Web Dev Basics 3

CS571: Building User Interfaces

Cole Nelson

Before Lecture

- Clone today's code to your machine.

Web Dev Basics 2 Recap

fetch()

```
fetch(url)
  .then((response) => response.json()) // implict return
  .then((data) => {
    // fetch has already parsed data from JSON to a JS object!
    // Do something with the data
  })
  .catch(error => console.error(error)) // Print errors
```

Fetching Jokes

Callback Functions

`then` and `catch` take a *callback function* as an argument.

Definition: A *callback function* (sometimes called a *function reference*) is passed into another function as an argument, which is then invoked inside the outer function to complete a routine or action.

More on callback functions

Learning Objectives

1. Understand the difference between imperative and declarative programming.
2. Be able to use declarative functions such as `map`, `filter`, `reduce`, ...
3. Know of other JavaScript syntactic sugars.
4. Be able to work with CSS libraries.

Declarative vs Imperative Programming

We typically prefer *declarative* (what) programming over *imperative* (how) programming.

Declarative array functions include `filter`, `map`, `some`, `every`, and `reduce`.

These take a *callback function* as an argument.

Warmup Problem

Given the following array, narrow the elements down such that we only keep elements that are numbers.

```
const things = ["dogs", 1.2, 0, false, {name: "Alice"}, -7]
```

Can you do this *imperatively*? Use `for (... of ...)`

Can you do this *declaratively*? Use `filter`

Solution

Done *imperatively*... (how)

```
const newThings = [];
for(let thing of things) {
  if (typeof thing === 'number') {
    newThings.push(thing);
  }
}
```

Done *declaratively*... (what)

```
const newThings = things.filter((thing) => typeof thing === 'number')
```

StackBlitz

`filter`

`filter` performs a function on each element of an array and *returns* an array of those elements whose function call returned true.

```
const shortNames = ["Bessy", "Rob", "Bartholomew"].filter(name => {
  if (name.length <= 5) {
    return true;
  } else {
    return false;
  }
});
console.log(shortNames);
```

StackBlitz

`filter`

Using *implicit* returns, this simplifies to...

```
const shortNames = ["Bessy", "Rob", "Bartholomew"].filter(name => name.length <= 5);
const longNames = ["Bessy", "Rob", "Bartholomew"].filter(name => name.length > 5);
console.log(shortNames);
console.log(longNames);
```

StackBlitz

`map`

`map` performs a function on each element of an array and *returns* an array of the the return of those function calls.

```
const nameLengths = ["Bessy", "Rob", "Bartholomew"].map(n => n.length);
console.log(nameLengths);
```

StackBlitz

Chaining Declarative Functions

Of those with short names, how many letters are in their name?

```
["Bessy", "Rob", "Bartholomew"]
  .filter(name => name.length <= 5)
  .map(name => name.length);
```

StackBlitz

Chaining Declarative Functions

Of those with short names, what are their names and do they have a *very* short name?

```
["Bessy", "Rob", "Bartholomew"]
  .filter(n => n.length <= 5)
  .map(n => {
    return {
      name: n,
      isVeryShort: n.length <= 3
    }
  });
```

StackBlitz

Your Turn!

Let's re-visit last lecture's data and use some of these declarative functions...

1. Can you `filter` to only show the 5-star reviews?
2. Can you `map` to only show the text before a ":" in the recipe's instructions?
3. Can you `map` to display the ingredients as a string?
   a. **Hint:** `Object.keys` returns an array!

`some(cb)` and `every(cb)`

`some(cb)` returns `true` if the callback returns true for *some* element of the array, `false` otherwise.

```
["sam", "jacob", "jess"].some(p => p === "jess"); // true!
```

`every(cb)` returns `true` if the callback returns true for *every* element of the array, `false` otherwise.

```
["sam", "jacob", "jess"].every(p => p === "jess"); // false!
```

`reduce(cb, start)`

`reduce` takes a 2-parameter callback (previous and current values) and a starting value.

```
[2, 4, -1.1, 7.2].reduce((prev, curr) => prev + curr, 1.2); // 13.3
```

PythonTutor

Your turn!

Let's re-visit last lecture's data and use some of these declarative functions...

1. Is there `some` instruction to bake?
2. Is `every` review 4 or 5 stars?
3. Using `reduce`, what is the average review rating?

