Interactive Web Programming

1st semester of 2021

Murilo Camargos
(**murilo.filho@fgv.br**)

Heavily based on **Victoria Kirst** slides

Schedule

Today:

- More custom events
- `this` and `bind()` revisited
- First-class functions

Tuesday (April 13):

- Asynchronous JavaScript
- `fetch`
- `Promises`

Announcements:

- HW3 is out. Due to **April 9**.
- HW 1, 2 and 3 will compose the A1 score.

A quick note on HW2

A quick note on HW2

DOM manipulation:

- Use `document.createElement(elementNameStr)` to **create HTML nodes**.
- A **container** can be **any HTML element** that **contains** one or more HTML elements or text nodes.
- Use `containerNode.appendChild(anotherNode)` to **append an HTML node** to another HTML node (semantically seen as a container).
- **Find elements** previously added to the DOM using `document.querySelector` or `document.querySelectorAll`: the first one returns **a Node**, while the other returns **a list of Nodes**. **Be aware of that!**

A quick note on HW2

Event listeners:

- Use `node.addEventListener(eventNameStr, functionVar)` to add an event listener to an **HTML node**.
- **Do not call** the functionVariable `node.addEventListener(eventNameStr, functionVar())`.
- Use `node.removeEventListener(eventNameStr, functionVar)` to remove an event listener from an **HTML node**.
- **Be aware of how your browser downloads and executes JS files!!!**
- **Hint:** for the select box, take a look at the "**change**" event: https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/change_event

A quick note on HW2

General stuff:

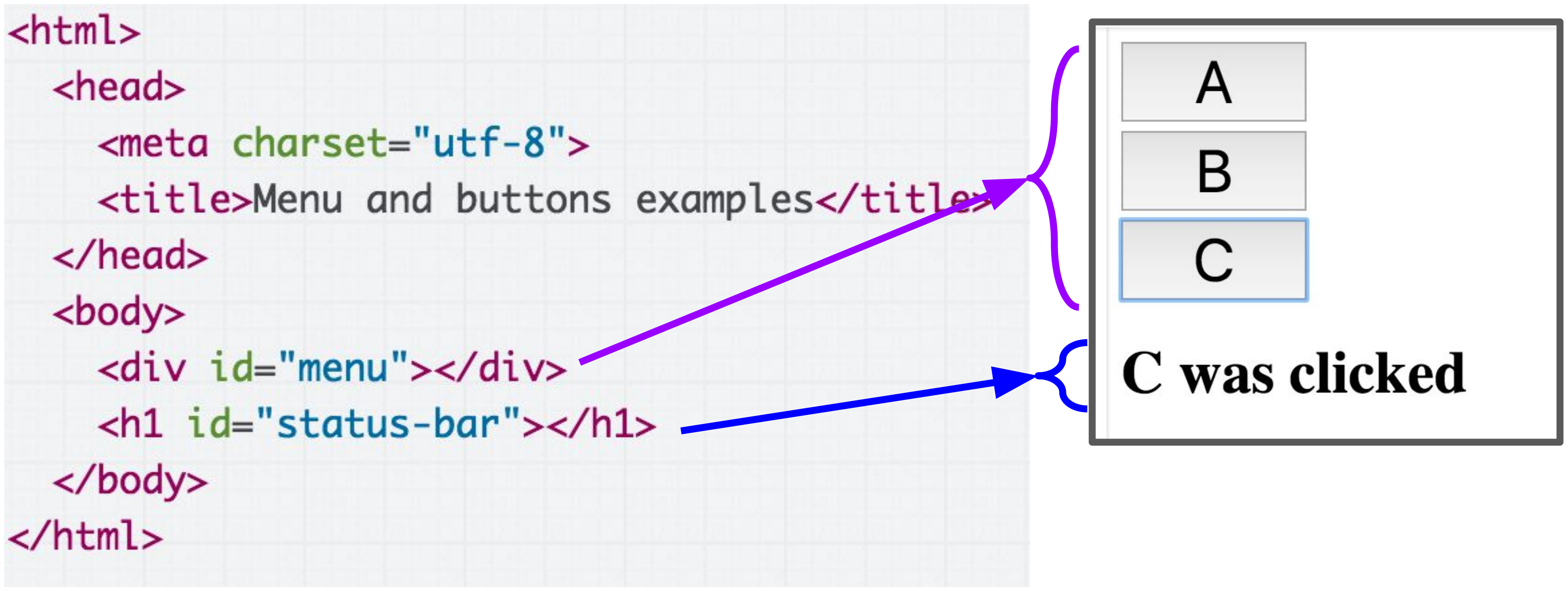
- **Always** commit your work on git, even if it doesn't work yet or is incomplete.
- I'll grade **everything** you've done!

On the deadlines:

- If you turn **HW1** until **April 12**, you can get up to **40%** of the grade.
- If you turn **HW2** until **April 12**, you can get up to **70%** of the grade.
- **April 12** will be the **last day** for the first block of homework to compose **A1 grade**: HW1, HW2, and HW3.

A quick review of ES6 classes

Example: Buttons

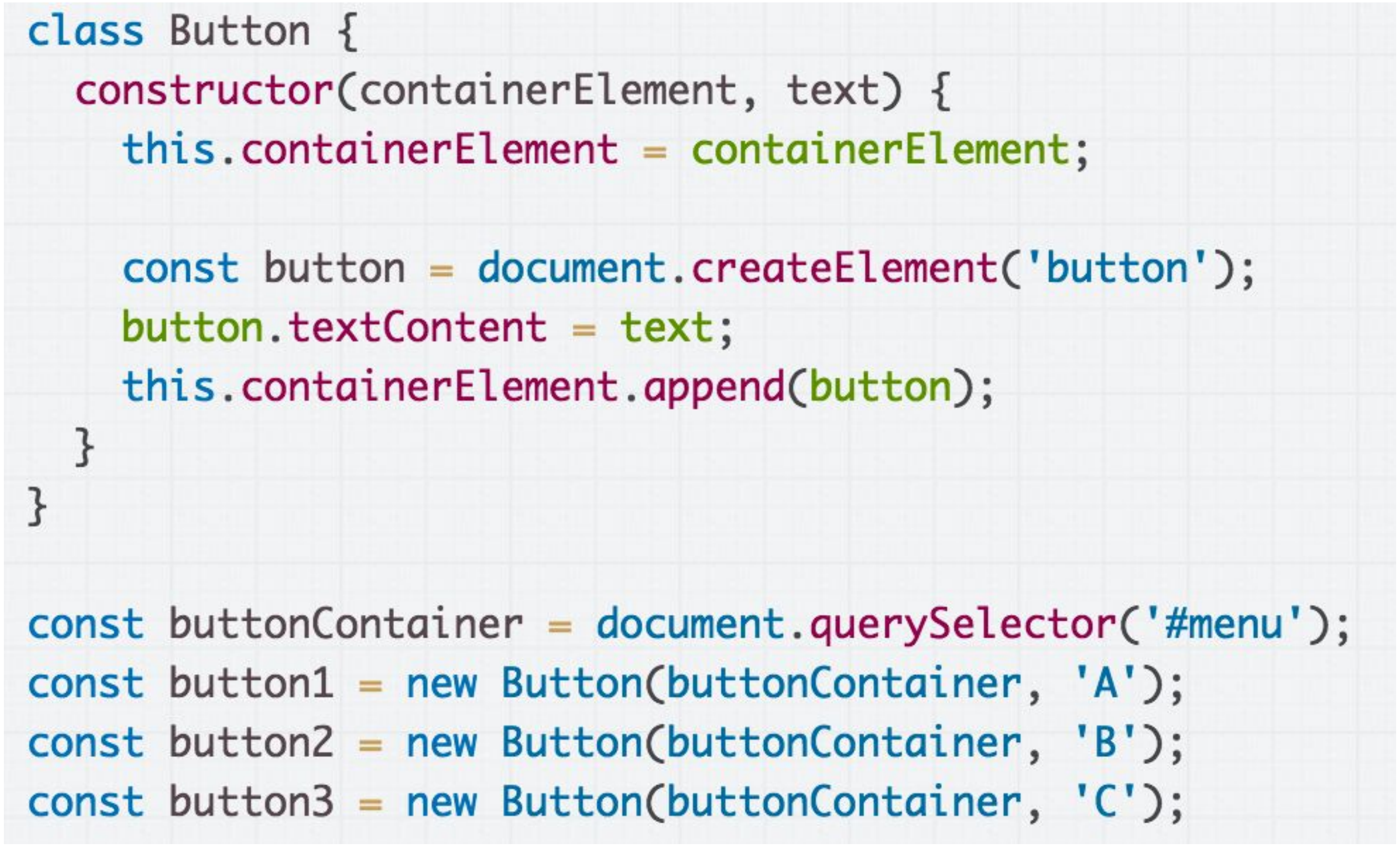


```
<html>
  <head>
    <meta charset="utf-8">
    <title>Menu and buttons examples</title>
  </head>
  <body>
    <div id="menu"></div>
    <h1 id="status-bar"></h1>
  </body>
</html>
```

C was clicked

We want to:

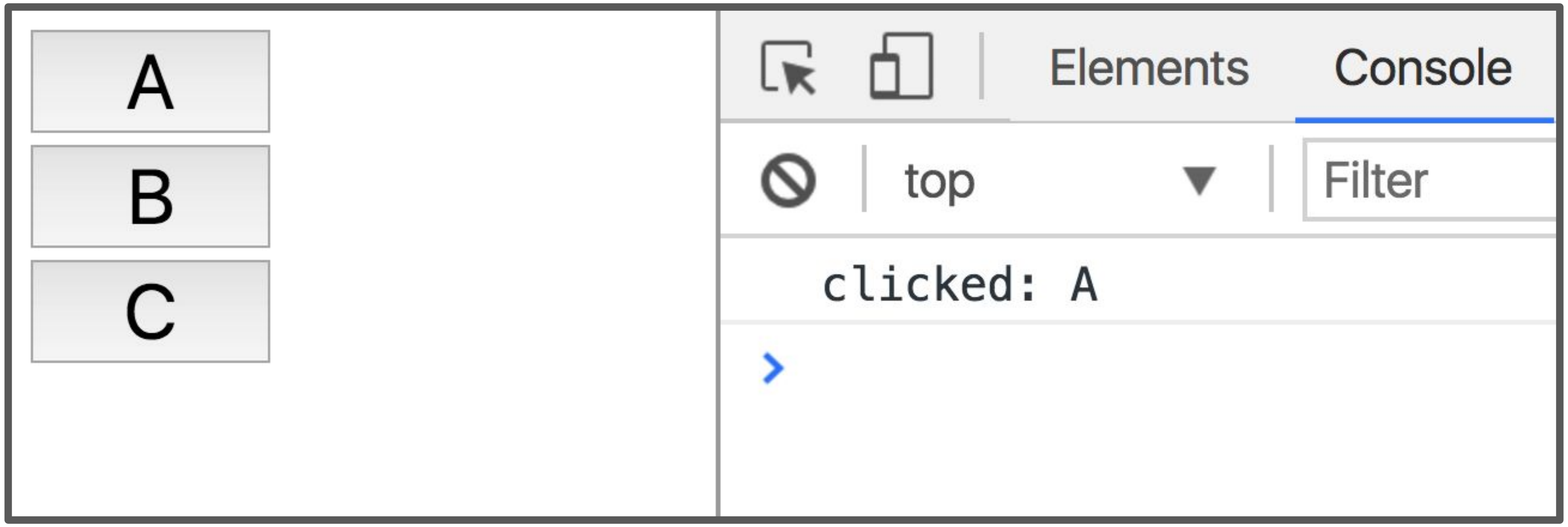
- Fill the `<div id="menu"></div>` with buttons A, B, and C
- Update the `<h1>` with the button that was clicked
- Live example



```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;

    const button = document.createElement('button');
    button.textContent = text;
    this.containerElement.append(button);
  }
}

const buttonContainer = document.querySelector('#menu');
const button1 = new Button(buttonContainer, 'A');
const button2 = new Button(buttonContainer, 'B');
const button3 = new Button(buttonContainer, 'C');
```

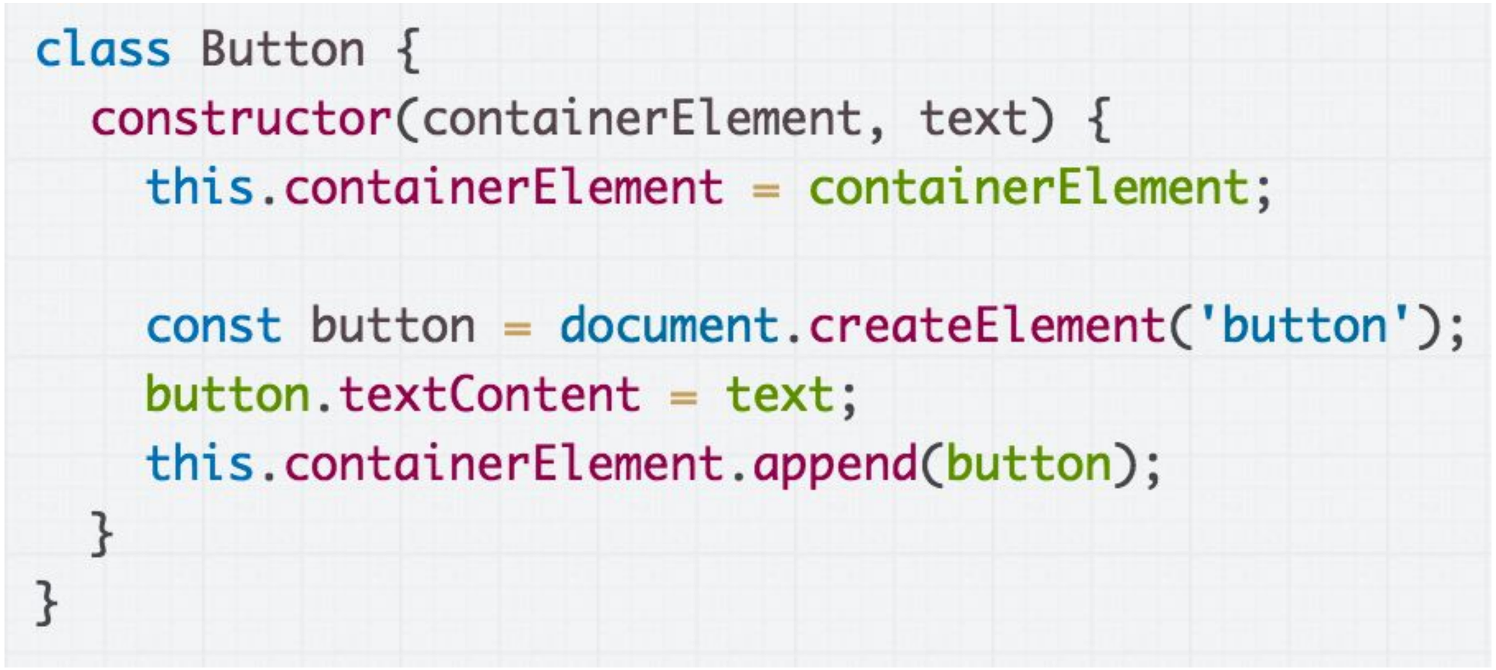
First step: Create a `Button` class and create three `Button`s. (CodePen)

Click handler for Button



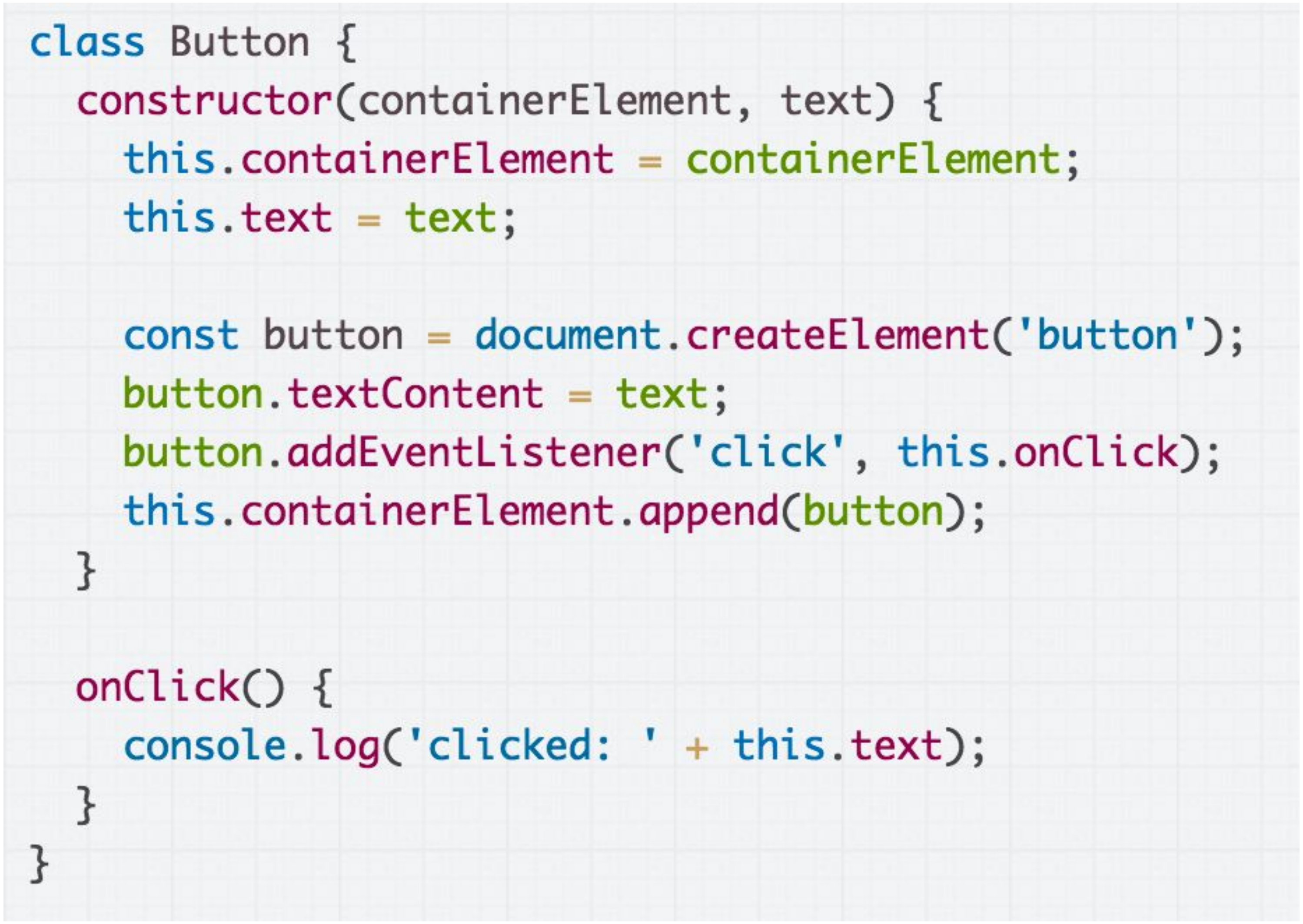
```
clicked: A
```

Let's make it so that every time we click a button, we print out which button was clicked in the console. (Live)



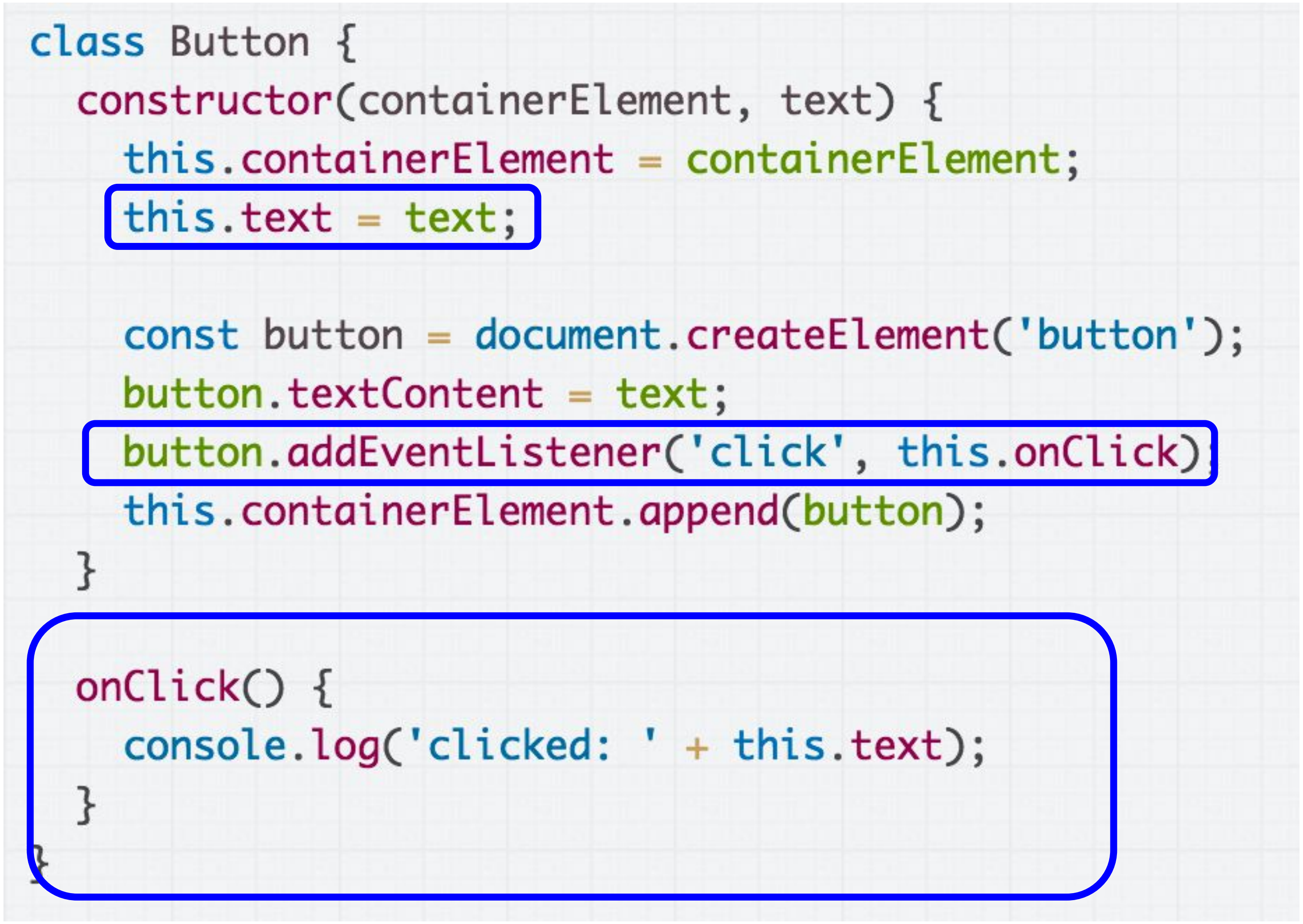
```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;

    const button = document.createElement('button');
    button.textContent = text;
    this.containerElement.append(button);
  }
}
```

Starting with this definition of `Button`...



