

# HTML CSS

- HTML and DOM
- CSS selectors
- Flexbox layout
- DOM manipulation with TypeScript

# HTML

- HTML stands for **H**ypertext **M**arkup **L**anguage
  - defines the meaning and structure of the interface  
i.e. widgets, widget containers, content
- Uses a *declarative* syntax

compared to *imperative* syntax which we used in SimpleKit so far



HTML (1990)  
**Tim Berners-Lee**

# html

- basic document
  - DOCTYPE, <html>
- <head>
  - <title>, meta elements for character set and viewport
- <body>
  - app content



# HTML Tag, Attribute, and Element

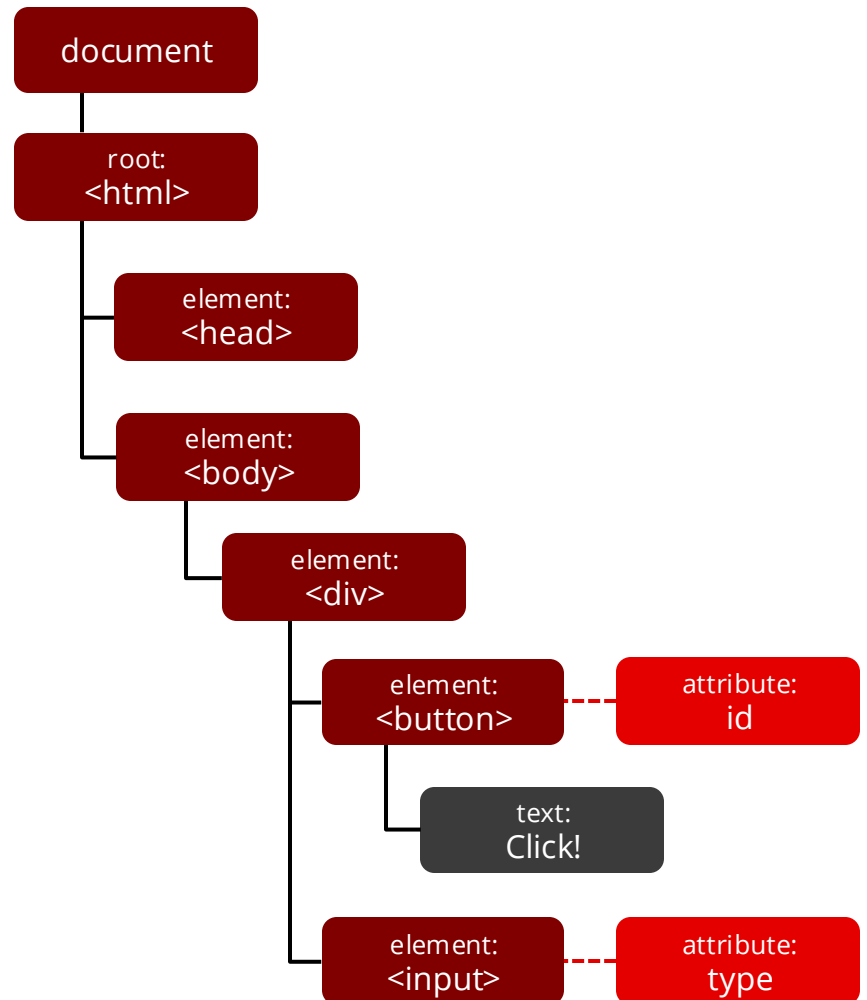
- The **tag** is syntax that defines an element and its attributes
  - `<tag>inner content</tag>`
  - `<tag />`
  - `<tag>`
- An **attribute** is extra information to define elements
  - `<tag attribute="information">`
- The **element** is what the tag and attributes create, e.g.:
  - `<input type="text" />` creates a textfield element
  - `<div>inner content</div>` creates a container element
  - `<button id="b" >Click!</button>` creates button element

The **id attribute** is a unique name for an element in the current page

# Document Object Model (DOM)

- A cross-platform and language-independent interface that treats an HTML document as a *tree structure* of node objects representing a part of the document
- Every element in the DOM is a *node*:
  - A web page is a **document** node
  - All HTML elements are **element** nodes
  - All HTML attributes are **attribute** nodes
  - Text enclosed by HTML elements are **text** nodes
  - Comments are **comment** nodes

```
<!DOCTYPE html>
<html lang="en">
  <head>
    ...
  </head>
  <body>
    <div>
      <button id="b">Click!</button>
      <input type="text" />
    </div>
  </body>
</html>
```



## <div> and <span>

- Generic HTML container widgets
- The div tag is a *block-level* element used for associating and grouping together nested elements
  - used very often, it's like SKContainer
- The span tag is an *inline element* used for associating and grouping together nested elements
  - Often used for styling

