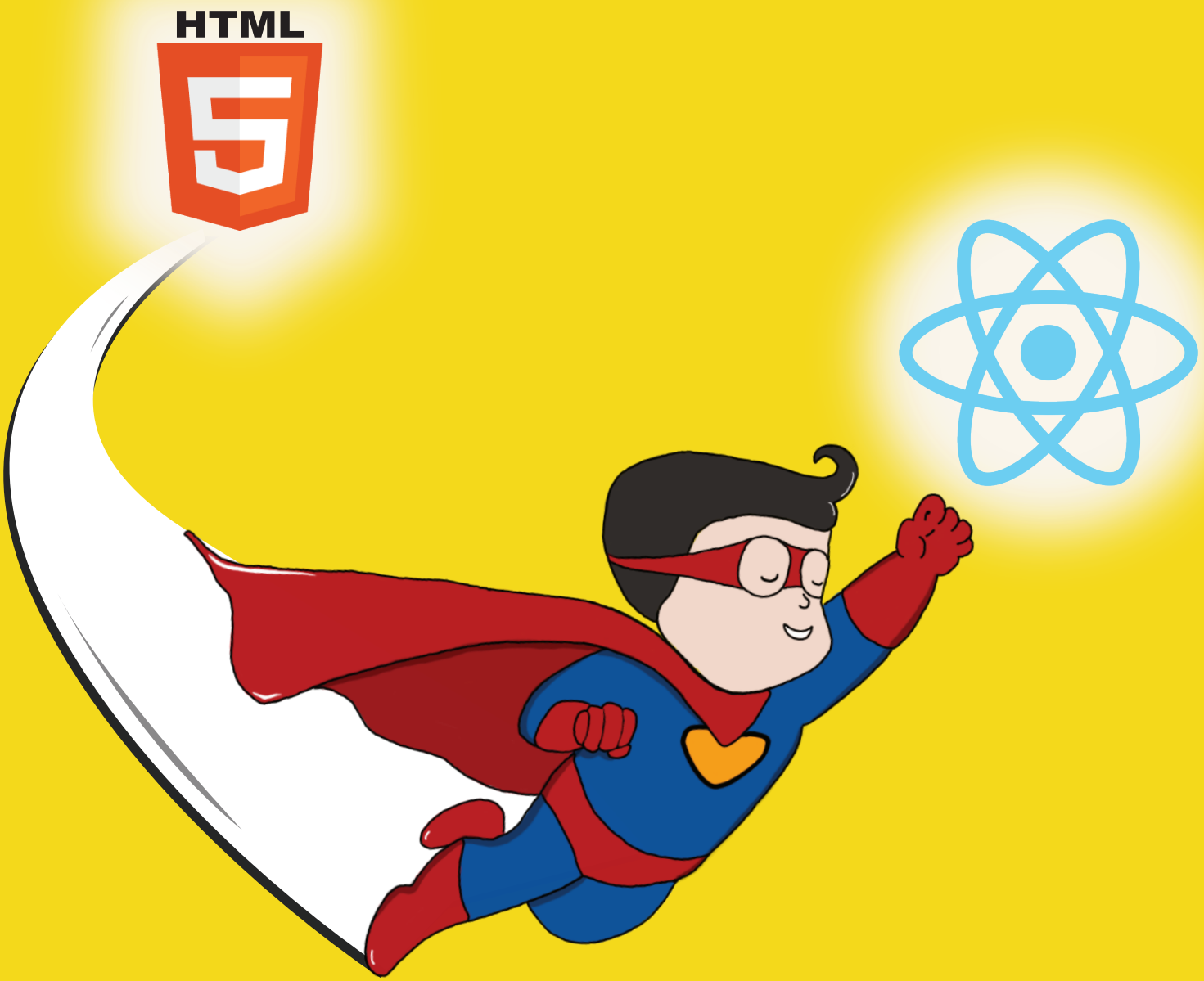


HTML TO REACT

The Ultimate Guide

NG NGNINJA ACADEMY

HTML and CSS

Table Of Content

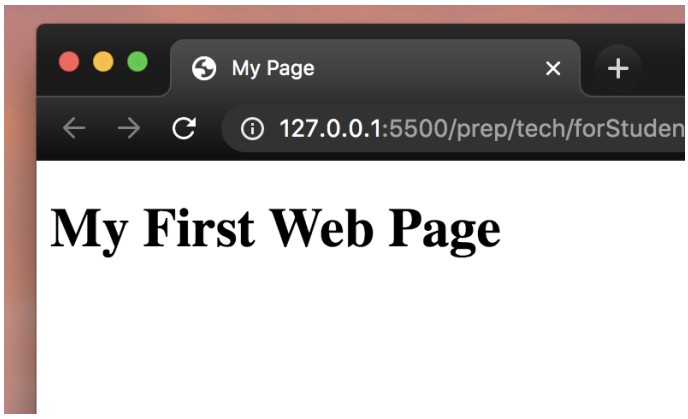
Module 1 - Getting Started

Create Your First HTML Page

- Create an `app` folder on your computer
- Create file named `index.html` inside `app` folder on your computer
- Open that page in your favorite text editor
    - I personally use VSCode

Write the following boilerplate code in `index.html` file

- This is your HTML boilerplate code
- You are telling the browser that `index.html` is an HTML file and render it as an HTML website
- `head` tag is where you declare meta-data, title, and link your style files
- `body` tag is where you actually start writing your web page codes
    - The visible part of the HTML document is between `<body>` and `</body>`
- `title` tag is used to define your page title
- `h1` tag is used to render a heading on your page




```
<!DOCTYPE html>
<html>
  <head>
    <title>My First HTML Page</title>
  </head>
  <body>
    <h1>My First Web Page</h1>
  </body>
</html>
```

Run Your First HTML File

- To run your application locally -
  - Save your changes to the `index.html` page
  - Then imply open your `index.html` file in the browser

Run the application on the Internet for FREE

- If you want to run your application on the Internet and share the URL with your partner follow these steps
- Go to Netlify Drop

- Drop the folder that contains your HTML and CSS file (if you have one) on that page where it says `Drag and drop your site folder here`
- And Voila! It should create a unique URL that you can simply share with your partner

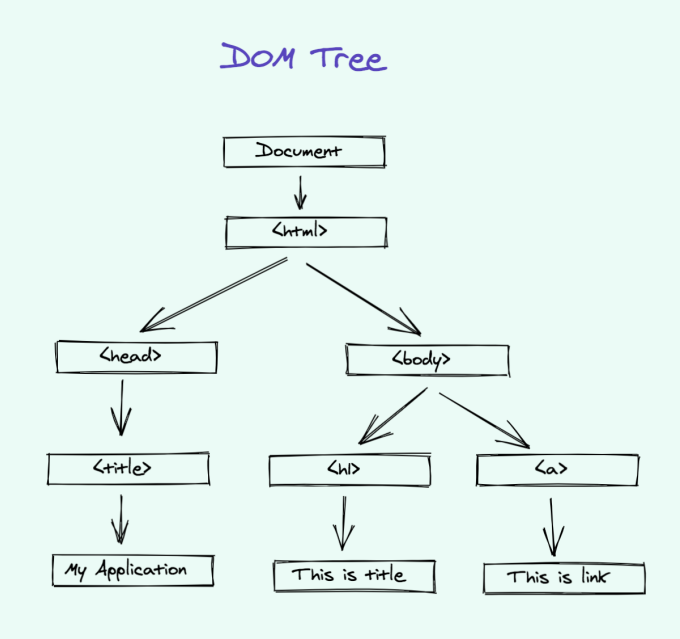
> You can see the working example here

If I do not put <! DOCTYPE html> will HTML5 work?

- No, browser will not be able to identify that it's a HTML document and HTML 5 tags will not function properly.
- Modern browsers are clever enough to render the HTML content, but it may not be optimized correctly.