Syntactic Sugar 

Just some "nice-to-haves"

` Template Literals

Shorthand for string concatenation; `${expr}` interpolates a JS expression.

```
const name = "Aven";
const website = "cs571.org";
const dt = new Date().toLocaleTimeString();
console.log("Hello " + name + ", welcome to " + website + "! It is currently " + dt + ".");
console.log(`Hello ${name}, welcome to ${website}! It is currently ${dt}.`);
```

`?` Ternary Operator

`if`, but shorthand! `expr ? ifTrue : ifFalse`

```
const age = 17;
let msg;
if (age >= 18) {
  msg = "You are old enough to vote";
} else {
  msg = "You are not old enough"
}
```

```
const age = 17;
const msg = age >= 18 ? "You are old enough to vote" : "You are not old enough";
```

`...` Spread Operator

Used to create something new with existing data.

We can spread arrays...

```
const cats = ["apricat", "barnaby", "bucky", "colby"];

const newCats = [...cats, "darcy"];
```

`...` Spread Operator

```
const defs = {
  erf: "a plot of land",
  popple: "turbulent seas"
}

const newDefs = {
  ...defs,
  futz: "waste of time"
}
```

... and also objects! These are both *shallow* copies.

Copying

```
let myComplexObj = {name: "Cole", pets: ["pepper", "spud"]};

let refCopy = myComplexObj;              // not really a copy at all!
let shallowCopy = {...myComplexObj};     // nested attributes not copied
let deepCopy = JSON.parse(JSON.stringify(myComplexObj)); // full copy!
```

Interactive Example - Reference Copy

Interactive Example - Shallow Copy

Interactive Example - Deep Copy

? Optional Chaining

Useful when you are uncertain if an object has a certain property.

```
const mgmt = { cfg: { options: { src: { port: 3761 } } } }

const port = mgmt?.cfg?.options?.src?.port;
```

Otherwise we would have to...

```
let port;
if (mgmt && mgmt.cfg && mgmt.cfg.options && mgmt.cfg.options.src) {
  port = mgmt.cfg.options.src.port;
}
```

`??` Null Coalescing Operator

If left-hand is `null` or `undefined`, use right-hand.

```
const IS_ENABLED = env.IS_ENABLED ?? true;

const USERNAME = document.getElementById("username").value ?? "";
```

How does this compare to ternary?

```
const IS_ENABLED = env.IS_ENABLED ? env.IS_ENABLED : true; // always true!
```

Try it on MDN

Note: JavaScript changes quickly!

Both `...` and `??` are not yet supported by the interactive example tool!

Working with CSS Libraries

What are CSS Libraries?

Definition: Software libraries that abstract away the low-level CSS implementation of user-facing elements.

Some popular libraries include...

- Bootstrap
- Foundation
- Semantic UI
- Pure
- UIkit

Bootstrap

getbootstrap.com

Why does the web look alike?

Many, many, many (new) websites use Bootstrap!

OHWL

Partner Finder

CS571

Bootstrap: get (oneself or something) into or out of a situation using existing resources.

Oxford Dictionary

How Bootstrap Works

Bootstrap provides us with...

- Layouts
- Content
- Components
- Utilities

There is much more!

Bootstrap Categories: Layouts

Containers are the most basic element of layouts.

```
<div class="container">
  ...
</div>
```

```
<div class="container-fluid">
  ...
</div>
```