# HTML Widgets

- `<button name="...">Hello</button>`
- `<input type="..." >`
  - "text" (for a textfield)
  - "checkbox"
  - "range" (for a slider)
  - ~~"button" for a button~~
- `<label>` for label associated with a widget
- `<textarea>` for editable multiline text
- `<select>` or `<datalist>` with `<options>` for a menu



# CSS

- CSS stands for **C**ascading **S**tyle **S**heets
  - visual style (colour, font, ...)
  - layout
  - animation
- Like HTML, uses a *declarative* syntax



CSS (1994)  
**Håkon Wium Lie**

# CSS Rule

- A *CSS rule* has three parts
  - selector
  - declaration block
  - one or more properties with values

```
div > div#b {  
  background-color: deepskyblue;  
  padding: 10px;  
}
```

The diagram illustrates the components of a CSS rule. A yellow callout box labeled 'selector' points to the text 'div > div#b'. The opening curly brace '{' is enclosed in a yellow box. Below the rule, two yellow callout boxes are present: one labeled 'property' pointing to 'padding' and another labeled 'value' pointing to '10px'.

# Where to Specify CSS

1. In HTML tag using element style attribute

```
<div style="padding: 10px;">
```

no selector  
needed

2. Inline in HTML document as a <style> element

```
<style>  
  div { padding: 10px; }  
</style>
```

3. link to file

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

in "style.css" file:

```
div {  
  padding: 10px;  
}
```

# CSS Selector

A pattern to select (or *find*) elements in the DOM

- Basic selectors

`div` { ... } select by tag type

`.foo` { ... } select *by class name*

`#a` { ... } select by *element id attribute*

`[type="text"]` { ... } select elements matching *attribute value*

- Hierarchical selectors

`div > div` { ... } select child elements by *parent-child relationship*

`div div` { ... } select child elements by *descendant relationship*

- Pseudo class selectors (there are many more ...)

`:first-child` select first child

`:hover` select when mouse is over element

- Combining selectors

`div#a` { ... } select elements matching all selectors

`div, #a` { ... } select elements matching at least one selector

Click a selector:

```
.intro
#Lastname
.intro, #Lastname
h1
h1, p
div p
div > p
ul + p
ul ~ table
*
p.myquote
[id]
[id=my-Address]
[id$=ess]
[id|=my]
[id^=L]
[title~=beautiful]
[id*=s]
:checked
:disabled
:enabled
:empty
:focus
p:first-child
p::first-letter
p::first-line
p:first-of-type
h1:hover
input:in-range
input:out-of-range
input:invalid
input:valid
p:lang(it)
p:last-child
p:last-of-type
tr:nth-child(even)
tr:nth-child(odd)
li:nth-child(1)
```

**Selector:**

`h1, p`

Selects all `<h1>` elements and all `<p>` elements.

Result:

```
<h1> Welcome to My
Homepage </h1>
<div class="intro">
<p> My name is Donald <span id="Lastname"> Duck. </span> </p>
<p id="my-Address"> I live in Duckburg </p>
<p> I have many friends: </p>
</div>
<ul id="Listfriends">
  • <li> Goofy </li>
  • <li> Mickey </li>
  • <li> Daisy </li>
  • <li> Pluto </li>
</ul>
<p class="myquote"> All my friends are great! <br>
But I really like Daisy!! </p>
<p lang="it" title="Hello beautiful"> Ciao bella </p>
<h3> We are all animals! </h3>
<p> <b> My latest discoveries have led me to believe that we
are all animals: </b> </p>
<table>
  Name Type of Animal
  Mickey Mouse
  Goofey Dog
```

Click the CSS Selectors and see the specified element(s) get selected. © w3schools.com

## CSS Selector Demo

- <https://www.w3schools.com/cssref/trysel.php>

# The CSS Cascade

- Defines the precedence of CSS rules when multiple declarations can apply to the same element

in your app

- Who specified: *agent (browser)* → **author** → *user*
- Where specified: *style.css or <style>* → inline style attribute
- When specified (*order* of rules in inline css or files)
- Importance using `!important`
- *Specificity* of rule

Also "layers", but we won't go into that

- **Specificity** is a standard method to determine which CSS rule declaration is most relevant to an element

- Essentially, the most specific CSS selector sets the style, e.g.

```
div#a { background-color: blue; }
```

```
div { background-color: red; }
```