DOM

- Document object model

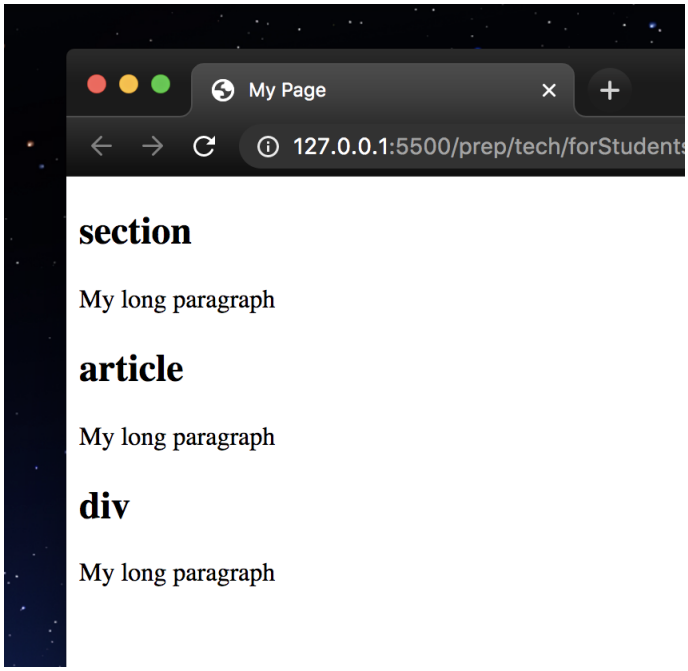


- HTML document is represented as tree
- Every tree node is an object
- `document` object represents the DOM tree
- `<html>` node is at the root
- `<head>` and `<body>` are its children
- The leaf nodes contain text of the document
- "My Application", "This is title", and "This is link" are the leaf nodes
- DOM api is available to capture user events and give access to its children

> NOTE: If we put something after `</body>` end tag, then that is automatically moved inside the `body`, at the end, as the HTML spec requires that all content must be inside `<body>`.

> NOTE: Modern browsers are smart. If the browser encounters malformed HTML, it automatically corrects it when making the DOM. For ex: browser will auto insert `<html>` tag at the top is not provided.

When should you use section, div or article?



section

- Group content with related single theme
- Like a subsection of long article
- Normally has heading and footer

```
<section>
  <h2>Subtitle</h2>
```

```
  <p>My long paragraph</p>
</section>
```

article

- Represents complete, self-contained content of page
- Could be a forum post, newspaper article, blog entry
- Its independent item of content
- Make sense on its own

```
<article>
  <h2>Subtitle</h2>
  <p>My long paragraph</p>
</article>
```

div

- Does not convey any meaning
- Its often called element of last resort
- Use it when no other element is suitable

```
<div>
  <h2>Subtitle</h2>
  <p>My long paragraph</p>
</div>
```

Headings

- Heading tags are part of semantic HTML
- They are used to define headings
- They go from `h1` to `h6`
- Size is largest for `h1` by default and smallest for `h6` by default

```
<h1>My Heading 1</h1>
<h2>My Heading 2</h2>
<h3>My Heading 3</h3>
<h4>My Heading 4</h4>
<h5>My Heading 5</h5>
<h6>My Heading 6</h6>
```

- Headings are used for SEO purposes
  - Search Engine Optimization
  - Useful for indexing pages and structure of the page
- You can format the font styles as per your requirements for these tags

```
<h1 style="font-size:72px;">My Heading</h1>
```

Paragraph

- `p` element is used for writing a paragraph of texts in HTML
- You can include `p` inside other elements like `div`, `section`, `article`

> TIP: don't add extra white spaces - the browser will remove any extra spaces and extra lines when the page is displayed. Use other HTML and CSS properties to add white spaces as per your requirements.

```
<div>
  <p style="font-size:12px;">This is a paragraph.</p>
</div>
```

Links / Anchor elements

- Links allow users to navigate
  - From one page to another
  - Or even from one section of the page to another section on the same page
- Links in HTML are called Hyperlinks
- Below link will take you to `https://www.google.com`
- `href` attribute specifies the target address

```
<a href="https://www.google.com">Google</a>
```

Images

- HTML allows you to add images on your website
- `src` attribute is where you specify the location of your image
    - It can be an image from the internet
    - Or from your local machine

```
<img src="https://via.placeholder.com/150">
```

- `img` tag also takes in another attribute called `alt`
- `alt` attributes provide a way to show an alternative text in case an image is not available for display
    - Example if the internet is not available, or user is on screen reader

Image Sizes

- You can size your image using `width` and `height` property or the `style` property

```
<img src="https://via.placeholder.com/150" alt="placeholder" width="150"
height="150">
<img src="https://via.placeholder.com/150" alt="placeholder"
style="width:150px; height:150px;">
```

Image as links

- You can make image clickable using `anchor` tags around them
- This can be used to navigate to another place by clicking on the image

```
<a href="https://www.google.com">
  <img src="https://via.placeholder.com/150" alt="placeholder" width="150"
height="150">
```

```
  </a>
```

Module 2 - Styling your Webpage

Create a CSS File

- Create `style.css` file inside `app` folder that you created above when creating your first HTML page
- Your styles will go in this `style.css` file

Then add your style in your `index.html` file

- You have to use the `link` tag to include your style file inside your HTML file

```
<head>
  <meta charset="utf-8" />
  <title>Valentine Gift</title>

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

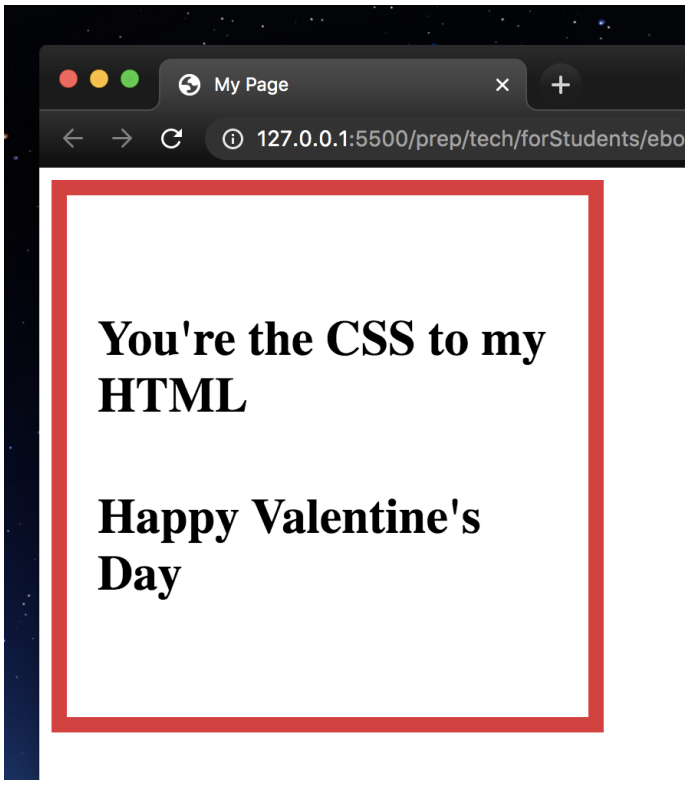
Now let's create the body of your Valentine's Day card

- Replace the `body` tag in your `index.html` file to match the following code
- You are adding `card` DIV which will be the container of your greeting card. We will add the styles later.
- Inside the `card` DIV add two H1 tags
  - These are your heading messages
  - H1 are the biggest headings available
  - You can also change the `font-size` as per your need
- We also assigned appropriate `classes` to our HTML so that we can style them later
  - You will learn about classes later

```
<body>
  <div class="card">
    <h1 class="quote">You're the CSS to my HTML</h1>
    <h1 class="message">Happy Valentine's Day</h1>
  </div>
</body>
```

Now let's add your first style

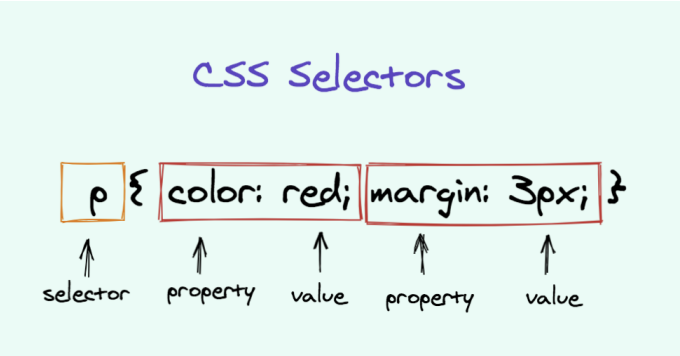
- Let's add styles for your valentine's day card
- We are using `.card` - class selector to grab the card DIV and style it
- Here we are just setting a nice red `border: 10px solid #E53038;`
- `height: 100vh;` is done to match out `body` tag's height - which is the full view-port height.
- `display: flex;` makes this `card` DIV a flex-box.
  - We are just making all our `flex-children` align in vertically and horizontally center position in one column.
  - NOTE: We will learn about flex-box in the later section.




```
.card {
  border: 10px solid #E53038;
  height: 300px;
  width: 300px;
  padding: 20px;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}
```

Selectors

- In layman terms selectors are used to grab a DOM element to apply styles to it
- There are different types of selectors you will learn below



- The images shows `tag` selector
- We are grabbing all the `<p>` tags in our HTML document and applying it `red` color and margin of `3px`
- `color` is the property. `red` is the property value
- `margin` is the property. `3px` is the property value
- There are numerous other values you can use to apply correct styles to your web page.

.class

- Selects all elements with given class name
- It will select all `p` with `class="myClass"`
- Write a period (.) character, followed by the name of the class

```
<p class="myClass">This is a paragraph</p>

.myClass {
  background-color: yellow;
```

```
}
```

child .class

- You can target child element using a class hierarchy

```
// syntax

.parent .child {
    background-color: yellow;
}
```

- You have to write parent class name followed by a space, and then followed by the child's class name
- Below example will add `background-color: yellow` to the paragraph

```
<div class="parent">
    <p class="child">This is a paragraph</p>
</div>


.parent .child {
  background-color: yellow;
}
```

#id

- Style the element with given `id` attribute
- In below example `myParagraph` is the `id` of the paragraph
- We select the elements by adding `#` followed by the `id` attribute of the element

```
<p id="myParagraph">This is a paragraph</p>

#myParagraph {
  background-color: yellow;
}
```

element tag

- You can directly select an element by its tag name and apply the styles
- In this case you don't have to mention `id` or the `class` - simply write the element name in your styles and add properties to it
- Below example will grab all the `p` elements and apply the style to it

```
<p>This is a paragraph</p>

p {
  background-color: yellow;
```

```
}
```

Mix n match

- Above mentioned selectors are basic and most common ways to select elements and apply the styles
- You can also mix and match any of the above selectors to apply the styles

Id and Class

- `id="myParagraph"` forms the Id selector
- `class="myClass"` forms the class selector

```
<p id="myParagraph" class="myClass">This is a paragraph</p>

#myParagraph.myClass {
  background-color: yellow;
}
```

Element and Class

- `p` is used as element selector
- `class="myClass"` forms the class selector

```
<p class="myClass">This is a paragraph</p>
```

```
p.myClass {
  background-color: yellow;
}
```

Advanced Selectors

adjacent selector

- Selects only the element which is preceded by the former element
- In this case, only the first paragraph after each ul will have red text

```
<ul></ul>
<p></p>

ul + p {
    color: red;
}
```

attributes selector

- Will only select the anchor tags that have a title attribute

```
a[title] {
    color: green;
}
```

- Will style all anchor tags which link to http://ngninja.com

```
a[href="http://ngninja.com"] {
    color: #1f6053; /* ngninja green */
}
```

- Star designates that the proceeding value must appear somewhere in the attribute's value
- This covers ngnin.com, ngninja.com, and even inja.com

```
a[href*="in"] {
    color: #1f6053; /* ngninja green */
}
```

- Attribute "contains" value
- Needs to be whole word

```
[title~="cat"] {
    border: 5px solid yellow;
}

<img src="cat1.jpg" title="small cat" width="150" height="113"> //
selected
<img src="cat2.gif" title="catssss" width="224" height="162"> // NOT
selected
<img src="dog.gif" title="dog" width="200" height="358"> // NOT selected
```

- Attribute "contains" value
- Does NOT have to be whole word

```
[title*="cat"] {
    border: 5px solid yellow;
}

<img src="cat1.jpg" title="small cat" width="150" height="113"> //
selected
<img src="cat2.gif" title="catssss" width="224" height="162"> // selected
<img src="dog.gif" title="dog" width="200" height="358"> // NOT selected
```

- Attribute "starts with" value
- The value HAS to be a whole word
    - Either whole word
    - Or word followed by –

```
[title|="cat"] {
    border: 5px solid yellow;
}

<img src="cat1.jpg" title="cat-like small" width="150" height="113"> //
selected
<img src="cat2.gif" title="cat" width="224" height="162"> // selected
<img src="dog.gif" title="dog" width="200" height="358"> // NOT selected
```

- Attribute "starts with" value
- The value DOEST NOT have to be a whole word

```
[title^="ca"] {
    border: 5px solid yellow;
}

<img src="cat1.jpg" title="cat-like small" width="150" height="113"> //
selected
<img src="cat2.gif" title="cat" width="224" height="162"> // selected
<img src="dog.gif" title="dog" width="200" height="358"> // NOT selected
```

- Attribute "ends with" value
- The value DOEST NOT have to be a whole word

```
[title$="at"] {
    border: 5px solid yellow;
}

<img src="cat1.jpg" title="cat-like small" width="150" height="113"> //
NOT selected
<img src="cat2.gif" title="cat" width="224" height="162"> // selected
<img src="dog.gif" title="dog" width="200" height="358"> // NOT selected
```

Backgrounds

- You can set different background of your elements
- Background of an element is the total size of the element, including padding and border (but not the margin)
- Below is the list of all the background properties

```
background-color
background-image
background-position
background-size
background-repeat
background-origin
background-clip
background-attachment
```

- You can set all the properties using one declaration
- These are some of the most commonly used background properties
- It adds `lightblue` background-color
- It adds `myImage.png` background image
- It set `no-repeat` background meaning it will not repeat the background image
    - It is defaulted to repeat both vertically and horizontally
- It sets `center` background-position

```
body {
  background: lightblue url("myImage.png") no-repeat center;
}
```

Colors

- The color property specifies the color of the text
- You can specify the color property on different elements by using different types of selectors
- You can specify colors by its name, it's hex value or its RGB value

```
h1 {
  color: red;
}

h1.myClass {
  color: #02af00;
}

h1#myId {
  color: rgb(111,111,111);
}
```

Borders

- You can add borders to your HTML elements
- Below is the list of all the border properties

```
border-width
border-style (required)
border-color
```

- In below example

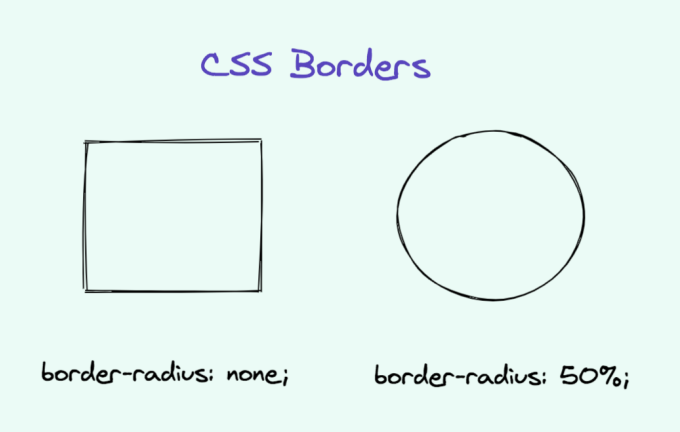
- `5px` is the border width
- `solid` is the border style
  - Other examples are `dotted, double, dashed`
- `red` is the border-color
  - You can specify colors by its name, it's hex value or its RGB value

```
h1 {
  border: 5px solid red;
}
```

Fun with Border Radius

Shapes

- Borders also take another property called `border-radius` using which you can give different shapes to your elements



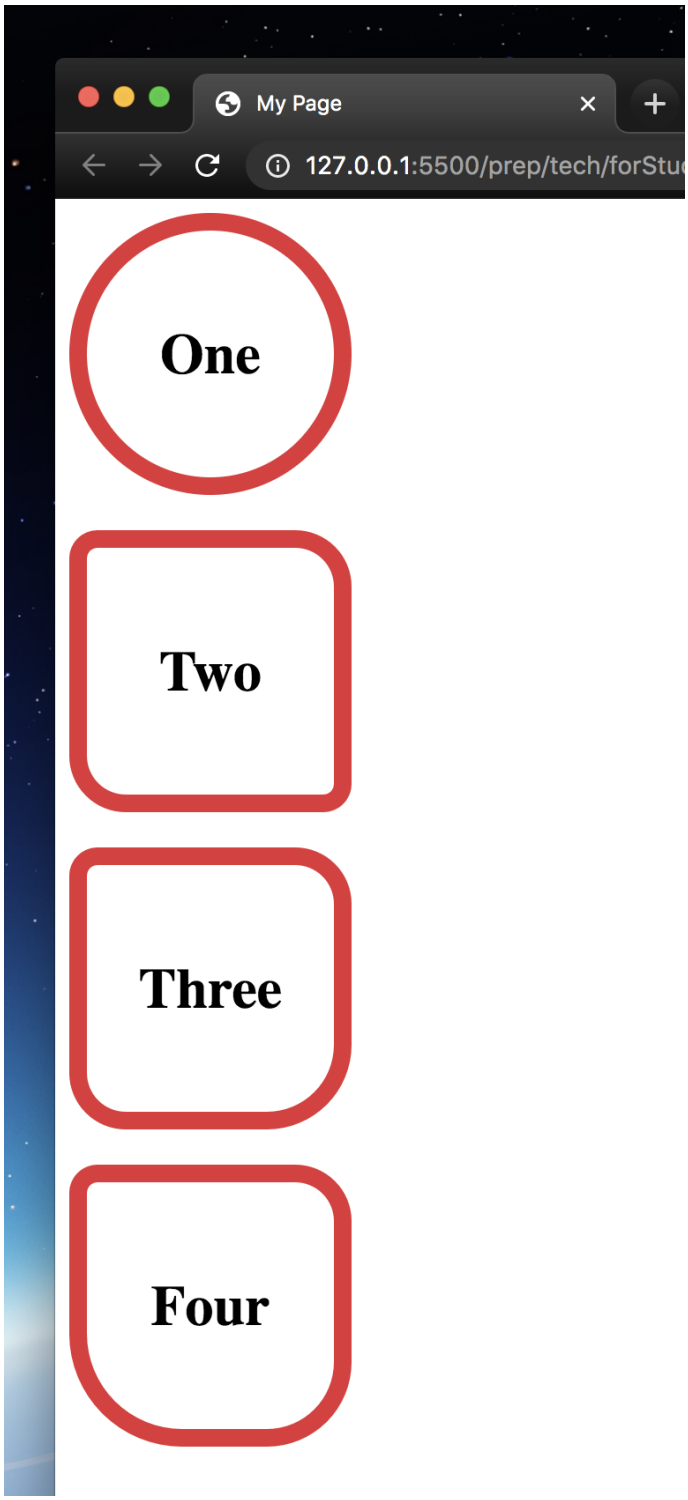
- In the above illustration we have a square on the left and circle on the right

- If you provide `border-radius` of `50%` it will turn your square into a circle