Grids

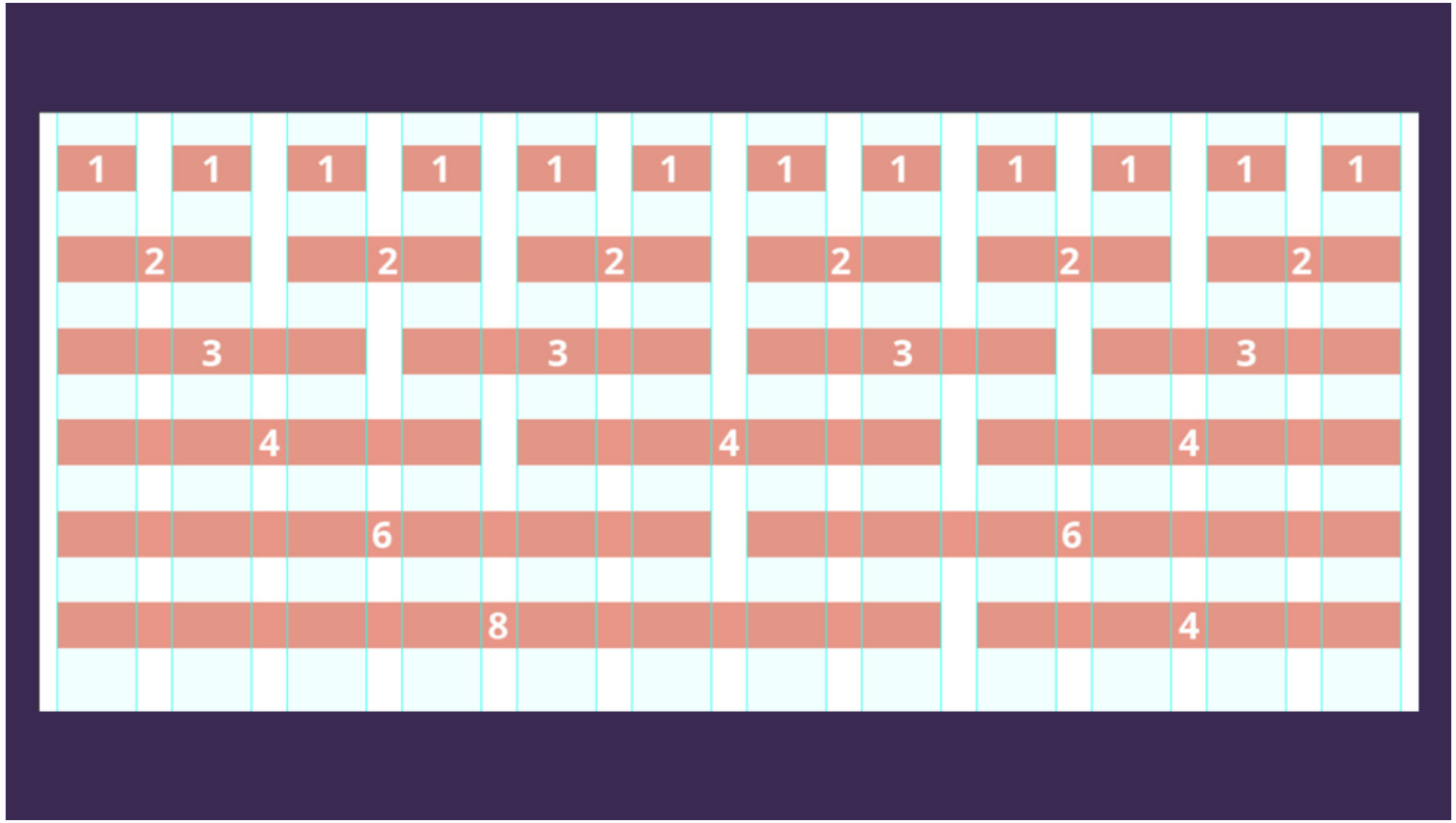
Often, containers will contain **rows** and **columns** to form a grid...

```
<div class="container">
  <div class="row">
    <div class="col-*"></div>
    <div class="col-*"></div>
  </div>
</div>
```

Where `*` is the *column span*.

StackBlitz

Grid System



Medium

Responsive Design

Definition: Responsive design adapts content to a variety of devices and screen sizes.

Width breakpoints determine whether the design will scale or be reorganized.

InVision

Responsive Design

What if we want our webpage to respond to different screen sizes, e.g. phone, tablet, and monitor?

```
<div class="container">
  <div class="row">
    <div class="col-12 col-md-6 col-lg-3"></div>
    <div class="col-12 col-md-6 col-lg-3"></div>
    <div class="col-12 col-md-6 col-lg-3"></div>
    <div class="col-12 col-md-6 col-lg-3"></div>
  </div>
</div>
```

Responsive Design Example

	Extra small <576px	Small ≥576px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Column ordering	Yes				

Bootstrap Categories: Content

Content styling includes basic HTML elements, typography, code, images, tables, figures.

Basic HTML examples:

```
<h1></h1>
<ul></ul>
<input/>
<button></button>
```

These will get the default Bootstrap styling.

Styling of other elements

```
<img src="..." class="img-fluid">
```

```
<table class="table">
  <thead class="thead-dark">
    <tr>
      <th scope="col">...</th>
      ...
```

```
<div class="table-responsive-sm">
  <table class="table">
  ...
```

Bootstrap Categories: Components

Components include all other visual/interactive elements that make up the design, e.g., buttons, forms, navbar, tooltips, etc.

```
<button type="button" class="btn btn-primary">Fill button</button>

<button type="button" class="btn btn-outline-primary">Outline button</button>

<div class="btn-group-toggle" data-toggle="buttons">
  <label class="btn btn-secondary active">
    <input type="checkbox" checked autocomplete="off"> Switch
  </label>
</div>
```

Bootstrap Categories: Utilities

Utilities are not elements themselves, but they modify/control other elements, e.g., adding rounded corners to an image.

```
<img src="..." class="rounded">
```

```
<div class="shadow p-3 mb-5 bg-white rounded">Shadow</div>
```

Your Turn!

Use Bootstrap to make your recipes *responsive*.

Example Home Page

See in CodePen

Also, see cs571.org for responsive design.

Additional Resources

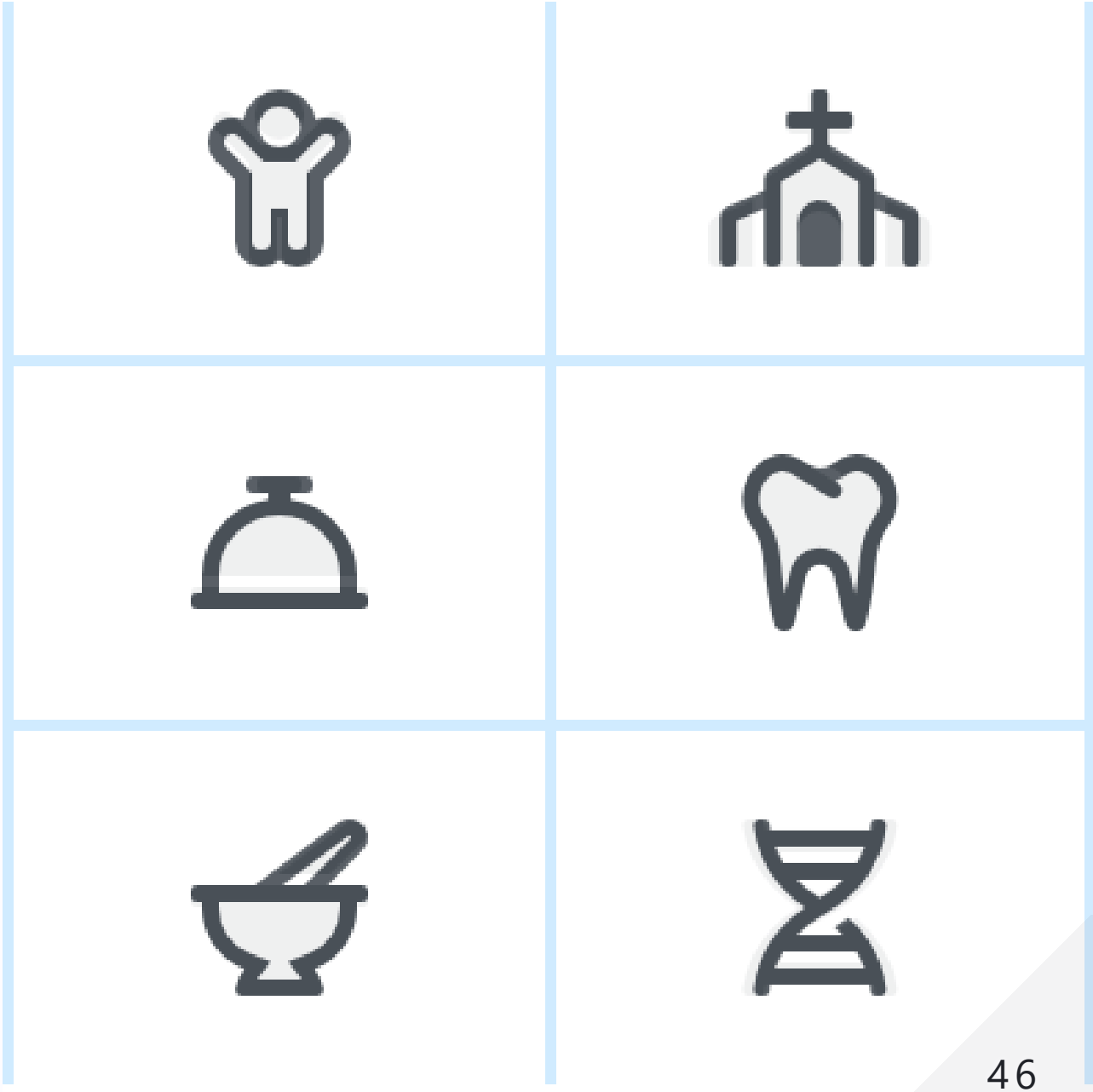
- Bootstrap Documentation
- Tutorial Republic
- W3 Schools

Assets

Asset libraries, e.g., icons, are usually used in conjunction with frameworks such as Bootstrap.

See icon libraries.

Image Source



Questions?