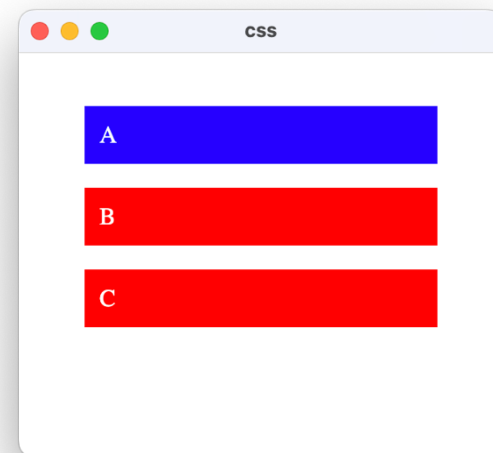
...

```
<div id="a">A</div>
```

div will be blue

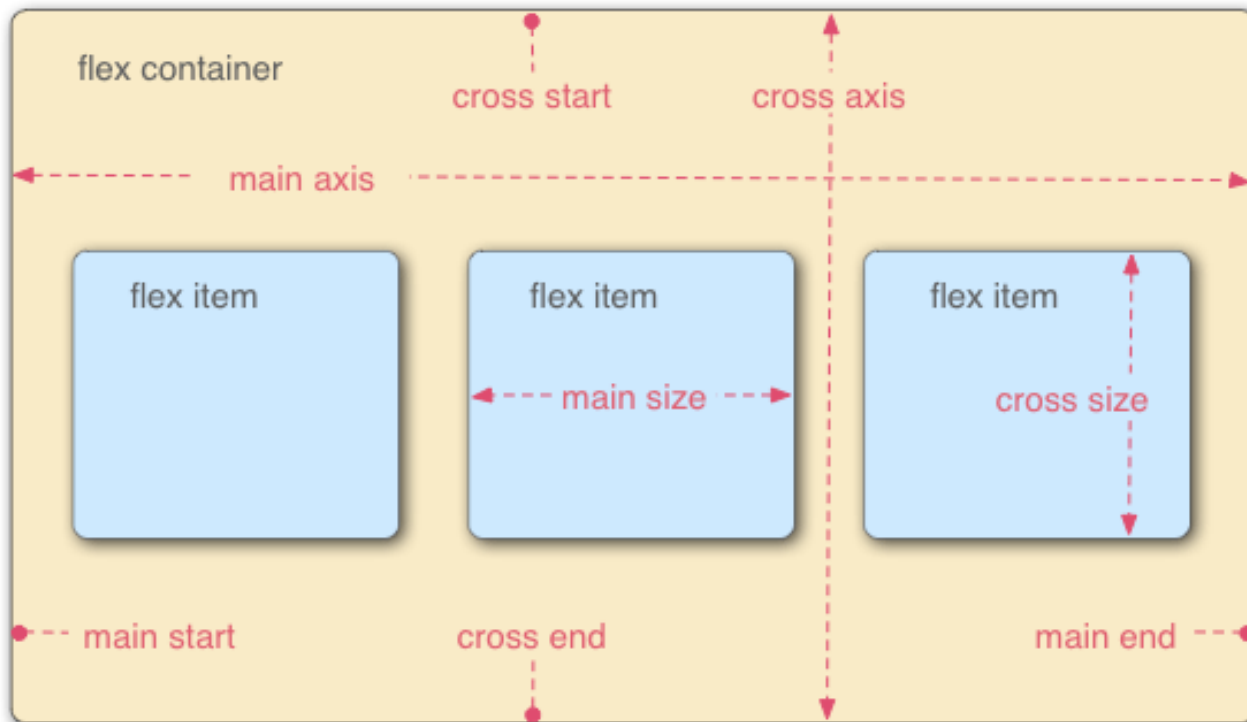
# CSS

- Property demos
  - set background-color and border
  - set width and height
  - set padding and margin
- Cascade demos
  - Create overlapping rules (try inline, try file, try both)
  - Change their order; try adding !importance
- Selector demos
  - by tag type; by class, by id, by attribute value
  - by hierarchy
  - pseudo class like hover and first-child
  - Combine selectors, multiple selectors



# CSS Flexbox Layout

- main axis (row or column)
- cross axis (perpendicular to main axis)
- flex container
- flex items





# Using Flexbox Layout

- Make parent a “flex container”
  - set CSS display property to “flex”  
`display: flex;`
- Children become “flex items”
  - CSS properties for flex items control growing, shrinking, etc.
  - CSS properties for parent control flex item alignment, gap, etc.

