Principles of Software Construction: Objects, Design, and Concurrency

Basic GUI concepts, HTML

Claire Le Goues **Bogdan Vasilescu**

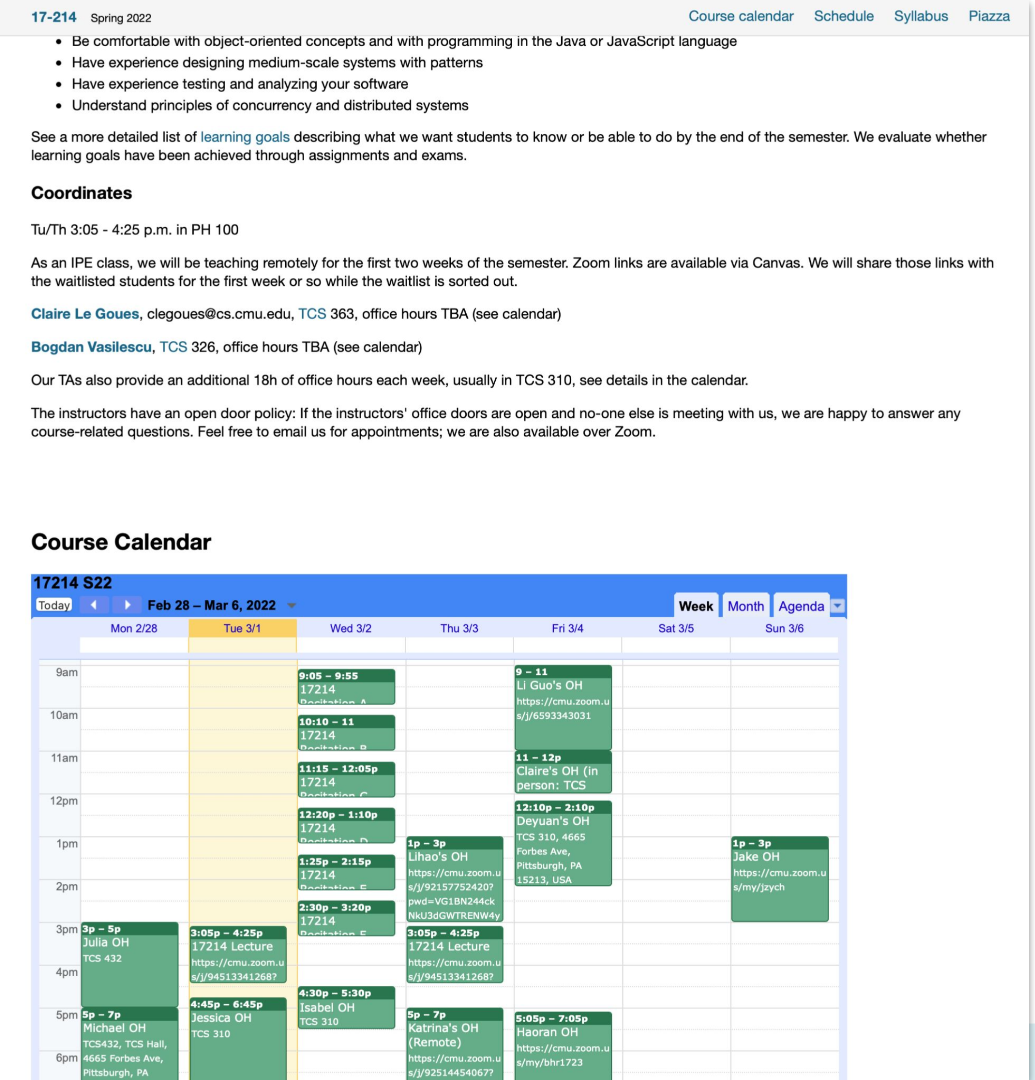
Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Today

- GUI Design
  - Concepts, strategies
  - Practical application in HTML, CSS, JS
- Dynamic Web Pages
  - Client/Server communication
  - Backend architecture

How To Make This Happen?



- Be comfortable with object-oriented concepts and with programming in the Java or JavaScript language
- Have experience designing medium-scale systems with patterns
- Have experience testing and analyzing your software
- Understand principles of concurrency and distributed systems

See a more detailed list of learning goals describing what we want students to know or be able to do by the end of the semester. We evaluate whether learning goals have been achieved through assignments and exams.

Coordinates

Tu/Th 3:05 - 4:25 p.m. in PH 100

As an IPE class, we will be teaching remotely for the first two weeks of the semester. Zoom links are available via Canvas. We will share those links with the waitlisted students for the first week or so while the waitlist is sorted out.

Claire Le Goues, clegoues@cs.cmu.edu, TCS 363, office hours TBA (see calendar)

Bogdan Vasilescu, TCS 326, office hours TBA (see calendar)

Our TAs also provide an additional 18h of office hours each week, usually in TCS 310, see details in the calendar.

The instructors have an open door policy: If the instructors' office doors are open and no-one else is meeting with us, we are happy to answer any course-related questions. Feel free to email us for appointments; we are also available over Zoom.

Course Calendar

17214 S22

Mon 2/28
- 3p – 5p Julia OH TCS 432
- 5p – 7p Michael OH TCS432, TCS Hall, 4665 Forbes Ave, Pittsburgh, PA

Tue 3/1
- 3:05p – 4:25p 17214 Lecture https://cmu.zoom.us/j/94513341268?
- 4:45p – 6:45p Jessica OH TCS 310

Wed 3/2
- 9:05 – 9:55 17214 Recitation A
- 10:10 – 11 17214 Recitation B
- 11:15 – 12:05p 17214 Recitation C
- 12:20p – 1:10p 17214 Recitation D
- 1:25p – 2:15p 17214 Recitation E
- 2:30p – 3:20p 17214 Recitation F
- 4:30p – 5:30p Isabel OH TCS 310

Thu 3/3
- 1p – 3p Lihao's OH https://cmu.zoom.us/j/92157752420?pwd=VG1BN244ckNkU3dGWTRENW4y
- 3:05p – 4:25p 17214 Lecture https://cmu.zoom.us/j/94513341268?
- 5p – 7p Katrina's OH (Remote) https://cmu.zoom.us/j/92514454067?

Fri 3/4
- 9 – 11 Li Guo's OH https://cmu.zoom.us/j/6593343031
- 11 – 12p Claire's OH (in person: TCS
- 12:10p – 2:10p Deyuan's OH TCS 310, 4665 Forbes Ave, Pittsburgh, PA 15213, USA
- 5:05p – 7:05p Haoran OH https://cmu.zoom.us/my/bhr1723

Sat 3/5

Sun 3/6
- 1p – 3p Jake OH https://cmu.zoom.us/my/jzych

GUI Design: what do we want?

