JavaScript 3

CS571: Building User Interfaces

Cole Nelson

Data Buddies Survey

Course Logistics

- Start on HWs early!
- Midterm Exam on Thursday, March 9th 5:45-7:15pm.
- Final Exam on Tuesday, May 9th 7:25-9:25pm.
  - Still will have an extra week for HW12.
  - Last week of lectures will be "fun" (async) lectures.
    - FullStack Development
    - Building *Secure* User Interfaces
- Canvas/HL alternatives... let me know!

JavaScript 2 Recap

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

Callback Functions

```
function greeting1(name) {
  alert('Hello ' + name);
}

const greeting2 = (name) => {
  alert('Whats up ' + name);
}

function processUserInput(callback) {
  const name = prompt('Please enter your name.');
  callback(name);
}

processUserInput(greeting1);
processUserInput(greeting2);
processUserInput((name) => alert("Welcome " + name));
```

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

Declarative vs Imperative Programming

We typically prefer *declarative* programming over *imperative* programming.

Declarative vs Imperative Programming

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

Last time we learned about `forEach`, `filter`, and `map`. Today we'll learn about `slice`, `concat`, `some`, `every`, and `reduce`!

What will we learn today?

- How to use other declarative functions?
- How to use spreading and null coalescing?
- How to perform data copying?
- How to work with CSS libraries?

`slice(begI, endI)` and `concat(arr)`

`slice` returns a shallow copy with an optional beginning (inclusive) and ending (exclusive) index.

```
["apple", "banana", "coconut", "dragonfruit"].slice(1, 3); // ["banana", "coconut"]
```

`concat` joins two arrays together.

```
["apple"].concat(["banana", "coconut"]); // ["apple", "banana", "coconut"]
```

`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!
```

Your turn!

Fetch data from our API and do "interesting" things!

`https://cs571.org/s23/week3/api/data`

1. Can you get this data in Postman? How about using JavaScript?
2. What were the names of the first 3 presidents?
3. Was there *some* president named Thomas?
4. Were *all* the presidents born in the 18th century?

`reduce(cb, start)`

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

```
[0, 4, -1.1, 7.2].reduce((prev, curr) => prev + curr, 0); // 10.1
```

Building Arrays with `reduce`

```
[2, 7, 3, 12, 27].reduce((prev, curr) => {
  if(curr % 2 === 0) {
    prev.push("even");
  } else {
    prev.push("odd");
  }
  return prev;
}, []); // array of even/odd
```

```
[2, 7, 3, 12, 27].reduce((p, c) => [...p, c % 2 === 0 ? "even" : "odd"], []);
```

`...` Spread Operator

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.

?? Null Coalescing Operator

```
const PORT_NUMBER = env.PORT_NUM ?? 80;

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

Your turn!

Using the same presidential data...

1. How many terms were served in total?
2. What are the unique political parties?
3. Construct an object mapping president name to political affiliation.

Data Copying

Why would we need to copy?

- We ❤️ the spread operator.
- Copying a template object.
- Passing an object to function where we don't want the object to be modified.
- Data safety.

Data Copying

`json.parse` and `json.stringify` can also be useful for deep data copying.^

^ lodash is the preferred way to copy.

Data Copying

Sometimes we wish to make copies of data, e.g. we want to duplicate an array of student objects.

1. Reference Copy
2. Shallow Copy
3. Deep Copy

Recall: Variables are containers -- they contain a primitive value or a *pointer* to an object.

Reference Copy

What will the output be?

```
let myBasket = {
  basketId: 154,
  items: ["Apples", "Bananas", "Grapes"]
};
let myRefCopyBasket = myBasket; // *
myRefCopyBasket.basketId = 999;
myRefCopyBasket.items.push("Zucchinis");

console.log(myBasket);
console.log(myRefCopyBasket);
```

Interactive Exercise

Reference Copy

```
let myBasket = {
  basketId: 154,
  items: ["Apples", "Bananas", "Grapes"]
};
let myRefCopyBasket = myBasket; // *
myRefCopyBasket.basketId = 999;
myRefCopyBasket.items.push("Zucchinis");

console.log(myBasket);
console.log(myRefCopyBasket);
```

```
{basketId: 999, items: ['Apples', 'Bananas', 'Grapes', 'Zucchinis']}
{basketId: 999, items: ['Apples', 'Bananas', 'Grapes', 'Zucchinis']}
```

Shallow Copy

What will the output be?

```
let myBasket = {
  basketId: 154,
  items: ["Apples", "Bananas", "Grapes"]
};
let myShallowCopyBasket = {...myBasket}; // *
myShallowCopyBasket.basketId = 999;
myShallowCopyBasket.items.push("Zucchinis");

console.log(myBasket);
console.log(myShallowCopyBasket);
```

Interactive Exercise

Shallow Copy

```
let myBasket = {
  basketId: 154,
  items: ["Apples", "Bananas", "Grapes"]
};
let myShallowCopyBasket = {...myBasket}; // *
myShallowCopyBasket.basketId = 999;
myShallowCopyBasket.items.push("Zucchinis");

console.log(myShallowCopyBasket);
console.log(myRefCopyBasket);
```

```
{basketId: 154, items: ['Apples', 'Bananas', 'Grapes', 'Zucchinis']}
{basketId: 999, items: ['Apples', 'Bananas', 'Grapes', 'Zucchinis']}
```