# flexbox items: grow, shrink, and basis

- **flex-grow**

- proportion to grow element to fill space
- 0 means don't shrink

- **flex-shrink**

- proportion to shrink element to fit into space
- 0 means don't shrink

- **flex-basis**

- *auto* means use "width or height" if set, else use content size
- number means use that as the "basis"

- *flex shorthand* (flex: grow shrink basis)

```
flex: 1 2 auto /* grow 1, shrink 2, basis auto */
```

```
flex: 1 /* grow 1, shrink 1, basis auto *
```

- *default:*

```
flex: 0 1 auto / * same as flex: initial */
```

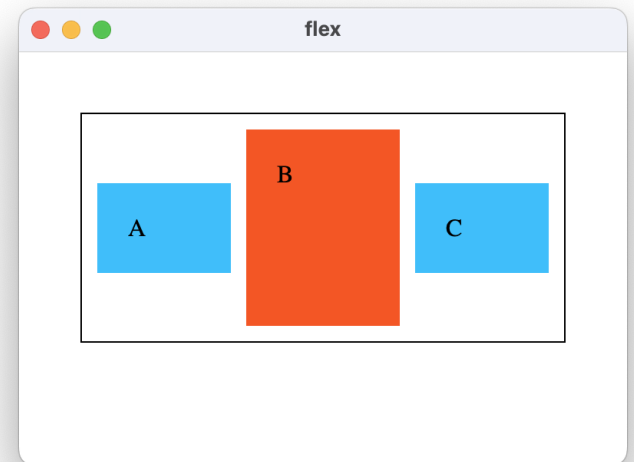
# flexbox alignment, justification, distribution

- **gap** to set gap between items
- **align-items** for container, how items align along *cross axis*  
align-items: < stretch, flex-start, flex-end, center >
- **align-self** for item, how it aligns along *cross axis*  
align-self: < stretch, flex-start, flex-end, center >
- **justify-content** for container, how items align on *main axis*  
justify-content: < flex-start, flex-end, center,  
space-between, space-around, space-evenly >