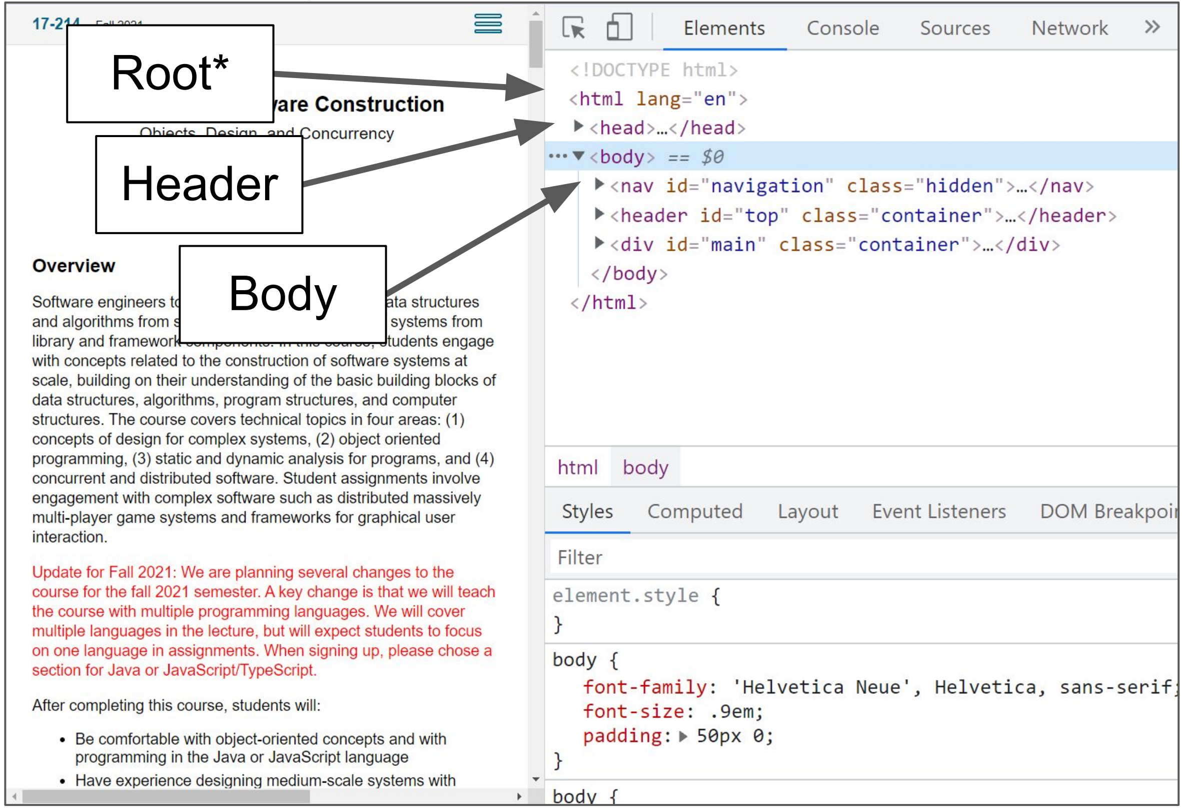
- Nested Elements
- Style Vocabulary
- Interactivity

GUI Design: what do we want?

- Nested Elements
  - HTML
- Style Vocabulary
  - CSS
- Interactivity
  - JavaScript

Anatomy of an HTML Page

Predefined elements



...are Construction

Objects, Design, and Concurrency

Overview

Software engineers to... ...ata structures and algorithms from s... ...systems from library and framework ... ...students engage with concepts related to the construction of software systems at scale, building on their understanding of the basic building blocks of data structures, algorithms, program structures, and computer structures. The course covers technical topics in four areas: (1) concepts of design for complex systems, (2) object oriented programming, (3) static and dynamic analysis for programs, and (4) concurrent and distributed software. Student assignments involve engagement with complex software such as distributed massively multi-player game systems and frameworks for graphical user interaction.

Update for Fall 2021: We are planning several changes to the course for the fall 2021 semester. A key change is that we will teach the course with multiple programming languages. We will cover multiple languages in the lecture, but will expect students to focus on one language in assignments. When signing up, please chose a section for Java or JavaScript/TypeScript.

After completing this course, students will:

- Be comfortable with object-oriented concepts and with programming in the Java or JavaScript language
- Have experience designing medium-scale systems with

```
<!DOCTYPE html>
<html lang="en">
 ▶<head>…</head>
…▼<body> == $0
   ▶<nav id="navigation" class="hidden">…</nav>
   ▶<header id="top" class="container">…</header>
   ▶<div id="main" class="container">…</div>
  </body>
</html>
```

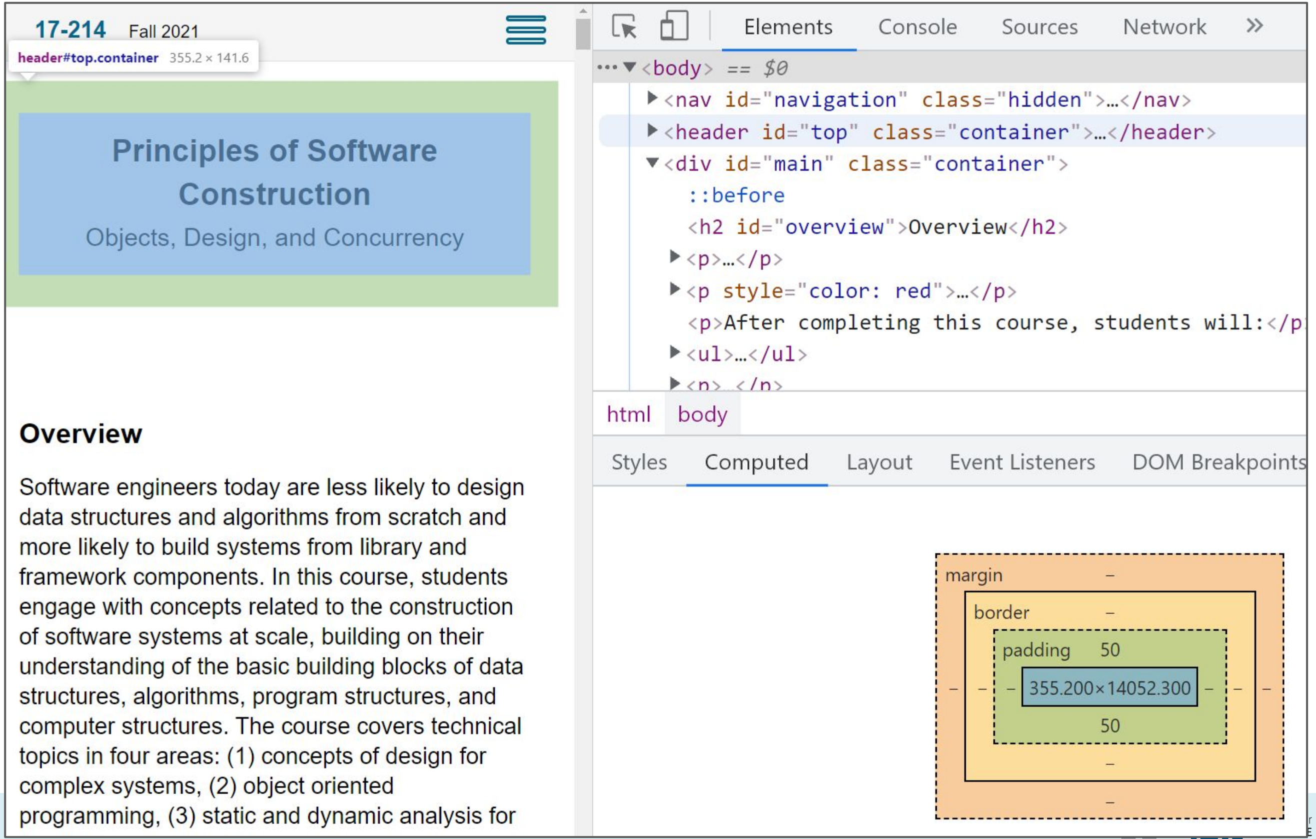
```
element.style {
}
body {
  font-family: 'Helvetica Neue', Helvetica, sans-serif;
  font-size: .9em;
  padding: ▶ 50px 0;
}
body {
```