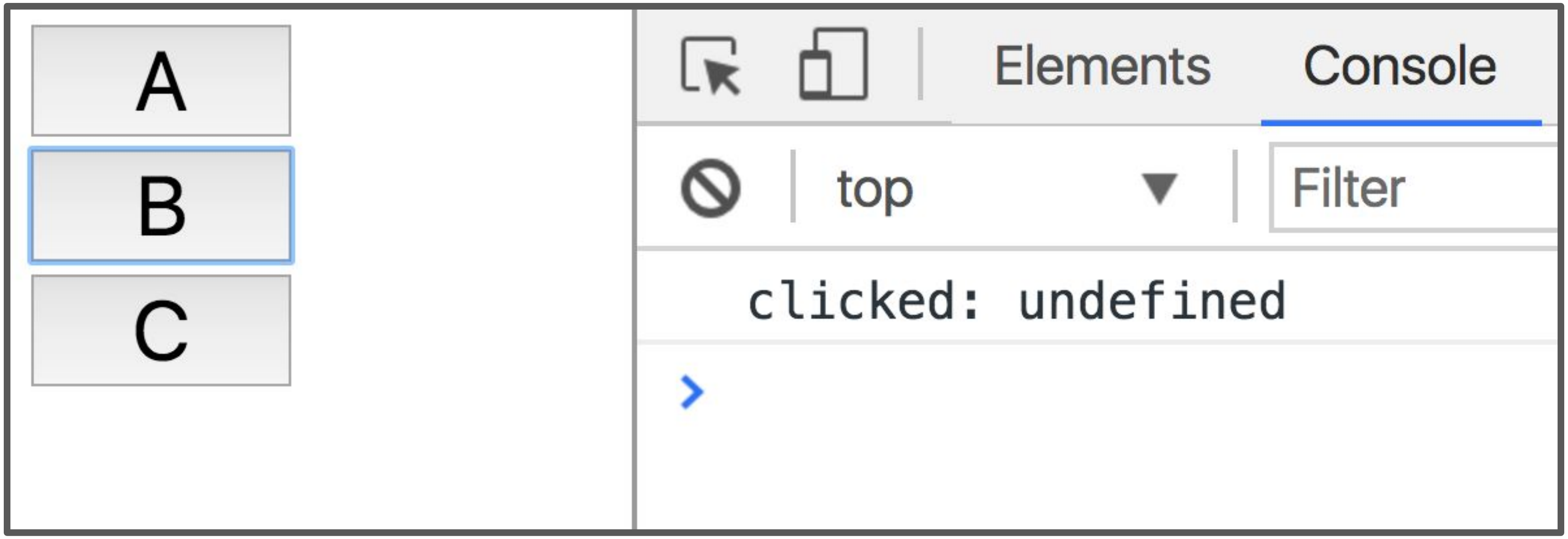
```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

An initial attempt might look like this. (CodePen)



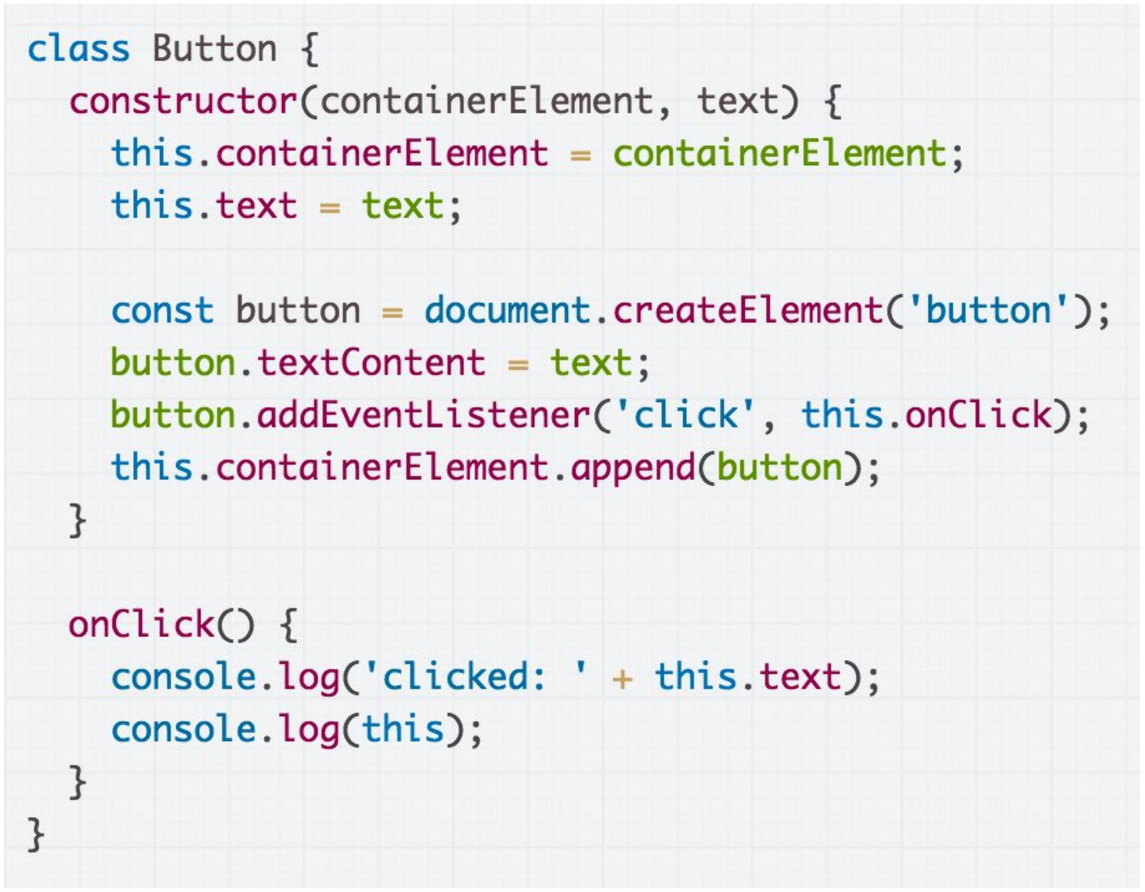
```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

An initial attempt might look like this. (CodePen)

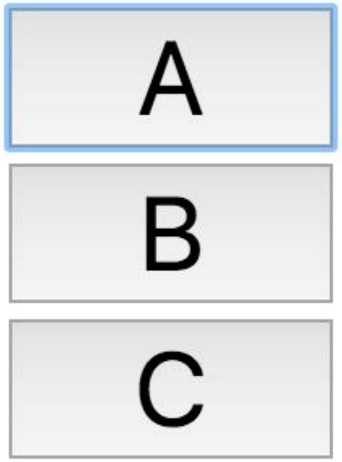


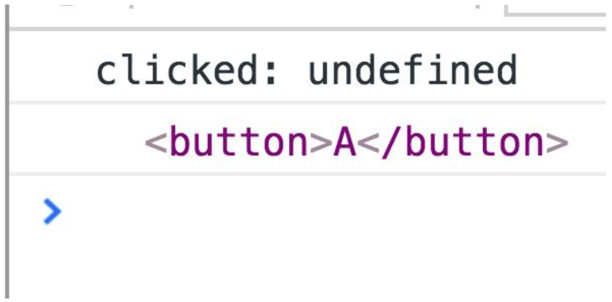
```
clicked: undefined
```

But when we run it, that gives us "clicked: undefined" (CodePen) **Why?**



```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
    console.log(this);
  }
}
```





```
clicked: undefined
  <button>A</button>
>
```

That's because the value of `this` in `onClick` is not the `Button` object; it is the `<button>` element to which we've attached the `onClick` event handler.

`this` in JavaScript

this in the constructor

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }
}
```

In the constructor of a class, this refers to the new object that is being created.

That's the same meaning as this in Java or C++.

this in the constructor

```
// Java
public class Point {
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public int x;
  public int y;
}
```

Here's roughly the equivalent code in Java. `this` refers to the new object that is being created.

`this` in Java

```
// Java
public class Point {
  ...

  String toString() {
    return this.x + ", " + this.y;
  }
}
```

In Java, `this` **always** refers to the new instance being created, no matter what method you're calling it from, or how that method is invoked.

`this` in JavaScript

```
class Point {
  ...

  toString() {
    return this.x + ", " + this.y;
  }
}
```

But in JavaScript, `this` **can have a different meaning** if used outside of the constructor, depending on the **context** in which the function is called.

`this` in JavaScript

```
toString() {
  return this.x + ", " + this.y;
}
```

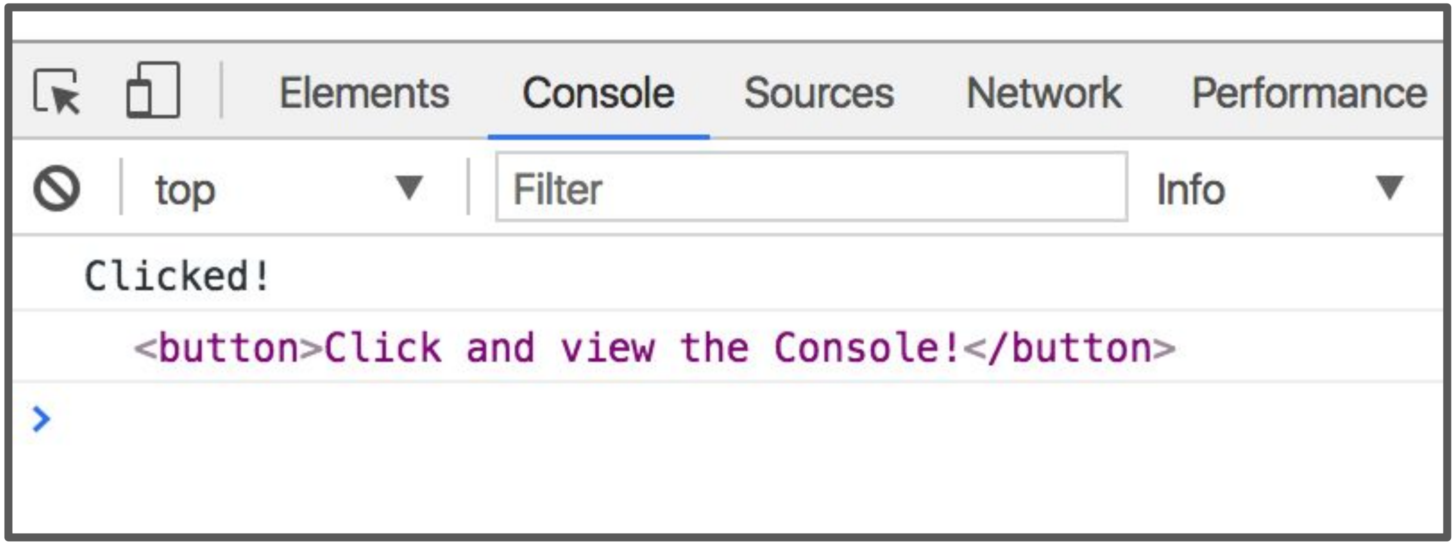
In JavaScript, `this` is:

- A implicit **parameter** that is passed to **every JavaScript function**, including functions not defined in a class!
- The value of the `this` parameter changes depending on how it is called.

`this` in `addEventListener`

```
function onClick() {
  console.log('Clicked!');
  console.log(this);
}

const button = document.querySelector('button');
button.addEventListener('click', onClick);
```

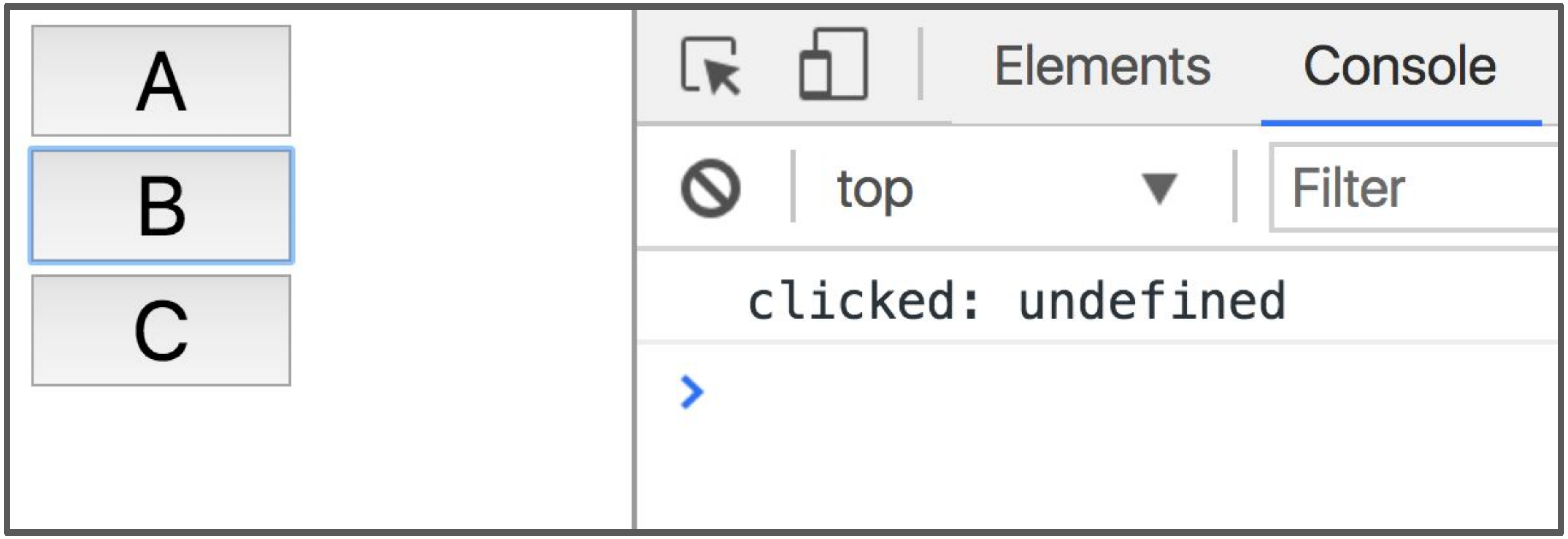
When used in an event handler, `this` is set to the **element to which that the event was added.** (mdn / CodePen / live)



```
Clicked!
  <button>Click and view the Console!</button>
>
```

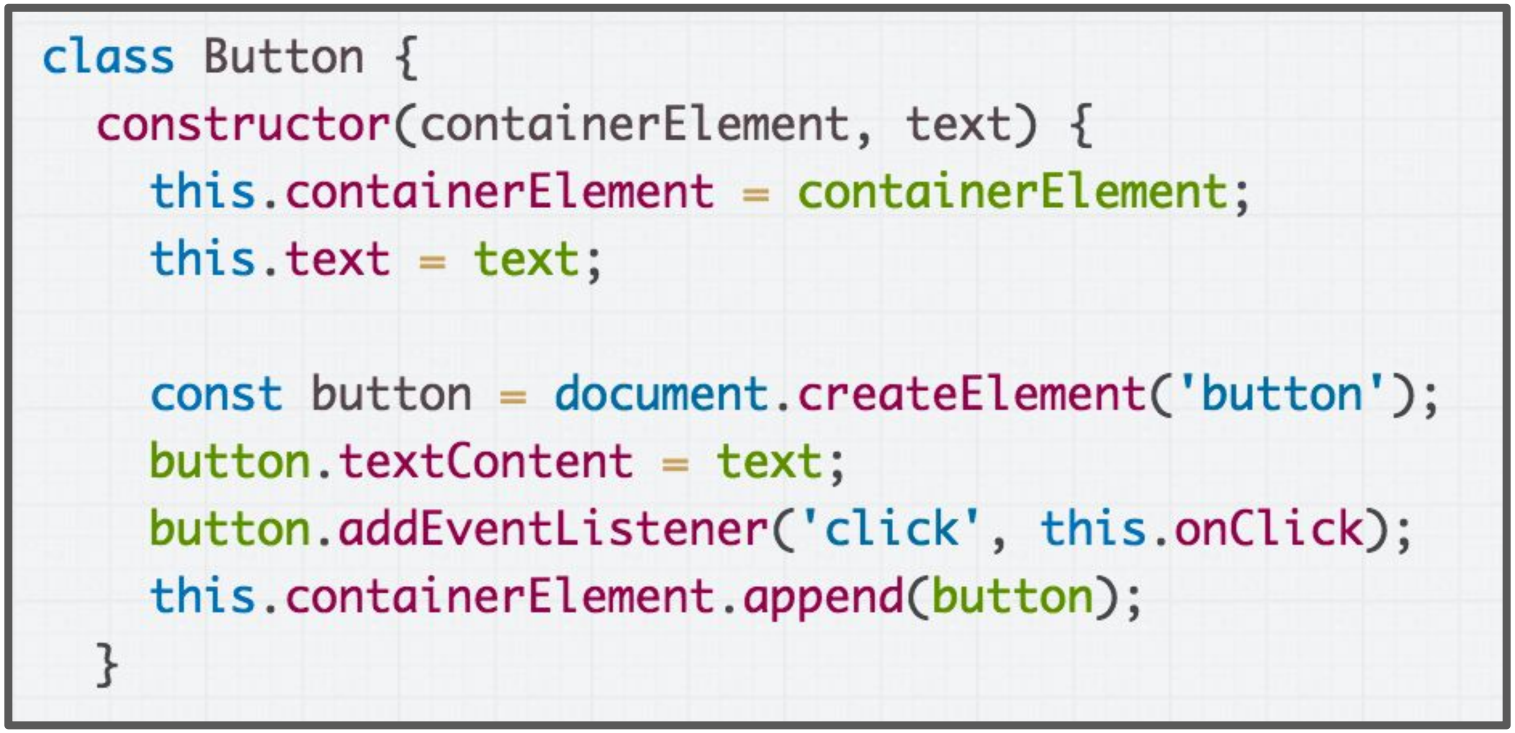
```
function onClick() {
  console.log('Clicked!');
  console.log(this);
}
const button = document.querySelector('button');
button.addEventListener('click', onClick);
```

In `onClick`, this refers to `<button>` because it `onClick` was invoked by `addEventListener`.



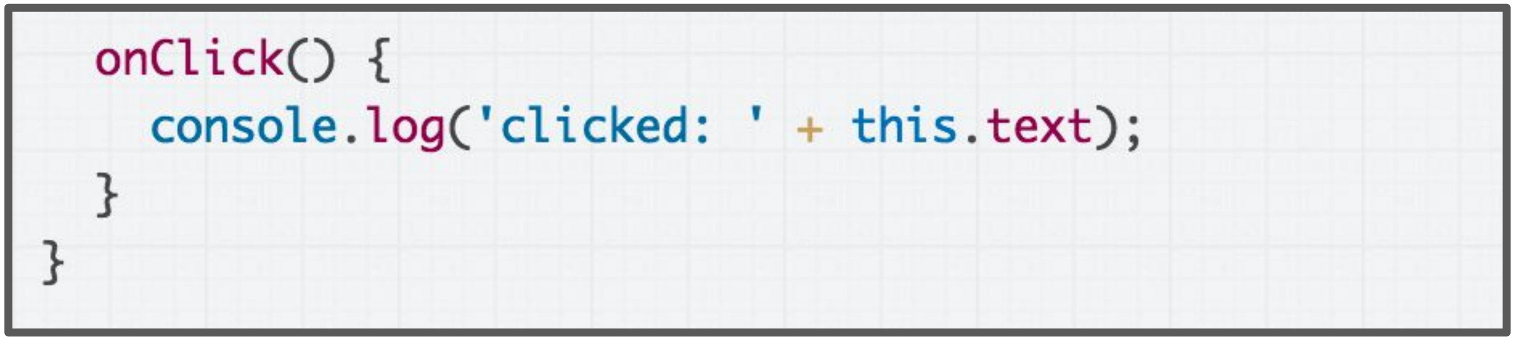
```
clicked: undefined
```

Let's revisit our undefined text… (CodePen)



```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }
```

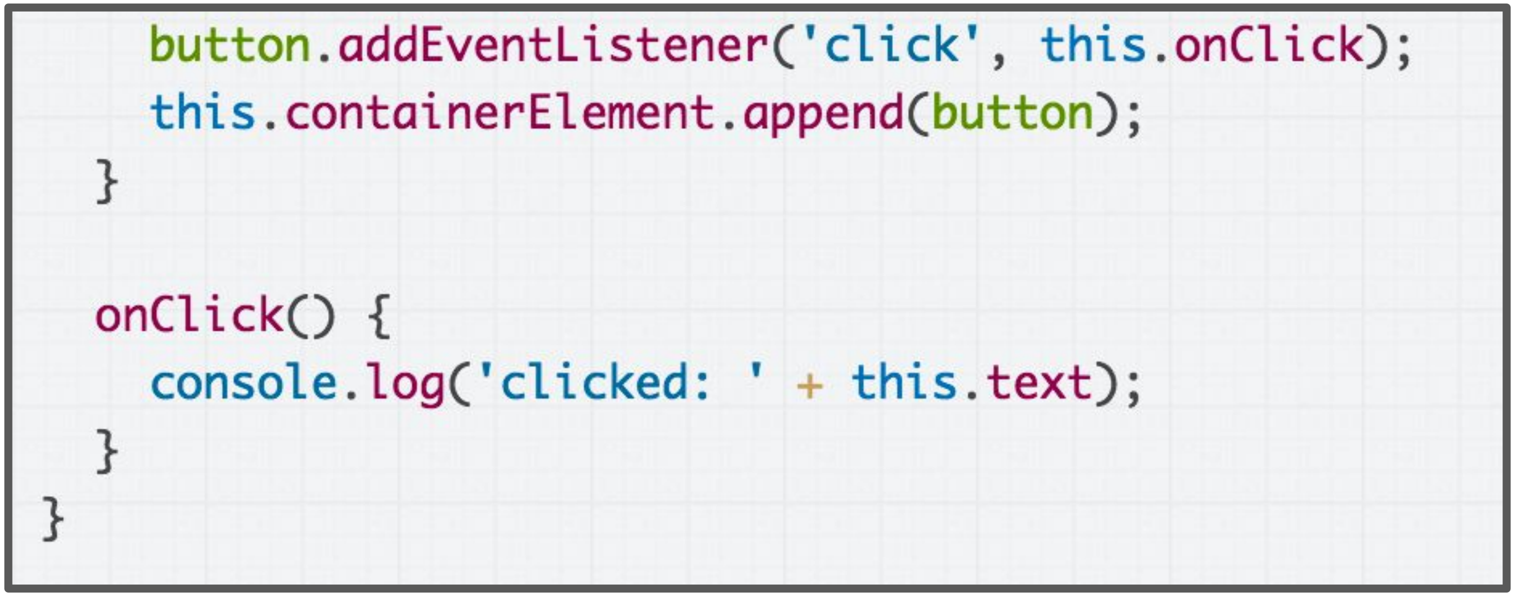
In the constructor, `this` refers to the new object we're creating. No problems here.