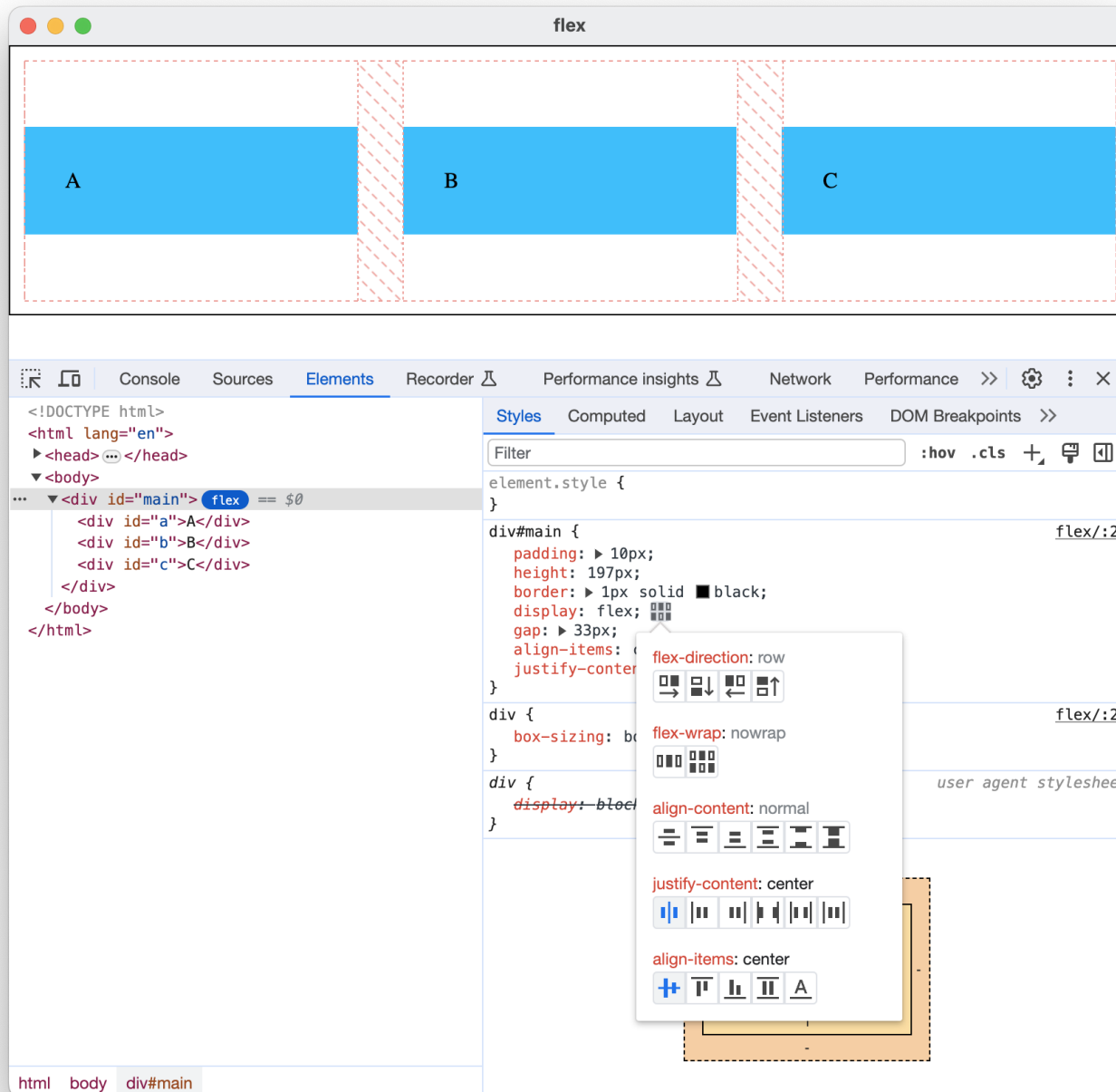
# flex

- Demos

- try different grow, shrink, basis
- try different align-items: stretch, flex-start, flex-end, center
- try different justify-content: flex-start, flex-end, center, space-between, space-around, space-evenly
- try changing item width
- override flex for div B



# Chrome Devtools Flexbox Visualization and Adjusting



The screenshot displays the Chrome Devtools interface with a browser window titled "flex" showing three blue rectangular items labeled A, B, and C arranged horizontally. The items are separated by vertical dashed lines with a diagonal hatching pattern, representing the flex gap. Below the browser window, the Elements panel shows the DOM tree with the following structure:

```
<!DOCTYPE html>
<html lang="en">
  <head> ... </head>
  <body>
    <div id="main"> flex == $0
      <div id="a">A</div>
      <div id="b">B</div>
      <div id="c">C</div>
    </div>
  </body>
</html>
```

The Styles panel shows the following CSS rules for the selected element:

```
element.style {
}

div#main {
  padding: 10px;
  height: 197px;
  border: 1px solid black;
  display: flex;
  gap: 33px;
  align-items: center;
  justify-content: center;
}

div {
  box-sizing: border-box;
}

div {
  display: block;
}
```

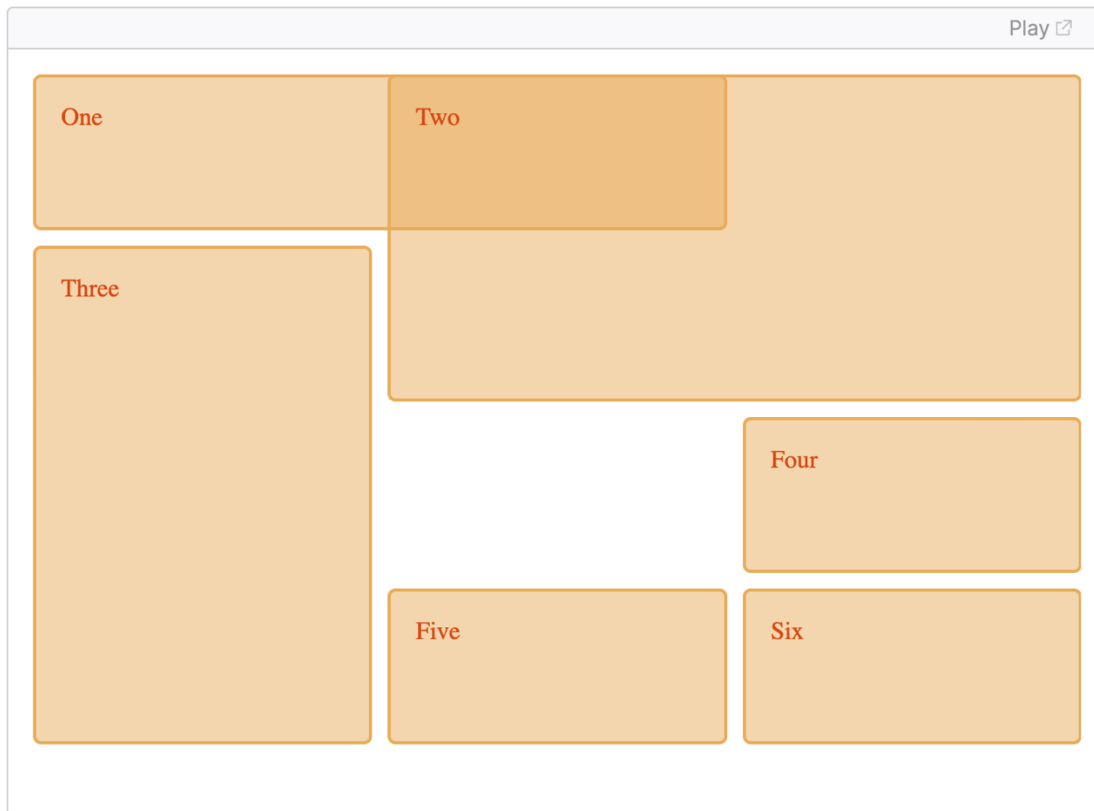
A tooltip menu is open over the flex properties, showing the following options:

- flex-direction: row (with icons for row, column, row-reverse, column-reverse)
- flex-wrap: nowrap (with icons for nowrap, wrap, wrap-reverse)
- align-content: normal (with icons for normal, stretch, flex-start, flex-end)
- justify-content: center (with icons for center, flex-start, flex-end, space-between, space-around, space-between, space-around, space-between)
- align-items: center (with icons for center, flex-start, flex-end, stretch)

# CSS Grid Layout

not covered in the course, but you should know it exists

- The CSS grid layout module divides a container into major regions, defining child relationships in terms of size, position, and layer.




# **“Vanilla” DOM Manipulation**

Approaches

Development steps

MVC

# Getting References to DOM Elements

- Get element using unique element id  
`document.getElementById("my-id")`
- Get element using *CSS selector syntax*  much more flexible method  
`querySelector("#my-id")`
- Best practices
  - specify the element type you expect with TypeScript "as"
  - throw a descriptive error if not found

Example:

```
const root = document.querySelector("div#app") as HTMLDivElement;  
if (!root) throw new Error("root div for app not found");
```



# (Vanilla) HTML Manipulation Approaches

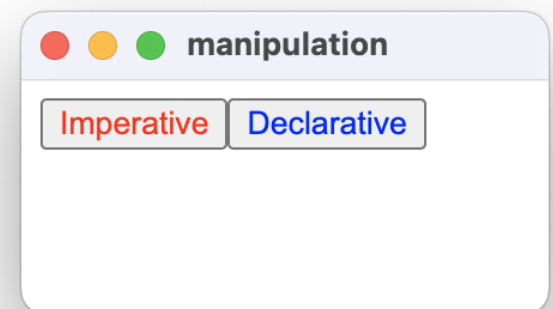
- Build HTML in **imperative** steps (like SimpleKit)
  - using `createElement`, `appendChild`, etc.
- Build HTML **declaratively** as a string
  - using `innerHTML` or `insertAdjacentHTML`

other approaches not covered


- Use HTML **templates**
  - using `<template>` tag
- **Web Components**
  - newer standard
  - You'd want to use the Lit library

# manipulation

- Get root of app (a div in html)
  - `querySelector`
- Add button using **imperative** approach
  - `createElement`, `innerText`, `style.color`, `appendChild`
- Add button using **declarative** approach
  - `insertAdjacentHTML` with HTML in string
- Demo
  - "afterbegin" and "afterend" for `insertAdjacentHTML`
  - Behaviour of `innerHTML` vs `insertAdjacentHTML`



# html Tagged Template Literals

- It's best practice to pass HTML templates through an html "tag"
  - To escape embedded HTML
  - To sanitize HTML
- VS Code recognizes template literals with an html tag 
  - HTML formatting with "Prettier" plug-in
  - HTML syntax highlights with "ES6 String HTML" plug-in
  - HTML expansion by adding to emmet included languages
- For Vanilla DOM projects, install an html tag function package, e.g.  
`npm install html-template-tag`
- Some web frameworks (like Preact) include an `html` tag function

# DOM Events

- DOM events dispatch essentially the same as SimpleKit
  - capture and bubble phases
  - `event.stopPropagation()` method
- Setting event handlers similar to SimpleKit
  - `button.addEventListener("click", (e) => { ... });`
- Basics of DOM events similar to SimpleKit
  - Event base class has properties `timeStamp`, `type`
  - `MouseEvent` has properties `x`, `y`
  - `KeyboardEvent` has property `key`

# manipulation

- Add “click” `addEventListener` to each button
- Note button reference is needed in declarative version