Technically, ‘document’ is the root with HTML as its only child

Anatomy of an HTML Page

Nested elements

- Sizing
- Attributes
- Text



Principles of Software Construction

Objects, Design, and Concurrency

Overview

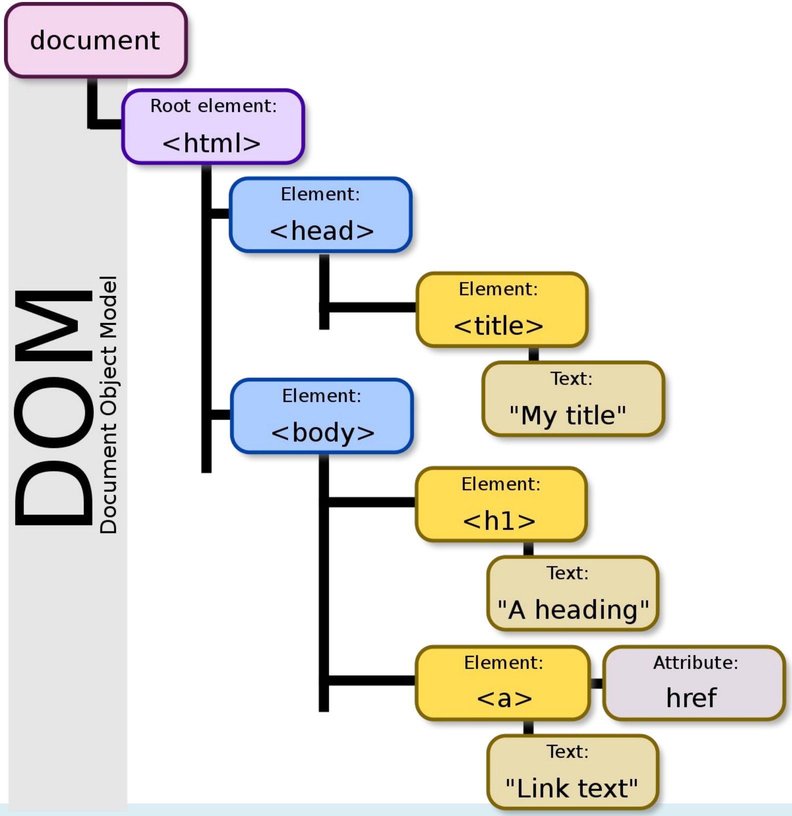
Software engineers today are less likely to design data structures and algorithms from scratch and more likely to build systems from library and framework components. In this course, students engage with concepts related to the construction of software systems at scale, building on their understanding of the basic building blocks of data structures, algorithms, program structures, and computer structures. The course covers technical topics in four areas: (1) concepts of design for complex systems, (2) object oriented programming, (3) static and dynamic analysis for

```
••• ▼<body> == $0
      ▶<nav id="navigation" class="hidden">…</nav>
      ▶<header id="top" class="container">…</header>
      ▼<div id="main" class="container">
         ::before
         <h2 id="overview">Overview</h2>
        ▶<p>…</p>
        ▶<p style="color: red">…</p>
         <p>After completing this course, students will:</p
        ▶<ul>…</ul>
        ▶<p>…</p>
```

Anatomy of an HTML Page

Many GUIs are trees

- Nested elements, recursively
- Some fixed positions (html, body)



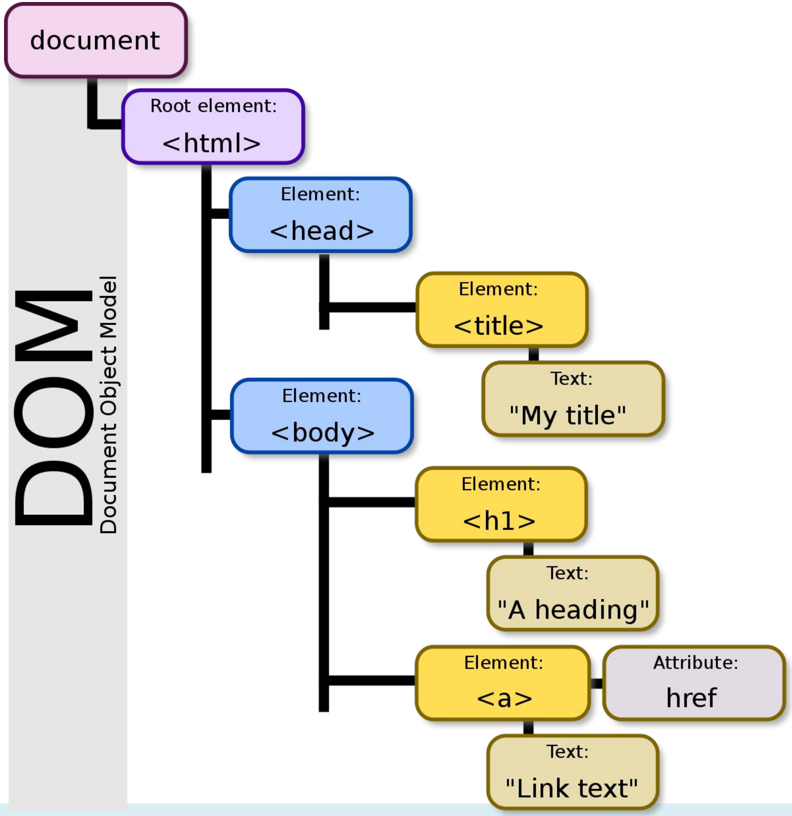
https://en.wikipedia.org/wiki/Document_Object_Model

Anatomy of an HTML Page

Many GUIs are trees

- Nested elements, recursively
- Some fixed positions (html, body)

How to implement this?



https://en.wikipedia.org/wiki/Document_Object_Model

The composite pattern

- Problem: Collection of objects has behavior similar to the individual objects
- Solution: Have collection of objects and individual objects implement the same interface
- Consequences:
    - Client code can treat collection as if it were an individual object
    - Easier to add new object types
    - Design might become too general, interface insufficiently useful

Composite

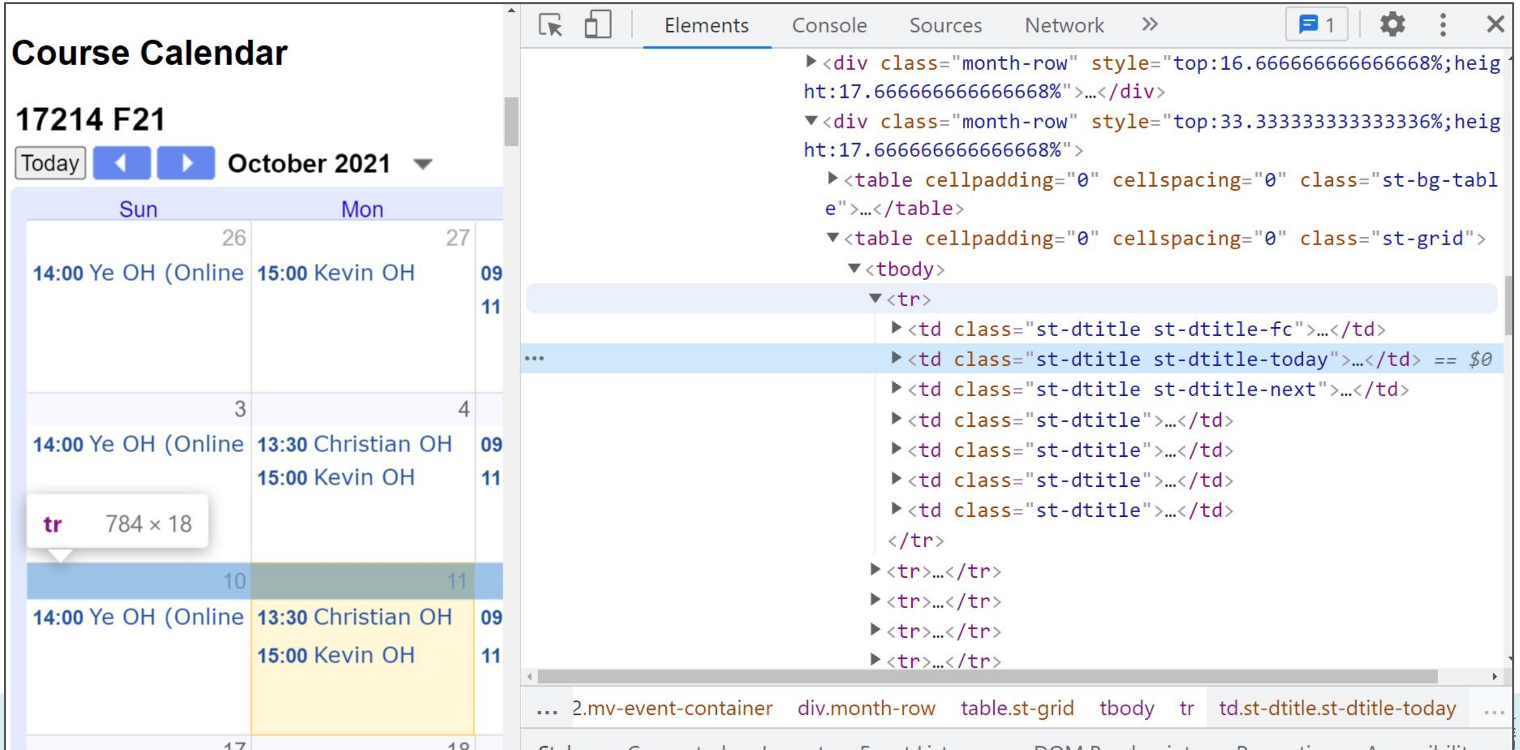
- Elements can contain elements
  - With restrictions
  - Need to deal with style, interaction
- In JS: HTMLElement
  - With child-classes e.g. HTMLDivElement, HTMLBodyElement
  - Navigation:
    - getElement*: locate by tag name, id, class, etc.
    - next/prev(Element)Sibling
    - childNodes, parent

A few Tags

- <html>
  - The root of the visible page
- <head>
  - Stores metadata, imports
- <p>
  - A paragraph
- <button>
  - Attributes include `name`, `type`, `value`
- <div>
  - Generic section -- very useful
- <table>
  - The obvious
- Many more; dig into a real page!

Style

Not only leaf-nodes have an appearance



Course Calendar

17214 F21

Today ◀ ▶ **October 2021** ▼

Sun	Mon	
26 **14:00** Ye OH (Online	27 **15:00** Kevin OH	**09** **11**
3 **14:00** Ye OH (Online	4 **13:30** Christian OH **15:00** Kevin OH	**09** **11**
10 **14:00** Ye OH (Online	11 **13:30** Christian OH **15:00** Kevin OH	**09** **11**

tr 784 × 18

```
▶<div class="month-row" style="top:16.666666666666668%;height:17.666666666666668%">…</div>
▼<div class="month-row" style="top:33.333333333333336%;height:17.666666666666668%">
  ▶<table cellpadding="0" cellspacing="0" class="st-bg-table">…</table>
  ▼<table cellpadding="0" cellspacing="0" class="st-grid">
    ▼<tbody>
      ▼<tr>
        ▶<td class="st-dtitle st-dtitle-fc">…</td>
        ▶<td class="st-dtitle st-dtitle-today">…</td> == $0
        ▶<td class="st-dtitle st-dtitle-next">…</td>
        ▶<td class="st-dtitle">…</td>
        ▶<td class="st-dtitle">…</td>
        ▶<td class="st-dtitle">…</td>
        ▶<td class="st-dtitle">…</td>
       </tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
```

… 2.mv-event-container div.month-row table.st-grid tbody tr td.st-dtitle.st-dtitle-today …

Style

Tags come with inherent & customizable style

- Inherent:
  - <div> is a `block` (full-width, with margin)
  - <span> is in-line
  - <h1> is large
- Customizable: add and override styles
  - Change font-styles, margins, widths
  - Modify groups of elements

Style: CSS

- Cascading Style Sheets
  - Reuse: styling rules for tags, classes, types
  - Reuse: not just at the leafs!

```
<span style="font-weight:bold">Hello again!</span>
```

vs.

```
<style type="text/css">
    span {
        font-family: arial
    }
</style>
```

Style: CSS

- Cascading Style Sheets
  - Reuse: styling rules for tags, classes, types
  - Reuse: not just at the leafs!
- What if there are conflicts?

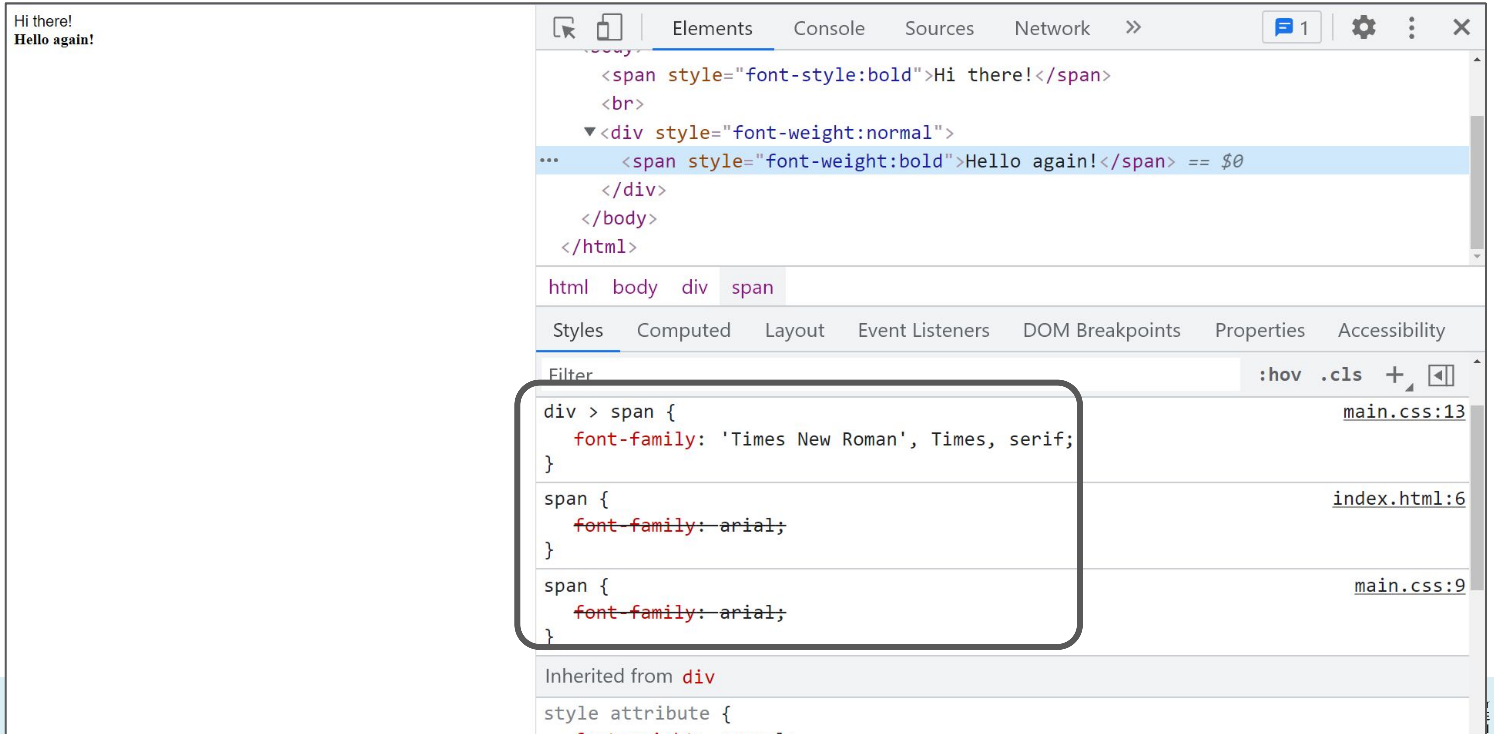
```
<div style="font-weight:normal">
  <span style="font-weight:bold">Hello again!</span>
</div>
```

  - Lowest element wins*

*Technically, there’s a whole scoring system

Style: CSS

What is happening here?



Hi there!
Hello again!

```
    <span style="font-style:bold">Hi there!</span>
    <br>
  ▼<div style="font-weight:normal">
…     <span style="font-weight:bold">Hello again!</span> == $0
    </div>
  </body>
</html>
```

```
div > span {                                              main.css:13
   font-family: 'Times New Roman', Times, serif;
}
span {                                                   index.html:6
   ~~font-family: arial;~~
}
span {                                                     main.css:9
   ~~font-family: arial;~~
}
Inherited from div
style attribute {
```

Style: CSS

- Cascading Style Sheets
  - Reuse: styling rules for tags, classes, types
  - Reuse: not just at the leafs!
- What if there are no conflicts?

```
<div style="font-family:arial">
  <span style="font-weight:bold">Hello again!</span>
</div>
```

  - How would you implement this?

Decorator

What is happening here?

- To compute the style of an element:
  - Apply its tag-default style
  - **Wrap** in added style rules (tag-specific or general)
    - Text: font-family, weight, etc.
  - Inherit parents’ style
    - Conflicts lead to overrides
- Makes *themes* really powerful

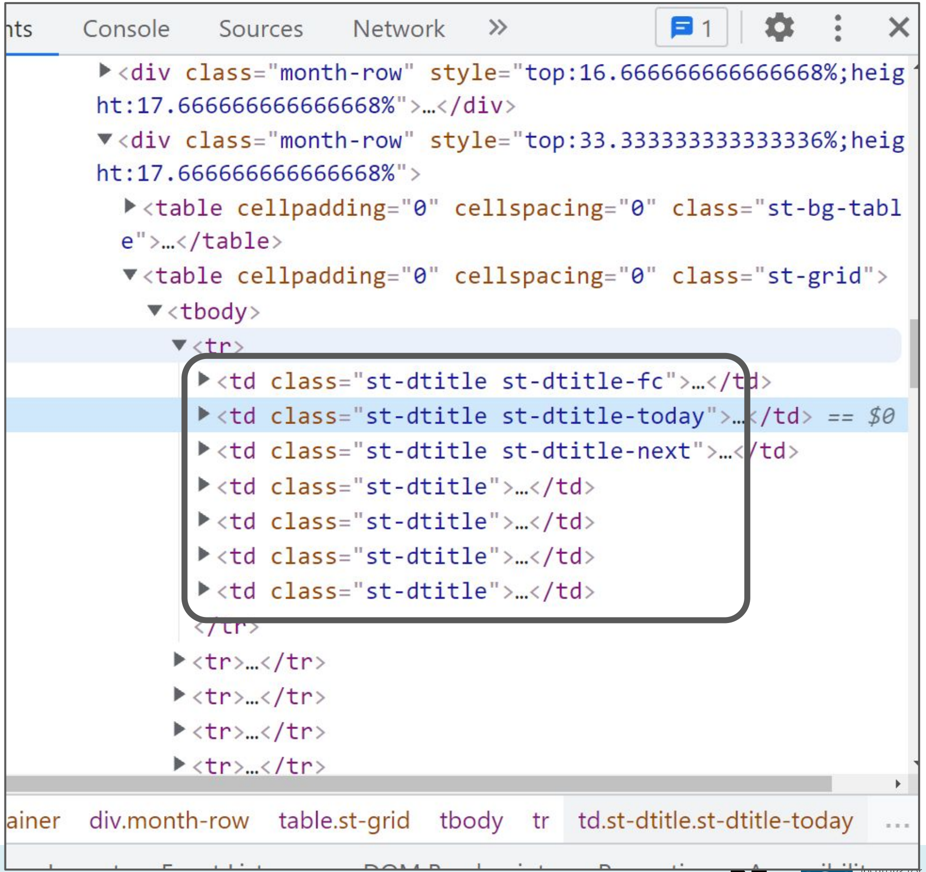
Technically, HTML is streamed top-to-bottom; CSS works bottom-up

CSS: classes

Let's not repeat custom style

- Use any nr. of class label(s)
- Class styles get added
- Facilitates reuse

How would you implement this?



nts
Console
Sources
Network
»
1

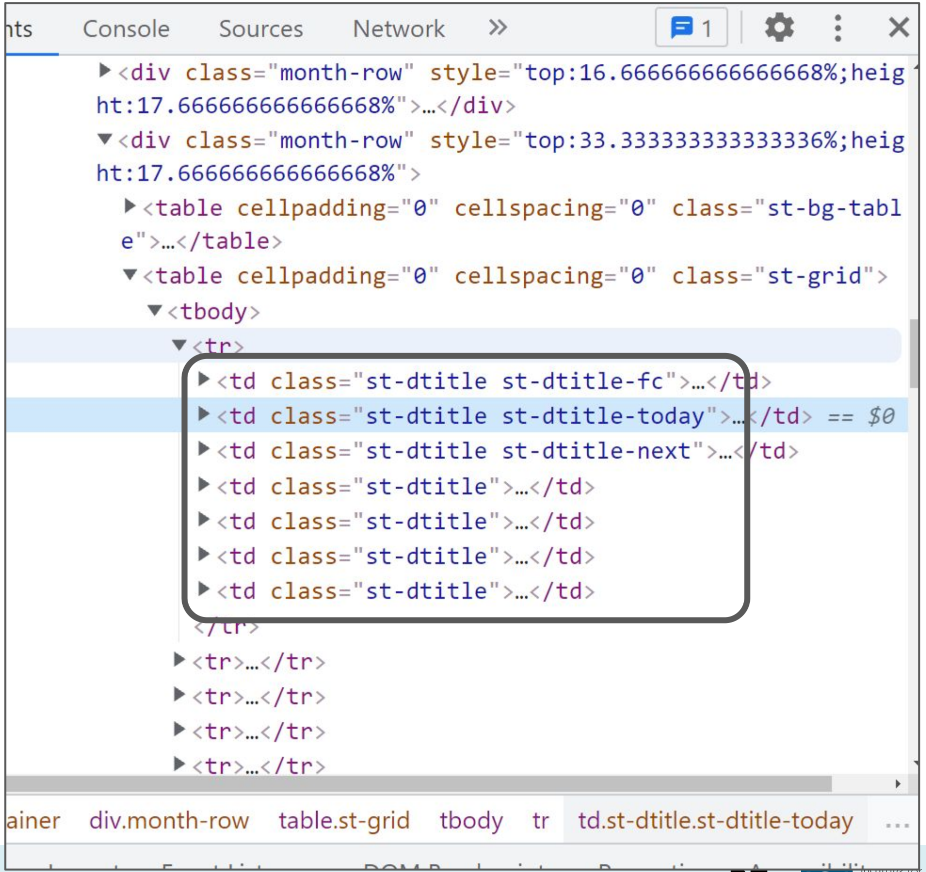
```
▶<div class="month-row" style="top:16.666666666666668%;height:17.666666666666668%">…</div>
▼<div class="month-row" style="top:33.333333333333336%;height:17.666666666666668%">
  ▶<table cellpadding="0" cellspacing="0" class="st-bg-table">…</table>
  ▼<table cellpadding="0" cellspacing="0" class="st-grid">
    ▼<tbody>
      ▼<tr>
        ▶<td class="st-dtitle st-dtitle-fc">…</td>
        ▶<td class="st-dtitle st-dtitle-today">…</td> == $0
        ▶<td class="st-dtitle st-dtitle-next">…</td>
        ▶<td class="st-dtitle">…</td>
        ▶<td class="st-dtitle">…</td>
        ▶<td class="st-dtitle">…</td>
        ▶<td class="st-dtitle">…</td>
       </tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
```

ainer div.month-row table.st-grid tbody tr td.st-dtitle.st-dtitle-today …

Strategy or Observer?

Either could apply

- Both involve callback
- Strategy:
  - Typically single
  - Often involves a return
- Observer:
  - Arbitrarily many
  - Involves external updates

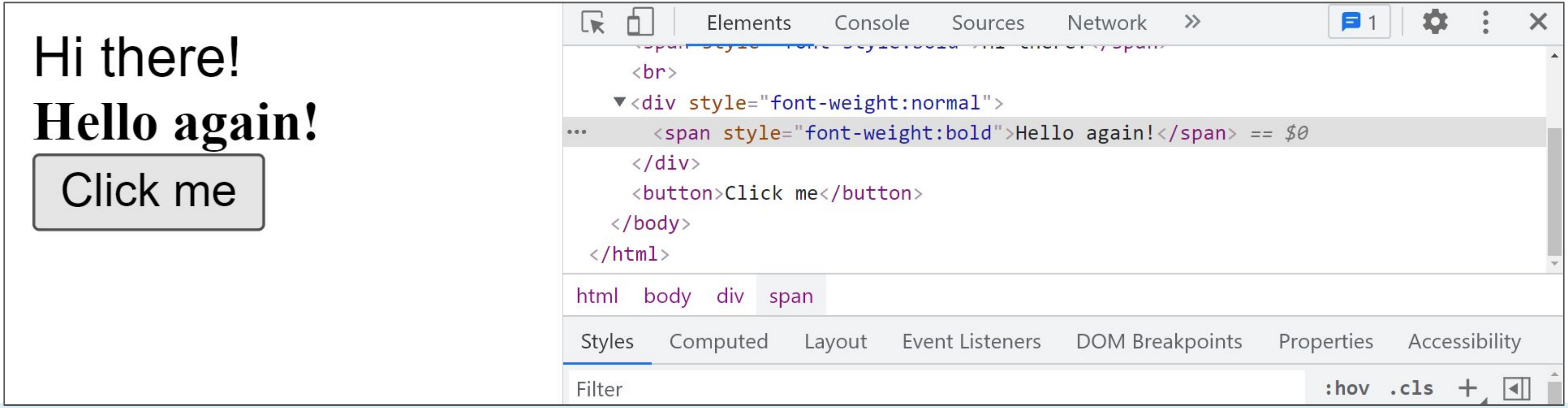


```
▶<div class="month-row" style="top:16.666666666666668%;height:17.666666666666668%">…</div>
▼<div class="month-row" style="top:33.333333333333336%;height:17.666666666666668%">
  ▶<table cellpadding="0" cellspacing="0" class="st-bg-table">…</table>
  ▼<table cellpadding="0" cellspacing="0" class="st-grid">
    ▼<tbody>
      ▼<tr>
        ▶<td class="st-dtitle st-dtitle-fc">…</td>
        ▶<td class="st-dtitle st-dtitle-today">…</td> == $0
        ▶<td class="st-dtitle st-dtitle-next">…</td>
        ▶<td class="st-dtitle">…</td>
        ▶<td class="st-dtitle">…</td>
        ▶<td class="st-dtitle">…</td>
        ▶<td class="st-dtitle">…</td>
       </tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
      ▶<tr>…</tr>
```

Interactivity

A GUI is more than a document

- How do we make it “work”?



Hi there!

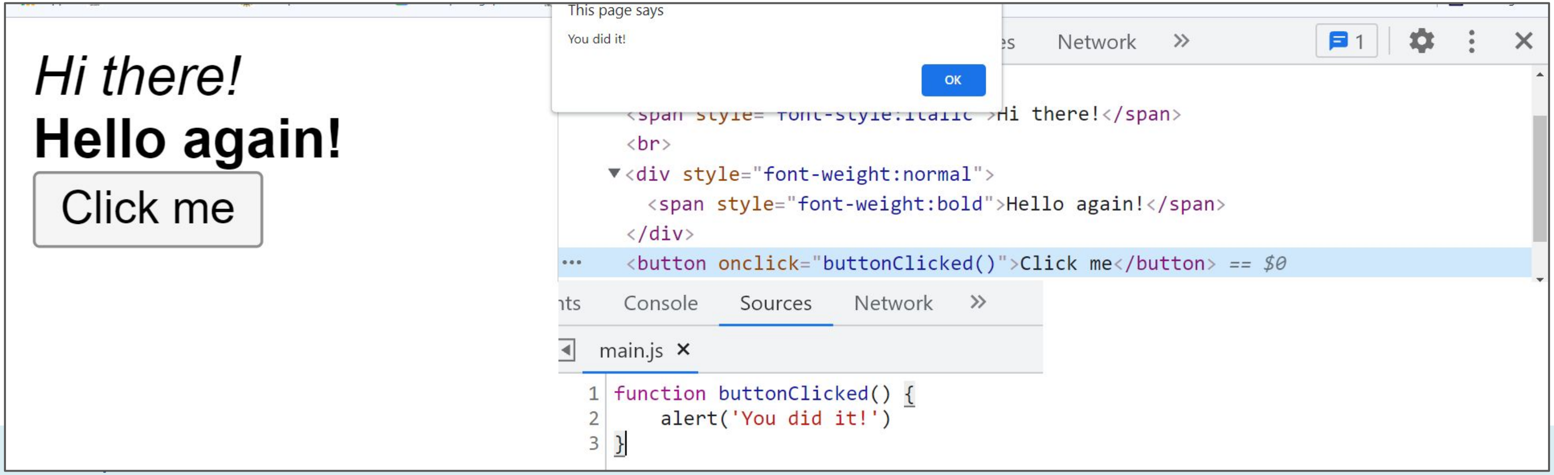
Hello again!

Click me

```
      <span style="font-style:bold">Hi there!</span>
      <br>
    ▼<div style="font-weight:normal">
•••     <span style="font-weight:bold">Hello again!</span> == $0
      </div>
      <button>Click me</button>
    </body>
  </html>
```

Actions: JavaScript

- Key: event listeners (what’s that pattern?)
- (frontend) JS is highly event-driven
  - Respond to window `onLoad` event, content loads (e.g., ads)
  - Respond to clicks, moves



This page says

You did it!

```
<span style="font-style:italic">Hi there!</span>
<br>
▼<div style="font-weight:normal">
  <span style="font-weight:bold">Hello again!</span>
</div>
<button onclick="buttonClicked()">Click me</button> == $0
```

```
function buttonClicked() {
    alert('You did it!')
}
```

Observer Pattern

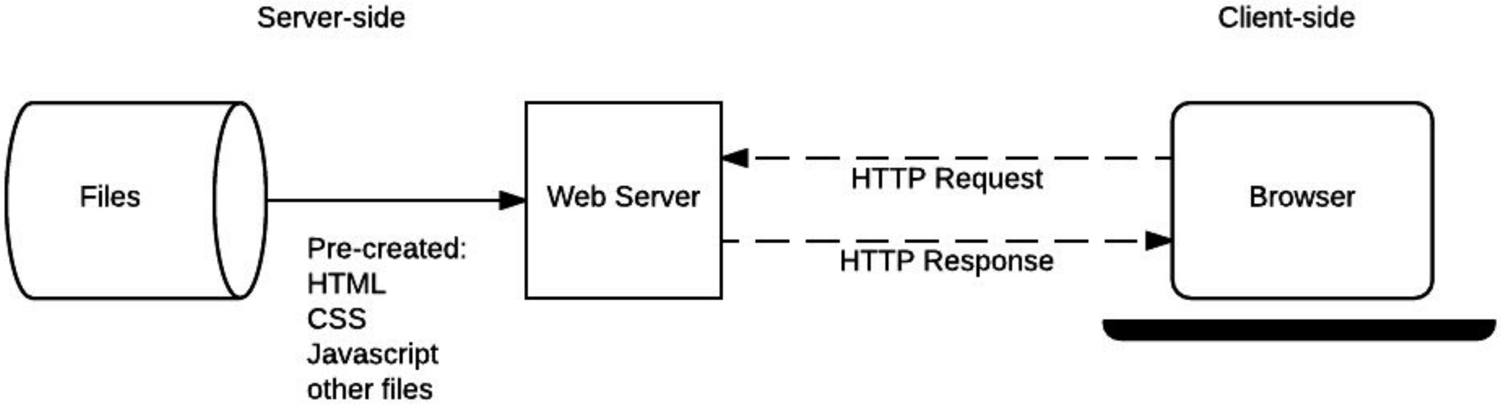
- Manages publishers and subscribers
  - Here, button publishes its ‘click’ events
  - `buttonClicked` subscribes to 1+ updates
- Flexibility and Reuse
  - Multiple observers per element
  - Shared observers across elements

Step Back

- What is our website now?
  - Layout, style, interaction
  - What is missing?

Static Web Pages

- Delivered as-is, final
  - Consistent, often fast
  - Cheap, only storage needed
- “Static” a tad murky with JavaScript
  - We can still have buttons, interaction
  - But it won’t “go” anywhere -- the server is mum



https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview#anatomy_of_a_dynamic_request

Static Web Pages

- Delivered as-is, final
    - Consistent, often fast
    - Cheap, only storage needed
- Maintain with *static website generators*
    - Or you’ll be doing a lot of copying
    - Coupled with themes => rapid development, deployment
    - Quite popular, e.g. hosting on GH Pages

Static Web Pages

- But …
  - No persistence (at least, not obviously)
  - No customizability (e.g., accounts)
  - No communication (payment, chat, etc)
  - Realistically, no intensive jobs

Dynamic Web Pages

- Client/Server
    - Someone needs to answer the website’s calls
        - Doesn’t need to be us!
    - Host a webserver
        - Serves pages, handles calls
        - For static pages too!
- We’ll show you more in recitation tomorrow (Wednesday)

Web Servers

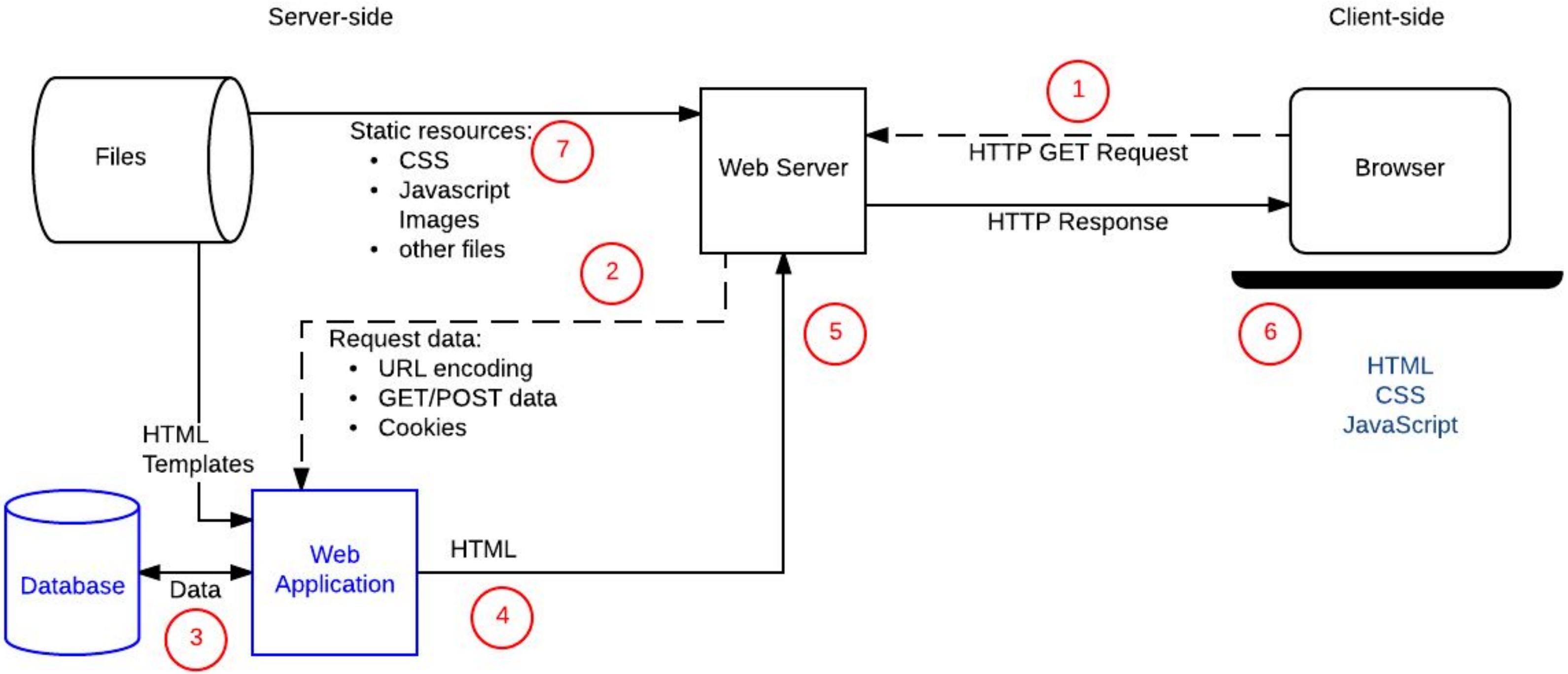
- Communicate via HyperText Transfer Protocol
  - URL (the address)
  - Method:
    - GET: retrieve data. Parameters in URL `...?key=value&key2=value2` and message body
    - POST: store/create data. Parameters in request body
    - Several more, rarely used
  - Responses:
    - *Status Code*:
      - We probably all know 404.
      - 2XX family is OK.
    - And possible data. E.g., entire HTML page.

Web Servers

- Communicate via HyperText Transfer Protocol
  - URL (the address)
  - Method:
    - GET: retrieve data. Parameters in URL \`...?key=value&key2=value2\` and message body
    - POST: store/create data. Parameters in request body
    - Several more, rarely used
  - Responses:
    - *Status Code*. We all know 404. 2XX family is OK.
    - And possible data. E.g., entire HTML page.
  - POST makes no sense for static sites!
  - As do GETs with parameters

Web Servers

Dynamic sites can do more *work*



https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview#anatomy_of_a_dynamic_request

AJAX

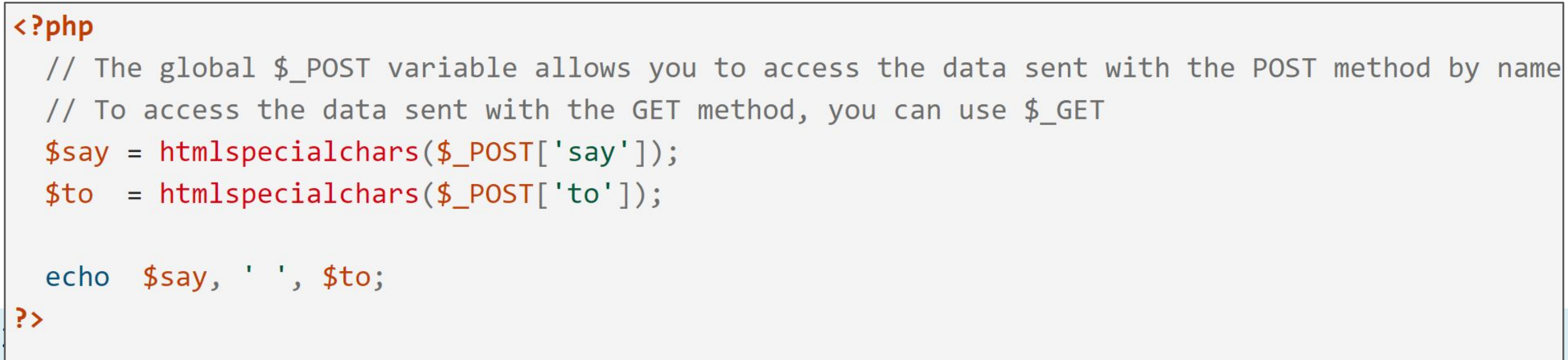
- Originally: “Asynchronous JavaScript and XML”
  - Updates parts of a page dynamically
  - Sends XMLHttpRequests with a callback
  - On return, check the code; handle success and failure.
  - Asynchronous, naturally decouples backend from UI

AJAX

- Originally: “Asynchronous JavaScript and XML”
  - Updates parts of a page dynamically
  - Sends XMLHttpRequests with a callback
  - On return, check the code; handle success and failure.
  - Asynchronous, naturally decouples backend from UI
- Slowly being phased out
  - Replace with `fetch`, which uses… Promises
    - More next week

How to Web App?

- Let’s avoid generating HTML from scratch on every call
  - Map requests to handler code
    - Fetch data, process
  - Generate and return HTML
- Historically: PHP
  - Modifies HTML pages server-side on request; strong ties to SQL



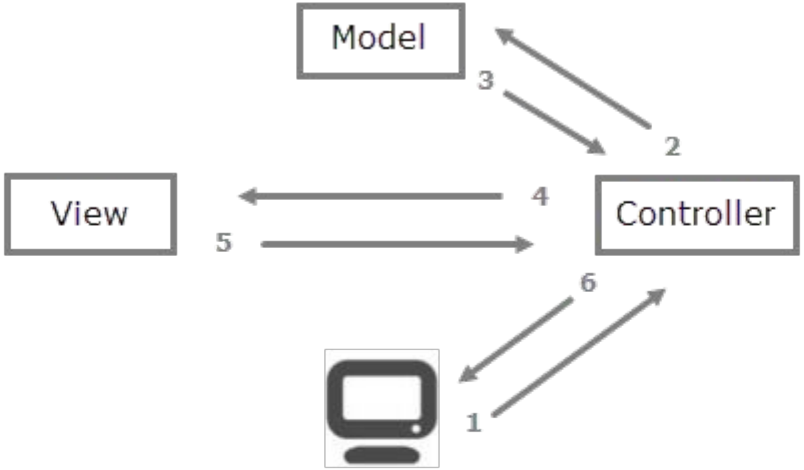
```
<?php
  // The global $_POST variable allows you to access the data sent with the POST method by name
  // To access the data sent with the GET method, you can use $_GET
  $say = htmlspecialchars($_POST['say']);
  $to  = htmlspecialchars($_POST['to']);

  echo  $say, ' ', $to;
?>
```

How to Web App?

- Let’s avoid generating HTML from scratch on every call
  - Map requests to handler code
    - Fetch data, process
  - Generate and return HTML
- Or use a framework
  - Python: Flask, Django
  - NodeJS: Express
  - Spring for Java
  - Many others, differences in **weight**, features

Model-View-Controller (MVC)



https://overiq.com/django-1-10/mvc-pattern-and-django/

MVC is ubiquitous

Separates:

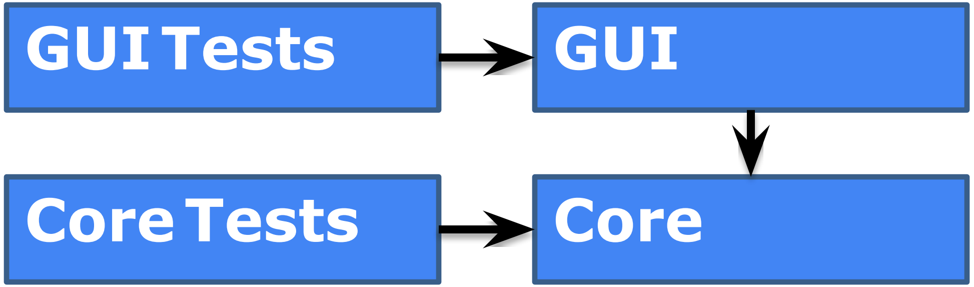
- Model: data organization
  - Interface to the database
- View: data representation (typically HTML)
  - Often called *templates* in web-dev; “view” is a bit overloaded
- Controller: intermediary between client and model/view
  - Typically asks model for data, view for HTML

Core implementation vs. GUI

- Core implementation: application logic
  - Computing some result, updating data
- GUI
  - Graphical representation of data
  - Source of user interactions
- Design guideline: *avoid coupling the GUI with core application*
  - Multiple UIs with single core implementation
  - Test core without UI

Separating application core and GUI

- Reduce coupling: do not allow core to depend on UI
- Create and test the core without a GUI
    - Use the Observer pattern to communicate information from the core (Model) to the GUI (View)



Summary

- GUIs are full of design patterns
  - Helpful for reuse, delegation in complex environments
- Covered the basics of HTML, CSS, JS, servers
  - Needed for dynamic web pages
  - Decouple the GUI; architect your backend
  - A lot more to learn (security, performance, privacy), but this will do
- You will build this
  - At a small scale