Today's schedule

Today:

- How to interact with page?
- More events
  - Keyboard events
  - Mobile events (intro only)

Break!

- Classes and objects in JavaScript
- `this` keyword and `bind`

Note

Assignment 1 released:

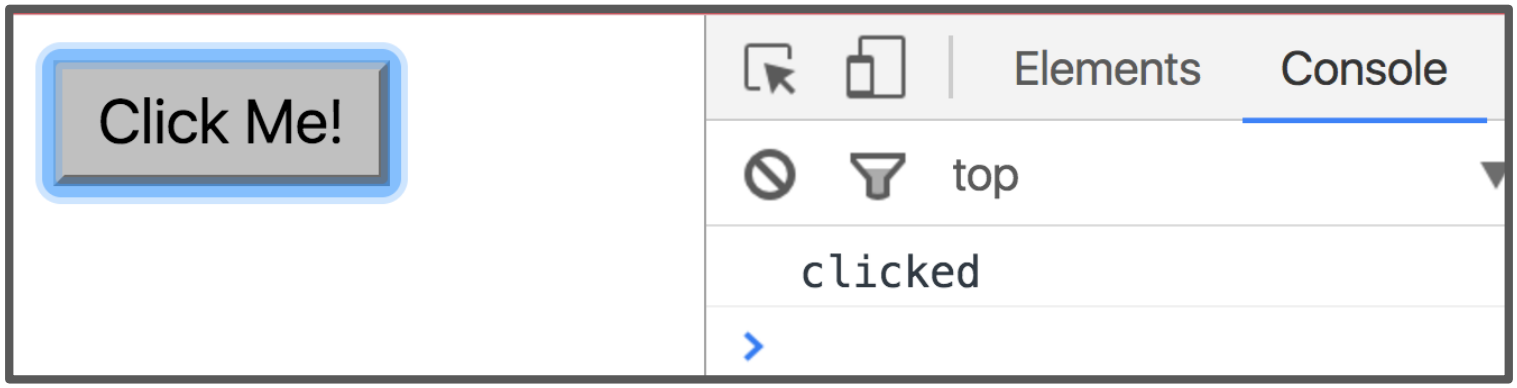
- Scope:
  - Week 01-04: HTML/CSS/ Modern JavaScript
  - No limited if you like OOP JavaScript (this week)

Start working on Assignment 1 immediately!

```html
<html>
▼<head>
    <meta charset="utf-8">
    <title>First JS Example</title>
    <script src="script.js" defer></script>
  </head>
▼<body>
    <button>Click Me!</button>
  </body>
</html>
```

```js
function onClick() {
  console.log('clicked');
}

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



How do we interact with the page?

A few technical details

The DOM objects that we retrieve from `querySelector` and `querySelectorAll` have types:

- Every DOM node is of general type `Node` (an interface)
- `Element` implements the `Node` interface
  (FYI: This has nothing to do with NodeJS, if you've heard of that)
- Each HTML element has a specific `Element` derived class, like `HTMLImageElement`

Attributes and DOM properties

Roughly every **attribute** on an HTML element is a **property** on its respective DOM object...

HTML

```
<img src="puppy.png" />
```

JavaScript

```
const element = document.querySelector('img');
element.src = 'bear.png';
```

(But you should always check the JavaScript spec to be sure.
In this case, check the HTMLImageElement.)

Adding and removing classes

You can control **classes** applied to an HTML element via `classList.add` and `classList.remove`:

```
const image =
document.querySelector('img');

// Adds a CSS class called "active".
image.classList.add('active');

// Removes a CSS class called "hidden".
image.classList.remove('hidden');
```

(More on classList)

Example: Present

Click for a present:

See the CodePen - much more exciting!

```
function openPresent() {
  const image = document.querySelector('img');
  image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

Finding the element twice...

```
function openPresent() {
  const image = document.querySelector('img');
  image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

This redundancy is unfortunate.

Q: Is there a way to fix it?

Finding the element twice...

```
function openPresent() {
  const image = document.querySelector('img');
  image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

This redundancy is unfortunate.

Q: Is there a way to fix it?

CodePen

Event.currentTarget

An Event element is passed to the listener as a parameter:

```
function openPresent(event) {
  const image = event.currentTarget;
  image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
  image.removeEventListener('click', openPresent);
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

The event's currentTarget property is a reference to the object that we attached to the event, in this case the <img>'s Element to which we added the listener.

Psst.. Not to be confused with `Event.target`

(Note: Event has both:

- ***event*`.target`**: the element that was clicked / "dispatched the event" (might be a child of the target)
- ***event*`.currentTarget`**: the element that the original event handler was attached to)

Example: Present

Click for a present:

It would be nice to change the text after the present is "opened"...

Some properties of `Element` objects

Property	Description
`id`	The value of the id attribute of the element, as a string
`innerHTML`	The raw HTML between the starting and ending tags of an element, as a string
`textContent`	The text content of a node and its descendants. (This property is inherited from `Node`)
`classList`	An object containing the classes applied to the element

Maybe we can adjust
the **`textContent`**!
CodePen

```
function openPresent(event) {
  const image = event.currentTarget;
  image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';

  const title = document.querySelector('h1');
  title.textContent = 'Hooray!';

  image.removeEventListener('click', openPresent);
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

We can select the h1 element then set its `textContent` to change what is displayed in the h1. (CodePen)

Another approach: Changing the elements

Add elements via DOM

We can create elements dynamically and add them to the web page via `createElement` and `appendChild`:

```
document.createElement(tag string)
element.appendChild(element);
```

Technically you can also add elements to the webpage via `innerHTML`, but it poses a security risk.

```
// Try not to use innerHTML like this:
element.innerHTML = '<h1>Hooray!</h1>';
```

Remove elements via DOM

We can also call remove elements from the DOM by calling the `remove`() method on the DOM object:

element`.remove();`

And actually setting the `innerHTML` of an element to an **empty string** is a fine way of removing all children from a parent node:

```
// This is fine and poses no security risk.
element.innerHTML = '';
```

```
function openPresent(event) {
  const newHeader = document.createElement('h1');
  newHeader.textContent = 'Hooray!';
  const newImage = document.createElement('img');
  newImage.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';

  const container = document.querySelector('#container');
  container.innerHTML = '';
  container.appendChild(newHeader);
  container.appendChild(newImage);
}

const image = document.querySelector('img');
image.addEventListener('click', openPresent);
```

CodePen

Hmm, the effect is slightly janky though:
The text changes faster than the image loads.

Q: How do we fix this issue?

`display: none;`

There is yet another super helpful value for `display`:

```
display: block;
display: inline;
display: inline-block;
display: flex;
display: none;
```

`display: none;` turns off rendering for the element and all its children. It's treated as if the element were not in the document at all...

`display: none;`

There is yet another super helpful value for `display`:

```
display: block;
display: inline;
display: inline-block;
display: flex;
display: none;
```

`display: none;` turns off rendering for the element and all its children. It's treated as if the element were not in the document at all…

…but the content (such as the images) is still loaded.

```html
<div id="gift-outside">
  <h1>Click for a present:</h1>
  <img src="https://s3-us-west-2.amazonaws.com/s.cdpn.io/1083533/gift-icon.png" />
</div>
<div id="gift-inside" class="hidden">
  <h1>Hooray!</h1>
  <img src="https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif" />
</div>
```

```css
.hidden {
  display: none;
}
```

We can add both views to the HTML, with one view hidden by default… (CodePen)

```
function openPresent(event) {
  const image = event.currentTarget;
  image.removeEventListener('click', openPresent);

  const giftOutside = document.querySelector('#gift-outside');
  const giftInside = document.querySelector('#gift-inside');
  giftOutside.classList.add('hidden');
  giftInside.classList.remove('hidden');
}

const image = document.querySelector('#gift-outside img');
image.addEventListener('click', openPresent);
```

Then we toggle the display state of the containers
by adding/removing the `hidden` class.
(CodePen)

Recap

Several strategies for updating HTML elements in JS:

1. Change content of existing HTML elements in page:

- Good for simple text updates

2. Add elements via `createElement` and `appendChild`

- Needed if you're adding a variable number of elements

3. Put all "views" in the HTML but set inactive ones to hidden, then update `display` state as necessary.

- Good when you know ahead of time what element(s) you want to display
- Can be used in conjunction with (1) and/or (2)

Other JavaScript events?

We've been doing a ton of JavaScript examples that involve `click` events...

Aren't there other types of events?

Other JavaScript events?

We've been doing a ton of JavaScript examples that involve `click` events…

Aren't there other types of events?

- Of course!
- Today we'll talk about:
  - **Keyboard events**
  - **Pointer / mobile events**

How do we listen
to keyboard events?

Keyboard events

Event name	Description
`keydown`	Fires when any key is pressed. Continues firing if you hold down the key. (mdn)
`keypress`	(deprecated) Fires when any **character** key is pressed, such as a letter or number. Continues firing if you hold down the key. (mdn)
`keyup`	Fires when you stop pressing a key. (mdn)

You can listen for keyboard events by adding the event listener to `document`:

```
document.addEventListener('keyup', onKeyUp);
```

KeyboardEvent.key

```
function onKeyUp(event) {
  console.log('onKeyUp:' + event.key);
}
document.addEventListener('keyup', onKeyUp);
```

Functions listening to a key-related event receive a parameter of `KeyboardEvent` type.

The `KeyboardEvent` object has a `key` property, which stores the string value of the key, such as `"Escape"`

- List of key values

Useful key values

Key string value	Description
`"Escape"`	The Escape key
`"ArrowRight"`	The right arrow key
`"ArrowLeft"`	The left arrow key

Example: key-events.html

MouseEvent

Event name	Description
`click`	Fired when you click and release (mdn)
`mousedown`	Fired when you click down (mdn)
`mouseup`	Fired when when you release from clicking (mdn)
`mousemove`	Fired repeatedly as your mouse moves (mdn)

*__`mousemove`__ only works on desktop, since there's no concept of a mouse on mobile.

TouchEvent

Event name	Description
`touchstart`	Fired when you touch the screen (mdn)
`touchend`	Fired when you lift your finger off the screen (mdn)
`touchmove`	Fired repeatedly while you drag your finger on the screen (mdn)
`touchcancel`	Fired when a touch point is "disrupted" (e.g. if the browser isn't totally sure what happened) (mdn)

*__`touchmove`__ only works on mobile (example)

`clientX` and `clientY`

```
function onClick(event) {
  console.log('x' + event.clientX);
  console.log('y' + event.clientY);
}
element.addEventListener('click', onClick);
```

MouseEvents have a `clientX` and `clientY` :

- `clientX`: x-axis position relative to the left edge of the browser viewport
- `clientY`: y-axis position relative to the top edge of the browser viewport

Implementing drag

When a user clicks down/touches an element…

Implementing drag

```
originX = 100;
```

Take note of the starting position.

Implementing drag

```
originX = 100;
newX = 150;
```

Then on `mousemove` / `touchmove`, make note of the new mouse position

Implementing drag

```
originX = 100;
newX = 150;
```

Move the element by the difference between the old and new positions.

Implementing drag

Then on release...

Implementing drag

… stop listening to `mousemove` / `touchmove`.

Dragging on mobile and desktop

Wouldn't it be nice if we didn't have to listen to different events for mobile and desktop?

PointerEvent

`PointerEvent`: "pointer" events that work the same with for both mouse and touch

- Not to be confused with `pointer-events` CSS property (completely unrelated)
- **Note:** In this case, Mozilla's documentation on `PointerEvent` is not great.
  - A Google blog post on `PointerEvent`

`PointerEvent` inherits from `MouseEvent`, and therefore has `clientX` and `clientY`

PointerEvent

Event name	Description
`pointerdown`	Fired when a "pointer becomes active" (touch screen or click mouse down) (mdn)
`pointerup`	Fired when a pointer is no longer active (mdn)
`pointermove`	Fired repeatedly while the pointer moves (mouse move or touch drag) (mdn)
`pointercancel`	Fired when a pointer is "interrupted" (mdn)

*__`pointermove`__ works on mobile and desktop!

… Except…

Our first controversial feature!

`PointerEvent` is **not** implemented on all browsers yet:

- Firefox implementation is in progress
- Safari outright opposes this API… since 2012.

Argh!!! Does this mean we can't use it?

Polyfill library

A polyfill library is code that implements support for browsers that do not natively implement a web API.

Luckily there is a polyfill library for `PointerEvent`: https://github.com/jquery/PEP

PointerEvent Polyfill

To use the PEP polyfill library, we add this `script` tag to our HTML:

```
<script src="https://code.jquery.com/pep/0.4.1/pep.js"></script>
```

And we'll need to add `touch-action="none"` to the area where we want `PointerEvents` to be recognized*:

```
<section id="photo-view" class="hidden" touch-action="none">
</section>
```

*Technically what this is doing is it is telling the browser that we do not want the default touch behavior for children of this element, i.e. on a mobile phone, we don't want to recognize the usual "pinch to zoom" type of events because we will be intercepting them via PointerEvent. This is normally a CSS property, but the limitations of the polyfill library requires this to be an HTML attribute instead.

Classes in JavaScript

Amateur JavaScript

So far the JavaScript code we've been writing has looked like this:

- Mostly all in one file
- All global functions
- Global variables to save state between events

It would be nice to write code in a **modular** way...

```
//
// Album view functions
//
let currentIndex = null;
function onThumbnailClick(event) {
  currentIndex = event.currentTarget.dataset.index;
  const image = createImage(event.currentTarget.src);
  showFullsizeImage(image);
  document.body.classList.add('no-scroll');
  modalView.style.top = window.pageYOffset + 'px';
  modalView.classList.remove('hidden');
}

//
// Photo view functions
//
function createImage(src) {
  const image = document.createElement('img');
  image.src = src;
  return image;
}

function showFullsizeImage(image) {
  modalView.innerHTML = '';

  image.addEventListener('pointerdown', startDrag);
  image.addEventListener('pointermove', duringDrag);
  image.addEventListener('pointerup', endDrag);
  image.addEventListener('pointercancel', endDrag);
  modalView.appendChild(image);
}

let originX = null;
function startDrag(event) {
  event.preventDefault();
  // Needed so clicking on picture doesn't cause modal dialog to close.
  event.stopPropagation();

  originX = event.clientX;
  event.target.setPointerCapture(event.pointerId);
}

function duringDrag(event) {
  if (originX) {
    const currentX = event.clientX;
    const delta = currentX - originX;
    const element = event.currentTarget;
    element.style.transform = 'translateX(' + delta + 'px)';
  }
}

function endDrag(event) {
  if (!originX) {
    return;
  }

  const currentX = event.clientX;
  const delta = currentX - originX;
  originX = null;

  let nextIndex = currentIndex;
  if (delta < 0) {
    nextIndex++;
  } else {
    nextIndex--;
  }

  if (nextIndex < 0 || nextIndex == PHOTO_LIST.length) {
    event.currentTarget.style.transform = '';
    return;
  }
```

ES6 classes

We can define **classes** in JavaScript using a syntax that is similar to Java or C++:

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

These are often called "**ES6 classes**" or "**ES2015 classes**" because they were introduced in the EcmaScript 6 standard, the 2015 release

- Recall that EcmaScript is the standard; JavaScript is an implementation of the EcmaScript standard

Wait a minute...

Wasn't JavaScript created in 1995?

And classes were introduced... 20 years later in 2015?

Q: Was it seriously not possible to create classes in JavaScript before 2015?!

Objects in JavaScript

In JavaScript, there are several ways to create blueprints for objects. Two broad approaches:

1. Functional
    a. This approach has existed since the creation of the JavaScript
    b. Weird syntax for people used to languages like Java, C++, Python
    c. Doesn't quite behave the same way as objects in Java, C++, Python
2. Classical
    a. This is the approach that just got added to the language in 2015
    b. Actually just "syntactic sugar" over the functional objects in JavaScript, so still a little weird
    c. But syntax is much more approachable

Objects in JavaScript

In JavaScript, there are several ways to create blueprints for objects. Two broad approaches:

1. **Functional**
    a. This approach has existed since the creation of the JavaScript
    b. Weird syntax for people used to languages like Java, C++, Python
    c. Doesn't quite behave the same way as objects in Java, C++, Python
2. **Classical**
    a. This is the approach that just got added to the language in 2015
    b. Actually just "syntactic sugar" over the functional objects in JavaScript, so still a little weird
    c. But syntax is much more approachable

This approach is quite controversial.

Class controversy

"There is one thing I am certain is a bad part, a very terribly bad part, and that is the new class syntax [in JavaScript]... [T]he people who are using `class` will go to their graves never knowing how miserable they were." (source)

-- Douglas Crockford, author of *JavaScript: The Good Parts*; prominent speaker on JavaScript; member of TC39 (committee that makes ES decisions)

Back to classes!

Public methods

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

`constructor` is optional.

Parameters for the constructor and methods are defined in the same they are for global functions.

You do not use the `function` keyword to define methods.

Public methods

```
class ClassName {
  constructor(params) {
    ...
  }
  methodOne() {
    this.methodTwo();
  }
  methodTwo() {
    ...
  }
}
```

Within the class, you must always refer to other methods in the class with the **`this.`** prefix.

Public methods

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

All methods are **public**, and you **cannot** specify private methods… yet.

Public methods

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

As far as I can tell, private methods aren't in the language only because they are still figuring out the spec for it. (They will figure out private fields first.)

Public fields

```
class ClassName {

  constructor(params) {
    this.fieldName = fieldValue;
    this.fieldName = fieldValue;
  }

  methodName() {
    this.fieldName = fieldValue;
  }
}
```

Define public fields by setting **`this.`*`fieldName`*** in the constructor… or in any other function.

(This is slightly hacky underneath the covers and there is a draft to add public fields properly to ES.)

Public fields

```
class ClassName {
  constructor(params) {
    this.someField = someParam;
  }
  methodName() {
    const someValue = this.someField;
  }
}
```

Within the class, you must always refer to fields with the `this.` prefix.

Public fields

```
class ClassName {

  constructor(params) {
    this.fieldName = fieldValue;
    this.fieldName = fieldValue;
  }

  methodName() {
    this.fieldName = fieldValue;
  }
}
```

You cannot define private fields… yet.

(Again, there are plans to add add private fields to ES once the spec is finalized.)

Instantiation

Create new objects using the new keyword:

```
class SomeClass {
  ...
  someMethod() { … }
}

const x = new SomeClass();
const y = new SomeClass();
y.someMethod();
```

Why classes?

Why are we even doing this?

Why do we need to use classes when web programming?

Why can't we just keep doing things the way we've been doing things, with global functions and global variables?

Why classes?

A: All kinds of reasons

- For a sufficiently small task, globals variables, functions, etc. are fine
- But for a larger website, your code will be hard to understand **and** easy to break if you do not organize it
- Using classes and object-oriented design is the most common strategy for organizing code

E.g. in the global scope, it's hard to know at a variable called "name" would be referring to, and any function could accidentally write to it.

- But when defined in a Student class, it's inherently clearer what "name" means, and it's harder to accidentally write that value

Organizing code

Well-engineered software is well-organized software:

- Software engineering is all about knowing
  1. What to change
  2. Where to change it
- You can read an existing codebase better if it is well-organized
  - *"Why do I need to read a codebase?"* Because you need to modify the codebase to add features and fix bugs

Other problems with globals

Having a bunch of loose variables in the global scope is asking for trouble

- Much easier to hack
  - Can access via extension or Web Console
  - Can override behaviors
- Global scope gets polluted
  - What if you have two functions with the same name? One definition is overridden without error
- Very easy to modify the wrong state variable

All these things are much easier to avoid with classes

Example: Present

Let's create a Present class inspired by our present example from last week.

Starter

How to design classes

You may be wondering:

- How do I decide what classes to write?
- How do I decide what methods to add to my class?

Disclaimer

This is not a software engineering class, and this is not an object-oriented design class.

As such, we will not grade your OO design skills.

However, this also means we won't spend too much time explaining *how* to break down your app into well-composed objects.

(It takes practice and experience to get good at this.)

One general strategy

"Component-based" approach: Use classes to add functionality to HTML elements ("components")

Each component:

- Has exactly one container element / root element
- Handles attaching/removing event listeners
- Can own references to child components / child elements

(Similar strategy to ReactJS, Custom Elements, many other libraries/frameworks/APIs before them)

Container element

One pattern:

```
<div id="present-container"></div>

const element =
  document.querySelector('#present-container');
const present = new Present(element);
// Immediately renders the present
```

Container element

A similar pattern:

```
<div id="present-container"></div>

const element =
  document.querySelector('#present-container');
const present = new Present();
// Renders with explicit call
present.renderTo(element);
```

Web: Almost total freedom

Unlike most app platforms (i.e. Android or iOS), you have almost total freedom over exactly how to organize your code

Pros:

- Lots of control!

Cons:

- Lots and lots and lots of decisions to make

Web: Almost total freedom

Unlike most app platforms (i.e. Android or iOS), you have almost total freedom over exactly how to organize your code

Pros:

- Lots of control!

Cons:

- Lots and lots and lots of decisions to make
- This is why **Web Frameworks** are so common: A web framework just make a bunch of software engineer decisions for you ahead of time (+provides starter code)

Don't forget `this`

```
// Create image and append to container.
const image = document.createElement('img');
image.src = 'https://s3-us-west-2.amazonaws.com/s.cdpn.io/1083533/gift-icon.png';
image.addEventListener('click', this._openPresent);
```

If the event handler function you are passing to `addEventListener` is a method in a class, you must pass "`this.`***functionName***" (finished)

"Private" with _

A somewhat common JavaScript coding convention is to add an underscore to the beginning or end of private method names:

```
_openPresent() {
   ...
}
```

I'll be doing this in this class for clarity, but note that it's frowned upon by some.

Solution: Present

CodePen finished

Present class

present.js

```
class Present {
  constructor(containerElement) {
    this.containerElement = containerElement;

    // Create image and append to container.
    const image = document.createElement('img');
    image.src = 'https://s3-us-west-2.amazonaws.com/s.cdpn.io/1083533/gift-icon.png';
    image.addEventListener('click', this._openPresent);
    this.containerElement.append(image);
  }

  _openPresent(event) {
    const image = event.currentTarget;
    image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
    image.removeEventListener('click', this._openPresent);
  }
}
```

Present class

main.js

```
const container = document.querySelector('#presents');
const present = new Present(container);
```

index.html

```
<head>
  <meta charset="UTF-8" />
  <title>Simple class: present</title>
  <link rel="stylesheet" href="styles/index.css">
  <script src="scripts/present.js" defer></script>
  <script src="scripts/main.js" defer></script>
</head>
<body>
  <div id="presents"></div>
</body>
```

`this` in event handler

```
class Present {
  constructor(containerElement) {
    this.containerElement = containerElement;

    // Create image and append to container.
    const image = document.createElement('img');
    image.src = 'https://s3-us-west-2.amazonaws.com/s.cdpn.io/1083533/gift-icon.png';
    image.addEventListener('click', this._openPresent);
    this.containerElement.append(image);
  }

  _openPresent(event) {
    const image = event.currentTarget;
    image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
    image.removeEventListener('click', this._openPresent);
  }
}
```

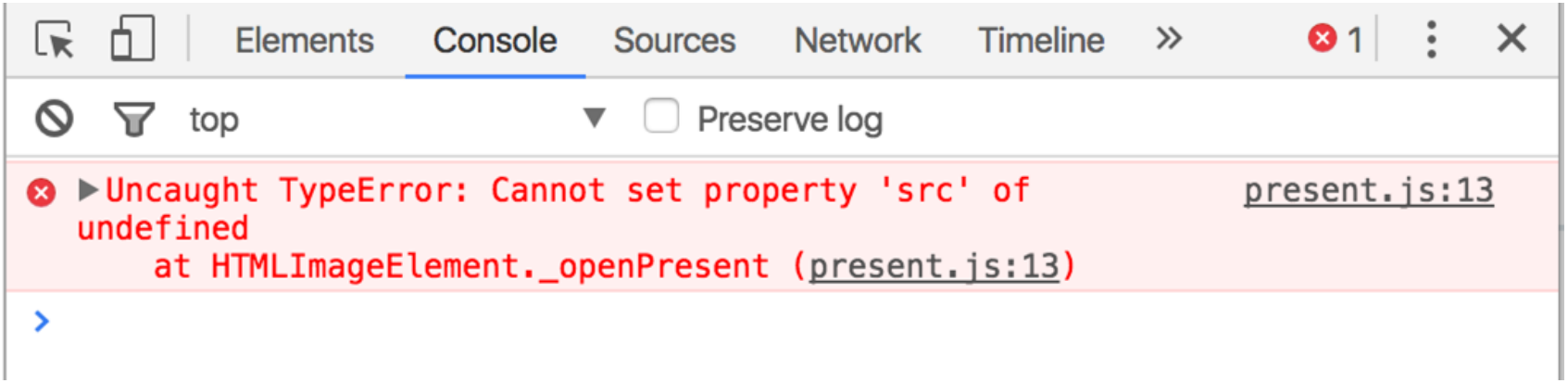
Right now we access the image we create in the constructor in `_openPresent` via `event.currentTarget`.

`this` in event handler

```
  _openPresent(event) {
    this.image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
    this.image.removeEventListener('click', this._openPresent);
  }
}
```

Q: What if we make the `image` a field and access it `_openPresent` via `this.image` instead of `event.currentTarget`?

`this` in event handler



Error message!

CodePen / Debug

What's going on?

JavaScript `this`

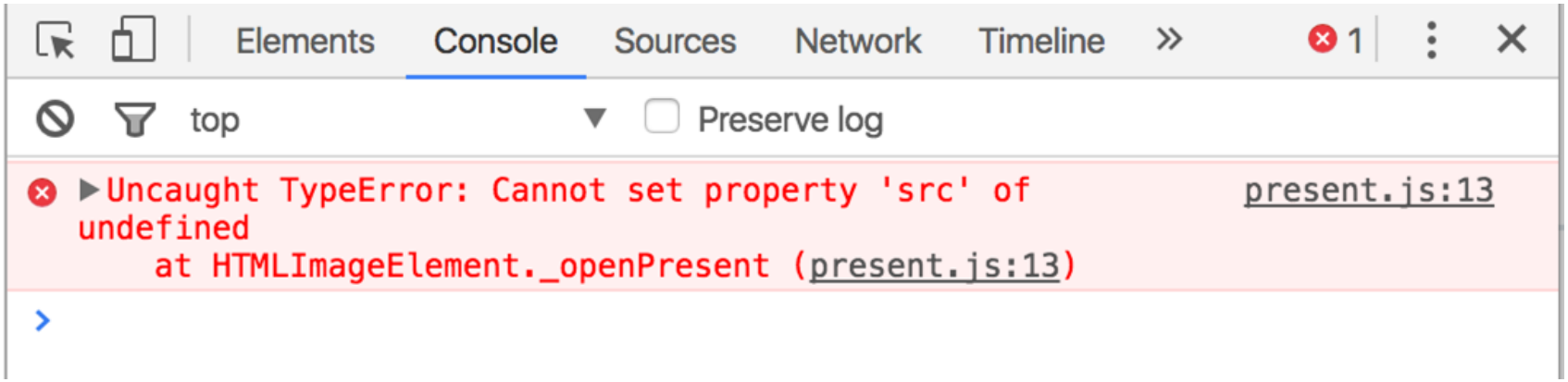
The `this` keyword in JavaScript is **dynamically assigned**, or in other words: `this` means different things in different contexts (mdn list)

- In our constructor, `this` refers to the instance
- When called in an event handler, `this` refers to… the element that the event handler was attached to (mdn).

`this` in event handler

```
  _openPresent(event) {
    this.image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
    this.image.removeEventListener('click', this._openPresent);
  }
}
```

That means `this` refers to the `<img>` element, not the instance variable of the class...



...which is why we get this error message.

Solution: `bind`

To make `this` always refer to the instance object for a method in the class (i.e. to get `this` to behave as you'd expect), you can add the following line of code in the `constructor`:

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

```
class Present {
  constructor(containerElement) {
    this.containerElement = containerElement;

    // Bind event listeners.
    this._openPresent = this._openPresent.bind(this);
```

Solution: `bind`

Now `this` in the `_openPresent` method refers to the instance object (CodePen / Debug):

```
_openPresent(event) {
  this.image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
  this.image.removeEventListener('click', this._openPresent);
}
```

Moral of the story:

Don't forget to `bind()` event listeners in your constructor!!

```
class Present {
  constructor(containerElement) {
    this.containerElement = containerElement;

    // Bind event listeners.
    this._openPresent = this._openPresent.bind(this);
```

One more time:

Don't forget to `bind()` event listeners in your constructor!!

Communicating between classes

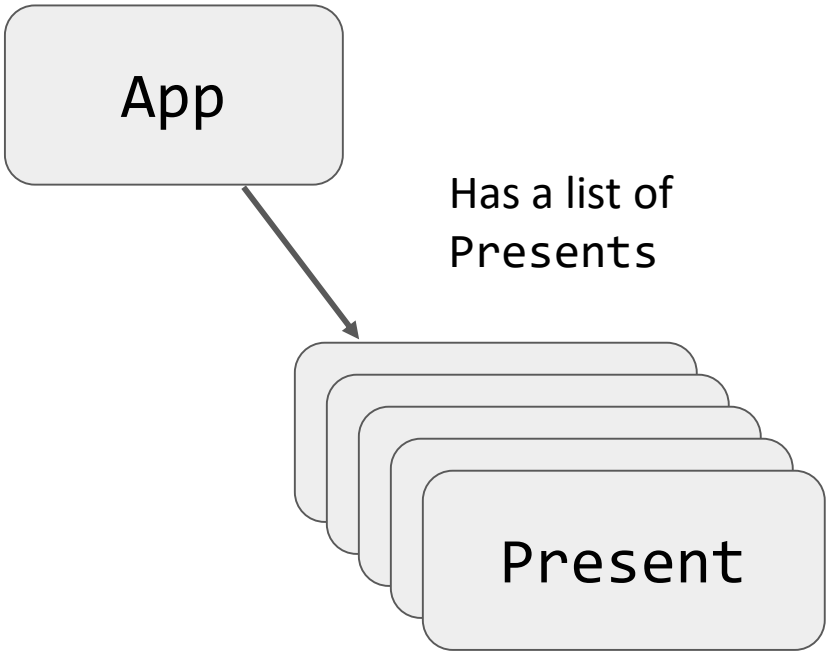
Multiple classes

Let's say that we have multiple presents now (CodePen):

Multiple classes

And we have implemented this with two classes:

- App: Represents the entire page
  - `Present`: Represents a single present



CodePen

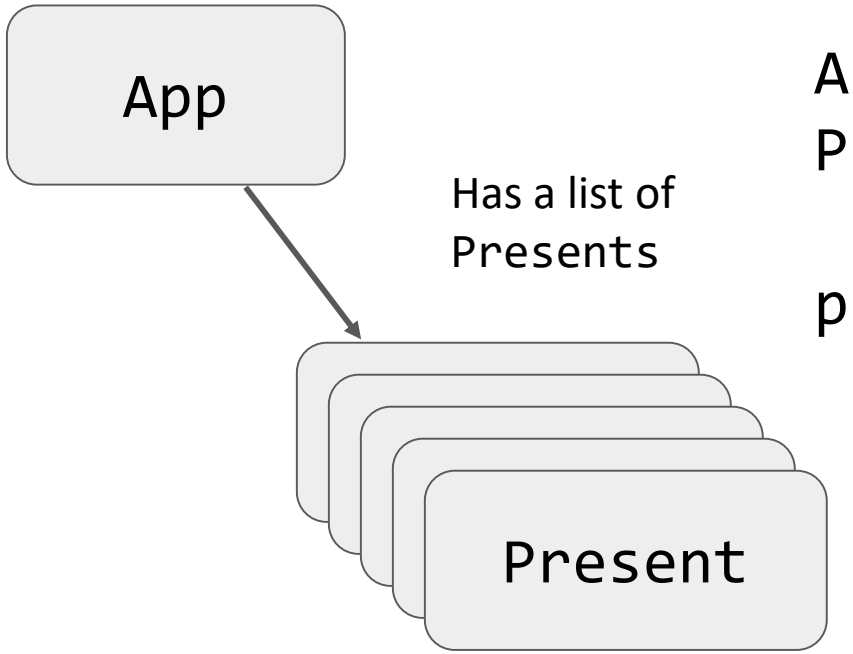
Communicating btwn classes

What if we want to change the **title** when all present have been opened? (CodePen)

Enjoy your presents!

Communication btwn classes

Communicating from App → `Present` is easy, since App has a list of the `Present` objects.

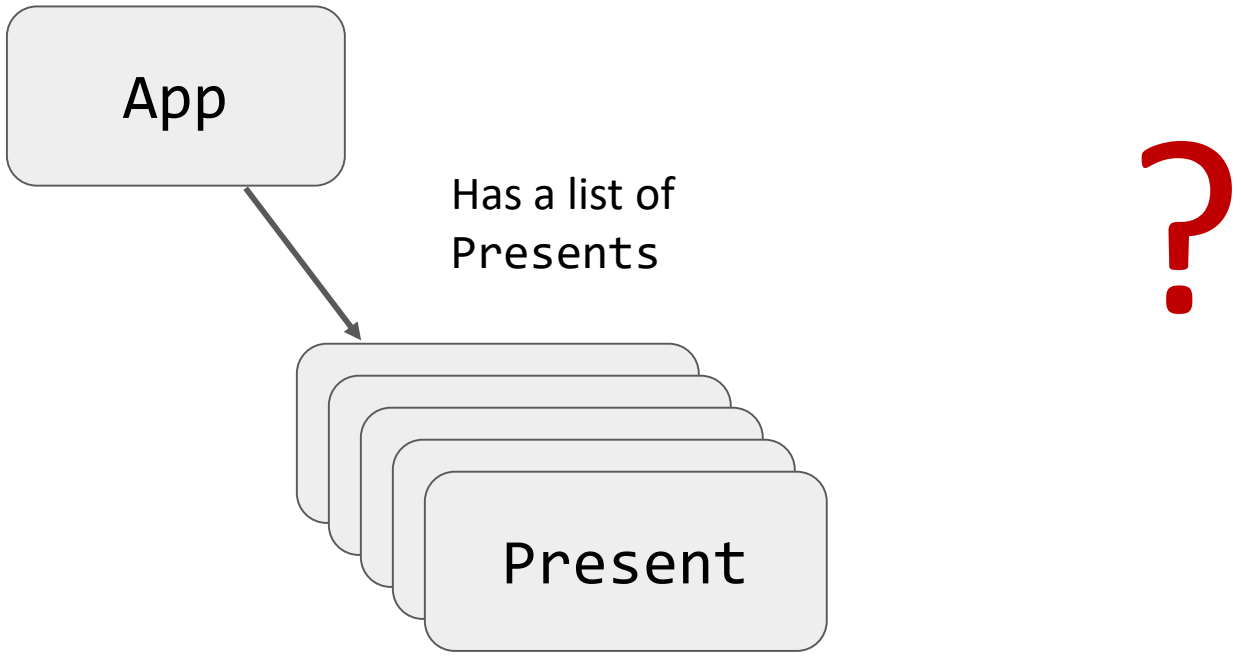


App can just call methods on `Present`:

```
present.doWhatever();
```

Communication btwn classes

However, communicating `Present` → App is not as easy, because `Presents` do not have a reference to App



More next week!