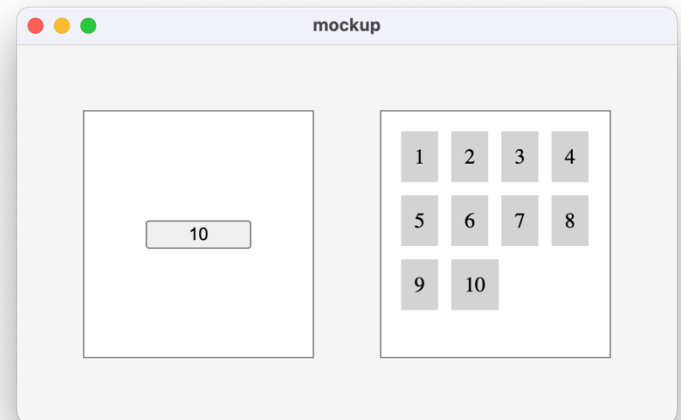


# General MVC Development Steps

1. Mock-up HTML as static page
  - think about how to identify each part using id, name, structure ...
2. Add CSS to create visual style, layout, etc.
3. Divide up HTML
  - result could be separate strings, HTML templates, etc.
4. Create Views for main parts of interface
  - Divide up CSS into views (css file or inline `<style>`)
  - build view from code or strings
  - create controllers using event listeners
  - attach everything to a root element for the view
  - implement Observer update method

# mockup

- HTML
  - predominately using `<div>` for containers
  - using element id to identify views
  - hard-coding state for mock-up
- CSS
  - in separate `style.css` file
  - standard resets
  - visual properties like `background-color`, `border`
  - layout using flexbox properties
  - note use of CSS selectors
  - `height: 100vh` for *full height div*



# MVC in Vanilla HTML Apps

- Model and Observer pattern is exactly the same
- Instead of inheriting from container, View is an **interface**
  - Extended from Observer
  - Has “root” property for reference to HTML node at root of view (usually a div)

```
import { Observer } from "../observer";
```

```
export default interface View extends Observer {  
  root: HTMLElement;  
}
```



# imperative

- Build DOM for views step-by-step in code:

```
document.createElement( ... )  
container.appendChild( ... )  
container.replaceChildren( ... )
```

- Demos

- div#app root

- views have a “root” element that is appended to parent:

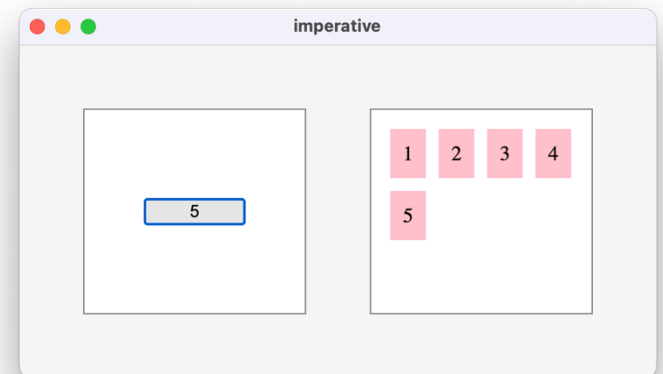
```
panel.appendChild(new LeftView(model).root);
```

- Separate css files with selectors that will apply only to view

- Style files can be imported in code, e.g.

```
import "./leftView.css";
```

- Button controller



# imperative

- **Demo: Is adding many elements to DOM slow?**
  - Start model count at 10000 (even 100000)
  - Reverse list of RightView divs to see update results
  - Create a console timer in LeftView increment button handler
- **Demo: Isn't adding elements to DOM *one-by-one* very slow?**
  - With setup above, switch to using a *DocumentFragment*

```
const fragment = document.createDocumentFragment();
```

```
[ ...Array( ... ) ].forEach((i) => {  
  const div = document.createElement("div");  
  ...  
  fragment.appendChild(div);  
});
```

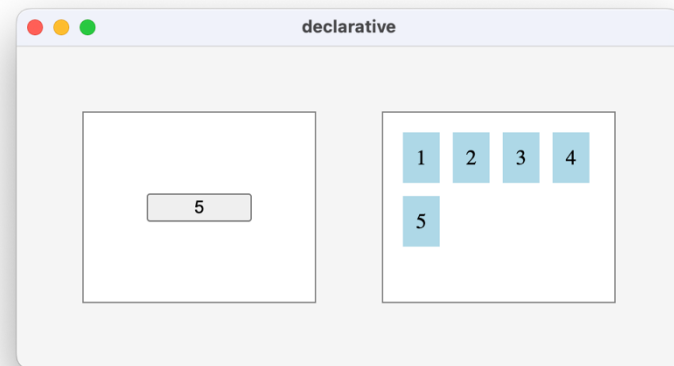
```
this.container.appendChild(fragment);
```