Deep Copy

What will the output be?

```
let myBasket = {
  basketId: 154,
  items: ["Apples", "Bananas", "Grapes"]
};
let myDeepCopyBasket = JSON.parse(JSON.stringify(myBasket)); // *
myDeepCopyBasket.basketId = 999;
myDeepCopyBasket.items.push("Zucchinis");

console.log(myBasket);
console.log(myDeepCopyBasket);
```

Interactive Exercise

Deep Copy

```
let myBasket = {
  basketId: 154,
  items: ["Apples", "Bananas", "Grapes"]
};
let myDeepCopyBasket = JSON.parse(JSON.stringify(myBasket)); // *
myDeepCopyBasket.basketId = 999;
myDeepCopyBasket.items.push("Zucchinis");

console.log(myBasket);
console.log(myDeepCopyBasket);
```

```
{basketId: 154, items: ['Apples', 'Bananas', 'Grapes']}
{basketId: 999, items: ['Apples', 'Bananas', 'Grapes', 'Zucchinis']}
```

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

How Bootstrap Works

Bootstrap provides us with...

- Layouts
- Content
- Components
- Utilities

There is much more!

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

Bootstrap Categories: Layouts

Containers are the most basic element of layouts.

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

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

```
<div class="container-{breakpoint}"> <!-- xs, sm, md, lg, xl-->
  ...
</div>
```

Containing a Grid

Basic usage of a grid...

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

Where `*` is *grid class* and `^` is *column size*.

You can specify multiple of these classes to make your website responsive to phone, tablet, and desktop!

	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>
```

Example Home Page

See in CodePen

Also, see cs571.org for responsive design.

Additional Resources

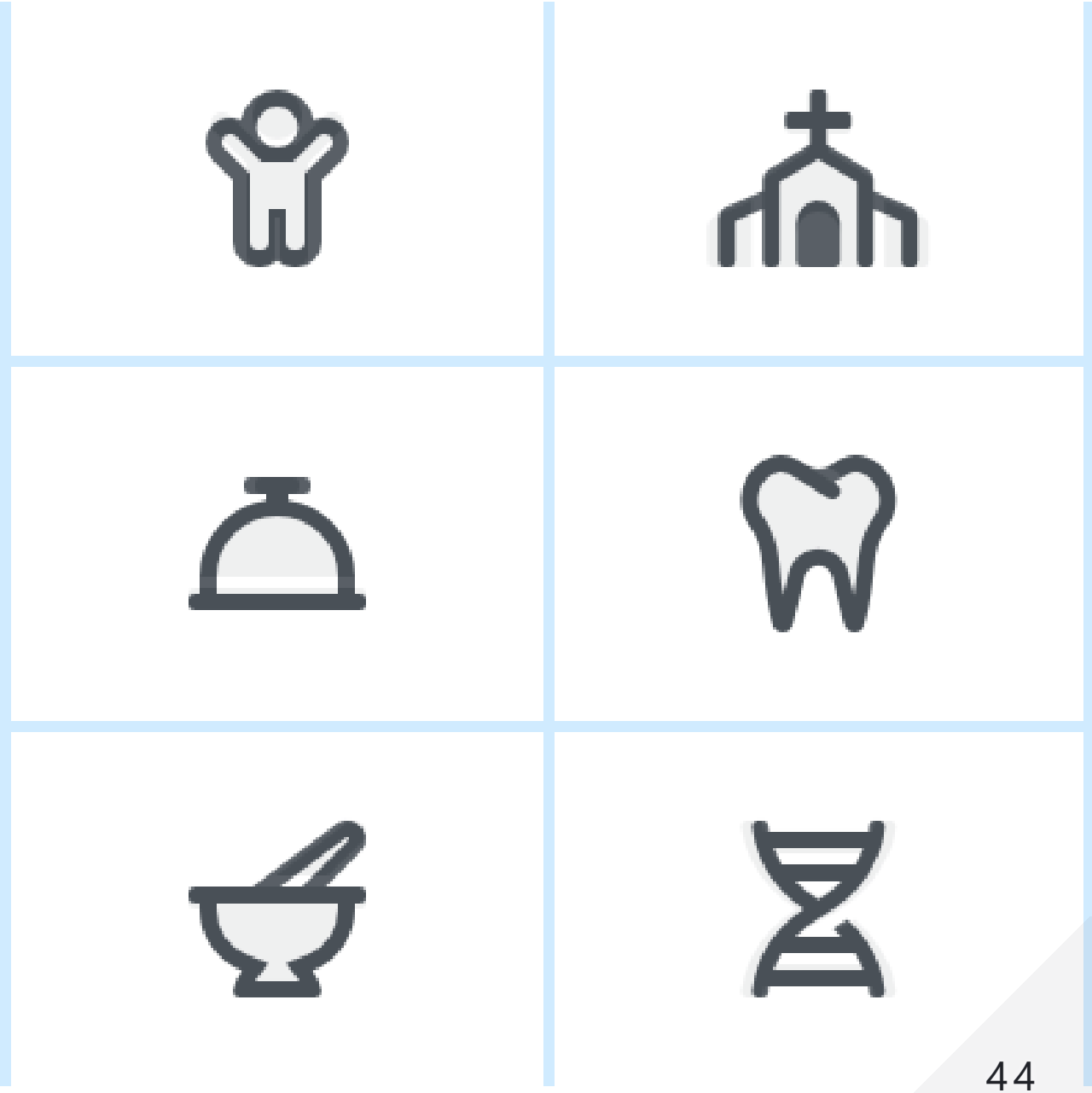
- Bootstrap Documentation
- Tutorial Republic
- W3 Schools

Assets

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

See icon libraries.

Image Source



Questions?