```
.square {
  border-radius: none;
}

.circle {
  border-radius: 50%;
}
```

Shorthand

- If `one` value is set, this radius applies to all 4 corners.
- If `two` values are set, the first applies to `top-left and bottom-right` corner, the second applies to `top-right and bottom-left` corner.
- If `three` values are set - the `second` value applies to `top-right and also bottom-left`.
- If `four` values apply to the `top-left, top-right, bottom-right, bottom-left` corner (in that order).

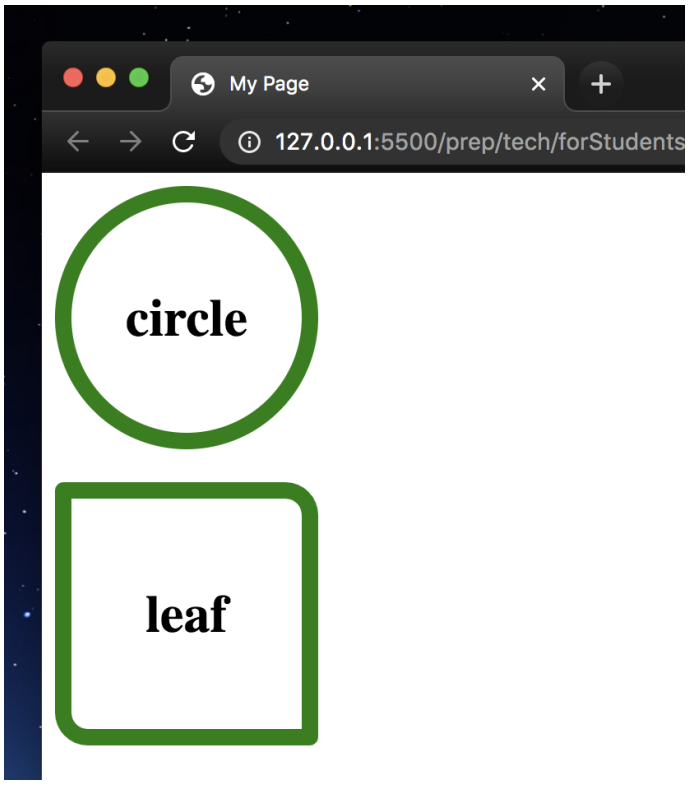



```
<div class="card one">
  <h1 class="">One</h1>
</div>
<div class="card two">
  <h1 class="">Two</h1>
</div>
<div class="card three">
  <h1 class="">Three</h1>
</div>
<div class="card four">
```

```
  <h1 class="">Four</h1>
</div>
```

```
// all 4 corners
.one {
  border-radius: 50%;
}

// 10% top-left and bottom-right,  20% top-right and bottom-left
.two {
  border-radius: 10% 20%
}

// 10% top-left, 20% top-right and also bottom-left, 30% bottom-right
.three {
  border-radius: 10% 20% 30%;
}

// top-left, top-right, bottom-right, bottom-left corner (in that order)
.four {
  border-radius: 10% 20% 30% 40%;
}

.card {
  border: 10px solid #E53038;
  height: 100px;
  width: 100px;
  padding: 20px;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  margin-bottom: 20px;
}
```

Circle and leaf



Circle

```
.circle {
    border-radius: 50%;
}
```

Leaf

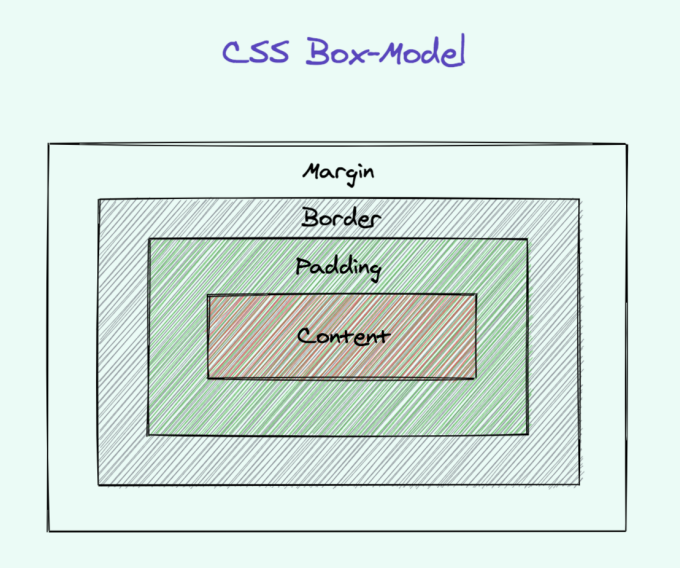
```
.leaf {
    border-radius: 5px 20px 5px;
```

```
}
```

Module 3 - Display and position your elements

Box model

- Imagine box that wraps around every HTML element
  - This refers to the box model
- It has margins, borders, paddings and the actual content
- Everything in a web page is a box where you can control size, position, background, etc.




```
div {
    width: 320px;
    padding: 10px;
```

```
    border: 5px solid gray;
    margin: 0;
}

320px (width)

- 20px (left + right padding)
- 10px (left + right border)
- 0px (left + right margin)
  = 350px
```

Total width of an element should be calculated like this:

```
Total element width = width + left padding + right padding + left border +
right border + left margin + right margin
```

Total height of an element should be calculated like this:

```
Total element height = height + top padding + bottom padding + top border
+ bottom border + top margin + bottom margin
```

Margins

- Margins are used to create space around elements - outside its border

```
<div>
  <p class="myParagraph">My Paragraph</p>
</div>

// styles

.myParagraph {
  margin: 20px;
}
```

- `margin: 20px;` gives the `p` element margin of `20px` around it from all the sides

Margin On Individual Sides

- You can also give margin to the elements on any particular side if you'd want

```
margin-top
margin-right
margin-bottom
margin-left
```

Margin Shorthands

- This shortcut can be used to give margin on all the sides

```
p {
  margin: 20px;
}
```

- The example below gives margin `20px` top and bottom
- And give margin `40px` left and right

```
p {
  margin: 20px 40px;
}
```

- The example below give margin `20px` top
- And give margin `40px` left and right
- And give margin `50px` bottom

```
p {
  margin: 20px 40px 50px;
}
```

Auto Margin

- `auto` value of margin sets the element horizontally center within its container
- Below `div` will take `200px` width and remaining space will be split equally between left and right margin

```
div {
  width: 200px
  margin: auto;
}
```

Paddings

- Paddings are used to generate space around the given element's content - inside its border

```
<div class="myDiv">
  <p>My Paragraph</p>
</div>

// styles

.myDiv {
  padding: 20px;
}
```

- `padding: 20px;` gives the `div` element padding of `20px`
- So, basically there will be `20px` space between `p` and `div` on all the sides

Padding On Individual Sides

- You can also give padding to the elements on any particular side if you'd want

```
padding-top
padding-right
padding-bottom
padding-left
```

Padding Shorthands

- To give padding on all the sides

```
div {
  padding: 20px;
}
```

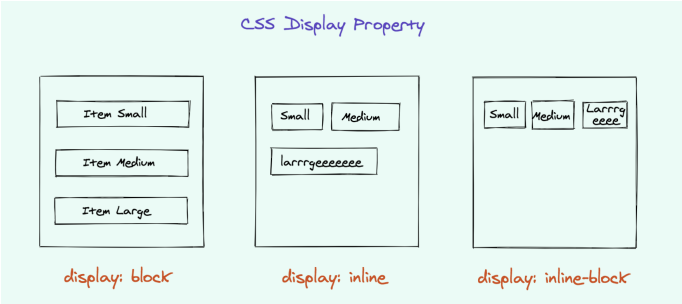
- The below example give padding `20px` top and bottom
- And give padding `40px` left and right

```
div {
  padding: 20px 40px;
}
```

- The below example give padding `20px` top
- And give padding `40px` left and right
- And give padding `50px` bottom

```
div {
  padding: 20px 40px 50px;
}
```

Display



Block

- This property stretches the element left to right as far as it can
- Default is block for `div, p, form, header, footer, section` (and some more)
- Such elements cannot be placed on the same horizontal line with any other display modes
    - Except when they are floated
- Like shown in the illustration -> every item stretches and uses up the entire row

Inline

- Inline element sits in line
    - Without disrupting the flow of other elements
- Like shown in the illustration -> every item takes up only the space it needs
    - Item wraps to the next row if there is no enough space
- `span, em, b` are examples of inline elements
- They take only width needed for it
- They do not honor vertical padding
    - No width
    - No height
    - They just ignores them
- Horizontal margin and padding are honored
- Vertical margin and padding are ignored

Inline-block

- This is just like inline element
- BUT they will respect the width and height
- Basically, they combine the properties of both block elements and inline elements
- The element can appear on the same horizontal line as other elements
- So, like the illustration shows if you set width you can fit all the items together in a single row

None

- These elements will not appear on the page at all
- But you can still interact with it through DOM
- NO space allocated on the page

Visibility Hidden

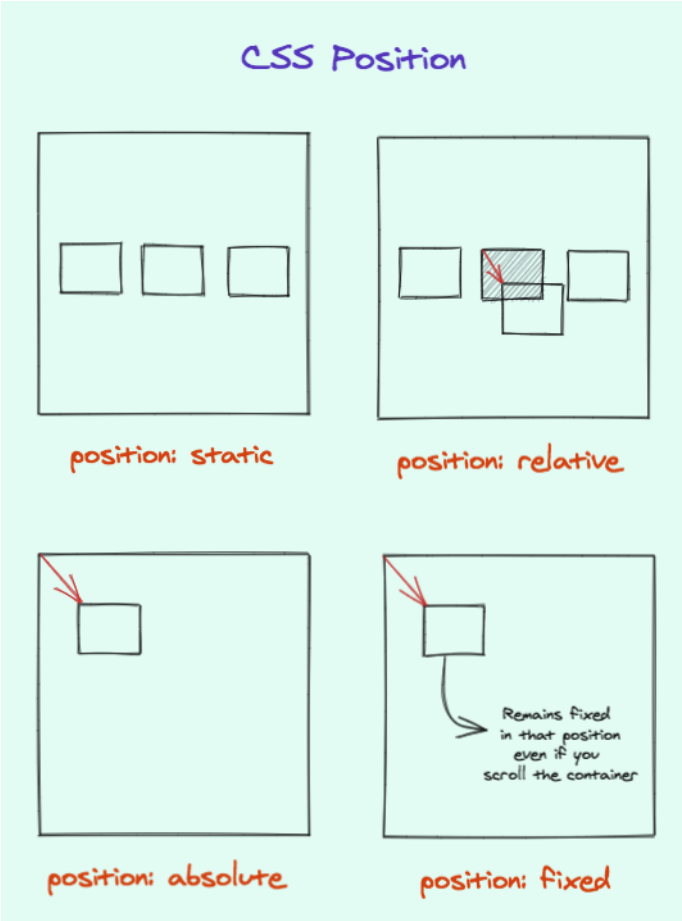
- Space is allocated for it on the page
- Tag is rendered on the DOM
- The element is just no visible

```
div {
  visibility: hidden;
}
```

Flex

- Flex property gives ability to alter its item's width/height to best fill the available space
- It is used to accommodate all kind of display devices and screen sizes
- Fills available space
    - Or shrink to prevent overflow

Positions



Static

- It is the default for every element
- Honestly, it doesn't mean much
    - It just means it will flow into the page as it normally would
- As shown in the illustrations blocks just falls in their default position
- Use it when you want to remove forcefully applied position to the element
- NOTE: `z-index` does not work on them

Relative

- The element is relative to itself
- Check out the example below

```
<div class=myDiv""></div>

.myDiv{
    position: relative;
    top: 10px;
    left: 10px;
}
```

- Now, it will slide down and to the left by `10px` from where it would normally be
  - Please refer the illustration above
- Without that "top" property - it would have just followed `position: static`

Absolute

- This property is very powerful
- It lets you place element exactly where you want
- Using top, left, bottom. and right to set the location
- REMEMBER!! - these values are relative to its parent
  - Where parent is absolute or relative
  - If there is no such parent then it will check back up to HTML tag and place it absolute to the webpage itself
- So, elements are removed from the "flow" of the webpage
- Not affected by other elements
- And it doesn't affect other elements
- NOTE: Its overuse or improper use can limit the flexibility of your site

```
<div class=myDiv""></div>

.myDiv{
    position: absolute;
  top: 10px;
```

```
    left: 10px;
}
```

- The above `div` will slide down and to the left by `10px` from its parent
    - Assuming the parent is either absolute or relative positioned

Fixed

- Positioned relative to the viewport
- Useful in fixed headers or footers
- Please refer the illustration above for better visualization

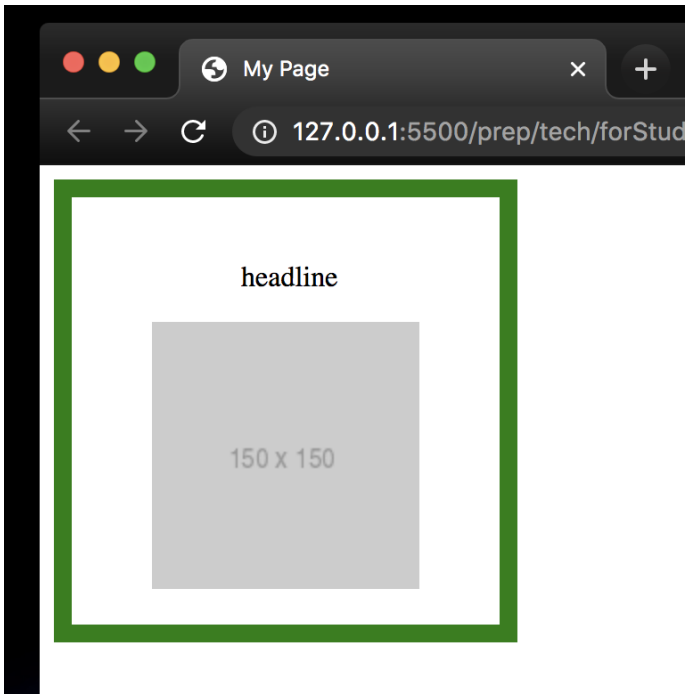
```
<div class=myDiv""></div>

.myDiv{
    position: fixed;
}
```

Centering:

- Center the line of text using `text-align` property


```
<div class="card">
  <h2 class="">long paragraph</h2>
  <p class="">headline</p>
</div>

P { text-align: center }
H2 { text-align: center }
```

- Center a paragraph block or an image using `margin` property
- This will horizontally center the elements




```
<div class="card">
  <p class="blocktext">
    headline
  </p>
  <img src="https://via.placeholder.com/150" />
</div>

P.blocktext {
    margin-left: auto;
    margin-right: auto;
    width: 50px;
}

IMG {
    display: block;
    margin-left: auto;
  margin-right: auto;
}

.card {
  border: 10px solid green;
  height: 200px;
  width: 200px;
  padding: 20px;
}
```

Centering Vertically

- We can use `transform` property along with setting the `top` of the element to center it vertically inside its parent
- Please not the parent container has to be positioned `relative` or `absolute` to center the child

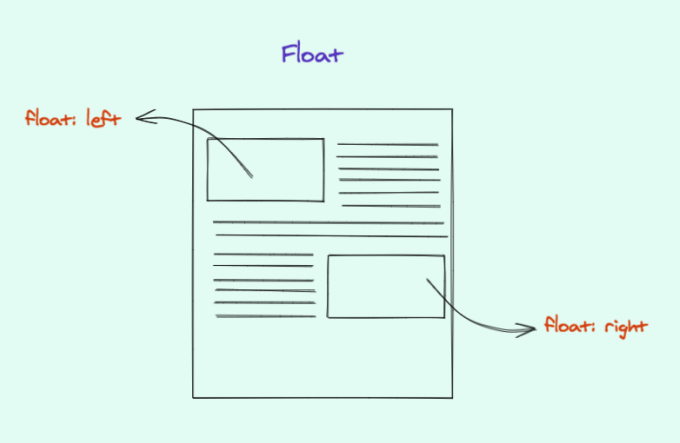

```
<div class="container">
  <p>This paragraph…</p>
</div>


div.container {
  height: 10em;
  position: relative;
  border: 2px solid blue;
}

div.container p {
  margin: 0;
  position: absolute;

  // 50% here means 50% of the height of the container
  top: 50%;

  // move the element up by half its own height. '50%' in 'translate(0,
-50%)' refers to the height of the element itself
  transform: translate(
    0,
    -50%
  );
}
```

- `top: 50%;` - 50% here means 50% of the height of the container
- `transform: translate(0, -50%);`
  - this will move the element up by half its own height.
  - `50%` in `translate(0, -50%)` refers to the height of the element itself

CSS Float

- Element can be pushed to the left or right
- Other elements can wrap around it
- It is used for positioning and formatting content
- Used with images and text which wraps around them

```
left - The element floats to the left of its container
right - The element floats to the right of its container
none - The element appears in its default position
inherit - The element inherits the float value of its parent
```




```
<p>
  my long text here...
  <img class="myImage1" src="myImage1.jpg" alt="myImage1">
</p>
<p>
  my long text here...
  <img class="myImage2" src="myImage2.jpg" alt="myImage2">
</p>

.myImage1 {
  float: left;
```

```
}

.myImage2 {
  float: right;
}
```

- Please refer the above code and the illustration to get better visualization
- We have defined two `p` tags and an `img` tag inside each paragraph elements
- We then set `float: left;` on `myImage1`
    - This pushes the image to the left and text to the right
- And set `float: right;` on `myImage2`
    - This pushes the image to the right and text to the left

Clearing Floats

- Used to control behavior of floating elements
- Clear is used to stop wrap of an element around a floating element
- Floats are headache when they are not cleared properly
- Problem1:
    - There are 2 sections side by side
    - You float LHS section..
    - The LHS section height is not does not match the RHS section

```
    ============
    | LHS  | RHS|
    |      |=====
    |      |
    =======
```

- problem2:
    - parent collapses when children are floated

```
==========
parent
==========

============
| LHS | RHS|
|     |    |
============


<div class="parent">
    <div class="lhs" >
        // float left
    </div>
    <div class="rhs">
        // float left
    </div>
</div>


// Fix:

==================
parent


============
| LHS | RHS|
|     |    |
============

====================


<div class="parent">
    <div class="lhs" >
        // float left
    </div>
    <div class="rhs">
        // float left
    </div>
    <div style="clear:both"></div>
</div>
```

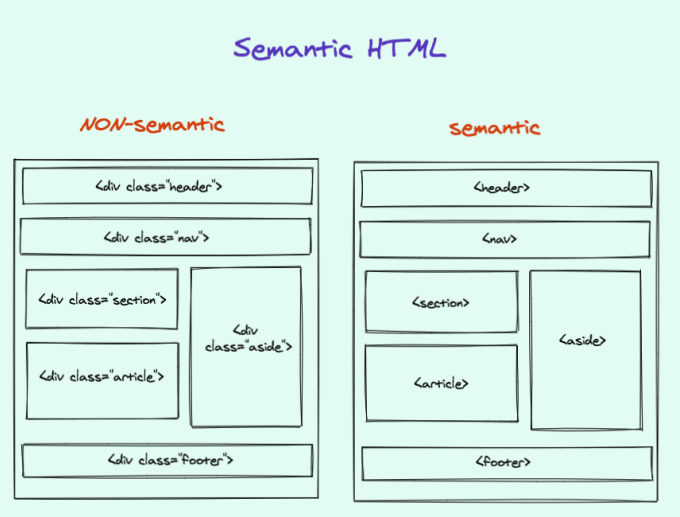
Methods to clear float:

- `clear`
  - takes in 3 values. left, right or both
  - used for problem 1
- `overflow:hidden`
  - great for ensuring parent does not collapse
  - this prop is set to parent
  - used for problem 2
- `clearfix`
  - its a hack.. it uses pseudo elements... and clear both property together
  - prop set to the parent
  - used for problem 2

Module 4 - Semantic HTML5

Semantic HTML?

- It is a coding style
- Semantic elements == elements with a meaning
- Good for SEO
- Good for accessibility
    - Especially for visually impaired people
    - Which rely on browser speech, screen readers to interpret page content clearly



- Check out the image above
- Every semantic element clearly describes its meaning to both the browser and the developer
- Avoid using simple `div` tags everywhere. It does not clearly express the intention.
- Use elements like `header`, `nav`, `section`, `article`, `aside`, `footer`
- All these elements are built with specific purpose - they are good for SEO too

More about Semantic HTML

- `<b></b>` for bold, and `<i></i>` for italic should not be used
  - They are just formatting
  - They do not provide indication of meaning and structure
- Use `<strong></strong>` and `<em></em>`
  - Provide meaning ---> emphasis for example
- Non-semantic elements: `<div> and <span>` - Tells nothing about its content
- Semantic elements: `<form>, <table>, and <article>` - Clearly defines its content

Difference between HTML and HTML5?

HTML

- Simple language for laying out text and images on a webpage

HTML 5

- More like an application development platform
- A new standard for HTML
- Better support for audio, video, and interactive graphics
- Supports offline data storage
- More robust exchange protocols
- Proprietary plug-in technologies like Adobe Flash, Microsoft Silverlight no longer needed
  - Because browsers can process these elements without any external support

Below are new semantic elements

```
- <article>, <aside>, <command>, <details>, <figure>, <figcaption>,
<summary>
- <header>, <footer>, <hgroup>, <nav>, <progress>, <section>, <time>
- <audio> and <video>
- <canvas>
```

What elements have disappeared in the latest HTML?

```
- <frame> and <frameset>
- <noframe>, <applet>, <bigcenter> and <basefront>
```

Difference between `<div>` and `<frame>`?

- `div` is a generic container for grouping and styling
- `frame` actually divides the page
- `frame` is no longer popular
- `iframes` are used instead
    - flexible
    - embed third-party elements like YouTube video

What is HTML5 Web Storage?

- With HTML5, browsers can store data locally
- It is more secure and faster than cookies
- You are able to store large information -> more than cookies
- They are `name/value` pairs
- 2 objects
    - `window.localStorage` - stores data with no expiration date
    - `window.sessionStorage` - stores data for one session (data is lost when the tab is closed)

localStorage:

- Stores the data with no expiration date
- Data is NOT deleted when browser is closed
- Available the next day, week, or year
    - It's not possible to specify expiration
    - You can manage its expiration in your app

```
// Store
localStorage.setItem("lastname", "Smith");

// Retrieve
document.getElementById("result").innerHTML =
localStorage.getItem("lastname");
```

sessionStorage:

- Stores the data for only one session
- Data deleted when browser is closed

```
if (sessionStorage.clickcount) {
    sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
} else {
    sessionStorage.clickcount = 1;
}

document.getElementById("result").innerHTML = "You have clicked the button
" +
sessionStorage.clickcount + " time(s) in this session.";
```

Module 5 - Flexbox intro and media query

Flexbox

- It provides efficient way to lay out, align and distribute space among items in a container
- The main idea - give the container the ability to alter its items' width/height (and order) to best fill the available space

Flex box container properties

- `display: flex`
  - defines a flex container
- `flex-direction: row | row-reverse | column | column-reverse;`
  - establishes main axis
  - defines the direction of children placement
- `flex-wrap: nowrap | wrap | wrap-reverse;`
  - allow items to wrap or nowrap as needed
- `justify-content`
  - defines the alignment along the main axis
  - X-axis for row, Y-axis for column
- `align-items`
  - defines the alignment along the cross axis
  - Y-axis for row, X-axis for column - opposite of `justify-content`
- children properties

Flex box item properties

- `order: <integer>; /* default is 0 */`
  - order in which the flex item appear inside the flex container
- `flex-grow: <number>; /* default 0 */`
  - defines the ability for a flex item to grow if necessary
  - accepts a unitless value that serves as a proportion
- `flex-shrink: <number>; /* default 1 */`
  - defines the ability for a flex item to shrink if necessary
- `flex-basis: <length> | auto; /* default auto */`
  - defines the default size of an element before the remaining space is distributed
  - can be a length (e.g. 20%, 5rem, etc.) or a keyword
  - `auto` keyword means "look at my width or height property"
- `flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]`
  - shorthand for flex-grow, flex-shrink and flex-basis combined
  - default is 0 1 auto
  - recommended that you use this shorthand property rather than set the individual properties
  - shorthand sets the other values intelligently
- `align-self`
  - allows the default alignment to be overridden for individual item
  - overrides container's `align-items` property

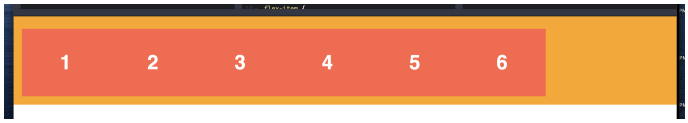
Flexbox Examples

- This is the boilerplate code we will use to demonstrate how flex box works

```
<div class="flex-container">
  <div class="flex-item">1</div>
  <div class="flex-item">2</div>
  <div class="flex-item">3</div>
  <div class="flex-item">4</div>
  <div class="flex-item">5</div>
  <div class="flex-item">6</div>
</div>


.flex-container {
  display: flex;
```

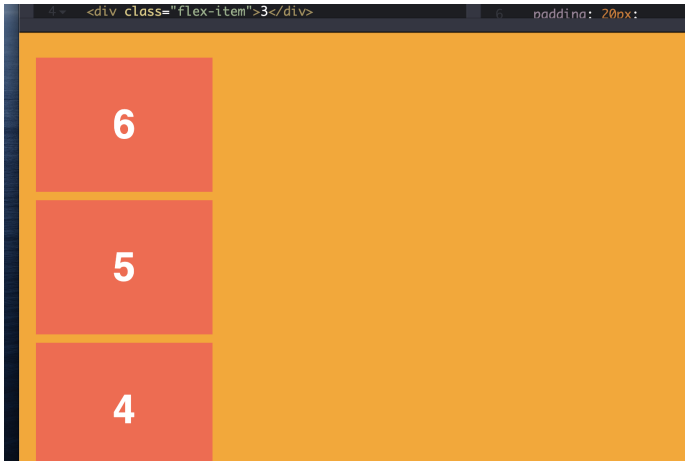
```
  padding: 20px;
  margin: 0;
  list-style: none;
  background: orange;
}

.flex-item {
  background: tomato;
  padding: 5px;
  width: 200px;
  height: 150px;
  margin-top: 10px;
  line-height: 150px;
  color: white;
  font-weight: bold;
  font-size: 3em;
  text-align: center;
}
```

- This is how the output looks for the above code

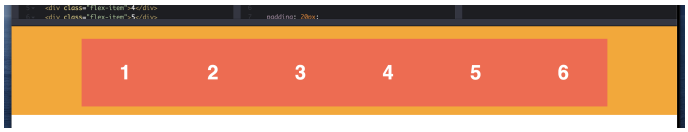



```
.flex-container {
  display: flex;
  flex-direction: column;
}
```

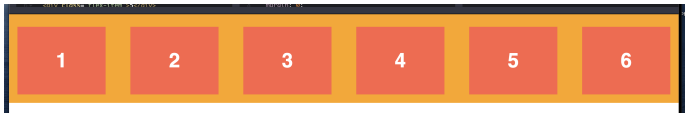

```
.flex-container {
  display: flex;
  flex-direction: column-reverse;
}
```



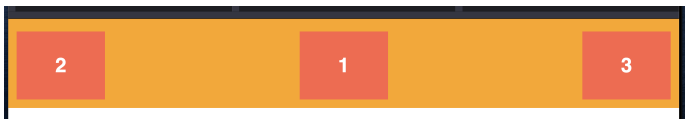

```
.flex-container {
  display: flex;
  flex-direction: row;
  justify-content: center;
}
```



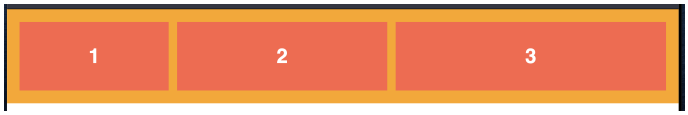

```
.flex-container {
  display: flex;
  flex-direction: row;
  justify-content: space-between;
}
```



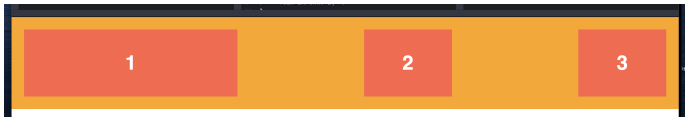

```
<div class="flex-container">
  <div class="flex-item second">1</div>
  <div class="flex-item first">2</div>
  <div class="flex-item third">3</div>
</div>


.first {
  order: 1;
}

.second {
  order: 2;
}

.third {
  order: 3;
}
```



```
<div class="flex-container">
  <div class="flex-item second">1</div>
  <div class="flex-item first">2</div>
  <div class="flex-item third">3</div>
</div>


.first {
  flex-grow: 1;
}

.second {
  flex-grow: 2;
}

.third {
  flex-grow: 3;
}
```

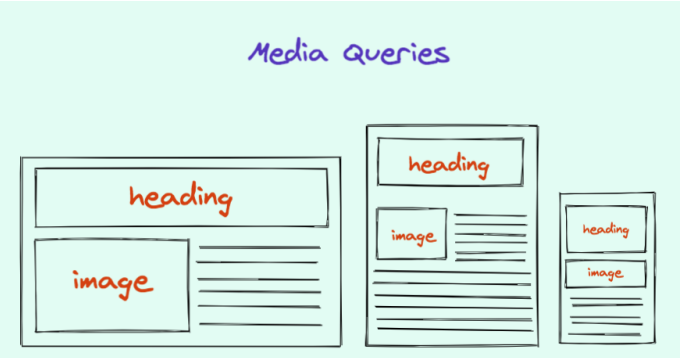



```
<div class="flex-container">
  <div class="flex-item second">1</div>
  <div class="flex-item first">2</div>
  <div class="flex-item third">3</div>
</div>


.first {
  flex-basis: 500px;
}
```



Media queries

- It was introduced in CSS3
- It uses `@media` rule to include CSS only if a certain condition is true



- Media queries allow you to target any platform you desire and write specific styles for that platform
- This is how you write responsive styles
  - Different styles for desktops vs tablets vs mobiles
- Simple example below

```
// If the browser window is smaller than 500px, the background color will
change to lightblue:

@media only screen and (max-width: 500px) {
    body {
        background-color: lightblue;
    }
}
```

- Below are the breakpoints for targeting devices of your interest

```
/* Extra small phones */
```

```
@media only screen and (max-width: 600px) {

}

/* Portrait tablets and large phones */
@media only screen and (min-width: 600px) {

}

/* Landscape tablets */
@media only screen and (min-width: 768px) {

}

/* Laptops/desktops */
@media only screen and (min-width: 992px) {

}

/* Large laptops and desktops */
@media only screen and (min-width: 1200px) {

}
```

Always Design for Mobile First

- It means style for mobiles first
- Then include desktop styles in media queries

```
/* Default styles for mobile phones */
.mobile-styles {
    width: 100%;
}


/* Styles for desktop in media queries */
@media only screen and (min-width: 768px) {
```

```
    /* Style For desktop: */
    .desktop-styles {
        width: 100%;
    }
}
```

Orientation: Portrait / Landscape

- These are queries which apply depending on the orientation of browser/device

```
@media only screen and (orientation: landscape) {
    body {
        background-color: lightblue;
    }
}
```

Let's talk about the sizes - `px` vs `em` vs `rem`

- `Pixels` are ignorant, avoid using them
- If you're setting all of your font-sizes, element sizes and spacing in pixels, you're not treating the end user with respect.
    - Users will have to zoom in and out with ctrl plus +/- depending on the device they are on
- `REMs` are a way of setting font-sizes based on the font-size of the root HTML element
- Allows you to quickly scale an entire project by changing the root font-size
- `em` is relative to the font size of its direct or nearest parent
    - When you have nested styles it becomes difficult to track `ems`
    - This is what REMs solve - the size always refers back to the root
- Both `pixels` and `REMs` for media queries fail in various browsers when using browser zoom, and `EMs` are the best option we have.

How to calculate PX from REM

EX: HTML font-size is set to 10px, paragraph font-size is set to 1.6rem

Then size in pixels is - `1.6rem * 10px = 16px`

More on `rem` vs `em`

- Both `rem and em` are relative units
- `rem` is only relative to the HTML (root) font-size
- `rem` is popularly used for margins, paddings, and sometimes font-sizes too

- em is relative to the font size of its direct or nearest parent
- em are encouraged to be used for media queries

CSS Grids

Flexbox

- Flexbox is 1 dimension
- Its either columns OR rows
- Complext 2-dimensional layouts not possible

Grids

- It is a grid based layout system
- CSS Grids are 2 dimensional
- You can affect columns and rows simultaneously

> TIP: You can use Grid and Flexbox together.

Example

- Let's check out the example below
- Create a `wrapper` DIV
    - `display: grid;` defines the DIV as Grid
- Then we create 2 rows
- Each has 2 columns
    - First column -> 70% of the space

- Second column -> 30% of the space
- `grid-template-columns: 70% 30%;` defines these two column

```
<div class="wrapper">
    // first row
  <div>70%</div>
  <div>30%</div>

  // second row... you don't have to specify rows like bootstrap
  <div>70%</div>
  <div>30%</div>
</div>

.wrapper {
    display: grid;

  // this will split children into 2 rows of 70% and 30%
    grid-template-columns: 70% 30%;
}
```

Example #2 - Gaps

- Here we create 3 columns
  - `40% 30% 30%` respectively
- `grid-column-gap` - used to give margin between columns
- `grid-row-gap` - used to give margin between rows
- `grid-gap` - used to give margin between rows and columns using single command

```
```

```
.wrapper {
    display: grid;
    grid-template-columns: 40% 30% 30%; // 3 columns in 1 row
    grid-column-gap: 1em; // margin between columns
    grid-row-gap: 1em; // margin between row
    grid-gap; 1em; // row and column both!!!
}
```

Example #3 - Fractions

- You don't have to spend time calculating percentages
- You can simply use "fractions"
- 3 DIVS with same width

```
grid-template-columns: 1fr 1fr 1fr;
```

- 3 DIVS
- Total space will be divided by 4
    - ex: 80 / 4 = 20px
    - 1st and 3rd DIV will take 20px space
    - 2nd DIV will take 40px space

```
grid-template-columns: 1fr 2fr 1fr;
```

Example #4 - Fractions Contd.

- You can use `repeat()` function instead of manually repeating the column sizes
- Below property will create 4 columns
- Each column width is `1fr`

```
  grid-template-columns: repeat(4, 1fr);
```

> TIP: fractions are recommended over pixels or %

- Extended example below

```
.wrapper {
  display: grid;

  // 3 columns in 1 row
  // divide into fractions...
  grid-template-columns: 1fr 2fr 1fr;

  // it will repeat fractions 4 times
  grid-template-columns: repeat(4, 1fr 2fr);

  grid-auto-row: 100px; // row height will be 100px
  grid-auto-row: minmax(100px, auto); // min height = 100px max height =
auto based on content
```

```
}
```

Nested grids

- You can also define nested Grids

```
<div class="wrapper">
  <div class="nested">
    <div></div>
    <div></div>
    <div></div>
  </div>
</div>


// now the nested div is also a grid container
.nested {
    display: grid;
  grid-template-columns: repeat(3, 1fr); // 3 columns of 1 fr
  grid-auto-rows: 100px;
}
```

Start-End points of Grid

- We can also specify start and end points of the grid columns and rows
- If you use this you may not want to specify sizes
- Check out the inline descriptions

```
// wrapper is grid container... of 4 columns in this example!!!
<div class="wrapper">
  <div class="box1"></div>
  <div class="box2"></div>
  <div class="box3"></div>
  <div class="box4"></div>
</div>


.box1 {
    grid-column: 1/3; // box1 spans from 1 to 3 columns on browser window
  grid-row: 1/3; // box1 spans from 1 to 3 rows on browser window
}

.box2 {
    grid-column: 3; // box2 spans takes spaces 3 and 4
  grid-row: 1/3; // same as box1
}

.box3 {
    grid-column: 2/4; // box3 will take space 2 to 4
  grid-row: 3; // it will take row space 3
}


// NOTE: in grids... we can overlaps things.. like below
// overlaps 1 and 3
// Sooo... you don't need negative margins and that crap CSS!!!
.box4 {
    grid-column: 1;
  grid-row: 2/4;
}
```

Module 6 - Quirks, tips, and tricks

FOUC

- Flash of un-styled content
- Happens when for a brief moment -> content is displayed with browser's default styles
- Happens when CSS is not loaded but the content is loaded

How to avoid it?

1.

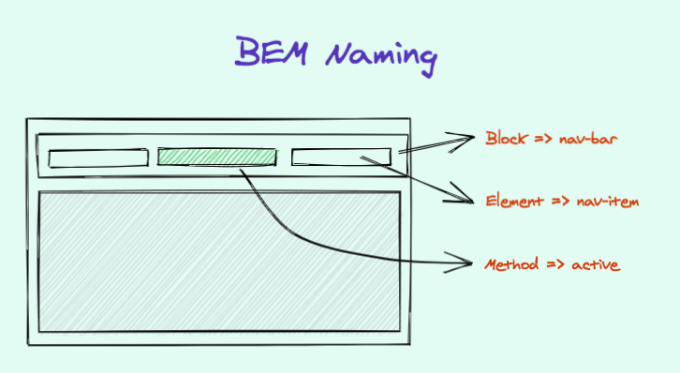
- You can use CSS to hide BODY
- Then when CSS is loaded then using JS -> set body to visible
- But note, if JS is disabled -> then users won't see anything ever!!!

2.

- Use JS to hide body
- And when JS and CSS are loaded -> use JS only to make body visible
- Write that JS code in HEAD -> so as soon as it is hit, HTML is hidden
- And when document is ready -> then show the HTML again

```
<html>
    <head>
        <!– Other stuff like title and meta tags go here –->
        <style type="text/css">
            .hidden {display:none;}
        </style>
        <script type="text/javascript" src="/scripts/jquery.js"></script>
        <script type="text/javascript">
            $('html').addClass('hidden');
            $(document).ready(function() { // EDIT: From Adam Zerner's
comment below: Rather use   load: $(window).on('load', function () {...});
                $('html').show(); // EDIT: Can also use
$('html').removeClass('hidden');
            });
        </script>
    </head>
    <body>
        <!– Body Content –->
    </body>
</html>
```

BEM naming convention



- Blocks, Elements, Modifiers -> BEM
- Naming convention to write styles so that everyone on your team understands the convention
- Blocks
  - Represents standalone entity that is meaningful on its own
  - Parent level elements
  - Class name selector only
  - No tag name or ids
  - `<ul class="nav"></ul>`
- Elements
  - Children of parent level components
  - They usually don't have any standalone meaning
  - Any DOM node within a block can be an element
  - Class name selector only
  - No tag name or ids
  - `<li class="nav__item"></li>`
  - 2 underscores -> it tells that item is child of nav
- Modifiers
  - Use them to change appearance, behavior or state
  - You can add it to blocks or elements
  - Keep the original "block" or "element" styles as it is - add new modifier class
  - Ex below - separate element `nav__item` class and separate modifier `nav__item--active` class
  - `<li class="nav__item nav__item--active"></li>`
  - 2 hyphens -> it tells that active is modifier class

```
<ul class="nav">
```

```
  <li class="nav__item">Home</li>
  <li class="nav__item nav__item--active">About</li> // active modifier
applied
</ul>
```

OOCSS - Object Oriented CSS

- Standard to structure your CSS in a modular fashion
- Can reuse CSS seamlessly
    - Meaning above NAV CSS can be used in any container -> section, page, dialog
- It follows a component based approach
- It enables us to abstract common styles together
    - Reduces duplication of the styles

```
// basic example

.global {
  width: 980px;
  margin: 0 auto;
  padding-left: 20px;
  padding-right: 20px;
}

.header {
  height: 260px;
}

.main {
  background-color: gray;
}

.footer {
  height: 100px;
  background-color: blue;
}
```

```
<header>
  <div class="header global">
    // your code
  </div>
</header>

<div class="main global">
  // your code
</div>

<footer>
  <div class="footer global">
    // your code
  </div>
</footer>
```

CSS User Agent Styles

- These are browser styles
- When browser renders a page it applies basic styles before you've even written a single style
- Each browser has its specific styles different from other browsers
    - It causes an inconsistency problem
- To solve this problem:
    - Normalize CSS approach and the CSS Reset approach
    - Normalize CSS as a gentle solution
    - Reset CSS as a more aggressive solution

Normalizing CSS

- Small CSS file that provides cross-browser consistency
- Provides default styles for HTML elements
- Make sure that all HTML elements renders the same way in ALL browsers - same padding, margin, border, etc..
- In some cases this approach applies IE or EDGE styles to the rest of the browsers

Reset CSS

- This approach says that we don't need the browsers' default styles at all
- We'll define in the project according to our needs
- `CSS Reset` resets all of the styles that come with the browser's user agent
- Grab sample CSS reset here
- The problem with CSS Resets is that they are ugly and hard to debug
- Solution - use Normalize CSS with little bit of CSS Reset
- Unlike an ordinary CSS reset, target specific HTML tags' styles rather than making a big list of tags.
    - Make it less aggressive and a lot more readable

Validate your CSS

- Use online tools to validate your CSS
- Validation Service can be used to check the correctness (validity)
- You might get important insights on what you are missing
- It Helps Cross-Browser, Cross-Platform and Future Compatibility
- Validating your web page does not ensure that it will appear the way you want it to.
    - It merely ensures that your code is without HTML or CSS errors.
- Tool - The Validation Service

Testing Strategies

- Do cross-browser testing
- Manually test Chrome, firefox
- Then manually test IE, Edge
- Then use tools like ..for compatibilities
    - `browsershots.org`
    - `IEtest`

Conditional CSS

- Use below conditional CSS for IE hacks

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

<!–[If IE]>
  <link type="text/css" href="IEHacks.css" />
<![endif]–->

<!–[if !IE]>
  <link type="text/css" href="NonIEHacks.css" />
<![endif]–->
```