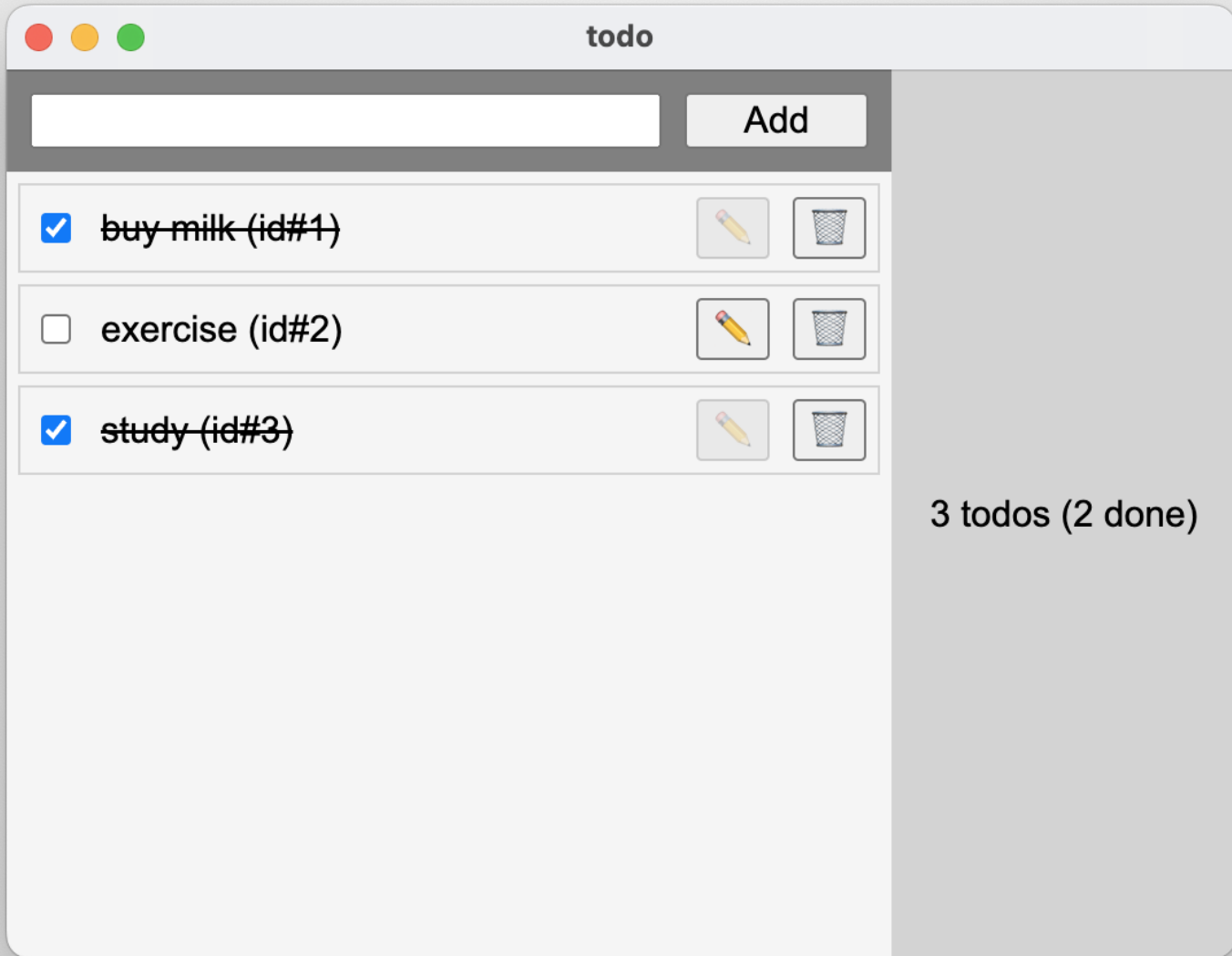
# declarative

- Set `innerHTML` to the result of a HTML tagged template literal
- Creating View root uses template element

```
var temp = document.createElement("template");
temp.innerHTML = html`
  <div id="left"><button id="increment">?</button></div>
`;
this.container = temp.content.firstChild as HTMLDivElement;
```

- Demos
  - extra code to get ref to button





# todo

- MVC todo app using Vanilla DOM manipulation
  - Using imperative DOM manipulation
  - Exact same Model as SimpleKit todo demo
  - Exact same nested View structure:  
FormView, ListView (with TodoView children), InfoView
- Demo
  - mockup.html to work out the layout and styles
  - Advanced CSS selectors for strike through text when todo finished

# Tutorials

- HTML

- [https://developer.mozilla.org/en-US/docs/Learn/Getting\\_started\\_with\\_the\\_web/HTML\\_basics](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics)

- CSS

- [https://developer.mozilla.org/en-US/docs/Learn/Getting\\_started\\_with\\_the\\_web/CSS\\_basics](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics)

- DOM Manipulation

- [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side\\_web\\_APIs/Manipulating\\_documents](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents)