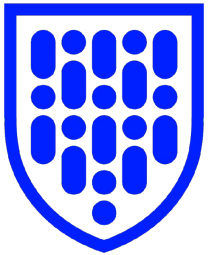


Institute of Data

2024

Software Engineering

Module 2

Front-End Development Basics

Institute of Data

Agenda: Module 2

- How does the **Internet** work
- Introduction to **IPs** and **ports**
- Introduction to **HTTP**
- How do **web browsers** work
- Introduction to **HTML**
- Introduction to **CSS**
- **Fluid layout** and **Responsive CSS**

Front-end development basics

Module 2
Part 1

How does the internet work

How does the Internet work?

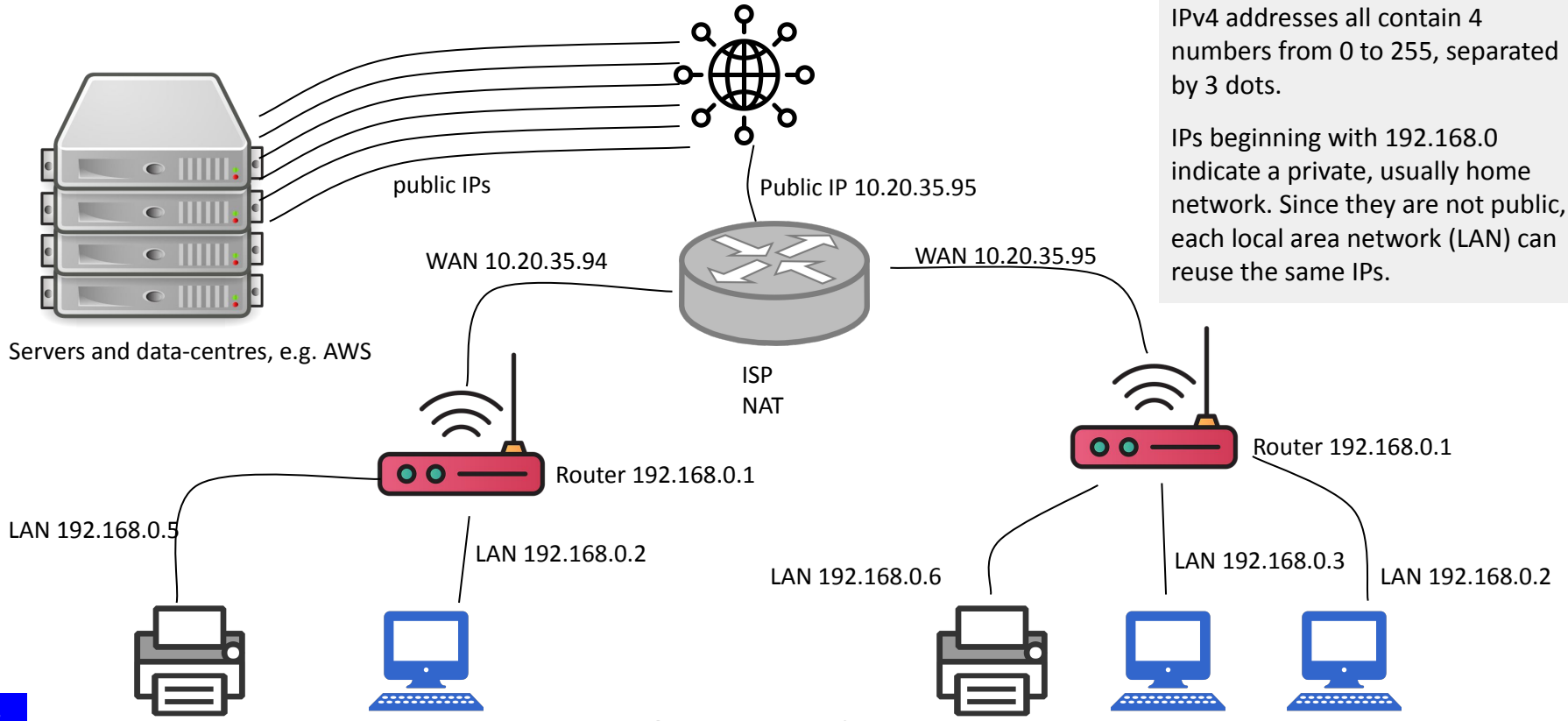
- Does it use a telegraph? Landline phones? The post office?
- This video clip is an entertaining view of how non-technical people may misunderstand the mystery of the internet:
- The Internet Speech The IT Crowd

Institute of Data

How does the Internet work?

- The **Internet** is a collection of interconnected devices which are spread across the globe, like computers, internet routers, basically any device that connects to the global network
- Each device on the **Internet** gets assigned an **IP (Internet Protocol) address**
- Devices can be accessed using assigned **IP addresses**
- Some **IP addresses** are publicly accessible while some others are private and hidden within a **local network**
- Whenever you use the internet, you in fact use **IP**

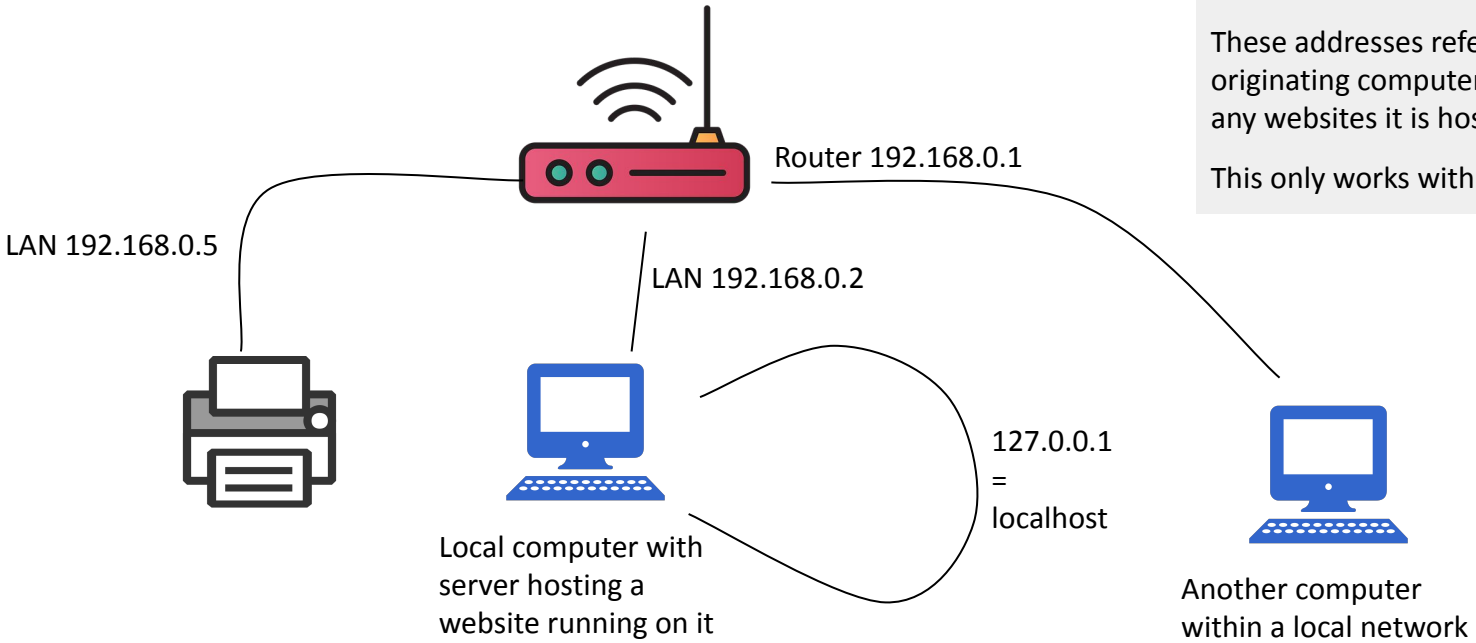
How does the Internet work?



IPv4 addresses all contain 4 numbers from 0 to 255, separated by 3 dots.

IPs beginning with 192.168.0 indicate a private, usually home network. Since they are not public, each local area network (LAN) can reuse the same IPs.

How does the Internet work?



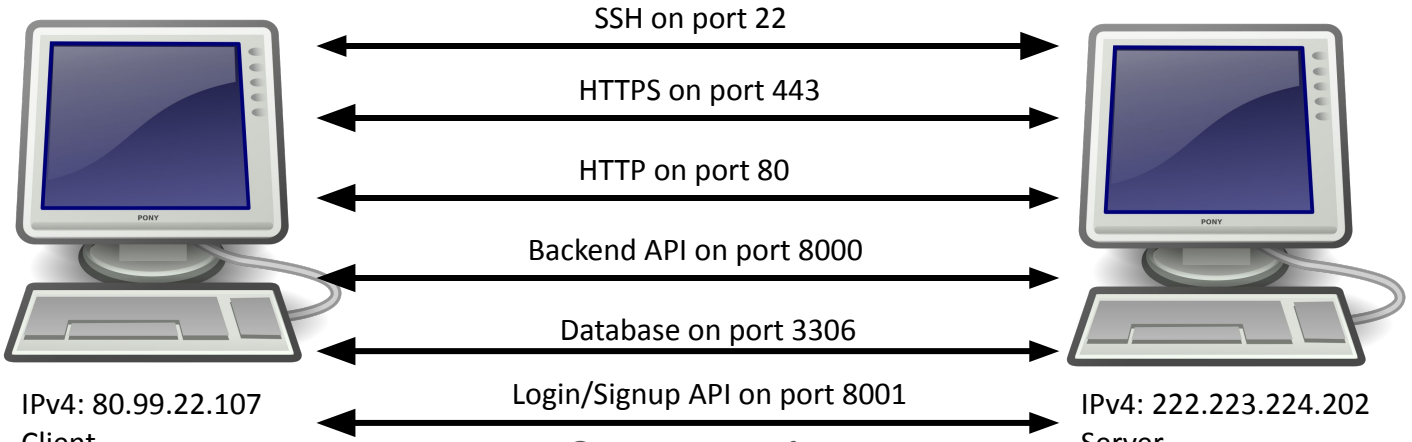
The special IP **127.0.0.1** is mapped to the reserved domain **localhost**. (The LiveServer VSCode extension makes use of this.)

These addresses refer (or loopback) to the originating computer, allowing us to access any websites it is hosting.

This only works within the local network.

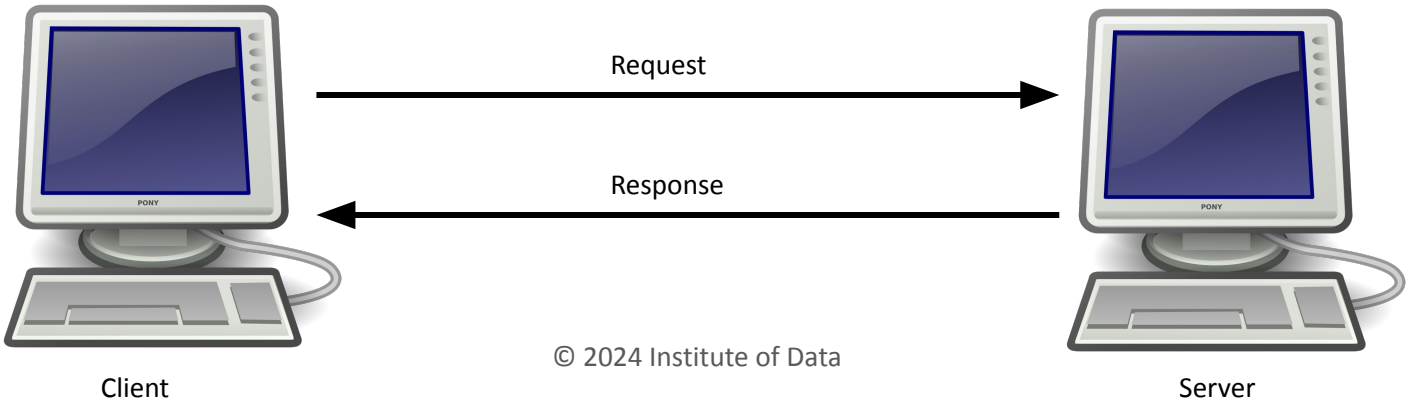
How does the Internet work? - ports

- Each computer connected to the Internet can allow incoming traffic on many different ports
- Some ports are reserved to be used by the specific application, e.g. when you browse the Internet you use port 80 for normal connections and port 443 for secure connections (which are mostly standard now)



Client, server, request and response

- Clients and server exchange information by the client **requesting** something from the server - requesting a web page, confirming purchase of an item, sending a form
- Both request and response contain **data** that client and server are exchanging and additionally some **metadata**
- Client may request establishing a connection that will allow the server to **stream** data to the user, like YouTube



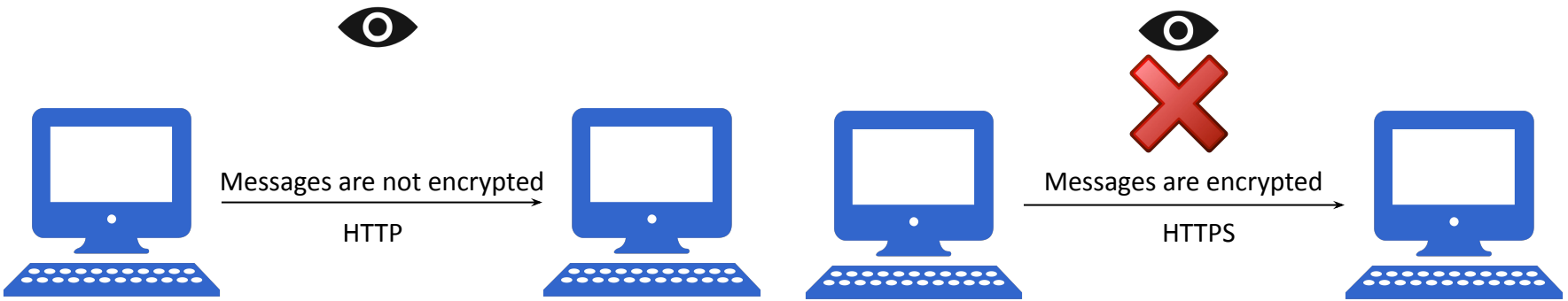
Institute of Data

HTTP - HyperText Transfer Protocol

- **HTTP** is a way of communicating based on the Request/Response model to exchange data over the internet (see here for more info)
- Web browsers, mobile apps and more send **requests** to the server containing information for processing and storage
- Requests often also expect a **response** from the server
  - asking to send a web page (HTML file) from server to client
  - asking to send static resources (JS, CSS, images)
  - asking to send data that is stored on the server, e.g. information about the calendar or emails in the mailbox (JSON)

HTTPS - secure version of HTTP

- Messages sent over **HTTPs** are **encrypted** using a security certificate
- Use of secure HTTPs is indicated by the **padlock icon** next to the URL in the browser
- HTTPs encryption protects against interception of messages and is especially important for **sensitive data** such as payment details
- Modern websites all use HTTPs as standard, for security and search rankings



Institute of Data

Website, Web page and URL

- A single website can consist of many webpages
- Traditionally each webpage is represented by a single HTML document
- A URL, or Uniform Resource Locator, is a string of text that defines where something is located on the Web, e.g.

  https://www.institutedata.com/picture.jpg

  https://www.institutedata.com/courses/software-engineering-program/

The URL always includes the **protocol** (**http** or **https**), followed by the **separator** **://**, followed by the **domain name** or **IP**.

After this may be a **port** preceded by a **:** (colon) and/or a **path** identifying a specific resource, with optional **parameters**. See here for more.

Website vs Webpage

A **webpage** is a single document on the Internet under a unique URL. A **website** is a collection of multiple webpages linked together under a common domain address.

https://www.institutedata.com (website)

File name: index.html (webpage)
URL: https://www.institutedata.com/index.html
URL: https://www.institutedata.com

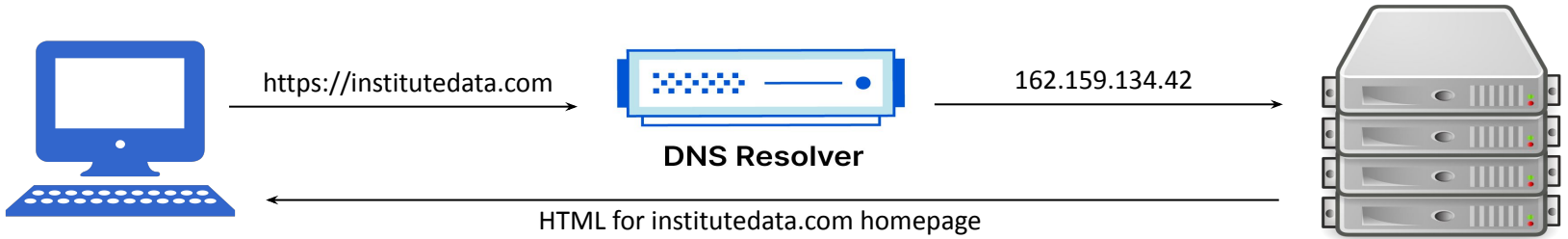
File name: images/image.jpg
URL: https://www.institutedata.com/images/image.jpg

File name: contact.html (webpage)
URL: https://www.institutedata.com/contact.html
URL: https://www.institutedata.com/contact

File name: courses/web-development.html (webpage)
URL: https://www.institutedata.com/web-development.html
URL: https://www.institutedata.com/web-development

What makes up a website?

- **Domain name**: all websites use a domain name such as institutedata.com, purchased from a domain registrar. DNS settings map this domain name to a static IP address, which is resolved during the request
- **Host**: the IP address uniquely identifies the server, or host, on the internet. This is a publicly available device, usually in a data warehouse, that receives the requests sent to the domain name. The code for the website is often hosted here (or spread across multiple servers), and handles processing the request and sending the response back to the client browser.



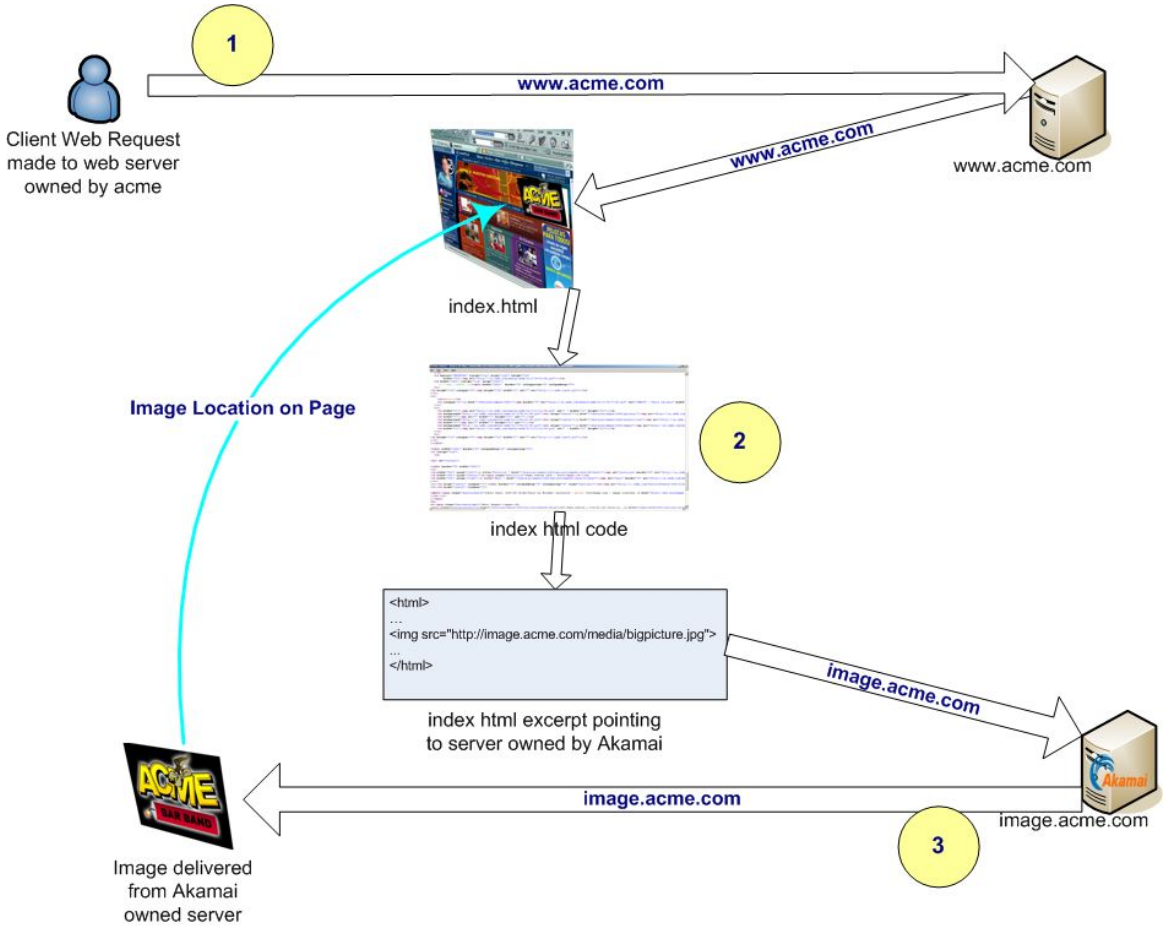
Institute of Data

How do web browsers work?

- **Web browsers** allow our computer (or client) to access **web pages** on the **Internet**.
- When we type in a **URL**, it is translated to an **IP** in a process of accessing the server hosting the **web page**; your **browser** always uses **IP** to display a **web page**
- When visiting a **web page** your **browser** downloads an **HTML** file from its **URL** and then executes the file inside the browser, thus creating a **web page**
- Most web pages also include other resources such as CSS, JS and images - these are also requested via their URLs and displayed in the client web browser

Institute of Data

How do web browsers work?



Institute of Data

Browser support

- Web development is a constantly evolving field with new features and feature proposals constantly being added to the HTML and CSS specifications
- Older browsers are not always capable of supporting new functionalities and there are still a lot of old browsers being used
- Web developers should write code that covers a wide range of devices. To check browser support for given HTML tag or CSS property you can use CanIUse or MDN Web docs
- There are special programs called preprocessors that preprocess code to increase browser support, e.g. autoprefixer for CSS

Institute of Data

Browser support - shimming and polyfilling

- The process of making new features work on old browsers is called either shimming or polyfilling
- Shims and polyfills commonly use JavaScript to mimic functionalities missing in older browsers
- Shims and polyfills are usually less performant than native features in new browsers
- We don’t need to write this code ourselves, polyfills are usually written and tested by experts and re-used as needed to standardise new features across all browsers

Institute of Data

Part 1 Review

- What is the internet?
- How are individual computers identified and accessed on the internet?
- What are some differences between local and public computers?
- What are ports used for?
- How do clients and servers communicate?
- What is HTTP and what is it used for?
- What's the difference between a website and a webpage?
- How does a web browser work?

Front-end development basics

Module 2
Part 2

HTML

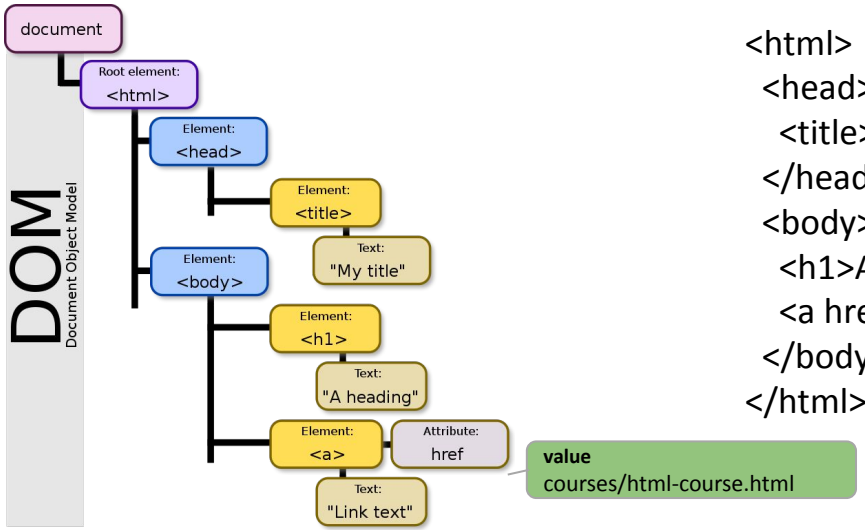
Institute of Data

HTML

- HTML stands for HyperText Markup Language
- HTML gives our website **structure** and **content** and provides it with metadata
- HTML tags - eg. `<div></div>` - are called HTML **elements** when the webpage loads
- HTML tags can have **attributes** assigned to them, e.g
  - `<html lang=”en”>`
  - `<img src=”images/image.png” alt=”My cat”>`

Document Object Model - DOM

- Browser loads the HTML from the Internet and then parses it to construct the **DOM** - an in-memory representation of the structure and content of the page
- The DOM is **hierarchical**: parent, child, sibling, descendant, direct descendant




```
<html>
 <head>
  <title>My title</title>
 </head>
 <body>
  <h1>A heading</h1>
  <a href=”courses/html-course.html”>Link text</a>
 </body>
</html>
```

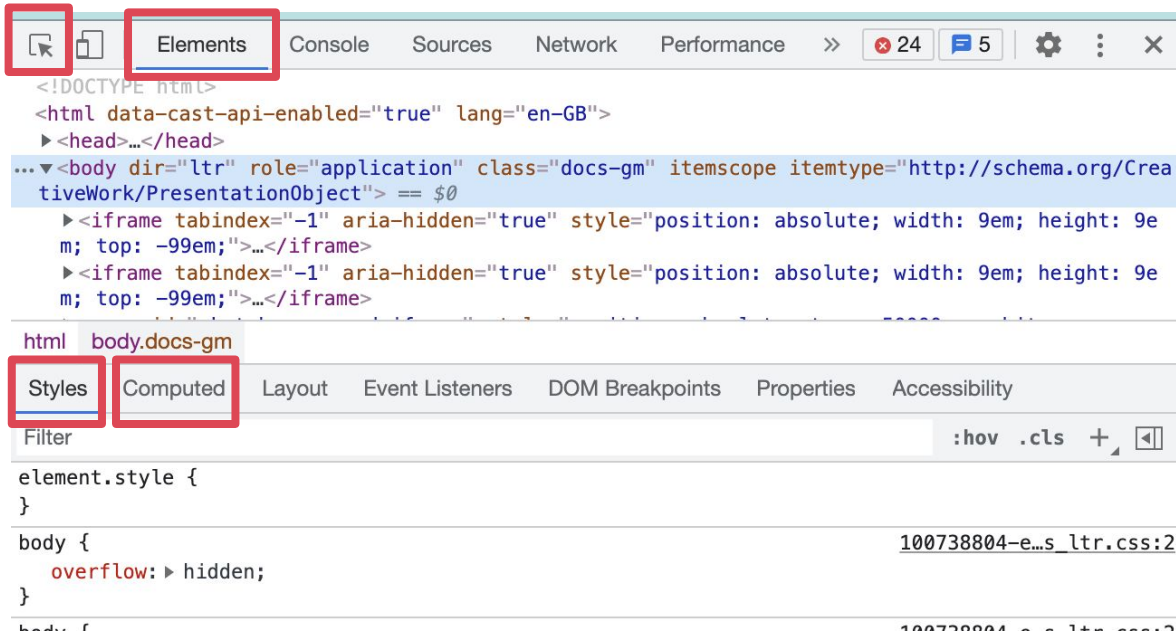
Institute of Data

Id and class

- **id** and **class** attributes can be added to each of the HTML elements
- **id** is a unique identifier of the element on the page, the same id shouldn’t repeat on the same page more than once (within HTML document)
- **class** allows us to assign one or multiple CSS classes to each HTML element - we can re-use classes multiple times
- With use of JavaScript you can find elements by their tagname, id or classes and then modify the DOM

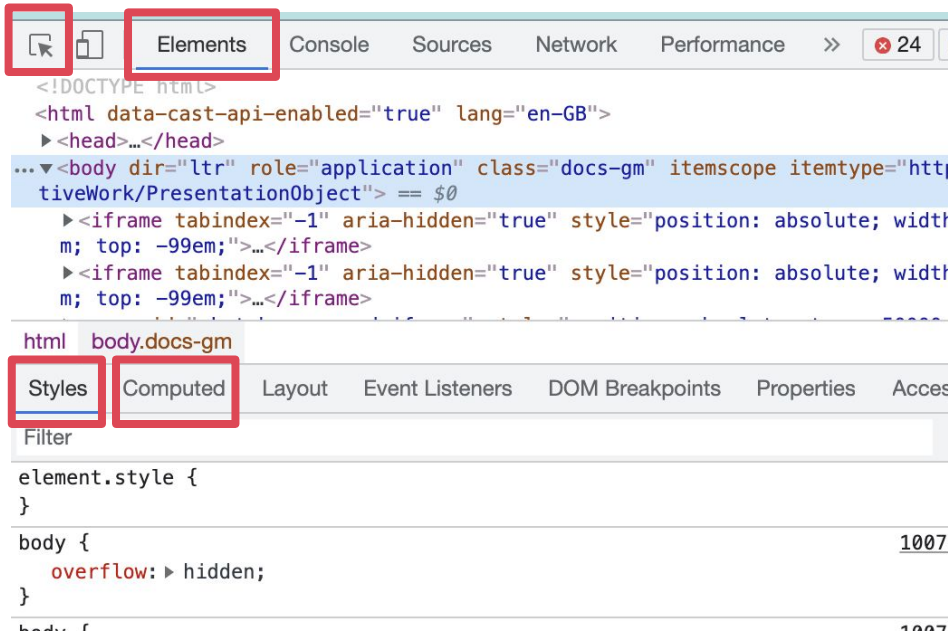
Chrome developers tools

- F12 - shortcut to open developer tools
- Inspection tool
- Elements tab
  - Styles subtab
  - Computed subtab
  - $ and $$ magic Chrome commands



Exercise - Chrome developers tools

- Go to your favourite website and try inspecting elements
- Have a look at the Styles tab and try changing some values (click at the end of a line to insert a new one)
- Changes made are only to the in-memory DOM, not to the source HTML



Head and metadata

- The `<head>` section isn't visible like other content of the page
- Used to describe the page for browsers and other interpreters such as search engines
- Page metadata - `<meta>` tag with `name` and `content` attributes:
  - `charset` and other languages, charset is always the first meta tag
  - `description` is used to display page description when displaying search results
  - `viewport` is used to define how the viewport is sized, especially for mobile
- `<link>` for the icon (favicon) and `<title>` for the page title
- `<link>` to external CSS and `<style>` for inline CSS styles
- `<script>` for both external JavaScript and inline JavaScript code
- Other types of metadata, e.g. OpenGraph or Twitter data

Institute of Data

Block elements

- Block elements occupy the entire horizontal space of their parent element, and vertical space equal to the height of its content
- Block elements can contain other block elements or inline elements (see here for more info)
- Block elements will start with a new line and end with a new line
- `<div>` is a generic block container
- More semantic block elements include `<h1>`, `<p>`, `<header>`, `<main>`, `<section>`, `<aside>`, `<footer>`
- Block elements height can be set using CSS

Institute of Data

Inline elements

- Inline elements occupy the space (height and width) equal to its content (see here for more info)
- Inline elements can contain text or other inline elements
- They neither start nor end with a new line, hence inline
- Usually used for distinguishing text within the same block
- `<span>` is a generic inline container
- More semantic inline elements include `<a>`, `<em>`, `<strong>`, `<q>`, `<label>`
- Inline elements height cannot be set using CSS

Institute of Data

Line break

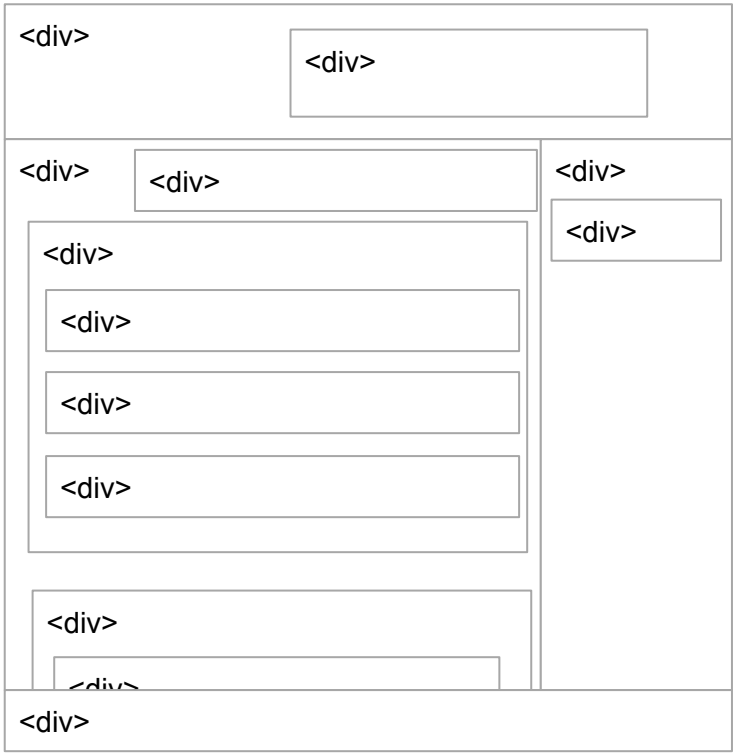
- `<br>` Creates a line break in a block of text
- Should only be used within a paragraph to shape the text - line breaks are not commonly used
- Do not use line breaks to create space in the design - this is a purely visual task and should be done with CSS
- Useful when writing poems or introducing a line break in an address
- With introduction of HTML5 both syntaxes are valid - `<br>` and `<br />` - the second is called a self-closing tag and is good practice

Institute of Data

Semantic elements

- A semantic element describes its meaning to the browser, search engine and the web developer
- `<div>` and `<span>` are generic, not semantic elements
- It is possible to style semantic and non-semantic elements so they look the same
- Using semantic elements improves accessibility of the page - visually impaired people use **screen readers** to browse the web
- Using semantic elements improves SEO (Search Engine Optimization) of the page - it ranks higher in search engines like Google

Semantic vs. generic elements



Sectioning Content

Use semantic sectioning elements to organize webpage content into logical parts. These elements create a structure to represent your page and navigate content, with headings to label each section. See the MDN reference for a complete list.

- `<header>` - introductory content at the top of the page, often including navigation, logo, branding, search form, primary contact details
- `<nav>` - a section of the page providing navigation links such as a menu
- `<main>` - the main content area inside the body of the document
- `<section>` - generic standalone section within the document, each of which should usually contain a heading tag
- `<article>` - self-contained, independent and reusable content such as a blog post, product, comment, newspaper article or card
- `<aside>` - sidebar or callout box content indirectly related to the main content
- `<footer>` - content at the bottom of the page typically containing copyright info, legal links, contact details or other related links.

Institute of Data

Headings

- Headings are wrapped with `<h1></h1>` … `<h6></h6>` elements
- Headings are used by search engines to index your page, the text inside them is especially important for SEO
- Each heading element represents a different level of content in the document: **h1** represents the main heading, **h2** represents a subheading etc.
- Preferably there's only one **h1** element per page
- Documents with too many high-level heading elements can be hard to navigate
- Headings 2-6 should be used sequentially for accessibility

Paragraphs

- Wrap each paragraph of text with a block-level `<p>` element to separate paragraphs onto new lines
- Using paragraphs will help people using screen readers navigate through the content
- Avoid using empty `<p>` tags to add spacing - use CSS margin/padding instead
- A `<p>` element may not contain other block level elements, but often contains other inline elements such as `<a>` or `<strong>`

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam orci nisi, egestas sed elit eu, feugiat placerat mauris. <p>
<p>Interdum et malesuada fames ac ante ipsum primis in faucibus. Ut eget turpis a mauris blandit porta. Phasellus vel odio odio. </p>
<p>Curabitur tempus molestie lorem quis dictum. Nulla facilisi. Nulla nec sapien nec nulla accumsan gravida sed at felis.</p>
```

HTML entities

- A HTML entity is a special string that starts with **&** (ampersand) and ends with **;**
- Entities are used to display reserved characters (&, <, >, “), invisible characters (non-breaking space) or other special characters
- `&` = **&** (reserved as beginning of HTML entity)
- `<` = **<** (reserved as beginning of HTML tag)
- `>` = **>** (reserved as end of HTML tag)
- `&quot;` = **“** (reserved as beginning of HTML attribute)
- ` ` = non-breaking space (invisible space-like character, won’t wrap)
- `&copy;` = **©** (copyright symbol)
- There are many more listed on the MDN and the HTML specification

Institute of Data

Ordered and unordered lists

- **Unordered** lists `<ul>` are used to mark up lists where order doesn't matter, they're usually represented as bullet points
- **Ordered** lists `<ol>` are used to mark up lists where order does matter, they're usually represented as numbered lists
- Both types of lists use `<li>` tags for individual **list items**
- Both types of lists can be **nested** - pay attention to where the `<li>` tags start and end
- We can change the **type** of numbering in ordered lists by using the `type` attribute for numeric (1,2,3), alpha-numeric (a,b,c) and roman (i,ii,iii) ordering.

Institute of Data

Emphasis and strong

- Emphasis and strong elements are used to stress certain words in a paragraph
- **Emphasis** (`<em>`) will make enclosed text **italic**
- **Strong** (`<strong>`) will make enclosed content **bold**
- Screen readers will read that content in a special tone of voice
- `<em>` and `<strong>` should be used in favour of the older italic (`<i>`), bold (`<b>`), underline (`<u>`) elements, as they focus on defining the **type of content** rather than **how** it should be displayed

Institute of Data

Other text elements

- Description lists (`<dl>`), terms (`<dt>`), details (`<dd>`)
- Block quotations (`<blockquote>`) and inline quotes (`<q>`)
- Citations (`<cite>`)
- Abbreviations or acronyms (`<abbr>`)
- Contact details (`<address>`)
- Superscript and subscript (`<sub>` , `<sup>`)
- Computer code (`<code>`) and preformatted text (`<pre>`)
- Times and dates (`<time>`)

See the MDN reference for a complete list with examples

Hyperlinks - anchor element

- **Anchor** element `<a>` links to another place on the Internet (hence the web)
- Link points to an URL with use of `href` attribute
- Almost anything can be turned into a link by wrapping in an `<a>` tag - text, an image or the entire section of a page
- Link can be in 5 different states: **normal**, **hover**, **focus**, **active** and **visited**
- `title` attribute will display when hovering over the link
- `target` attribute specifies where to display the linked URL (target="_blank")
- `download` attribute informs the browser about the filename of the downloaded file

Institute of Data

Comments

- Comments are used to include comments for ourselves and developers in the HTML code
- Comments aren’t displayed in the HTML page
- Comments are used to document, point out something unusual, tricky or unresolved, or clarify the start/end of various sections
- It is good practice to comment your code
- `<!-- Here goes arbitrary text - a comment -->`

Institute of Data

Multimedia elements

- Multimedia HTML tags allow us to embed various elements like **images**, **videos** or **audio** into our website
- Images and videos will occupy space equal to their size
- Historically audio and video were embedded on the web with use of plugins like flash, now we have native HTML5 `<audio>` and `<video>` elements
- Audio and video can be controlled with custom JavaScript code, or via built-in controls

Institute of Data

Image

- `<img src="images/my_photo.jpg” alt="Me on holidays">`
- `alt` attribute adds alternative text to the image that search engines and screen readers can make use of. It is also displayed for users who use turned off images to save bandwidth
- If an image is purely decorative then you should include it to the webpage using CSS `background` property, to enhance screen-readers’ users experience
- You can use `<figure>` and `<figcaption>` if you plan to add any caption to your image. These elements can also be used to annotate any other type of content, e.g. tables, audio

Institute of Data

Video

- `src`, `controls`, `autoplay`, `loop`, `muted`, `poster` and `preload` attributes can be used to customise videos
- Including a `<p>` tag inside a `<video>` tag displays fallback text when the video cannot be displayed
- Not all browsers support all video formats. We can embed the same video in different **formats** using a `<source>` tag
- Videos when resized will keep their aspect-ratio
- Auto-playing videos with sound is not recommended or supported by some browsers due to poor user experience
- It is possible to add subtitles to the video using `<track>`

Institute of Data

Audio

- `<audio>` is very similar to `<video>`
- `src`, `controls`, `autoplay`, `loop`, `muted` and `preload` attributes can be used to customise audio
- Including a `<p>` tag inside an `<audio>` tag displays fallback text when the audio cannot be played
- Not all browsers support all audio formats. We can embed the same audio in different **formats** using using a `<source>` tag
- By default audio doesn't have any display - only when using `controls` attribute are the controls are displayed

Institute of Data

Tables

Cells	

- Display structured data using `<table>`, `<tr>` and `<td>`
- Used for **tabular data** and **not for layouts**
- Single cell can span multiple rows and columns via the `colspan` and `rowspan` attributes
- `<colgroup>` can be used to style individual columns
- Accessibility of tables can be improved with `<thead>`, `<tbody>` and `<tfoot>` tags, as well as the `scope` attribute
- Generally don't respond well on smaller, vertical screens

Forms

- Forms are everywhere where we want to collect data from a user - names, passwords, files, financial transactions, register/login
- Form data is usually sent to the server for processing and storage
- Form root element
  `<form action="/order" method="POST">`
- `<input>` types are: text, password, email, number, checkbox, radio, file, color, date, range, search, submit and more - see the MDN reference
- Labelling inputs with `<label>` is very important for accessibility and SEO
- `<select>` and `<textarea>` are special types of inputs with their own tag

Institute of Data

HTML - iframe

- `<iframe>` allows us to embed content of another web page within our web page
- This is sometimes used to ensure payment security, where users provide their payment details only within the iframe of their bank or other known payment provider
- Often used for embedding YouTube videos, to take advantage of their optimised streaming
- Often used for embedding Google Maps, to take advantage of built-in zoom, pan, marker features

Institute of Data

HTML - canvas

- `<canvas>` allows us to paint arbitrary shapes, graphics and animations
- requires JS code to implement the canvas scripting API
- use `width` and `height` attributes to set appropriate size
- is not as accessible as standard HTML
- JS Paint

Lab 2.1: Working with HTML

- **Purpose:**
  - Practice using HTML elements
- **Resources:**
  - W3Schools
  - Mozilla Developers Network - HTML

HOMEWORK

- Go to the Mozilla website and read about HTML elements - HTML elements reference
- Read about HTML forms - Web form building blocks
- Practice using semantic elements to structure the page and markup content

Front-end development basics

Module 2
Part 3

CSS

Institute of Data

What is CSS?

- Styling language that is used to alter the display of web pages
  - appearance - changing colours, fonts, sizes, etc
  - layout - arranging elements on the page
- CSS stands for Cascading Style Sheets - multiple styling rules are applied in order of precedence and specificity
- Many different ways of achieving the same result
- CSS syntax

```
selector {
  property: value;
  property: value;
}
```

CSS Selectors

- CSS Selectors are used to target or define the HTML elements that you want to style
- There are four types of selectors: **class**, **type**, **id** and **attribute** selectors
- Most common and most useful is the **class** selector
- Universal selector ***** selects all HTML elements
- Selectors can be combined together (without spaces) to create more specific selectors for the same element, or with a comma to apply the same rules to multiple elements

```
.class-selector {}
#id-selector {}
div /* type selector */ {}
input[type] {} /* attribute selector - inputs with defined type attribute */
a[title=”institute of data”] {} /* attribute selector*/
* {} /* universal selector */
div.class-selector#id-selector {} /* specific selector matches type, class and id */
h1, h2, h3, h4 {} /* selects multiple heading elements */
```

Selectors - Combinators

- Combinators allow us to style elements in relation to their position in the DOM
- Descendant combinators use a **space** to separate ancestor-descendant elements
- Child (direct descendant) combinators use a **>** to select only direct children of the parent (other matching descendants won't be selected)
- Pseudo-classes allow us to style elements in a certain state, e.g. a link that is hovered
- Pseudo-elements allow us to style certain parts of HTML elements, e.g. first line of a paragraph or generate content **before** or **after** an element using CSS

```
.class-selector div {} /* descendant combinator - targets <div> inside class-selector */
.parent > .child {} /* direct descendant - targets child class directly inside parent */

a:hover {} /* :hover pseudo-class targets only links that are hovered */

.title::before { content: '-'; } /* pseudo-element inserted before each .title */
```

Institute of Data

Specificity

- When styling elements with different selectors, the browser must decide which CSS style (rules) to apply in a scenario of conflicting styles
- More specific selectors take precedence over less specific ones
- Type selectors < class selectors < id selectors < inline style
- Avoid overly specific selectors
- When classes of equal specificity conflict, the last one defined in the CSS file takes precedence
- Adding `!important` to any CSS rule will force it to take precedence and override natural specificity ordering
- Avoid using `!important` unless absolutely necessary, eg. to override inline styles

Institute of Data

Naming conventions

- Traditionally we name CSS classes using **hyphen-case**, but we can also use **camelCase** or **PascalCase**
- Object Oriented CSS - `.leftPanel .logo-image`
- Functional CSS - `.m-2`, `.h-20`, `.row`, `.border-1`, `.underlined`
- Use small but meaningful names to reduce file size
- When working on a big project, we split parts of the website into smaller pieces and then use class names that are relevant to the given section, e.g. `.container`, `.menu`, `.sidebar`
- For small projects it is common to use **type** selectors
- Keep class names **consistent** by adhering to the project style guide

Examples

```
<header>
   <nav>
       <ul class="menu" id="header-nav">
           <li>Menu Item #1</li><li>Menu Item #2</li>
       </ul>
   </nav>
</header>
<main class="content">
   <section id="about">
       <h1>About</h1>
       <ul><li>bullet point</li></ul>
       <button>Click Here</button>
   </section>
</main>
```

Place this HTML inside the `<body>` tags of a new file called `mod2css.html`, and add another section with a unique id, heading, content and a button.

```
body { background: lightgray; font-family: sans-serif; }
body, ul, li { margin: 0; }
header { background: midnightblue; color: white; }
nav ul.menu { list-style: none; padding: 0; display: flex; }
nav ul.menu li { padding: 1em; }
main { max-width: 80%; margin: 1em auto; background: white; }
section#about { border: 1px solid grey; padding: 1em; }
button { background: midnightblue; color: white; font-weight:
bold; padding: 0.5em 1em; }
#about button { font-size: 18px; }
```

Examine each of the different selectors and work out which HTML elements they target. Next, add the CSS to the HTML file and verify that your understanding is correct.

- Remove the second line above and observe what changes
- Why do the two `<ul>`s appear differently?
- How are the two buttons alike and different - why?

Institute of Data

Inheritance - initial and inherit

- `initial` and `inherit` values can be applied to any CSS property
- `initial` will set the value of the property to its default value
- `inherit` will make an element inherit the property value from the parent
- All CSS properties can be categorised as either **inherited** (inherits styles from parents, eg. font, color) or **non-inherited** (initial or default property value unless specified, eg. border, margin)
- form inputs do not inherit font styles from parents by default

```
p {
 border: 1px solid green;
 color: red
}
```

The quick brown fox jumps over the lazy dog

The child tag content above will inherit the color but not the border

Institute of Data

Dimension units

- Dimension units are used with properties like **width**, **margin**, **padding**, **font-size** to specify **fixed** or **relative** sizing
- Most common dimension units are `px` or `pt` (fixed size), `em`, `rem`, `%`, `vh`, `vw` (relative size)
- Good practice to define a base font size in px in the **body**, then use em or rem for margin/padding relative to the font size
- `1em` will be the size of the parent element font, `1rem` will be the size of the root element (html) font
- `vh` (view-height) and `vw` (view-width) are used to define elements in relation to the size of the viewport, eg. `100vh` is 100% or the full height of the browser viewport

Institute of Data

Display

- Allows us to override default **block** vs **inline** element display, as well as define other custom display models such as flex and grid
- `display: none` will hide the element completely
- One value syntax (traditional):
  display: block|inline|inline-block|flex|inline-flex|grid|inline-grid
- Two value syntax (new, see the MDN definition):
  display: [block|inline][flow|flex|grid], e.g. display: block flex;
- Controls both the **outer** display of the element itself and the **inner** display of its children

Position

- **position**: `static (default) | relative | absolute | fixed | sticky`
- Defines the position of an element, specified using **top** / **right** / **bottom** / **left** offsets
- **Absolutely** positioned elements are removed from the document flow and explicitly positioned using **offset** properties, eg. `top: 50px` or `right: 0`
- If relative position is set on the parent, then all the absolutely positioned children are positioned in relation to the **parent** - otherwise they are relative to the **page**
- **Fixed** positioned elements don't scroll, but are fixed relative to the viewport, eg. a header always at the top or footer always at the bottom, even when scrolling
- **Sticky** elements activate when scrolling the page - as long as the parent is in view, the sticky element will remain in position as specified by any offset properties

Institute of Data

Colors

- Colors are used to color backgrounds, borders, outlines and text (font, shadow, underline)
- Named colors - `red`, `green`, `slateblue`
- RGB functional notation - `rgba(255,99,71,0.7)`, `rgb(255,99,71)`
- Hexadecimal string notation - `#rrggbb`, `#rgb`, `#rrggbbaa`, `#rgba`
- R = red, G = green, B = blue, A = alpha. Alpha indicates transparency, where 0, or 00, is a fully transparent color, and f, or ff, is fully opaque
- Colors and accessibility - ensure sufficient contrast for legibility
- Palette generators (eg. coolors) can create accessible, attractive colour schemes using appropriate contrast

Styling text

- Font styles - `font-family`, `font-size`, `color`, `font-weight`, `text-transform`, `text-decoration`, `text-shadow` (experiment here)
- Text layout styles - `text-align`, `line-height`, `letter-spacing`, `word-spacing`
- Font families - `sans-serif`, `serif`, `monospace`, `cursive`, `fantasy`
- Default fonts - Arial / Verdana / Tahoma (sans-serif), Times New Roman / Georgia / Garamond (serif), Courier New (monospace), Brush Script MT (cursive)
- Font stacks - comma-separated list of fonts with fallbacks
- Google Fonts is an excellent library of licensed, web-safe fonts to extend the defaults
- `@font-face` rule can be used to load custom fonts in the WOFF/2 format
- When converting/using custom fonts be sure to check the licensing agreements

Background

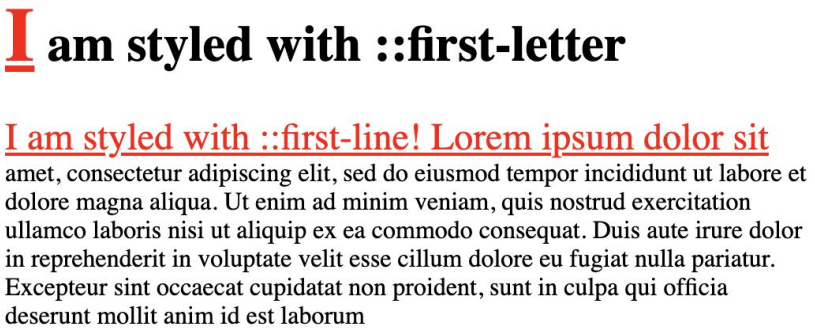
- Decorative images should be included using the CSS `background` property, rather than the `<img>` tag, because they belong to appearance and not content. See the MDN Reference for examples
- A lot of `background` properties are used to position and scale the background image of an HTML element in a way that it will always show the part of an image that's relevant, e.g. on mobile we need to scale down and position the image
- It is possible to add multiple backgrounds to one HTML element
- Gradients can fill the background of an element with multiple colours

Styling links with pseudo-classes

- Link selectors - `a`, `a:link`, `a:hover`, `a:active`, `a:visited`, `a:focus`
- The colon `:` indicates a **pseudo-class**, used to differentiate links in different states without needing actual classes (see here for more info)
- By default links are underlined, unvisited links are blue and visited are purple. Hovering a link changes cursor to a hand
- You can navigate links on any page just by pressing the Tab key and focusing on different links
- Good practices when styling links:
  - underline or highlight in some other way
  - make them react when hovered or focused
  - change cursor when hovering over the link

Pseudo-elements and CSS-generated content

- Pseudo-elements - `::before`, `::after`, `::first-letter`, `::first-line` - these don't exist in the HTML, only in CSS for styling purposes
- Can be used with other selectors, e.g `div::after` or `.class::first-line`
- Generate content by combining the CSS `content` rule and `::before` and `::after` pseudo-elements - can be used for text or images/icons
- Double colon `::` is for **pseudo-elements**, single colon `:` is for **pseudo-classes**
- Style the first letter or first line of text
- Experiment here



Institute of Data

Calculate CSS property value - `calc()`

- Supports 4 operators: `+`, `-`, `*`, `/` and parentheses e.g.

```
calc(100% - (40px + 5% / 2))
```

- Uses standard mathematical operator precedence rules
- length, angle, percentage, number, time or any integer can be the result of a calculation using the `calc` function
- `+` and `-` characters must be surrounded by whitespaces, it's recommended to also surround `*` and `/`
- `calc()` makes it easy to set a dynamic width of an element with a fixed margin and/or border (see here for an example)

Lab 2.2: Working with CSS

- **Purpose:**
  - Learning and using various CSS rules
- **Resources:**
  - W3Schools
  - Mozilla Developers Network - CSS

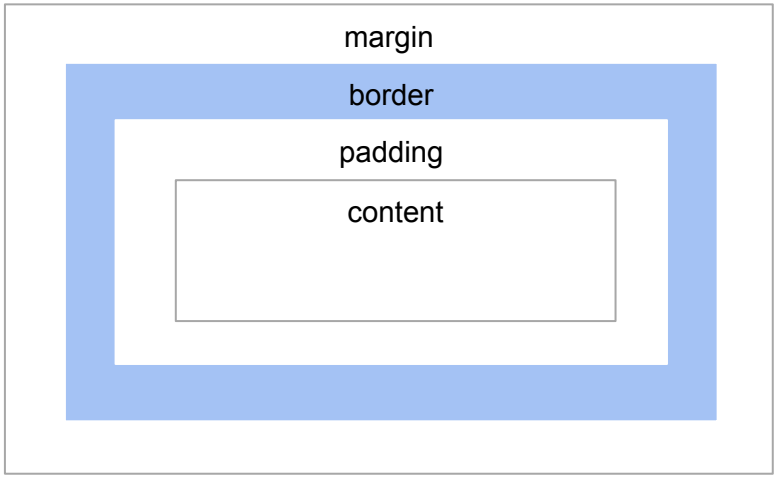
Institute of Data

Layout - normal document flow

- Normal layout document flow consists of block and inline elements
- `block` - vertically-based
- `inline` - horizontally-based
- `inline-block` - horizontally-based with some of the block properties to allow sizing, margin, border
- Important to start with a solid, well-structured HTML document which is readable in normal flow, then start overriding and customising layout with CSS
- Floats - legacy way of creating layouts with horizontal blocks

Box sizing properties

- **margin** / **padding**: 5px | 5px 10px | 0 10px 5px -10px;
  - the one, two or four value syntax specifies values for all four sides of the box
  - margins can be negative
- **width**: 50px; **height**: 100px;
- **min-width**: 20px; **max-width**: 100vw;
- **min-height**: 100vh; **max-height**: 200px;
- **box-sizing**: border-box | content-box
- **border-radius**: 10px | 50%;
- **outline** / **border**: 1px solid blue;



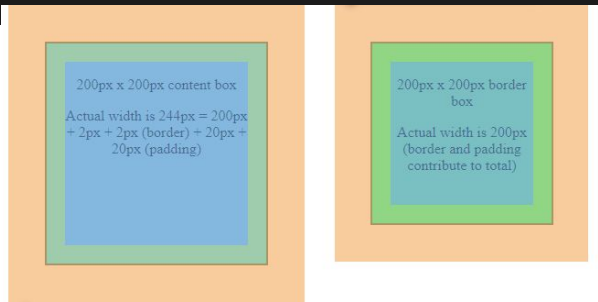
All of the above properties contribute to the overall size of the box.

See MDN on Sizing for more info.

Box model

- **box-sizing: content-box** is the browser default method of calculating the total width and height of any block-level HTML element (see here for more details)
  - Takes the width and height of the block, then adds on padding and border
  - Width can be greater than expected, eg. if using 25% width for four equal columns with padding
- **box-sizing: border-box** is an alternative model that includes padding and border in the total width
  - More intuitive, especially for column-based layouts, often specified using the universal selector:
    `* { box-sizing: border-box; }`

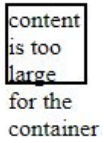
```
<style>
   .box { padding: 20px; border: 2px solid
black; margin: 40px; width: 200px; height:
200px; text-align: center; }
   .content-box { box-sizing: content-box;
background: lightblue; }
   .border-box { box-sizing: border-box;
background: lightgreen; }
</style>

<section class="content-box box">
   <p>200px x 200px content box</p>
   <p>Actual width is 244px = 200px + 2px +
2px (border) + 20px + 20px (padding)</p>
</section>

<section class="border-box box">
   <p>200px x 200px border box</p>
   <p>Actual width is 200px (border and
padding contribute to total)</p>
</section>
```



Overflow

If the content inside a box is larger than its defined height and/or width, it can overflow the container. We can control how this overflow behaves with CSS.

- CSS doesn't hide overflow content by default, but allows it to display outside its container.




```
<section style="border: 2px solid black; width: 50px; height: 50px;">
   content is too large for the container
</section>
```

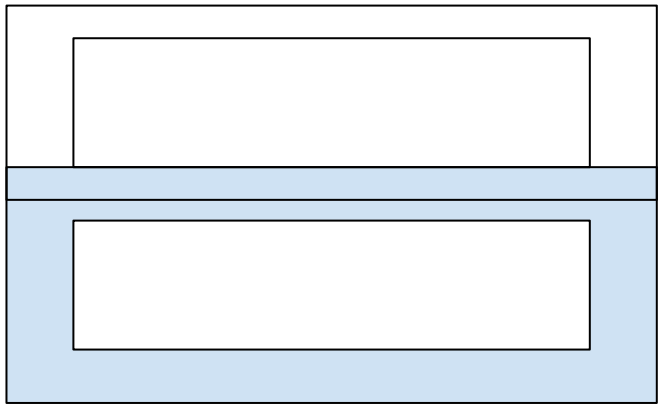
Fixed size containers are generally not recommended if there is potential for overflow; instead use min-width/min-height and/or relative sizing units such as em or %

Overflow alternatives (see here for more detail):

- `overflow: hidden;` Hides any overflow and prevents it being visible
- `overflow: scroll;` Adds vertical and horizontal scrollbars to the container
- `overflow-x / overflow-y: scroll;` Adds only vertical or horizontal scrollbar
- `overflow: auto;` Adds scrollbars only when content overflows

Margin collapsing

- The top and bottom (vertical) margins of a block element are sometimes **collapsed** into a single margin whose size is the **larger** of the two margins
- This applies to **adjacent sibling** elements (those next to each other in the DOM), and also to **empty block** elements, where the top and bottom margins will collapse into a single (larger) value instead of adding together



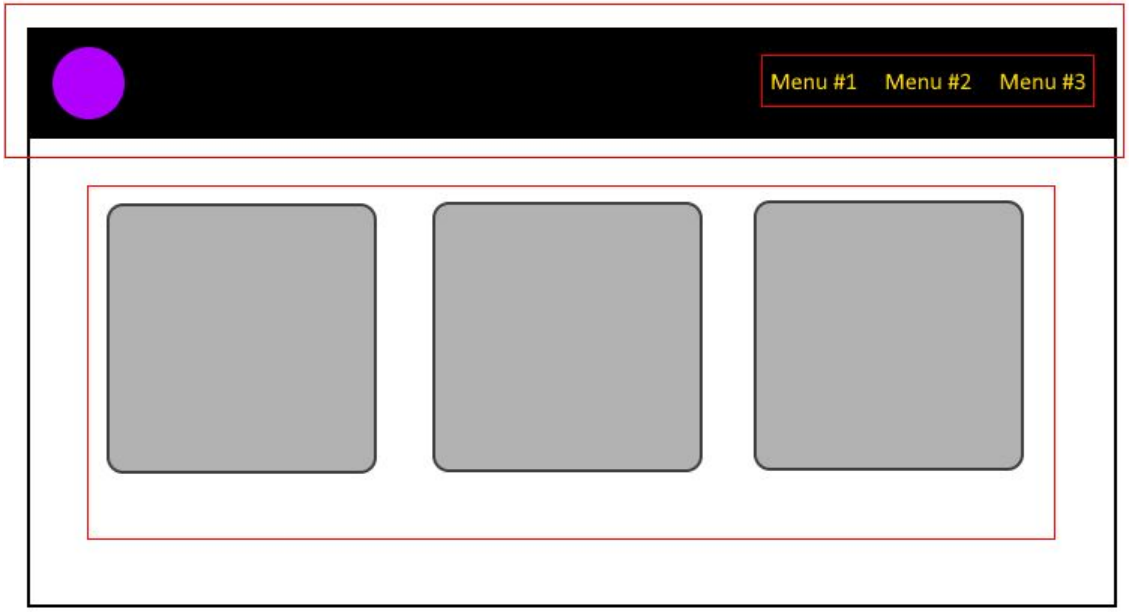
Collapsed margins

- There are exceptions to this rule, such as when using `display: flex` or `display: grid` (see here for more info)

Flexbox Layout

- Modern, efficient way to create layouts, columns and any horizontal block elements
- Aligns items vertically, horizontally or both
- Easily defines how to distribute remaining space by either increasing the spacing or enlarging or shrinking the items
- Works well with elements of unknown or dynamic size
- Responds well on screens of various sizes, giving the flex container the ability to alter the width/height and order of its items to best fill the available space
- See A Guide to Flexbox for details: `display: flex;` is enough to start with

Flexbox Layout Class Exercise



The basic desktop web layout to the left uses horizontal block elements. It can be built using several flexbox containers, indicated by the red boxes.

Using the starter code on the next slide, add CSS to make each of the 3 areas a **flex container** which distributes extra space as needed to achieve the layout.

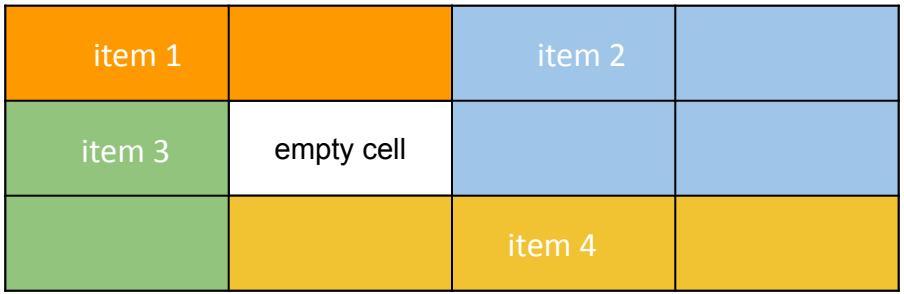
Experiment with flexbox rules such as `justify-content`, `align-items`, `flex-wrap`, `flex-direction`, `order`, `gap` *and* `flex-grow` *to see how each behaves.*

Flexbox Layout Class Exercise

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flexbox Demonstration</title>
    <style> <!-- adjust these styles to use flexbox and experiment with various property values -->
        body { margin: 0; font-family: sans-serif; font-size: 18px; }
        header { background: black; padding: 1em; }
        #logo { width: 60px; height: 60px; border-radius: 50%; background: #B200FF; }
        .menu { list-style: none; padding: 0; margin: 0; color: yellow; }
        main { max-width: 80%; margin: 1em auto; }
        .box { border-radius: 1em; min-width: 300px; min-height: 300px; border: 2px solid #404040; background: #B2B2B2; margin: 1em; }
    </style>
</head>
<body>
    <header> <!-- flex container #1 -->
        <div id="logo"></div>
        <nav id="main-menu">
            <ul class="menu"> <!-- flex container #2-->
                <li>Menu #1</li><li>Menu #2</li><li>Menu #3</li>
            </ul>
        </nav>
    </header>
    <main>
        <section id="boxes"> <!-- flex container #3 -->
            <div class="box"></div><div class="box"></div><div class="box"></div>
        </section>
    </main>
</body>
</html> <!-- this CodePen also demonstrates flexbox properties and usage: https://codepen.io/chriscoyier/pen/kNZRER -->
```