```
  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

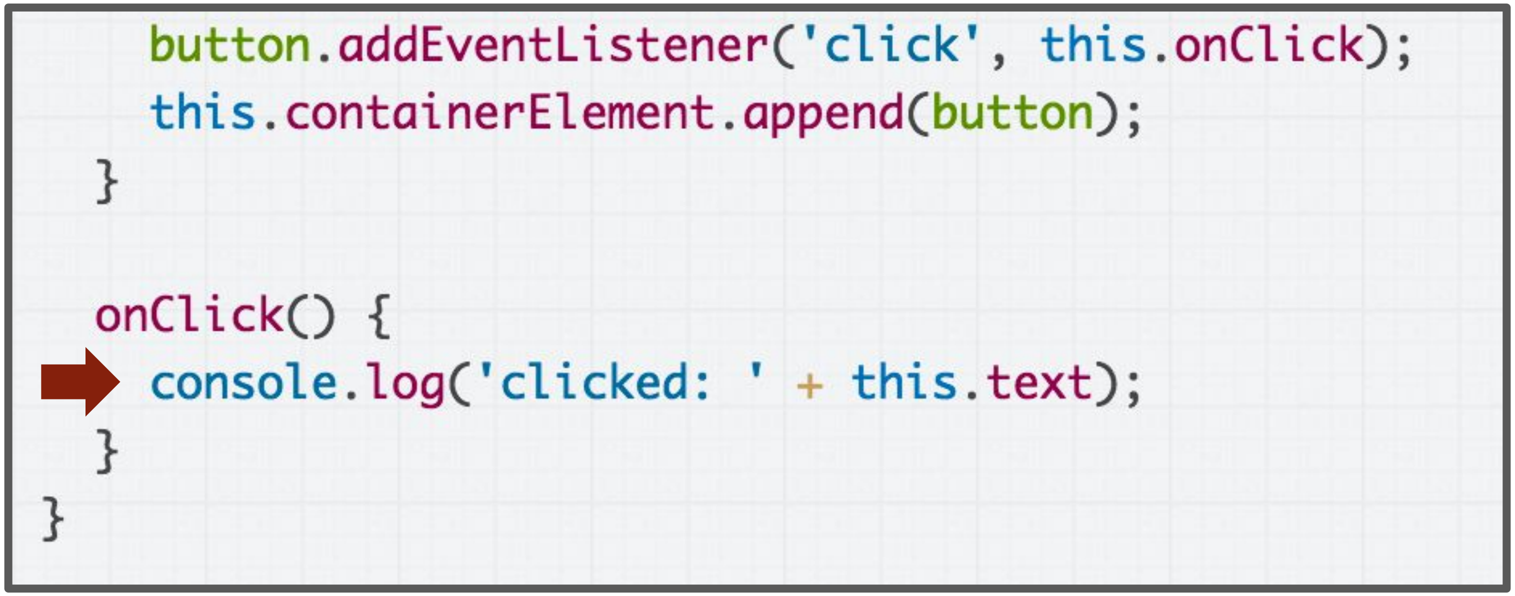
But in `onClick`, `this` will mean something different depending on how the function is called.

That is because we are using `this` in a function that is **not** a constructor.



```
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

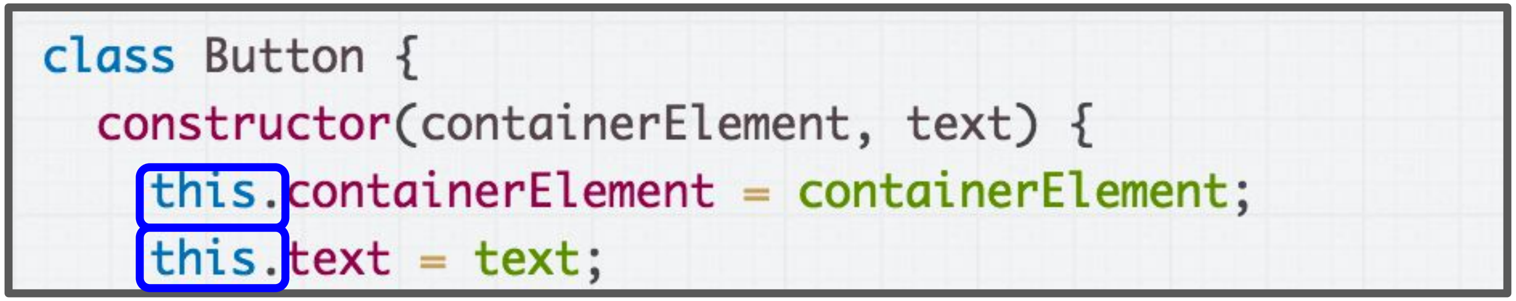
Specifically, because `onClick` is attached to the `<button>` via `addEventListener`...



```
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

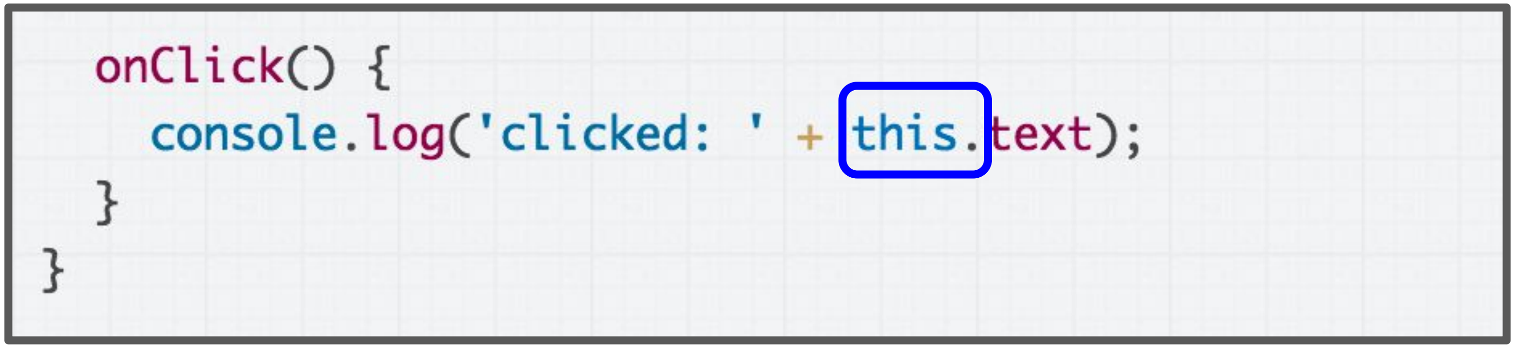
...we know the value of `this` will be the `<button>` element when the click event is fired and invokes `onClick`.

Since [`HTMLButtonElement`] doesn't have a `text` property, `this.text` is `undefined`.



```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;
```

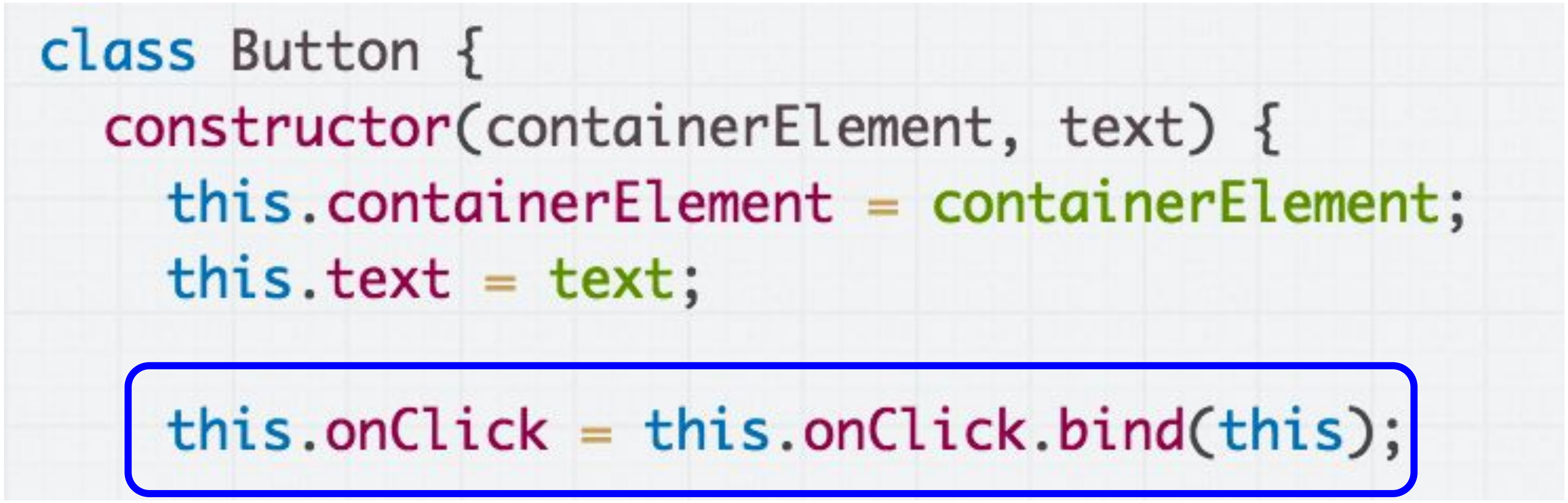
...



```
  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

It'd be nice if we could set the value of "`this`" in `onClick` to be the `Button` object, like it is in the constructor.

"Bind" the value of `this`



```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);
```

That is what this line of code does:

"Hey, use the current value of `this` in `onClick`"

(And the current value of this is the new object, since we're in the constructor)

CodePen / Live

`bind` in classes

```
constructor() {
  const someValue = this;
  this.methodName = this.methodName.bind(someValue);
}
```

This is saying:

- Make a copy of ***methodName***, which will be the exact same as ***methodName*** except `this` in ***methodName*** is always set to the `someValue`
- The value of `someValue` is `this` to `bind()`, which is the value of the new object since we are in the constructor

`bind` in classes

```
constructor() {
  this.methodName = this.methodName.bind(this);
}
```

And of course, you don't need the intermediate `someValue` variable.

CodePen / Live

One more time…

this in the constructor

this in the constructor refers to the new object you are creating.

```
constructor(x, y) {
  this.x = x;
  this.y = y;
}
```

this in a function

`this` in a function that is **not** a constructor has a different value, depending on **how the function is called**.

```
onClick() {
  console.log(this.x);
  console.log(this.u);
}
```

- When invoked as a response to an event, the `this` in `onClick` will be `Event.targetElement`, or the element onto which the `onClick` event handler was attached.

A consistent this

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

Right now, this in the constructor always refers to the new object we're creating...

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

But `this` in `onClick` function refers to a different value, depending on how onClick is called.

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

It'd be nice if we could make the "`this`" value in `onClick`:

- Refer to the new object we're constructing, instead of things like the dom element, etc
- And make it always refer to the new object we're constructing

A consistent `this`

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.onClick = this.onClick.bind(someParam);
  }
```

That's what **bind** does:

- It is saying, "Hey, in the `onClick` function, I want the `this` value to always be ***someParam***," i.e. the value that we are passing as a parameter to `bind`.

A consistent this

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.onClick = this.onClick.bind(someParam);
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

We want the value of `this` in `onClick` to be the value of the new object being created.

In other words, we want ***someParam*** to be the value of the new object being created.

A consistent **this**

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.onClick = this.onClick.bind(someParam);
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

In the constructor, how do we access the new object being created?

A consistent this

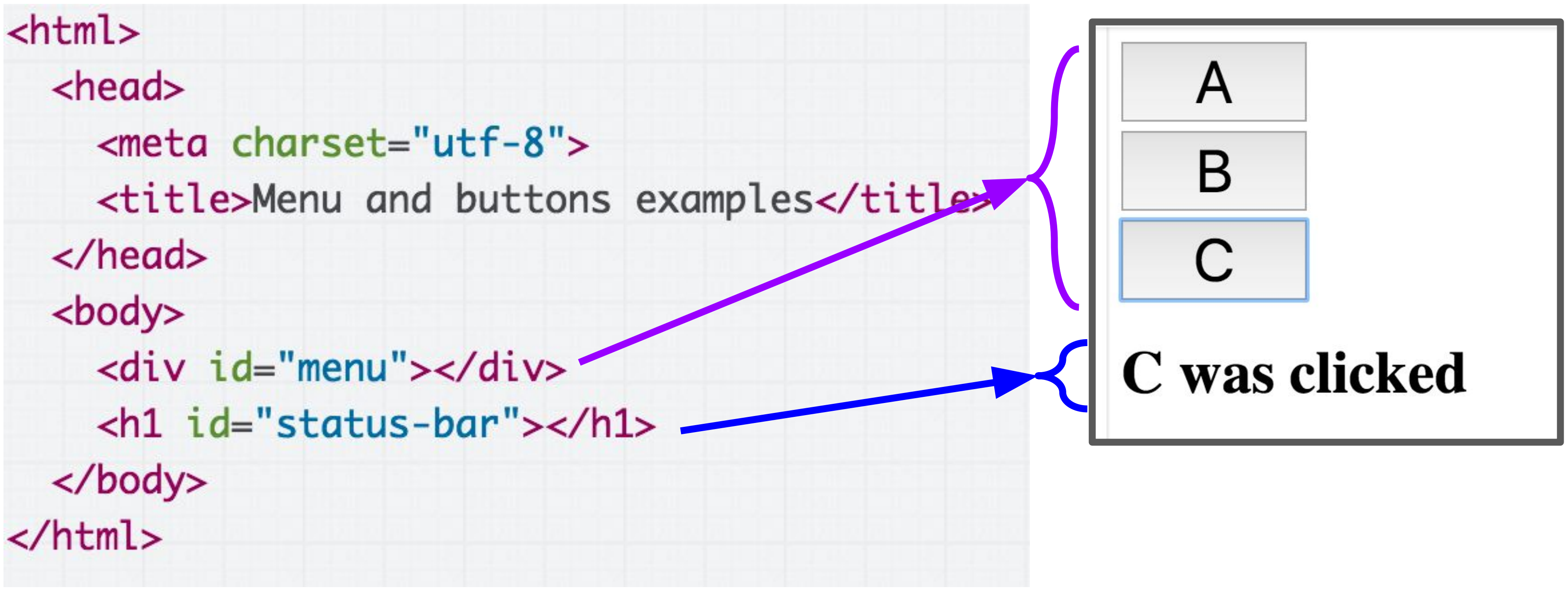
```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.onClick = this.onClick.bind(this);
  }

  onClick() {
    console.log(this.x);
    console.log(this.u);
  }
}
```

In the constructor, the new object is referenced by `this`.

Now the `this` in `onClick` always referring to the new object.

What were we trying to do again?

Example: Buttons



```
<html>
  <head>
    <meta charset="utf-8">
    <title>Menu and buttons examples</title>
  </head>
  <body>
    <div id="menu"></div>
    <h1 id="status-bar"></h1>
  </body>
</html>
```

C was clicked

We want to:

- Fill the `<div id="menu"></div>` with buttons A, B, and C
- Update the `<h1>` with the button that was clicked
- Live example

(Contrived) OO example

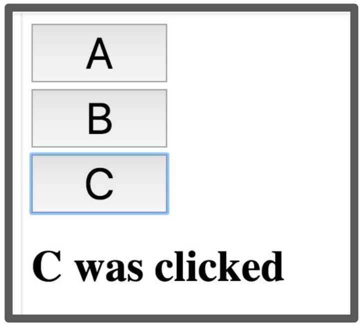
For practice, we'll write this using 2 classes:

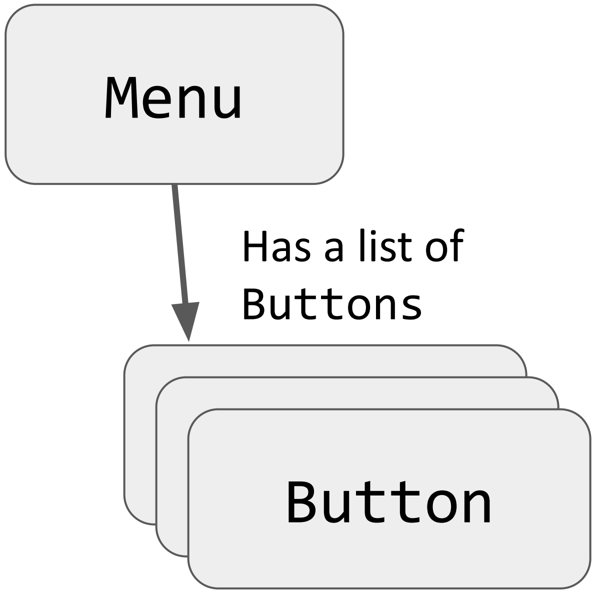
Menu:

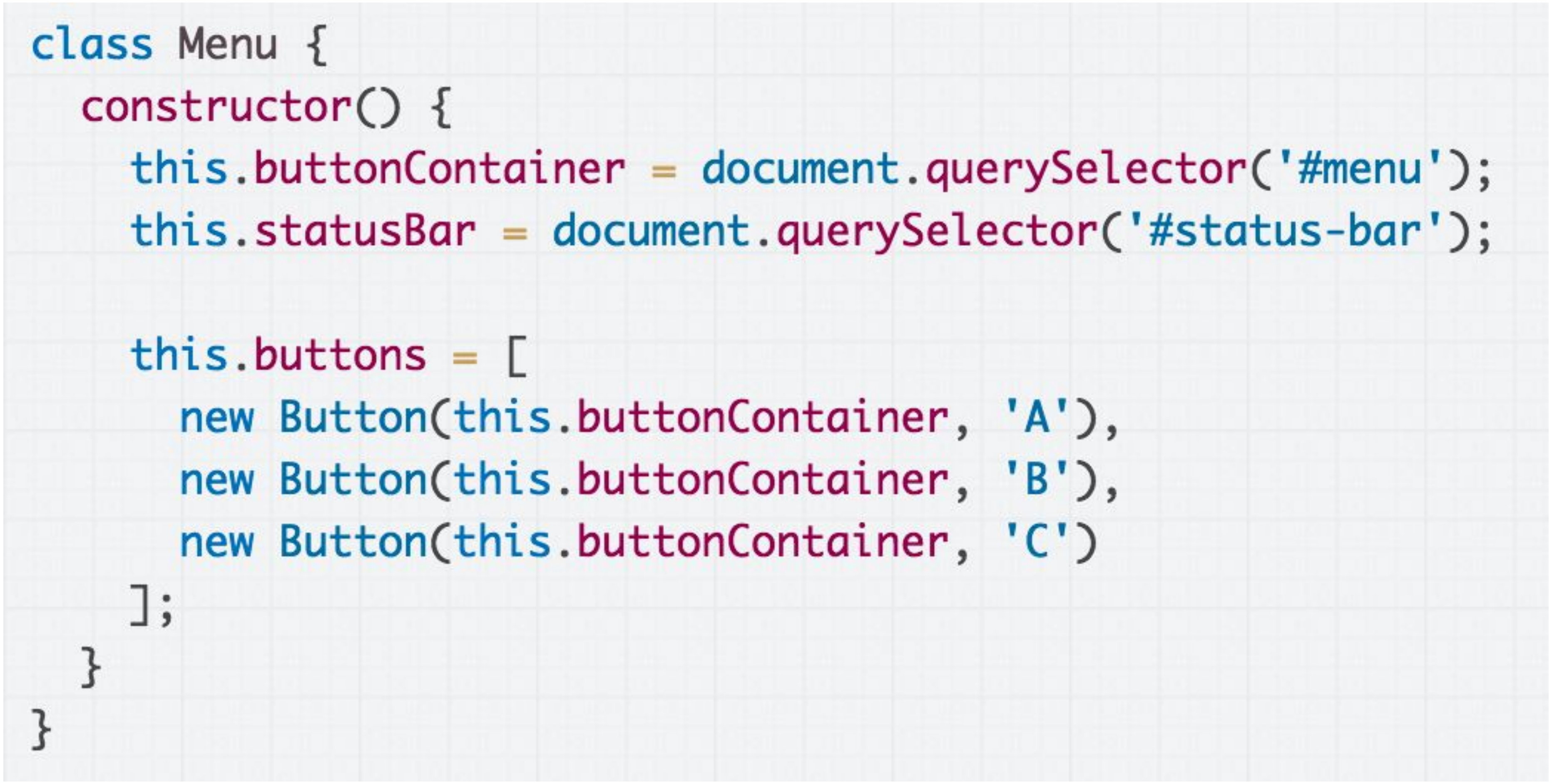
- Has an array of `Buttons`
- Also updates the <h1> with what was clicked

Button:

- Notifies Menu when clicked, so that `Menu` can update the <h1>

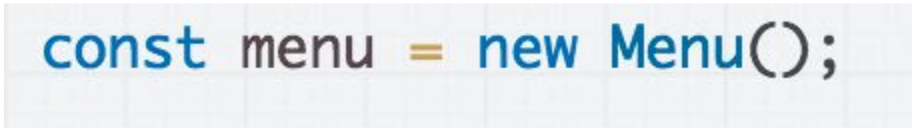






```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }
}
```

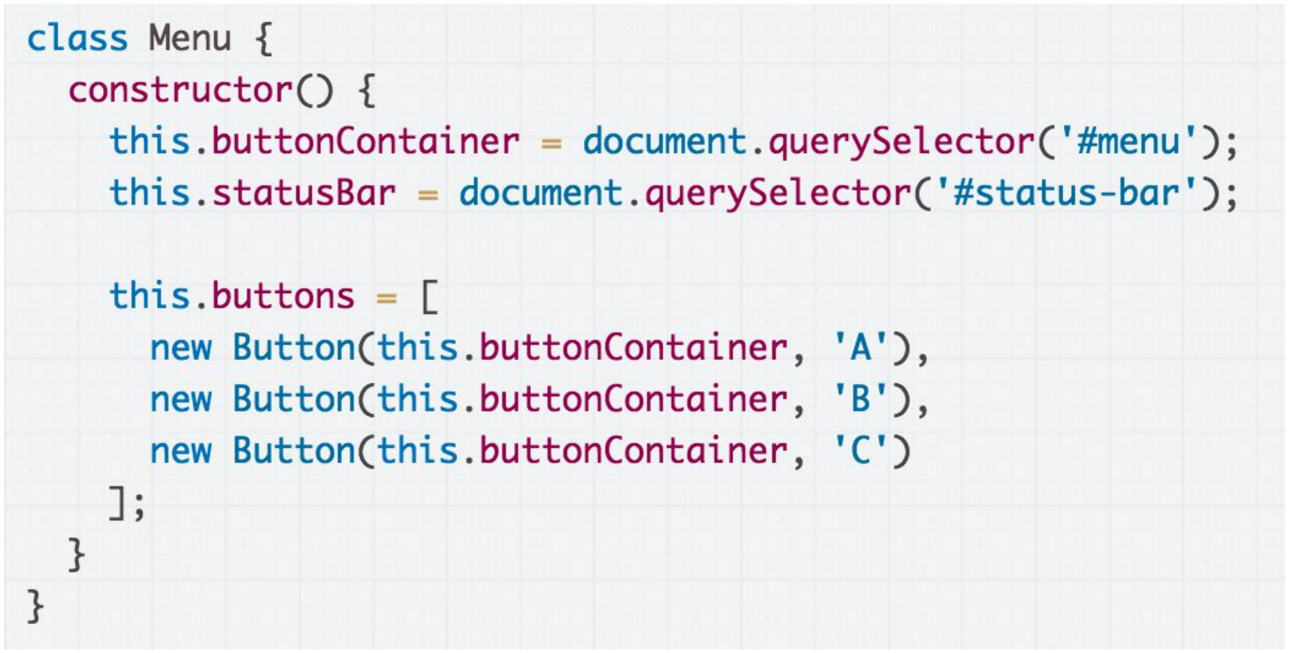
Partial solution: We create a Menu class, which creates the Buttons (CodePen)



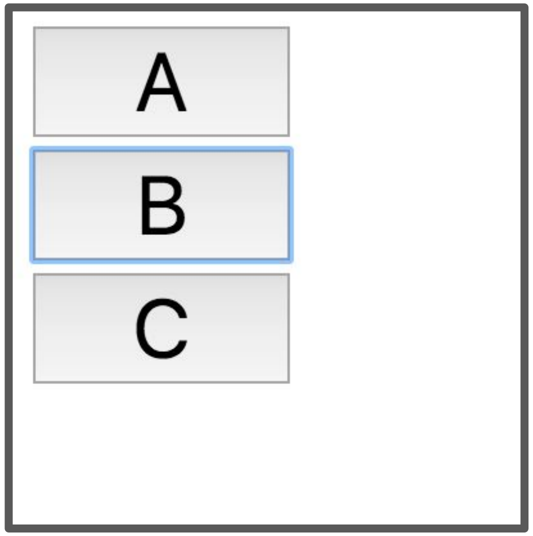
```
const menu = new Menu();
```

Then we create the `Menu` (and the menu creates the `Buttons`) when the page loads. (CodePen)

Update Menu when Button clicked



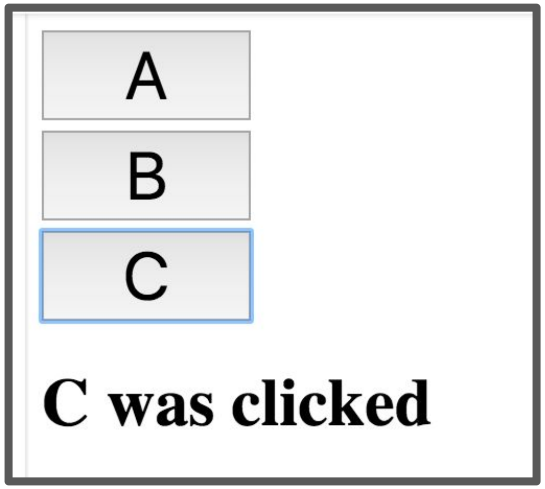
```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }
}
```



Our current Menu doesn't do much.

Update Menu when Button clicked

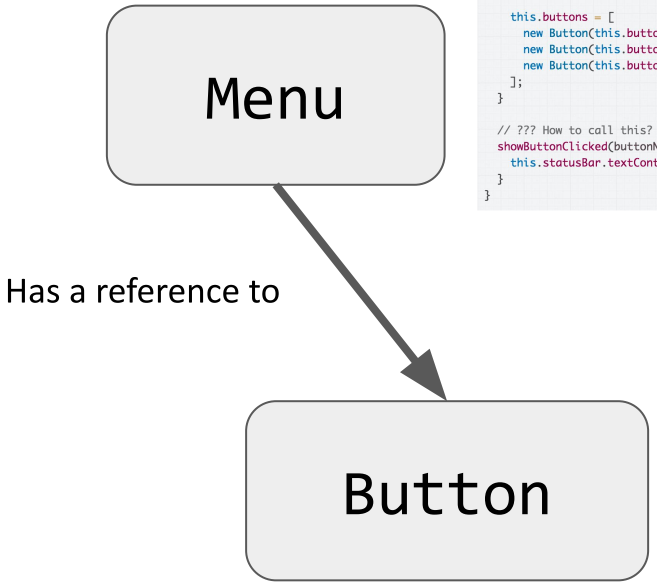
```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }

  // ??? How to call this?
  showButtonClicked(buttonName) {
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

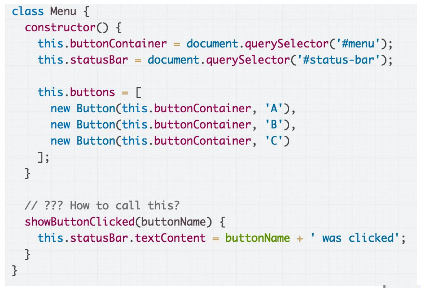


C was clicked

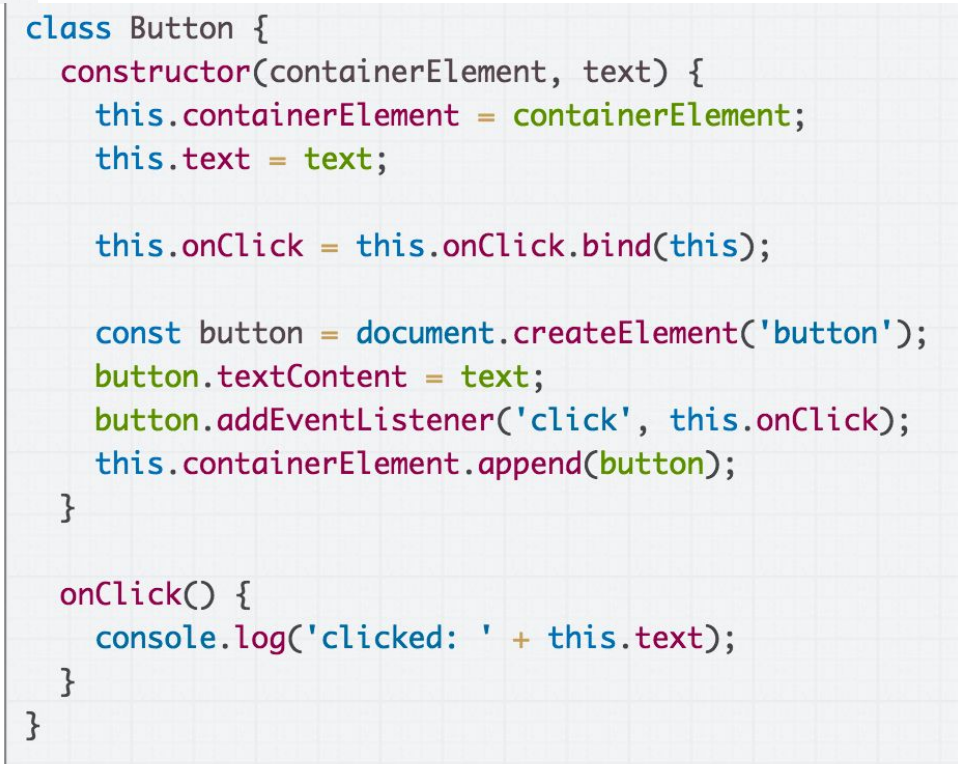
We want the Menu to update the <h1> when one of the Buttons are clicked. **How do we do this?**

Communicating upstream





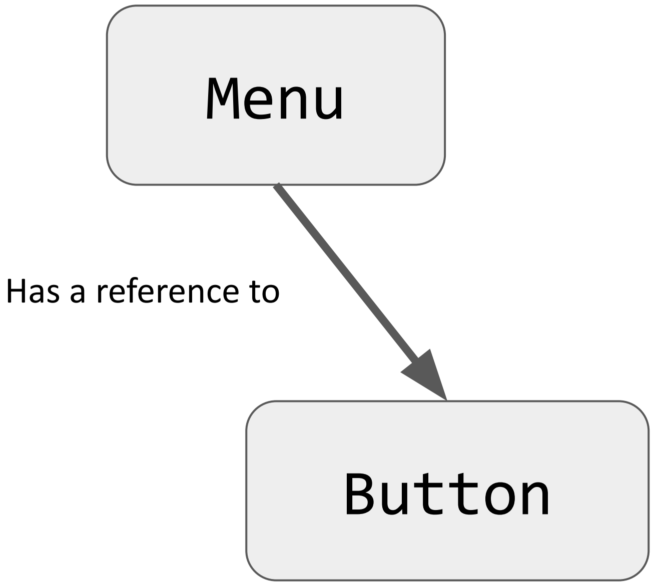
```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }

  // ??? How to call this?
  showButtonClicked(buttonName) {
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

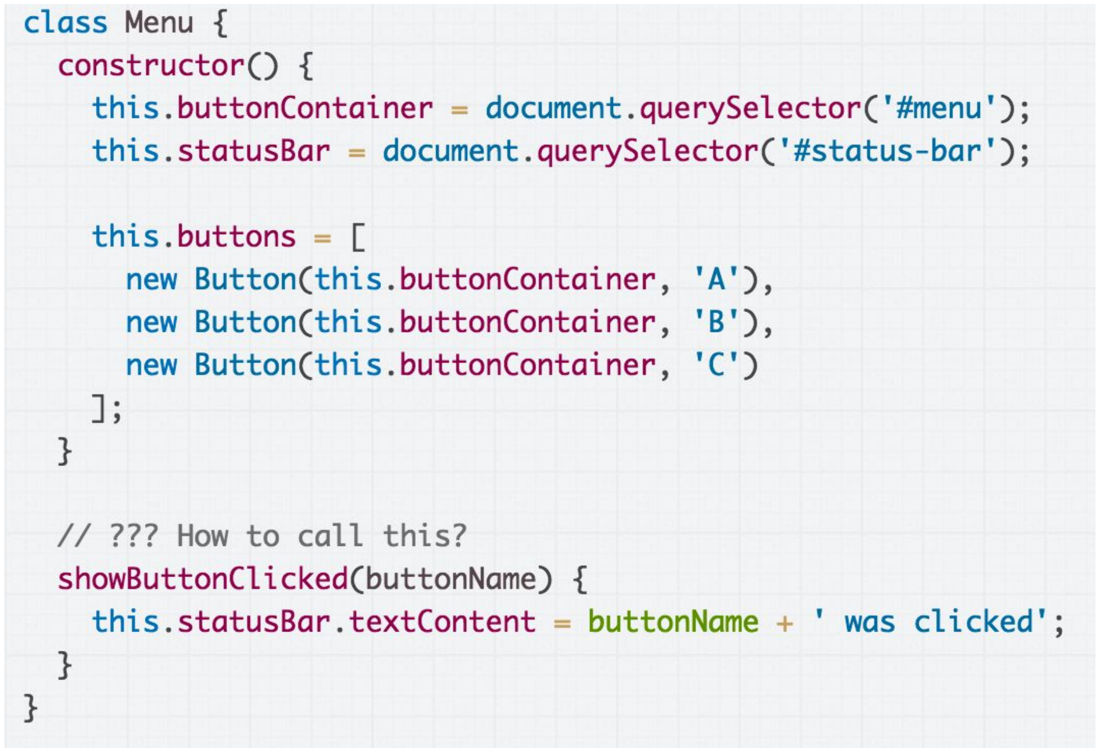


```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

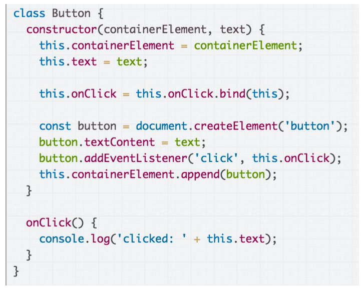
Button is the thing that knows it was clicked...

Communicating upstream





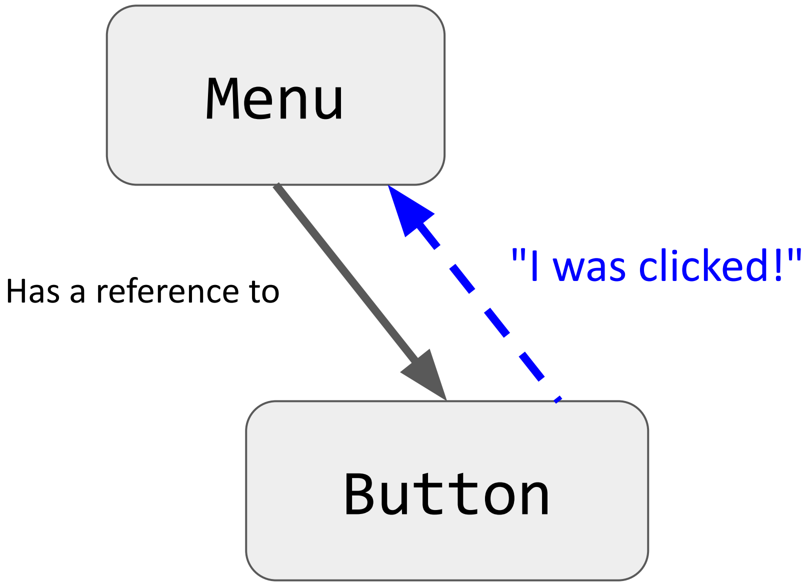
```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }

  // ??? How to call this?
  showButtonClicked(buttonName) {
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```



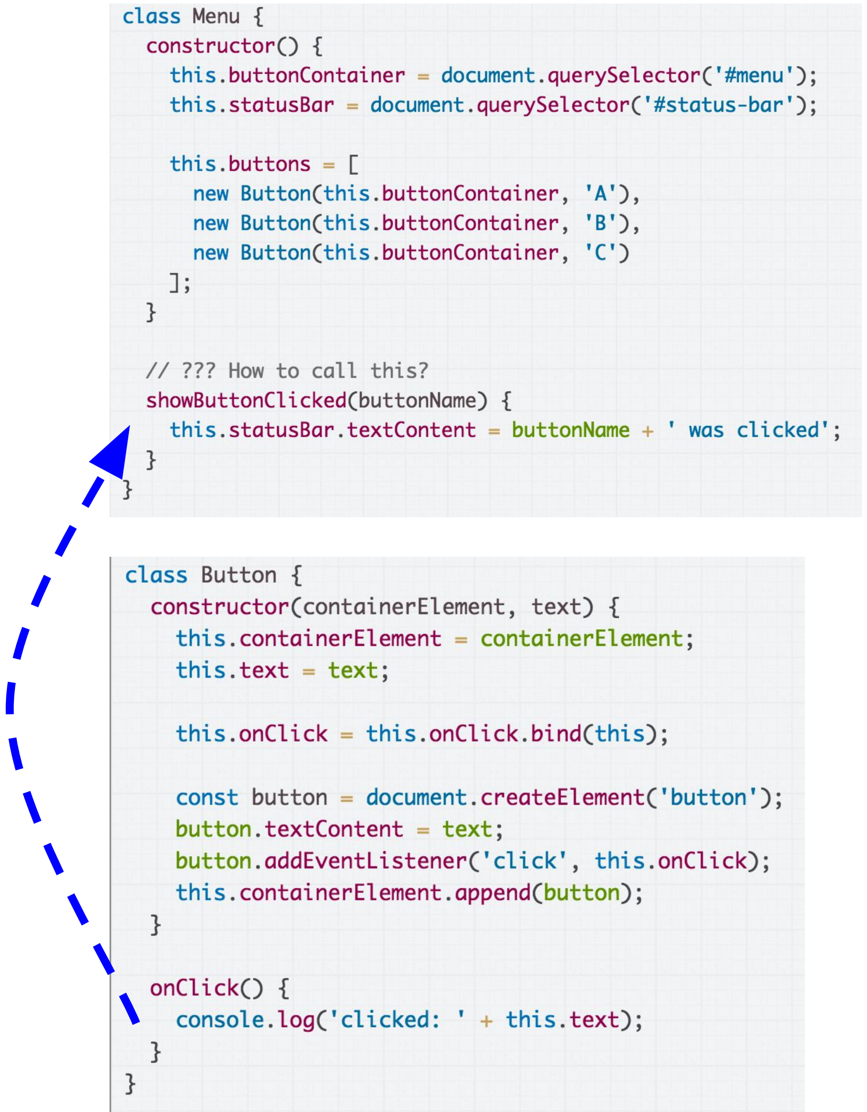
```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

But Menu is the thing that can update the header.

Communicating upstream



It needs to be possible for a Button to tell the Menu that it has been clicked.



```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }

  // ??? How to call this?
  showButtonClicked(buttonName) {
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

One strategy for doing this: Custom events

Custom Events

You can listen to and dispatch Custom Events to communicate between classes (mdn):

```
const event = new CustomEvent(
    eventNameString, optionalParameterObject);
element.addEventListener(eventNameString,
functionName);
element.dispatchEvent(eventNameString);
```

Custom Events on `document`

CustomEvent **can only be listened to / dispatched on HTML elements,** and not on arbitrary class instances.

Therefore we are going to be adding/dispatching events on the `document` object, so that events can be globally listened to/dispatched.

`document.addEventListener(`***eventNameString, functionName***`);`

`document.dispatchEvent(`***eventNameString***`);`

Define a custom event

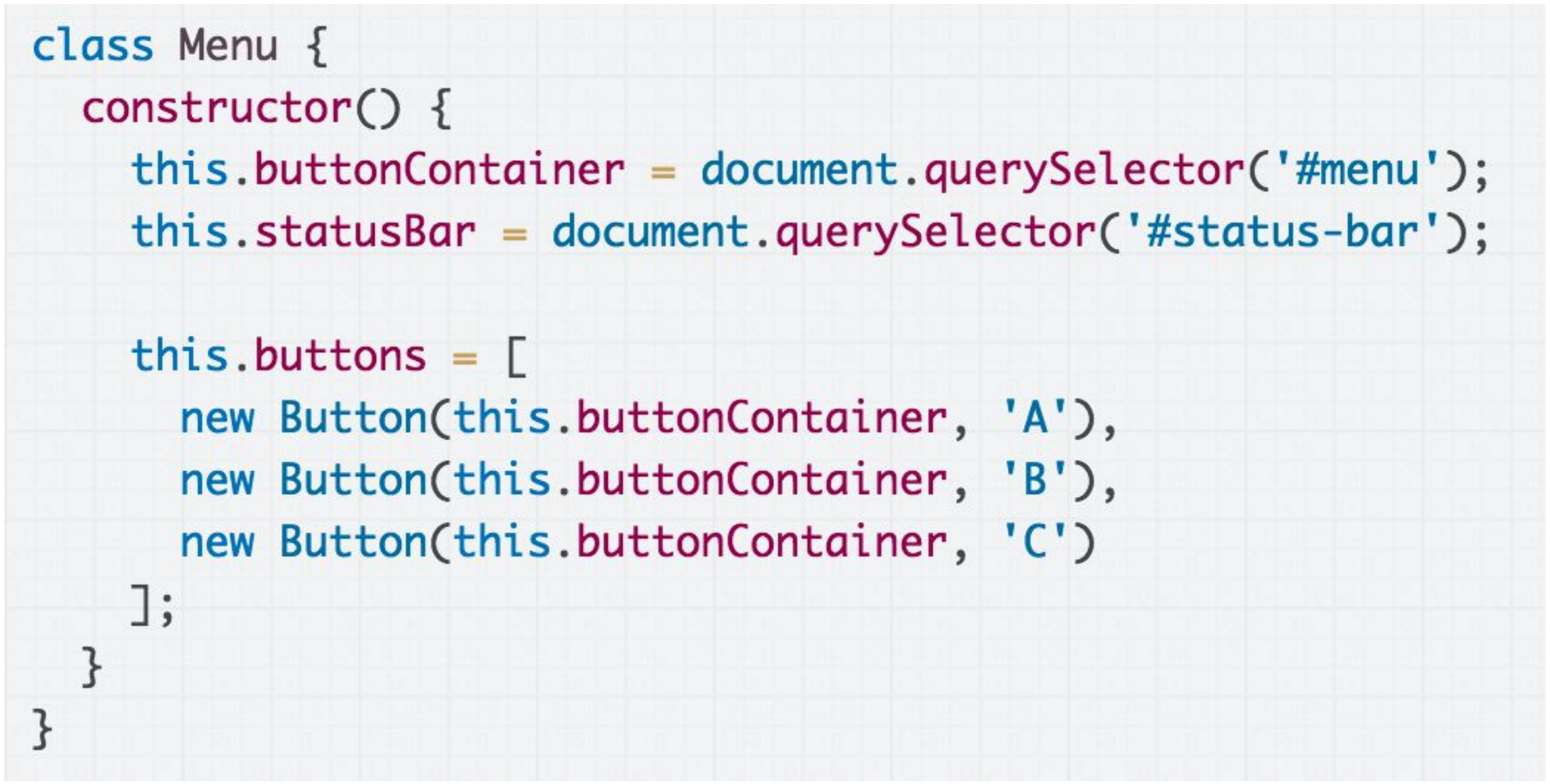
We'll define a custom event called `'button-click'`:

Menu will listen for the event:

```
document.addEventListener(
    'button-click', this.showButtonClicked);
```

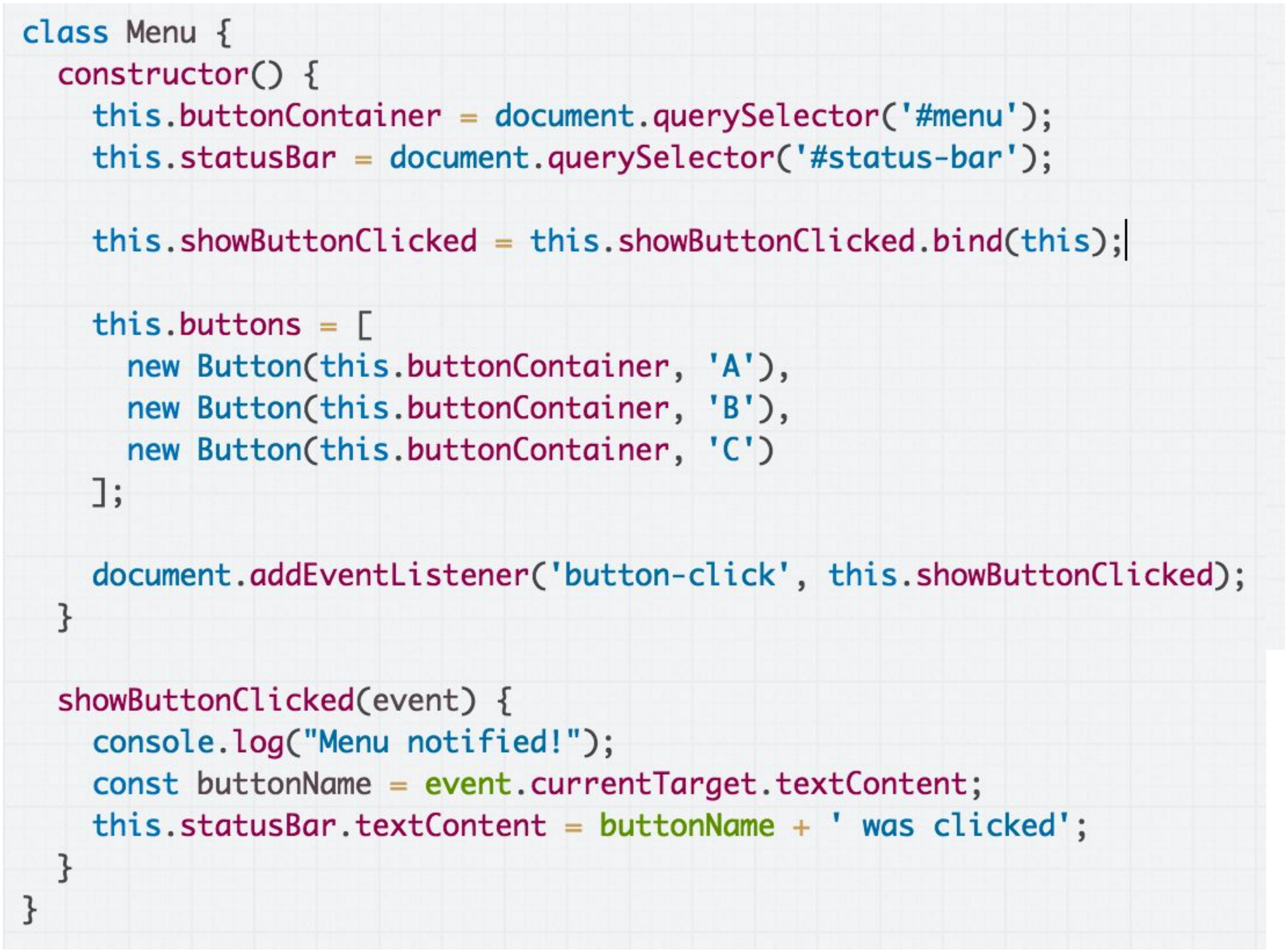
Button will dispatch the event:

```
document.dispatchEvent(
    new CustomEvent('button-click'));
```



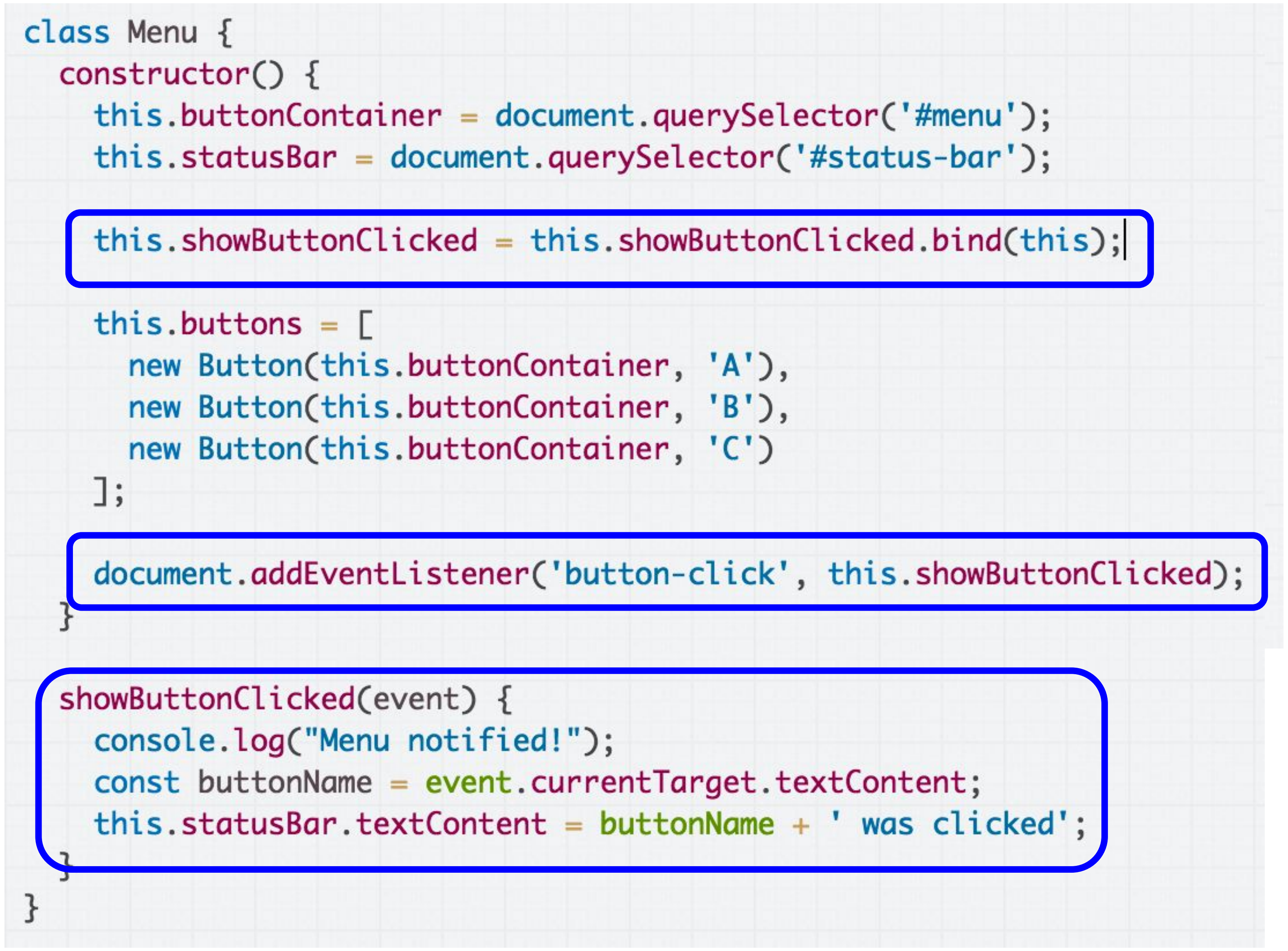
```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];
  }
}
```

A first attempt: We should listen for the custom `'button-click'` event in `Menu`.

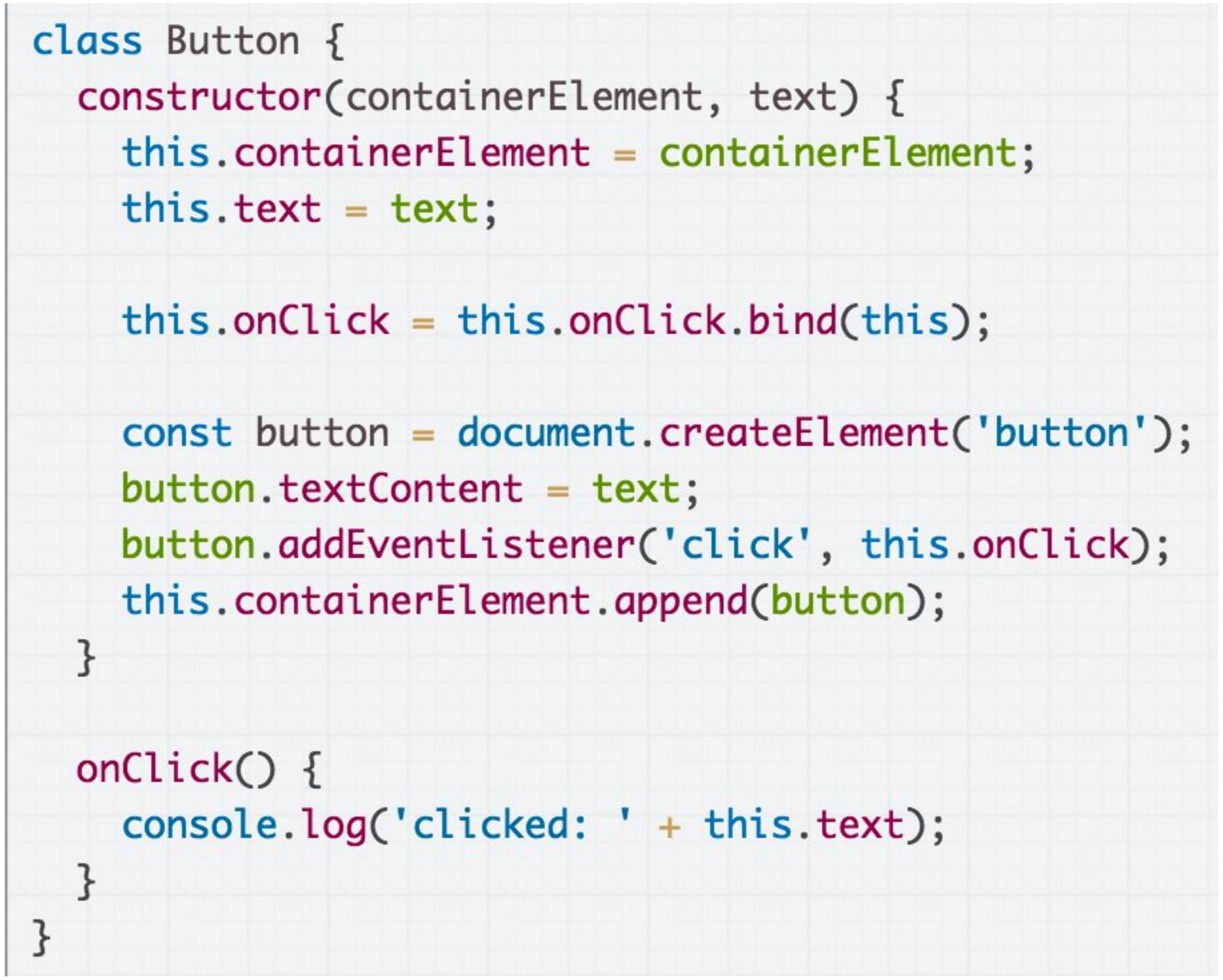


```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.showButtonClicked = this.showButtonClicked.bind(this);

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];

    document.addEventListener('button-click', this.showButtonClicked);
  }

  showButtonClicked(event) {
    console.log("Menu notified!");
    const buttonName = event.currentTarget.textContent;
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

A first attempt: Listen for the custom `'button-click'` event in `Menu`. **Note the call to bind!** (CodePen)

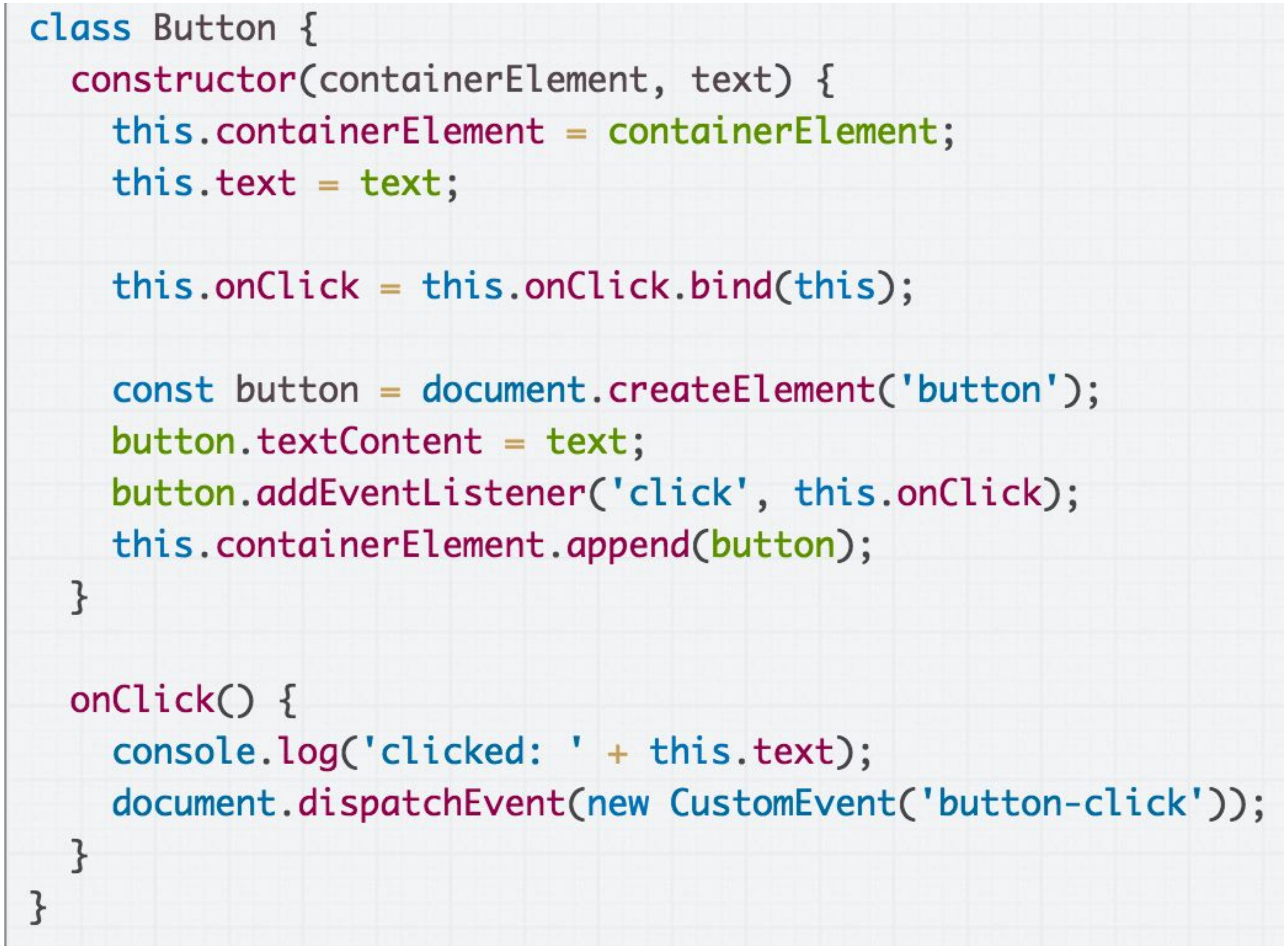

```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.showButtonClicked = this.showButtonClicked.bind(this);

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];

    document.addEventListener('button-click', this.showButtonClicked);
  }

  showButtonClicked(event) {
    console.log("Menu notified!");
    const buttonName = event.currentTarget.textContent;
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

A first attempt: Listen for the custom `'button-click'` event in `Menu`. **Note the call to bind!** (CodePen)



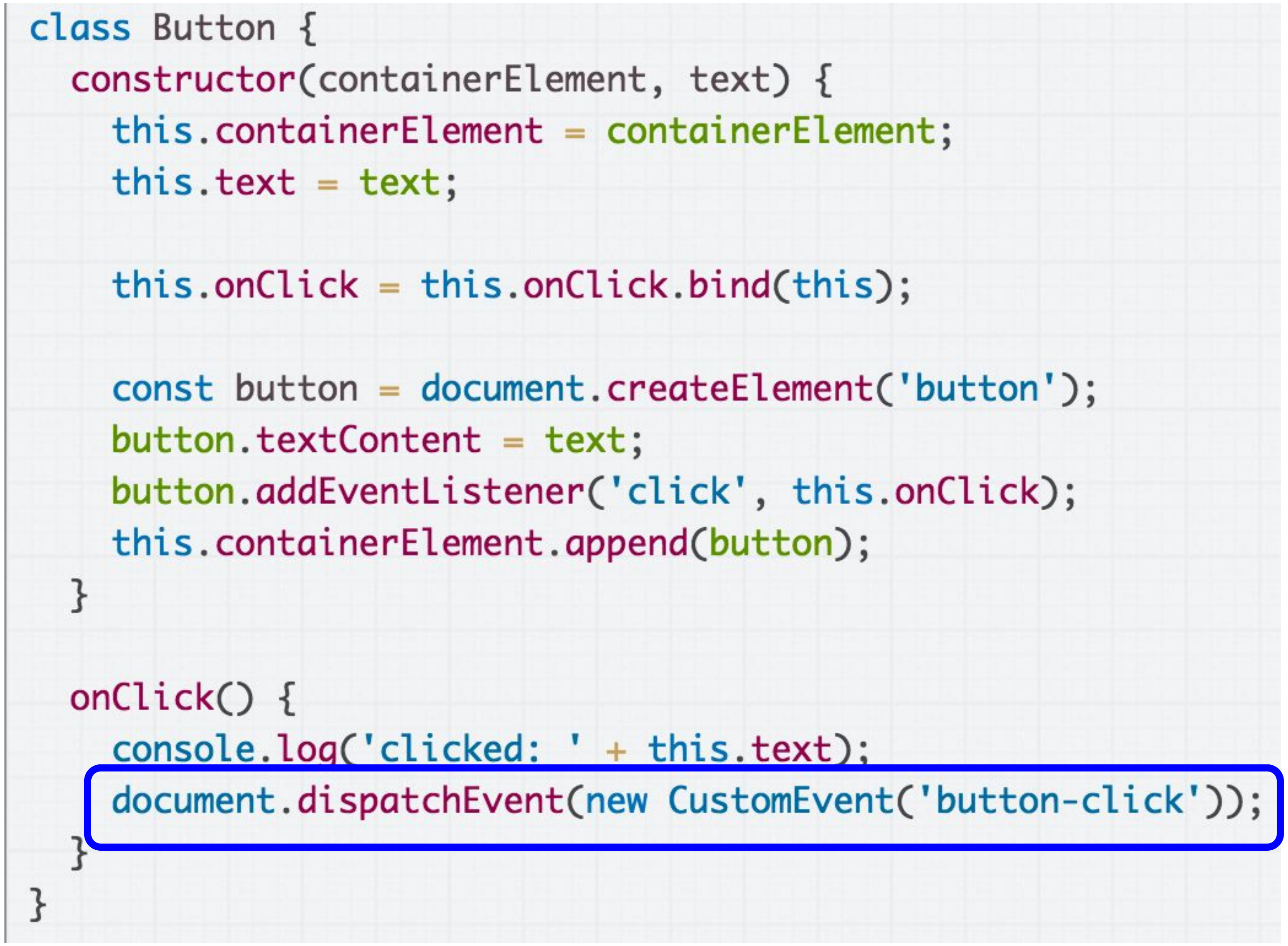
```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
  }
}
```

Then we want to dispatch the `'button-click'` event in the `onClick` event handler in `Button`.



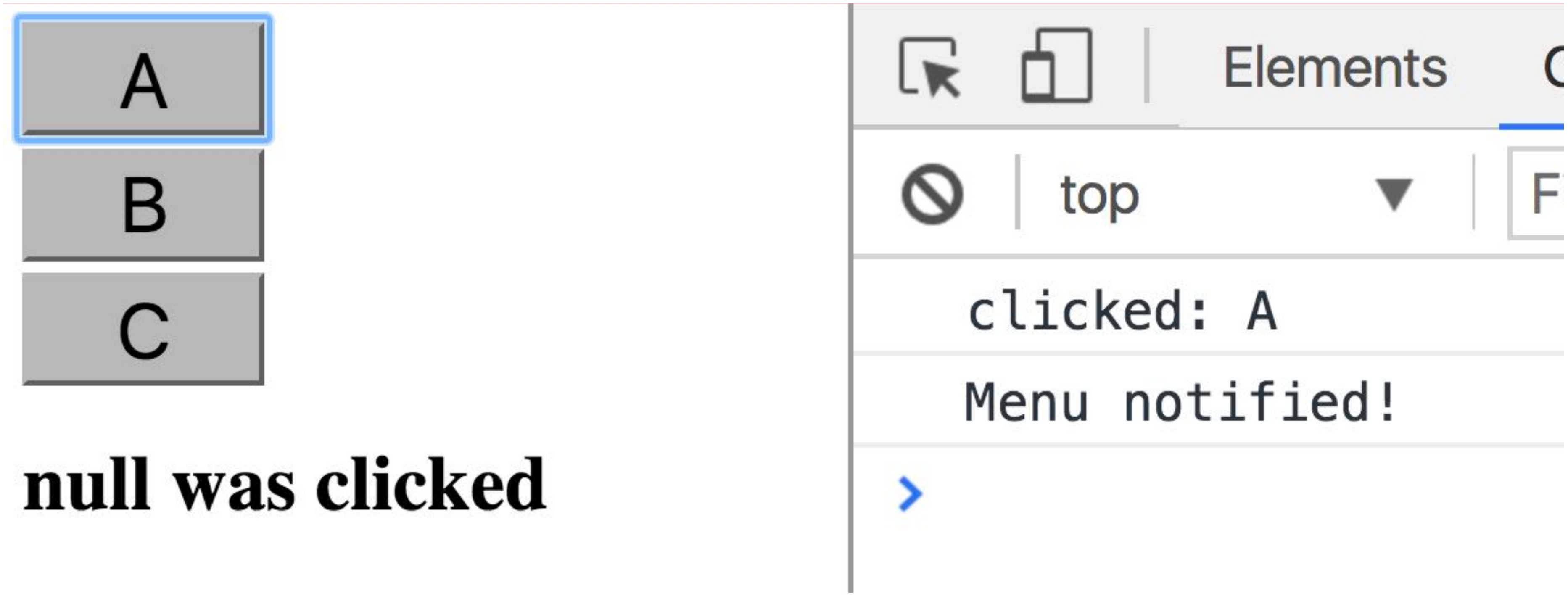
```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
    document.dispatchEvent(new CustomEvent('button-click'));
  }
}
```

Dispatch the `'button-click'` event in the `onClick` event handler in `Button` (CodePen).



```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
    document.dispatchEvent(new CustomEvent('button-click'));
  }
}
```

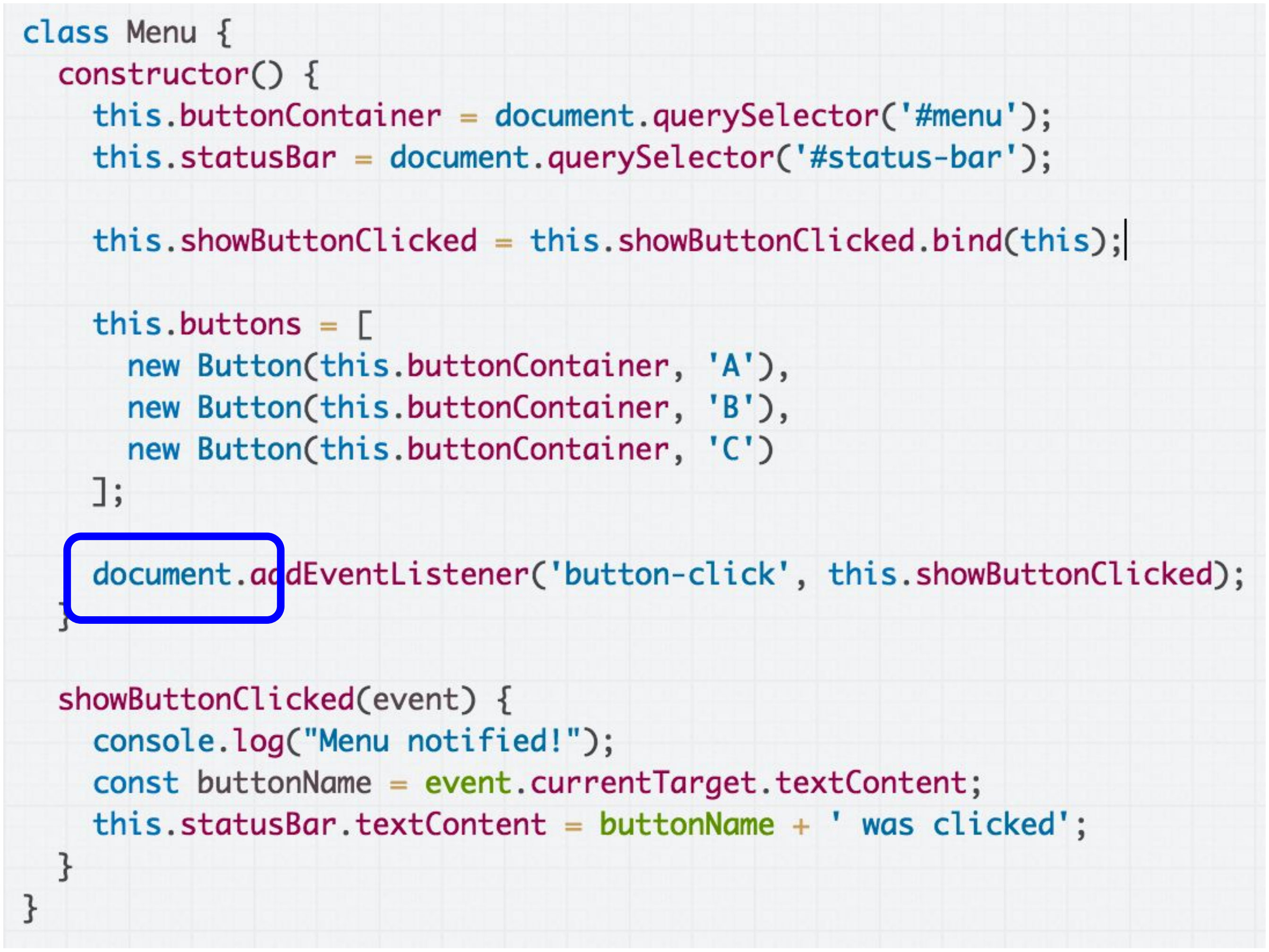
Dispatch the `'button-click'` event in the `onClick` event handler in `Button` (CodePen).



null was clicked

```
clicked: A
Menu notified!
```

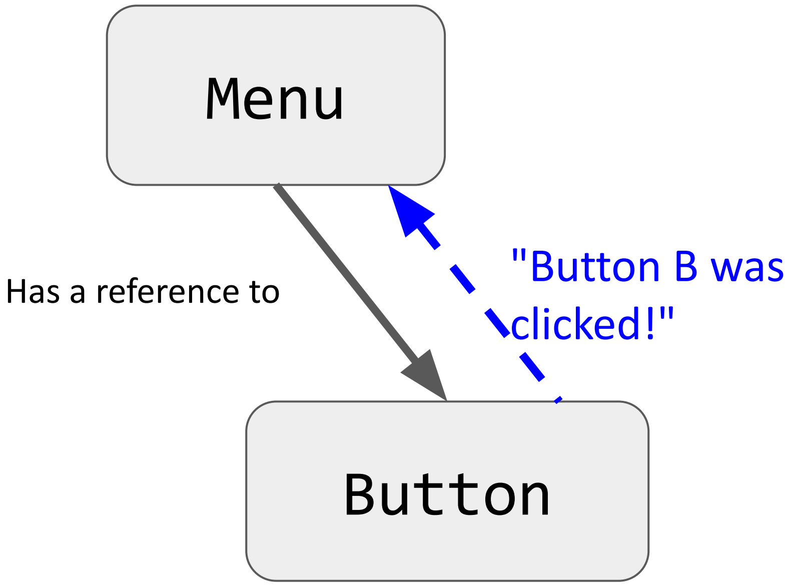
When we try it out, the event dispatching seems to work… but our output is "null was clicked"
(CodePen / Live)



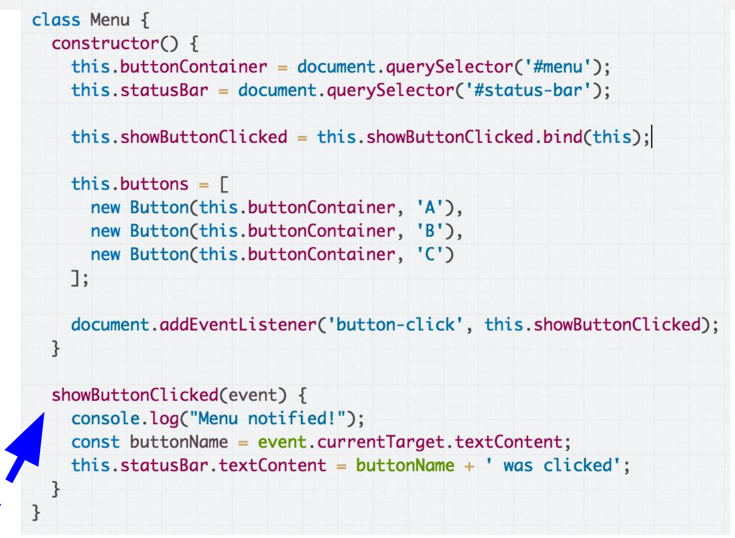
```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.showButtonClicked = this.showButtonClicked.bind(this);

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];

    document.addEventListener('button-click', this.showButtonClicked);
  }

  showButtonClicked(event) {
    console.log("Menu notified!");
    const buttonName = event.currentTarget.textContent;
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

The problem is we are adding custom event listeners to `document`, meaning `event.currentTarget` is going to be `document`, and not `<button>`

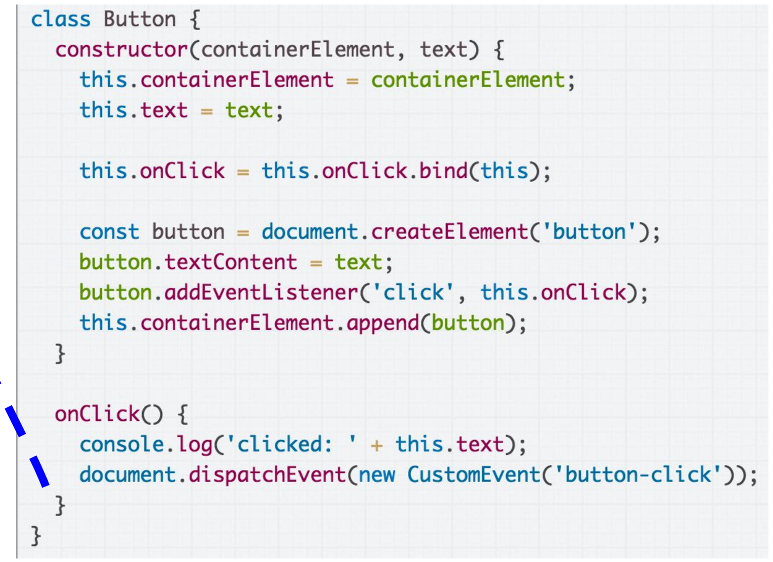
Communicating upstream



Menu knows some button was clicked… How do we tell the Menu which button was clicked?



```
class Menu {
  constructor() {
    this.buttonContainer = document.querySelector('#menu');
    this.statusBar = document.querySelector('#status-bar');

    this.showButtonClicked = this.showButtonClicked.bind(this);

    this.buttons = [
      new Button(this.buttonContainer, 'A'),
      new Button(this.buttonContainer, 'B'),
      new Button(this.buttonContainer, 'C')
    ];

    document.addEventListener('button-click', this.showButtonClicked);
  }

  showButtonClicked(event) {
    console.log("Menu notified!");
    const buttonName = event.currentTarget.textContent;
    this.statusBar.textContent = buttonName + ' was clicked';
  }
}
```

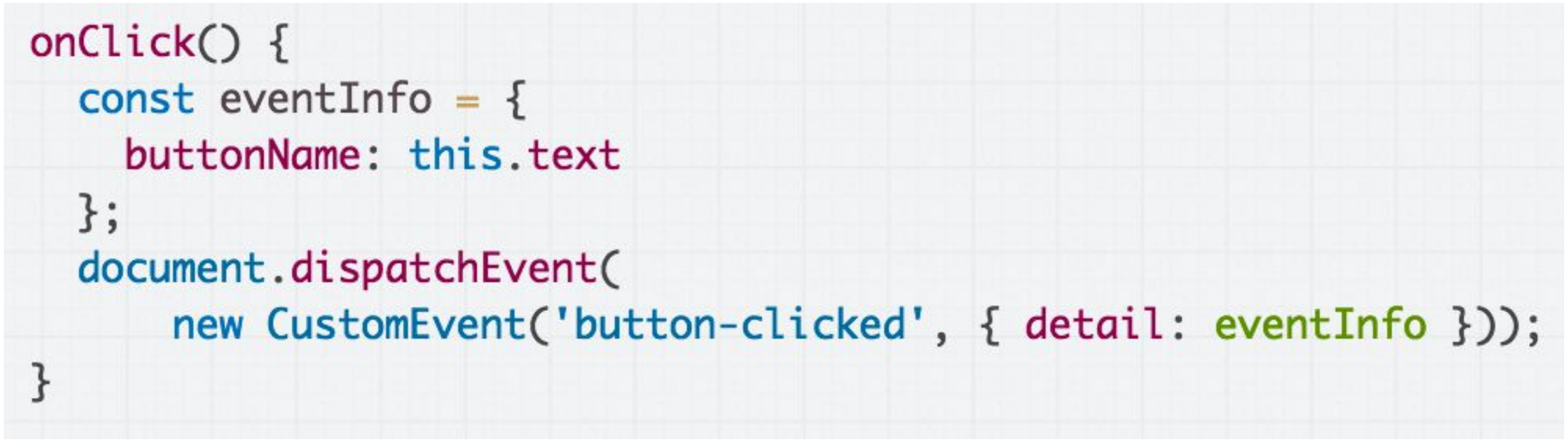


```
class Button {
  constructor(containerElement, text) {
    this.containerElement = containerElement;
    this.text = text;

    this.onClick = this.onClick.bind(this);

    const button = document.createElement('button');
    button.textContent = text;
    button.addEventListener('click', this.onClick);
    this.containerElement.append(button);
  }

  onClick() {
    console.log('clicked: ' + this.text);
    document.dispatchEvent(new CustomEvent('button-click'));
  }
}
```

CustomEvent parameters

You can add a parameter to your [CustomEvent]:

- Create an object with a `detail` property
- The value of this `detail` property can be whatever you'd like.

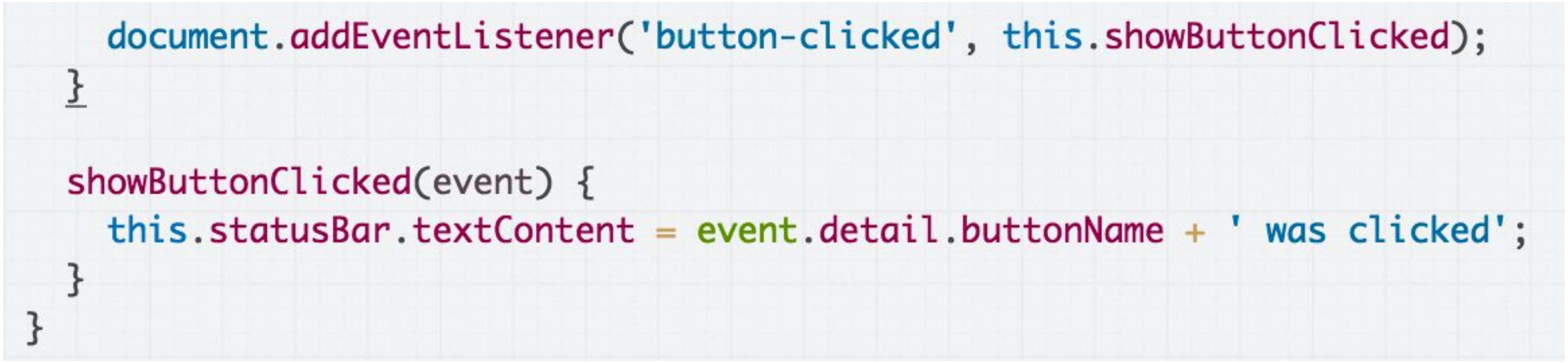


```
onClick() {
  const eventInfo = {
    buttonName: this.text
  };
  document.dispatchEvent(
      new CustomEvent('button-clicked', { detail: eventInfo }));
}
```

CustomEvent parameters

You can add a parameter to your CustomEvent:

- The event handler for your CustomEvent will be able to access this detail property via Event.detail



```
    document.addEventListener('button-clicked', this.showButtonClicked);
  }

  showButtonClicked(event) {
    this.statusBar.textContent = event.detail.buttonName + ' was clicked';
  }
}
```

Finished CodePen

First-class functions

Recall: `addEventListener`

Over the last few weeks, we've been using **functions** as a parameter to `addEventListener`:

```
image.addEventListener(
    'pointerdown', onDragStart);

image.addEventListener(
    'click', this._openPresent);
```

Q: How does this actually work?

First-class functions

Functions in JavaScript are objects.

- They can be saved in variables
- They can be passed as parameters
- They have properties, like other objects
- They can be defined without an identifier

(This is also called having **first-class functions**, i.e. functions in JavaScript are "first-class" because they are treated like any other variable/object.)

First-class functions

Functions in JavaScript are objects.

- They can be saved in variables
- They can be passed as parameters
- They have properties, like other objects
- They can be defined without an identifier

(This is also called having **first-class functions**, i.e. functions in JavaScript are "first-class" because they are treated like any other variable/object.)

???

First-class functions

Functions in JavaScript are objects.

- They can be saved in variables
- They can be passed as parameters
- They have properties, like other objects
- They can be defined without an identifier

(This is also called having **first-class functions**, i.e. functions in JavaScript are "first-class" because they are treated like any other variable/object.)

???

Isn't there like… a fundamental difference between "code" and "data"?

Let's take it all the way back to first principles…

Back to the veeeeery basics

What is code?

- A list of instructions your computer can execute
- Each line of code is a **statement**

What is a function?

- A labeled group of **statements**
- The statements in a function are executed when the function is invoked

What is a variable?

- A labeled piece of **data**

Recall: Objects in JS

Objects in JavaScript are sets of property-value pairs:

```
const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing']
};
```

- Like any other value, Objects can be saved in **variables.**
- Objects can be passed as parameters to functions

Back to the veeeeery basics

What is code?

- A list of instructions your computer can execute
- Each line of code is a **statement**

What is a function?

- A labeled group of **statements**
- The statements in a function are executed when the function is invoked

What is a variable?

- A labeled piece of **data**

What could it mean for a function to be an object, i.e. a kind of data?

Function variables

You can declare a function in several ways:

```
function myFunction(params) {
}

const myFunction = function(params) {
};

const myFunction = (params) => {
};
```

Function variables

```
function myFunction(params) {
}

const myFunction = function(params) {
};

const myFunction = (params) => {
};
```

Functions are invoked in the same way, regardless of how they were declared:

```
myFunction();
```

```
const x = 15;
let y = true;

const greeting = function() {
  console.log('hello, world');
}
```

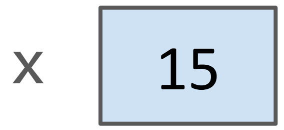
"A function in JavaScript is an object of type `Function`"

In the interpreter's memory:

```
const x = 15;
let y = true;

const greeting = function() {
  console.log('hello, world');
}
```

"A function in JavaScript is an object of type `Function`"

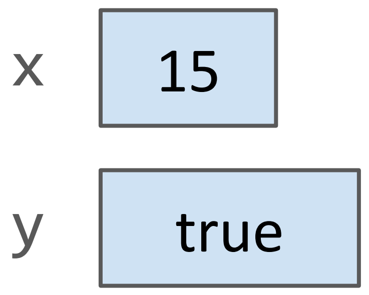
In the interpreter's memory:



```
const x = 15;
let y = true;

const greeting = function() {
  console.log('hello, world');
}
```

"A function in JavaScript is an object of type `Function`"

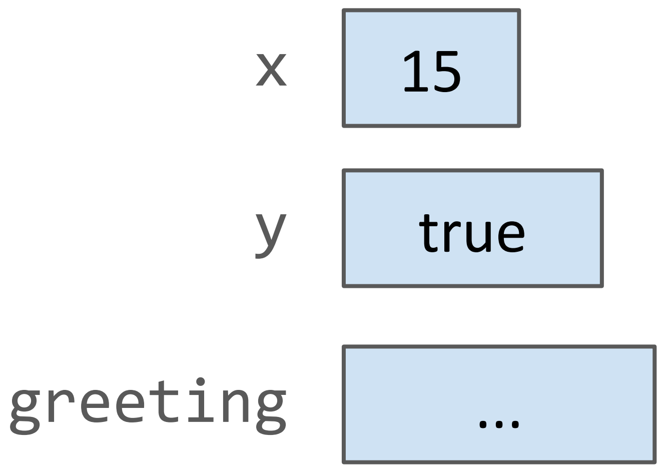
In the interpreter's memory:



```
const x = 15;
let y = true;

const greeting = function() {
  console.log('hello, world');
}
```

"A function in JavaScript is an object of type `Function`"

In the interpreter's memory:



```
const x = 15;
let y = true;

const greeting = function() {
  console.log('hello, world');
}
```

"A function in JavaScript is an object of type `Function`"

What this really means:

- When you declare a function, there is an object of type `Function` that gets created alongside the labeled block of executable code.

Function properties

```
const greeting = function() {
  console.log('hello, world');
}

console.log(greeting.name);
console.log(greeting.toString());
```

When you declare a function, you create an object of type `Function`, which has properties like:

- `name`
- `toString`

CodePen

Function properties

```
const greeting = function() {
  console.log('hello, world');
}

greeting.call();
```

`Function` objects also have a `call` method, which invokes the underlying executable code associated with this function object.

CodePen

Function properties

```
const greeting = function() {
  console.log('hello, world');
}

greeting.call();
greeting();
```

`()` is an operation on the `Function` object (spec)

- When you use the `()` operator on a Function object, it is calling the object's `call()` method, which in turn executes the function's underlying code

Code vs `Functions`

Important distinction:

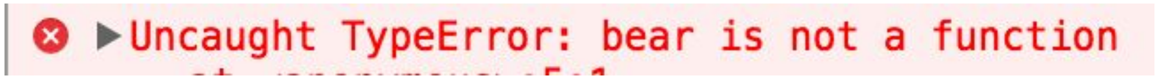
- Function, the executable code
  - A group of instructions to the computer
- `Function`, the object
  - A JavaScript object, i.e. a set of property-value pairs
  - `Function` objects have executable code associated with them
  - This executable code can be invoked by
    - *functionName*`();` *or*
    - *functionName*`.call();`

Note: Function is special

Only Function objects have executable code associated with them.

- Regular JS objects **cannot** be invoked
- Regular JS objects **cannot** be given executable code
    - I.e. you can't make a regular JS object into a callable function

```
const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing']
};
bear(); // error!
```



```
▶Uncaught TypeError: bear is not a function
```

Function Objects vs Objects

```
function sayHello() {
  console.log('Ice Bear says hello');
}

const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: sayHello
};
bear.greeting();
```

CodePen

But you can give your object `Function` properties and then invoke those properties.

Function Objects vs Objects

```
function sayHello() {
  console.log('Ice Bear says hello');
}

const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: sayHello
};
bear.greeting();
```

CodePen

The `greeting` property is an object of `Function` type.

Why do we have `Function` objects?!

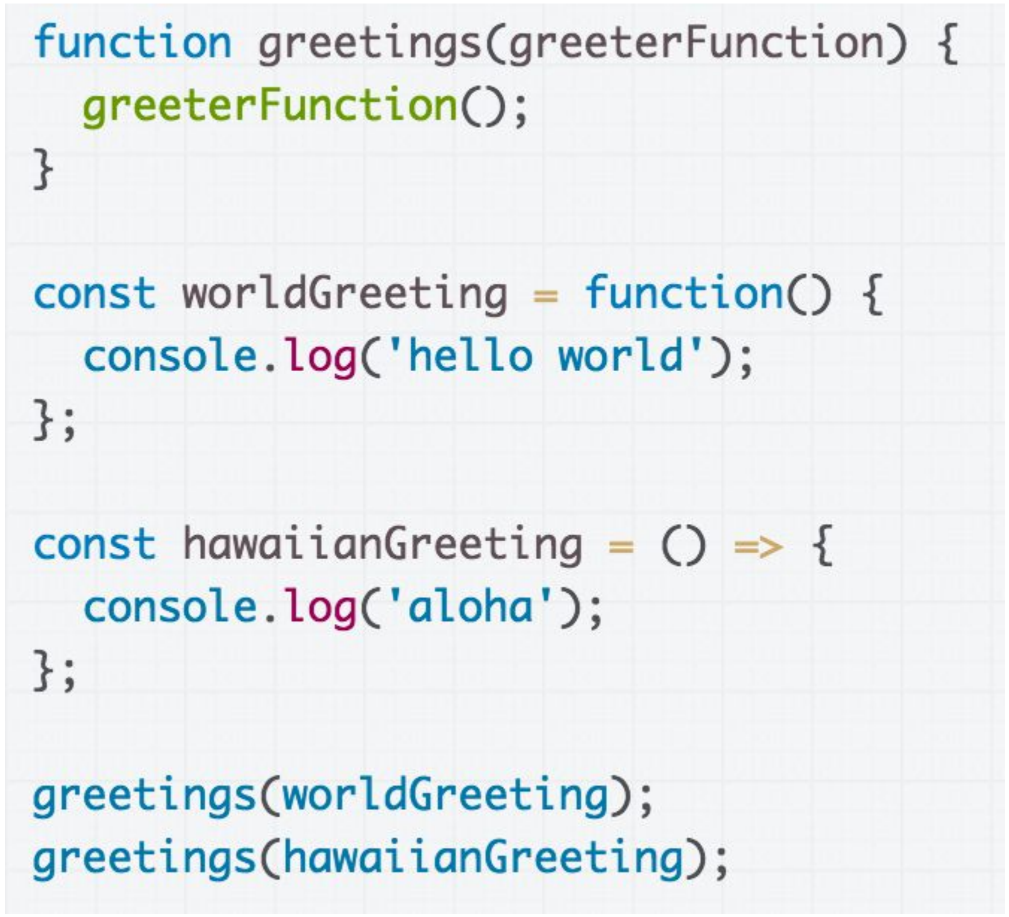
Callbacks

`Function` objects **really** come in handy for event-driven programming!

```
function onDragStart(event) {
    ...
}
dragon.addEventListener('pointerdown', onDragStart);
```

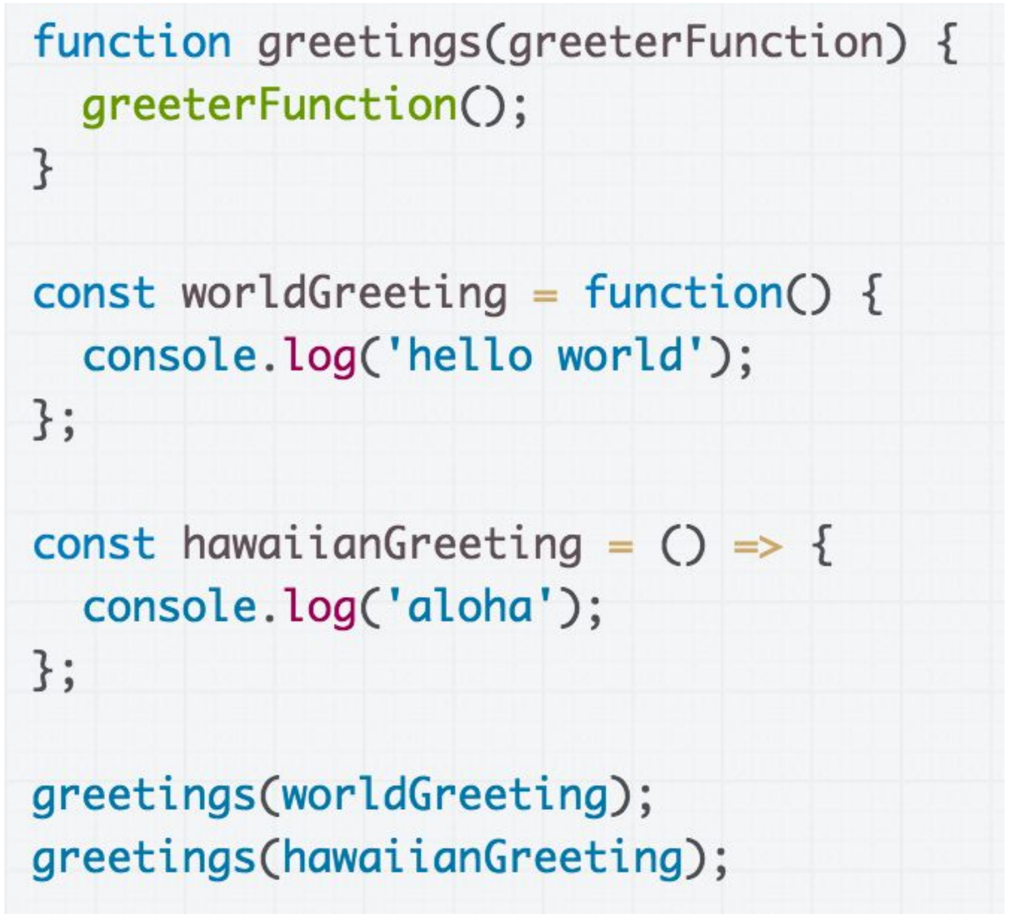
Because every function declaration creates a `Function` object, we can pass `Functions` as parameters to other functions.

Simple, contrived example



```
function greetings(greeterFunction) {
  greeterFunction();
}

const worldGreeting = function() {
  console.log('hello world');
};

const hawaiianGreeting = () => {
  console.log('aloha');
};

greetings(worldGreeting);
greetings(hawaiianGreeting);
```

CodePen



```
function greetings(greeterFunction) {
  greeterFunction();
}

const worldGreeting = function() {
  console.log('hello world');
};

const hawaiianGreeting = () => {
  console.log('aloha');
};

greetings(worldGreeting);
greetings(hawaiianGreeting);
```

CodePen

This example is really contrived!

Aside from `addEventListener`, when would you ever want to pass a `Function` as a parameter?

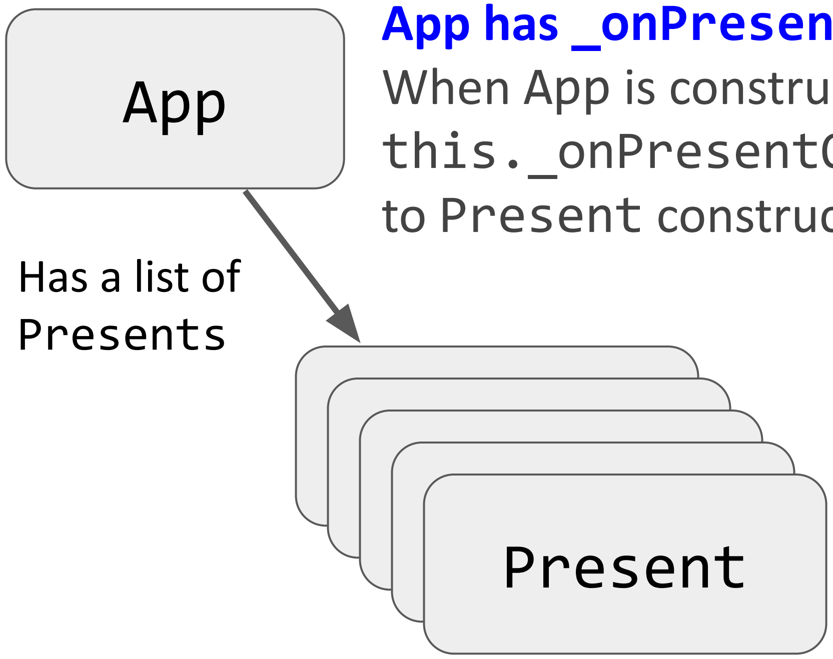
A real example: Callbacks

Another way we can communicate between classes is through callback functions:

- **Callback**: A function that's passed as a parameter to another function, usually in response to something.

Callback: Present example

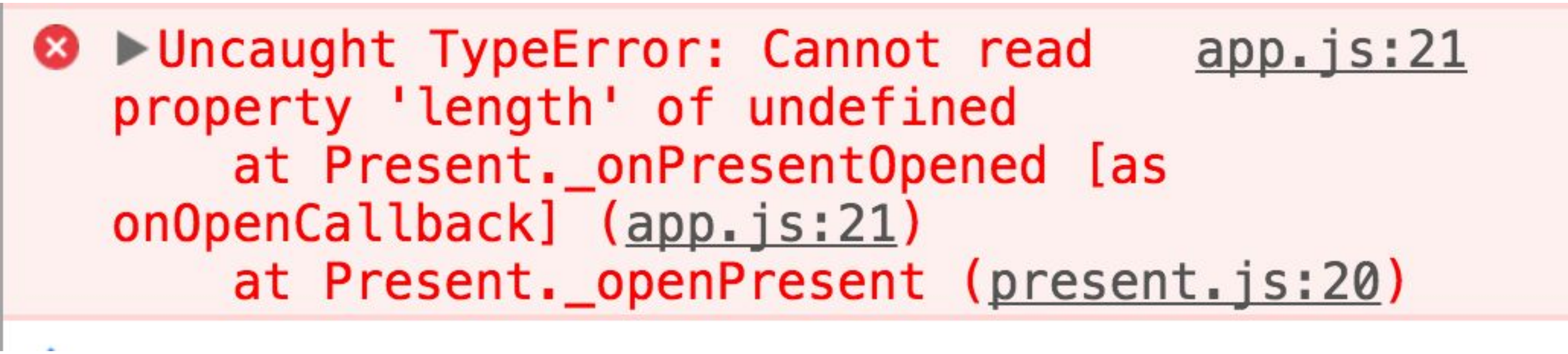
Let's have `Presents` communicate with App via callback `parameter`: (CodePen attempt)



App has _onPresentOpened method
When App is constructing `Presents`, pass its `this._onPresentOpened` method as parameter to `Present` constructor

Each `Present` saves an `onOpenCallback` parameter in the constructor
When the present is opened, fire the callback

this in event handler


```
▶Uncaught TypeError: Cannot read    app.js:21
property 'length' of undefined
    at Present._onPresentOpened [as
onOpenCallback] (app.js:21)
    at Present._openPresent (present.js:20)
```

Say, it's another error in our event handler...

this in a method

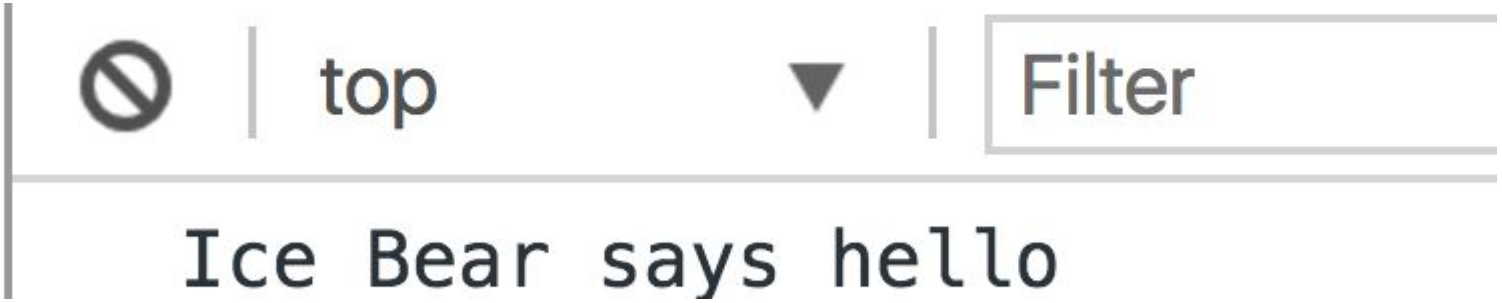
```
function sayHello() {
  console.log(this.name + ' says hello');
}

const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: sayHello
};
bear.greeting();
```

CodePen

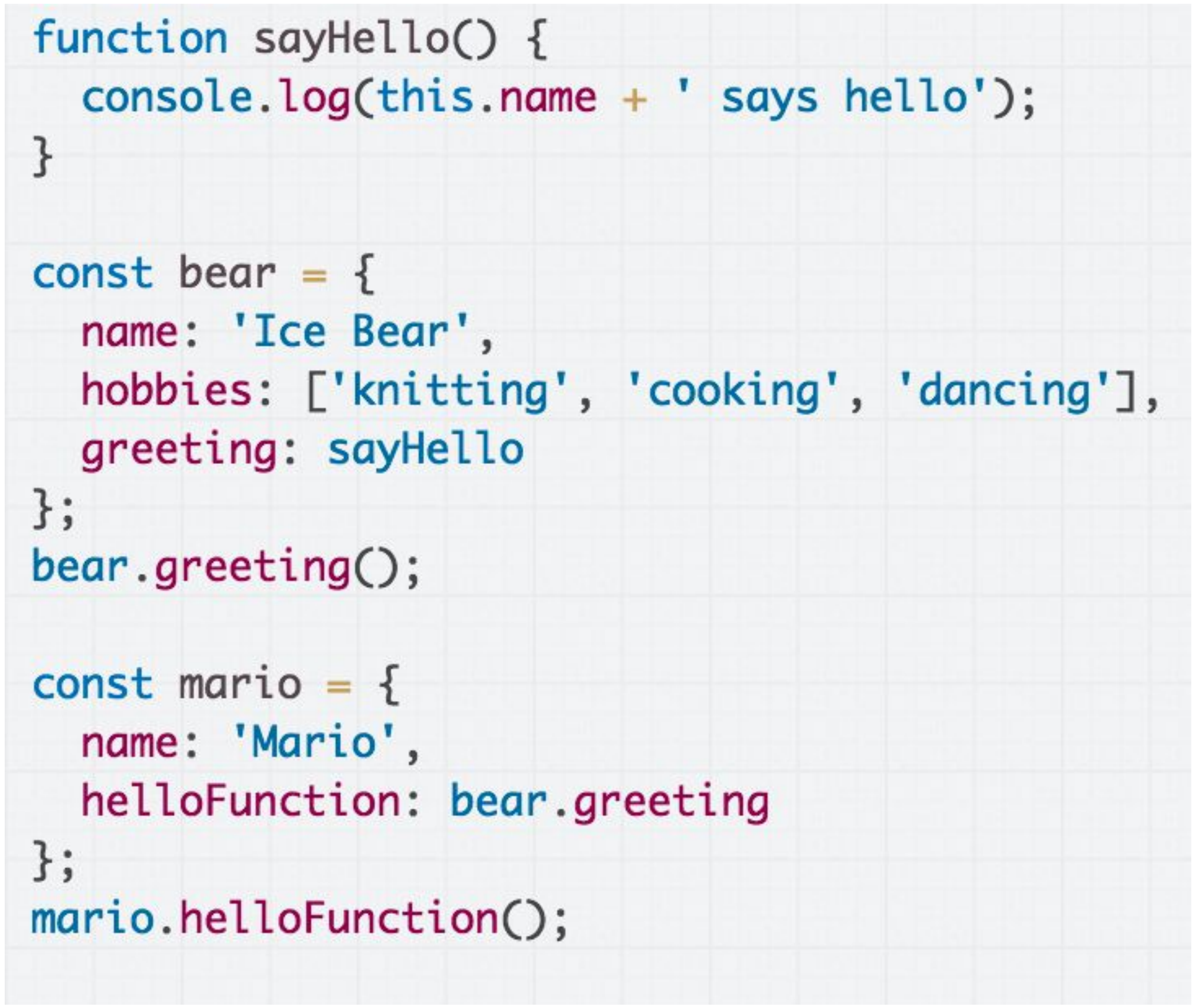
When we use this in a function that is not being invoked by an event handler, **this is set to the object on which the method is called.**

this in a method

```
function sayHello() {
  console.log(this.name + ' says hello');
}

const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: sayHello
};
bear.greeting();
```



```
Ice Bear says hello
```



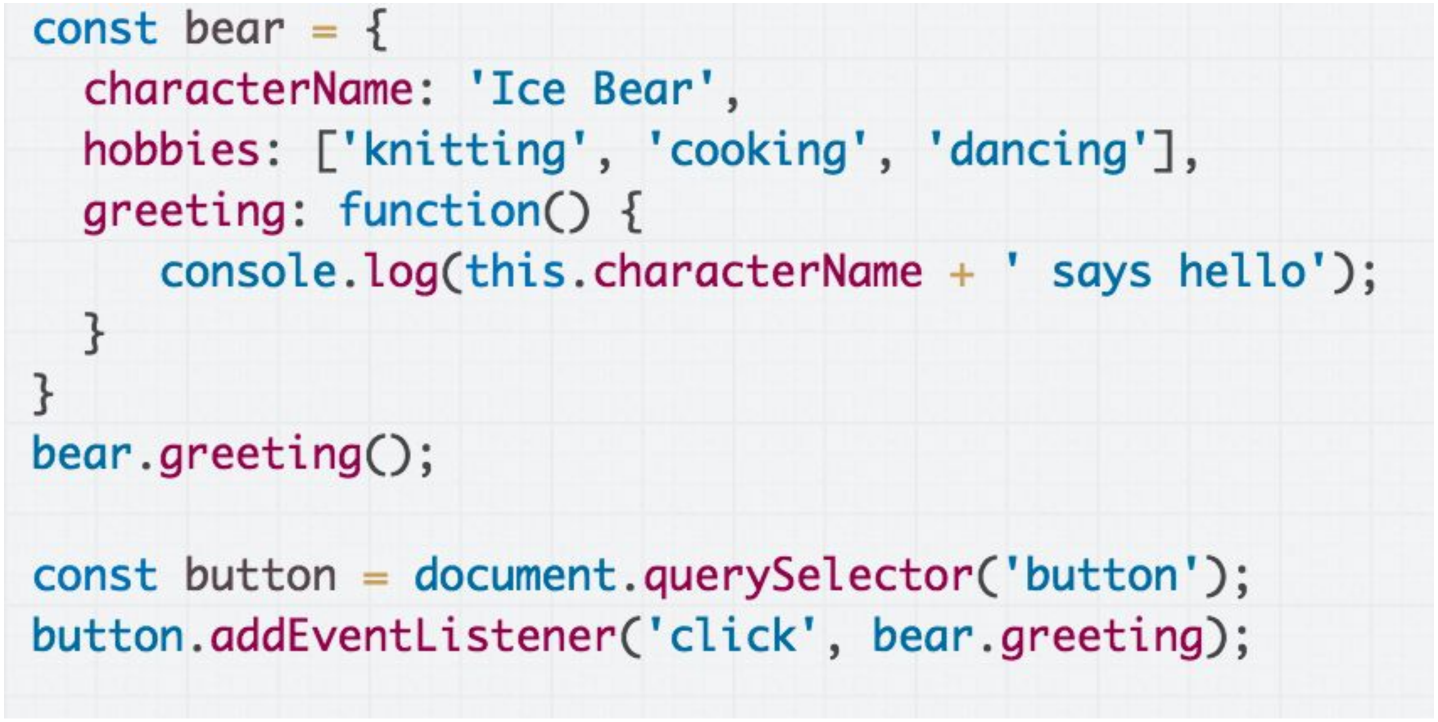
```
function sayHello() {
  console.log(this.name + ' says hello');
}

const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: sayHello
};
bear.greeting();

const mario = {
  name: 'Mario',
  helloFunction: bear.greeting
};
mario.helloFunction();
```

What is the output of the code above?

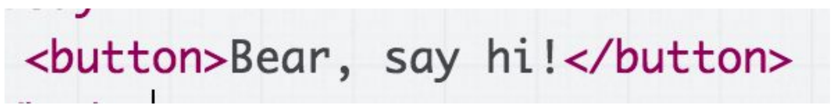
(CodePen)

```
function sayHello() {
  console.log(this.name + ' says hello');
}

const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: sayHello
};
bear.greeting();

const mario = {
  name: 'Mario',
  helloFunction: bear.greeting
};
mario.helloFunction();
```

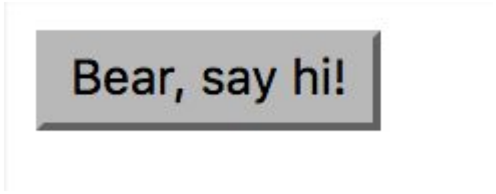
```
Ice Bear says hello
Mario says hello
```



```javascript
const bear = {
  characterName: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: function() {
      console.log(this.characterName + ' says hello');
  }
}
bear.greeting();

const button = document.querySelector('button');
button.addEventListener('click', bear.greeting);
```

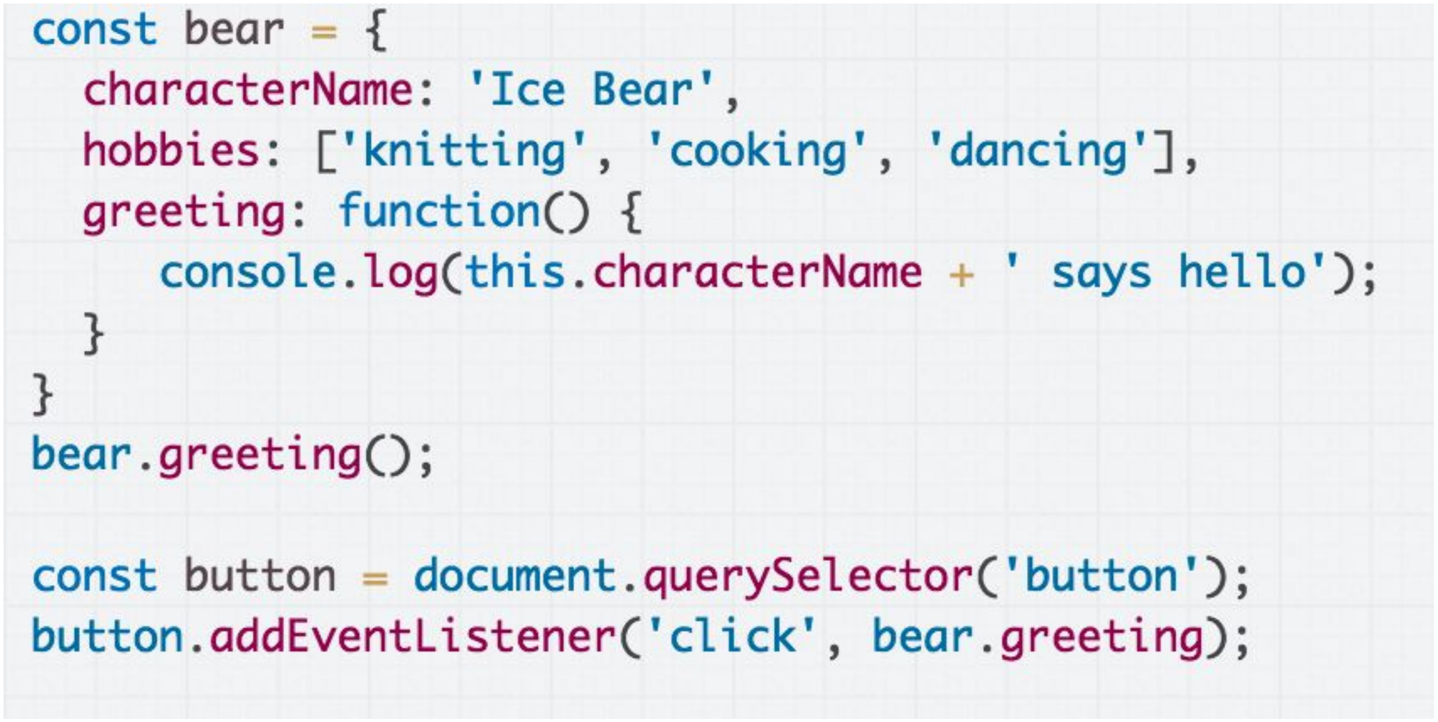


```html
<button>Bear, say hi!</button>
```

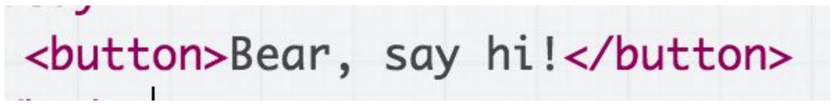


What is the output of the code above, if we click the button?

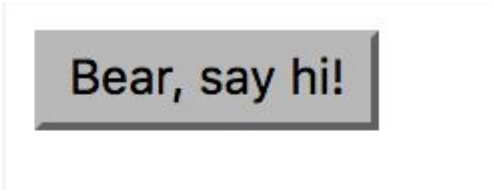
(CodePen)

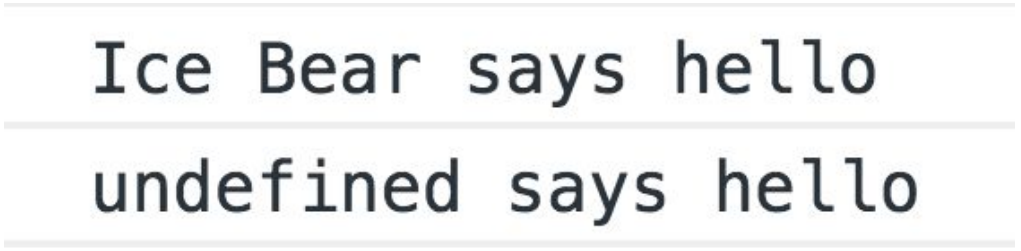


```javascript
const bear = {
  characterName: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: function() {
     console.log(this.characterName + ' says hello');
  }
}
bear.greeting();

const button = document.querySelector('button');
button.addEventListener('click', bear.greeting);
```

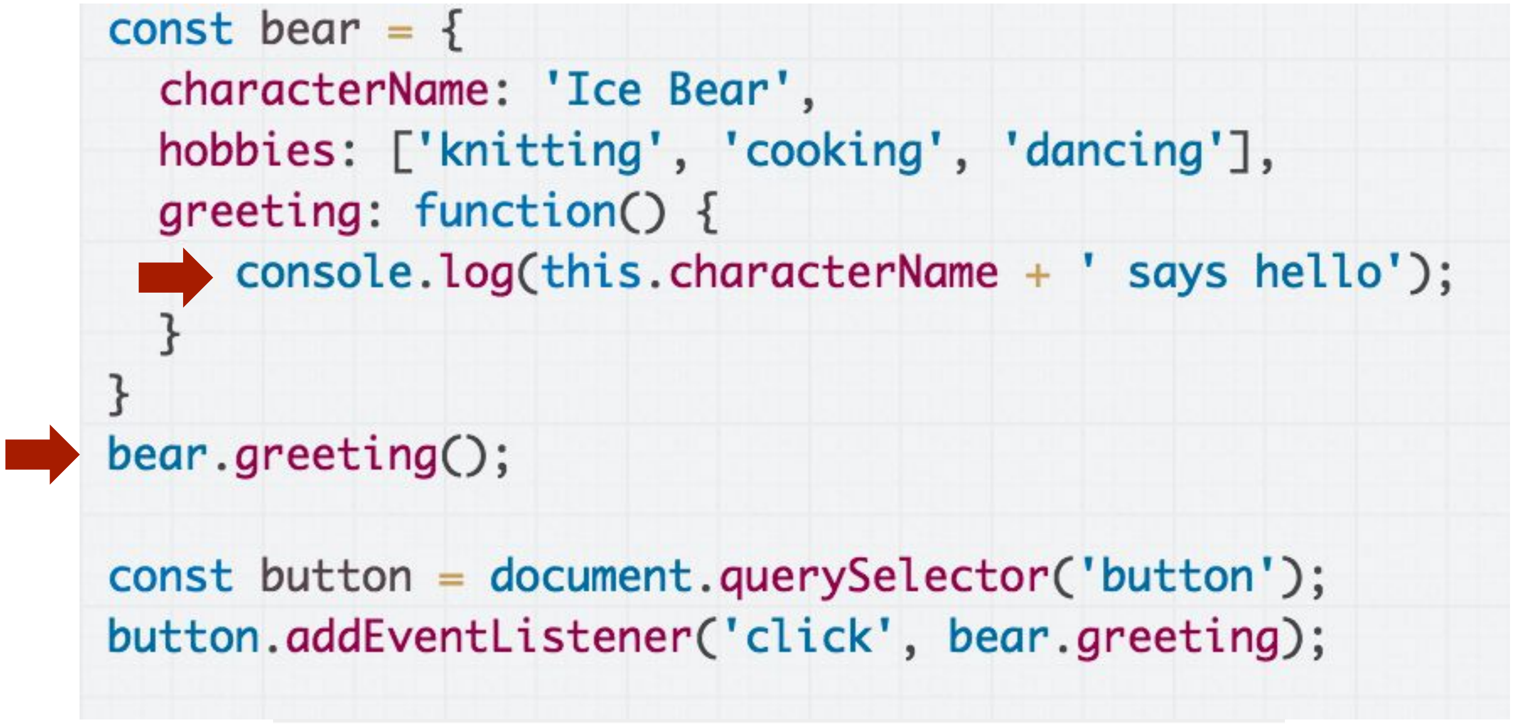


```html
<button>Bear, say hi!</button>
```





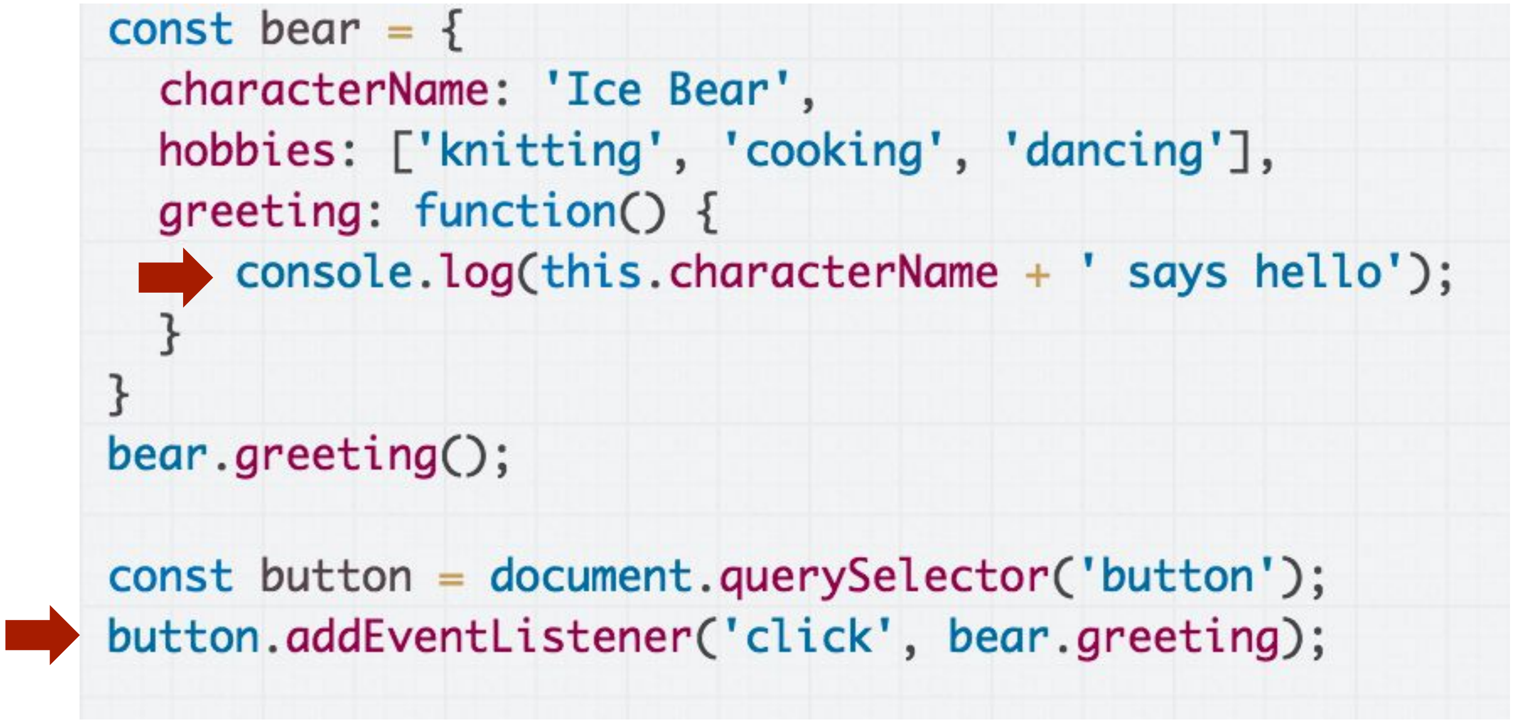
```
Ice Bear says hello
undefined says hello
```



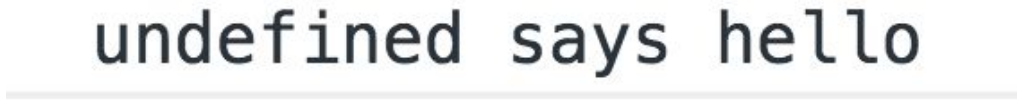
```
const bear = {
  characterName: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: function() {
    console.log(this.characterName + ' says hello');
  }
}
bear.greeting();

const button = document.querySelector('button');
button.addEventListener('click', bear.greeting);
```

```
Ice Bear says hello
```

When called as a method, the value of `this` is the object on which the method was called.



```
const bear = {
  characterName: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing'],
  greeting: function() {
    console.log(this.characterName + ' says hello');
  }
}
bear.greeting();

const button = document.querySelector('button');
button.addEventListener('click', bear.greeting);
```



```
undefined says hello
```

But when called from an event handler, `this` is the DOM object to which the event was attached.

Since `<button>` doesn't have a `characterName` property, we see "`undefined says hello`"

`bind`, revisited

- `this` is a **parameter** to passed to every function in JavaScript.
- JavaScript assigns this to be a different value depending on how it is used.
  - When called as a **method**, `this` is the object on which the method was called
  - When called from an **event handler**, `this` is the DOM element on which the event handler was attached

`bind`, revisited

someFunction*.**`bind(`valueOfThis***`);`

The `bind()` method:

- Returns a new function that is a copy of ***someFunction***
- But in this new function, `this` is always set to ***valueOfThis***, no matter how the function is invoked

`bind` in classes

```
constructor() {
  const someValue = this;
  this.methodName = this.methodName.bind(someValue);
}
```

This is saying:

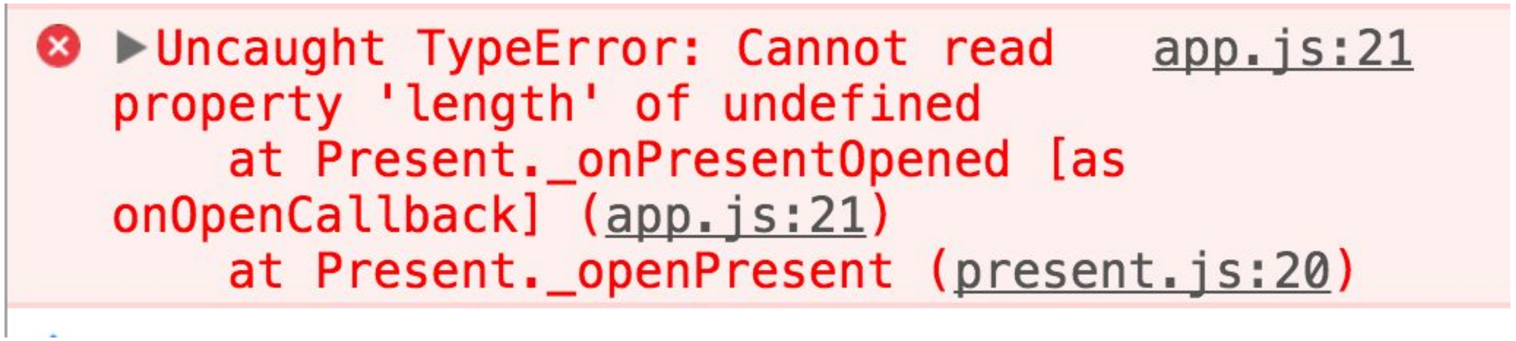
- Make a copy of ***methodName***, which will be the exact same as ***methodName*** except `this` in ***methodName*** is always set to the `someValue`
- The value of `someValue` is `this` to `bind()`, which is the value of the new object since we are in the constructor

`bind` in classes

```
constructor() {
  this.methodName = this.methodName.bind(this);
}
```

And of course, you don't need the intermediate `someValue` variable.

Callback: Present example



```
▶Uncaught TypeError: Cannot read    app.js:21
property 'length' of undefined
     at Present._onPresentOpened [as
onOpenCallback] (app.js:21)
     at Present._openPresent (present.js:20)
```

We can fix this error message by binding the method:

[CodePen solution]