CSS Grid Layout

- Two-dimensional grid-based layout over both rows AND columns - as opposed to flexbox which is one-dimensional (row OR column)
- Grid can do things Flexbox cannot do and vice versa
- Older versions of IE have limited support, but most modern browsers support it well
- Use `display: grid;` to start with, then customise using various `grid-` properties
- See A Guide to CSS Grid



CSS Grid Layout

- CSS Grids use a `grid-template` to define the number and proportional size of rows and columns in the grid
- `fr` is a special **fr**actional sizing unit used to flexibly assign space, in conjunction with keywords such as `min-content`, `max-content` and `auto`
- Grids use several in-built CSS functions to layout content responsively:
  - `fit-content()` fits content between min and max limits
  - `minmax()` is extremely useful for relatively constraining grid item length
  - `repeat()` saves typing when setting up a template with many rows/columns
- Grids can use either fixed or flexible templates; they can be complex or simple
- See these demos: fixed grid, flexbox grid, flexible grid

Lab 2.3: Working with CSS layouts - flexbox

- **Purpose:**
  - Learning and using the most important concept in layouts - flexbox
- **Resources:**
  - CSS tricks guide to flexbox

Lab 2.4: Build your own website

- **Purpose:**
  - The goal of this lab is to build your own resume-website from scratch
- **Resources:**
  - W3Schools
  - Mozilla Developers Network - HTML
  - Mozilla Developers Network - CSS

Front-end development basics

Module 2
Part 4

Responsive CSS and advanced CSS concepts

CSS variables (Custom properties)

- Define CSS variables (usually within the `:root` pseudo-element) that can be reused throughout the website
- Color system - easy way of reusing (and updating) main website colors
- Design system - easy way of reusing CSS stylings to provide a consistent look & feel to the website
- Set using custom property notation (`--prop-name`) and accessed using `var()` function
- Can't be used inside media queries or container queries
- Practice [using CSS variables here]

```
:root {
  --primary-color: #FFEE77;
  --secondary-color: #77EEFF;
  --border-1: solid 5px var(--secondary-color);
  --spacing: 5px;
}
```

```
.box {
  color: var(--secondary-color);
  background: var(--primary-color);
  border: var(--border-1);
  padding: calc(var(--spacing) * 4);
}
```

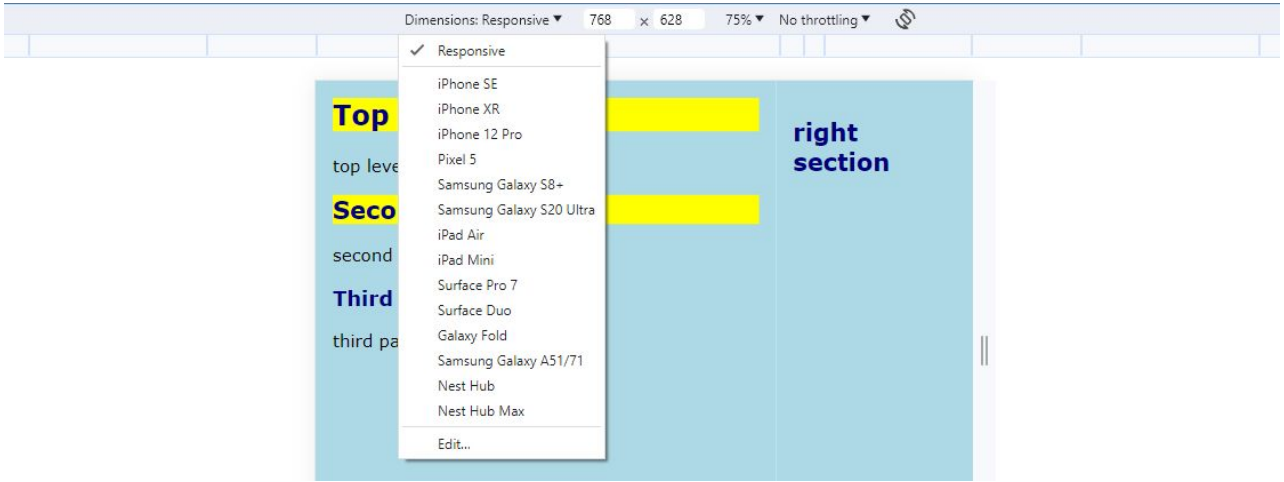
Institute of Data

Responsive design - fluid layout

- Goal of responsive design is to ensure that a website resizes naturally and looks good on all devices, from small mobiles to large 4k displays
- Modern web developers should aim to use code that automatically adjusts, but still need to consider different layouts for different devices
- You can use developer tools to emulate different screen sizes in the browser, or just resize the browser window
- It's common to use a different set of font-sizes for mobile and desktop, and to use relative margin, padding and sizing units such as `em` or `rem` to adjust based on the font size

Chrome developer tools - device toolbar

- The ‘device toolbar’ in Developer Tools can be used to emulate screens with smaller or bigger resolution than our actual screen, at common sizes
- Useful for developing and testing websites on different devices



Institute of Data

Mobile-first vs. desktop-first

- **Desktop-first** design (traditional) involves designing and building webpages primarily for larger, landscape-oriented desktop and laptop sized screens, then adjusting for smaller mobile screens
- **Mobile-first** design (modern) targets primarily portrait-oriented mobile devices, and adjusts as needed for larger screens
- Why **mobile-first** or **desktop-first**?
    - Website owners know what devices are used by their customers and try to direct their approach in a way that will generate the best conversion rates
    - If some UX problem can be solved on mobile then it also can be solved on the bigger devices

Responsive CSS - media queries

- Media queries let you apply CSS only when the user's device and browser setup matches the condition (see MDN Media Queries for more information)
- The most common @media queries are `min-width` and `max-width` that test the viewport size, but there are also others that work with environment properties, like whether the device is a touchscreen or a printed version of the page
- Use of `min-width` is associated with the mobile-first design approach, `max-width` with desktop-first - usually you'll try to use only one type of media-query in a project
- `media-type` can be one of `screen`, `print`, or `all` (screen is most common)

```
@media screen and (min-width: 800px) and (max-width:
1200px) {
    body {
       font-size: 15px;
    }
}
```

```
@media media-type and (media-feature-rule) {
  /* CSS rules to apply */
}
```

Responsive CSS - breakpoints

- Breakpoints are defined screen widths at which different media queries are activated
- Custom breakpoints should be chosen at the points where layout begins to break
- In practice it is common to use default breakpoints from CSS frameworks such as Bootstrap
- To cope with half-pixel accuracy on displays like retina, you can subtract a fraction out of max-width media-query
- Breakpoints are associated with different devices like mobile, tablet, laptop, desktop, TV etc.

name	value	typical device	min-width	max-width (notice the subtracted fraction)
xs	0px	phone (portrait)	min-width: 0px	-
sm	600px	phone (landscape)	min-width: 600px	max-width: 599.98px
md	960px	tablet	min-width: 960px	max-width: 959.98px
lg	1280px	laptop/desktop	min-width: 1280px	max-width: 1279.98px
xl	1920px	desktop	min-width: 1920px	max-width: 1919.98px

Institute of Data

Responsive images

- We want to display higher resolution images on larger screens and smaller resolution images in smaller screens
- There are two approaches to responsive images: CSS and HTML based
- CSS-based approach uses `background-image` and `@media` queries
- HTML-based approach uses `<img>` tag with `srcset` and `sizes` attributes - see Responsive Images on the MDN
- Art direction - changing an image to a different one, or a different crop of the same image, to suit needs of different displays
- For decorative images use CSS background property, for images that convey certain meaning use HTML `<img>` tag

Institute of Data

Reset CSS

- Not all the default CSS settings of the browser are the same
- Developers include reset CSS on their websites to ensure that initial CSS for all the websites is the same
- Reset CSS can be used to remove any extra styling from all the elements or leave some of the default styles
- Provides a simple, reliable baseline to build upon
- Common resets: Eric Meyer, Normalize, Josh Comeau
- Lightweight file of well-tested CSS rules to simplify development and maintain consistency across browsers

Institute of Data

Design tools

- Very often in your web developer career, the designs for the websites to build will be provided by the UI designer. Those are usually shared via design tools like Figma or Adobe XD
- Quite often, a lot of CSS can be copied from the design tools - fonts, colours, borders, text styling, margin, padding; but unfortunately not layouts
- Layouts should be based on well-structured semantic HTML that matches the design, using CSS flexbox or grid models
- Popular design tools include - InVision, Figma, Sketch, Adobe XD

Lab 2.5: Build your own responsive website

- **Purpose:**
  - The goal of this lab is to add responsive media-queries to the website you built in the previous exercise to ensure it looks great on both desktop and mobile
- **Resources:**
  - CSS tricks - guide to media queries

Institute of Data

Links

- HTML Element Reference
- Web technology for developers
- CSS- tricks
- Stack Overflow