CS193X: Web Programming Fundamentals

Spring 2017

Victoria Kirst
(vrk@stanford.edu)

Today's schedule

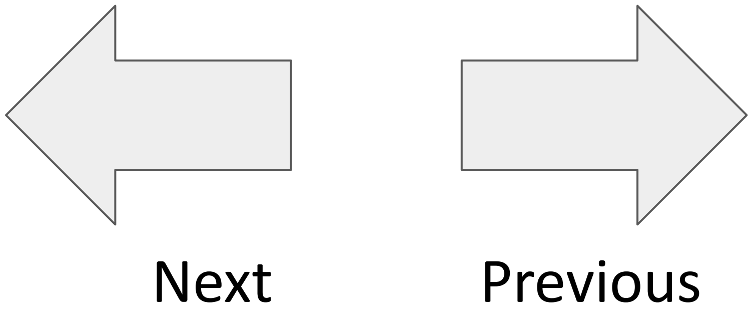
Today

- Mobile events
- Simple CSS animations
- Classes and objects in JavaScript
- `this` keyword and `bind`
- **HW2 due; HW3 assigned**
- Victoria has office hours 2:30 - 4pm

Custom swipe events

- There are no gesture events in JavaScript (yet).
- That means there is no "Left Swipe" or "Right Swipe" event we can listen to. (Note that drag does not do what we want, nor does it work on mobile)

To get this behavior, we must implement it ourselves.



`transform`

`transform` is a strange but powerful CSS property that allow you to translate, rotate, scale, or skew an element.

`transform: translate(`***x***`, `***y***`)`	Moves element relative to its natural position by ***x*** and ***y***
`transform: translateX(`***x***`)`	Moves element relative to its natural position horizontally by ***x***
`transform: translateY(`***y***`)`	Moves element relative to its natural position vertically by ***y***
`transform: rotate(`***deg***`)`	Rotates the element clockwise by ***deg***
`transform: rotate(10deg) translate(5px, 10px);`	Rotates an element 10 degrees clockwise, moves it 5px down, 10px right

Examples

`translate` vs `position`

Can't you use `relative` or `absolute` positioning to get the same effect as `translate`? What's the difference?

- `translate` is much faster
- `translate` is optimized for animations

See comparison (article):

- Absolute positioning (click "10 more macbooks")
- `transform: translate` (click "10 more macbooks")

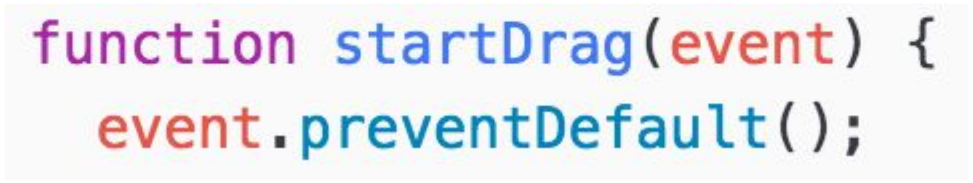
Dragon walk

Let's make it possible to drag this dragon across the sidewalk:

CodePen / live

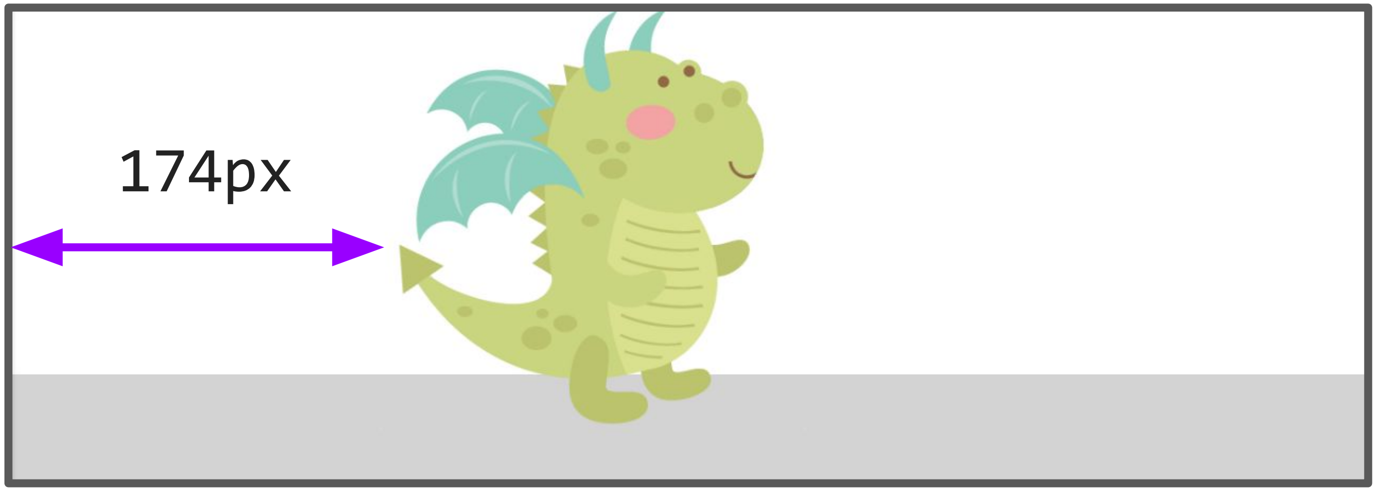
`preventDefault()`

On desktop, there's a default behavior for dragging an image, which we need to disable with [`event.preventDefault()`]:



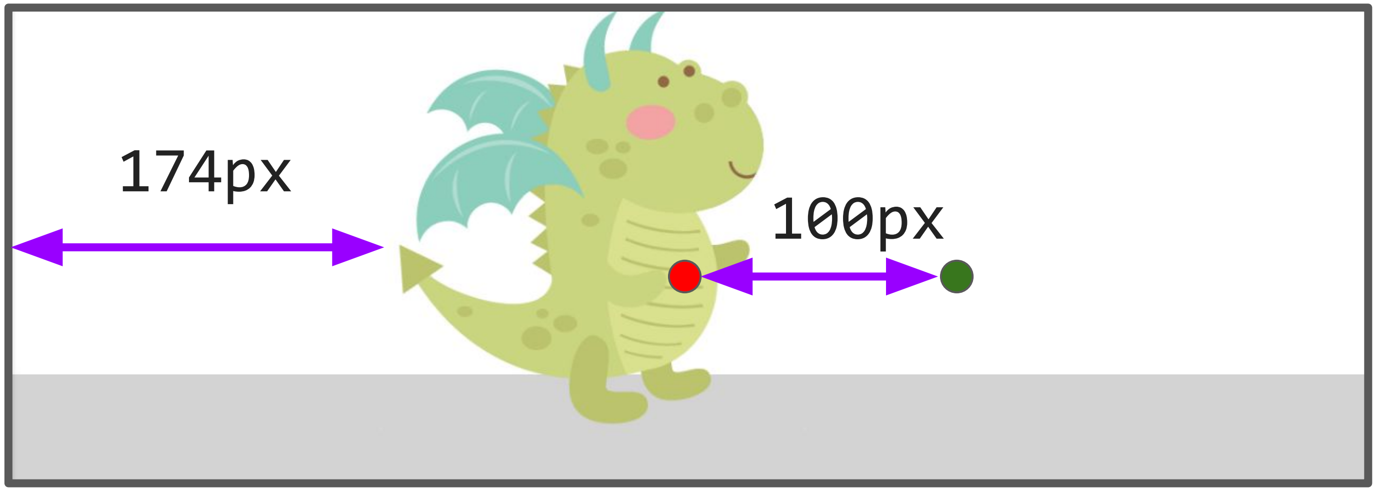
```
function startDrag(event) {
  event.preventDefault();
```

Dragon walk bug (buggy code)



Our dragon is already translated in the X direction by 174px...

Dragon walk bug (buggy code)

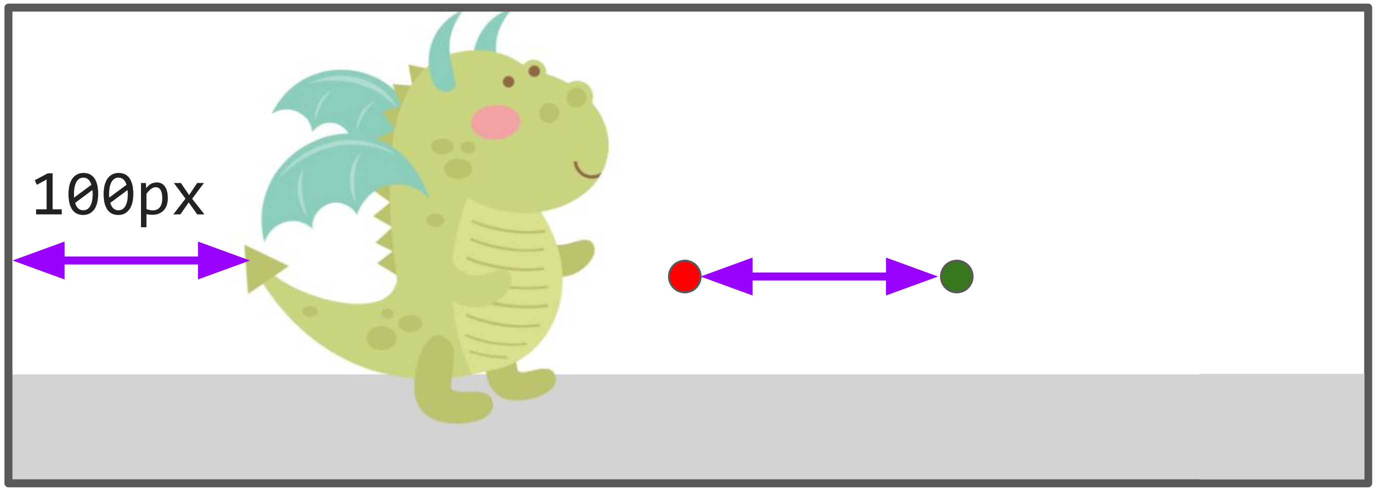


So if we drag again….

🟥 originX

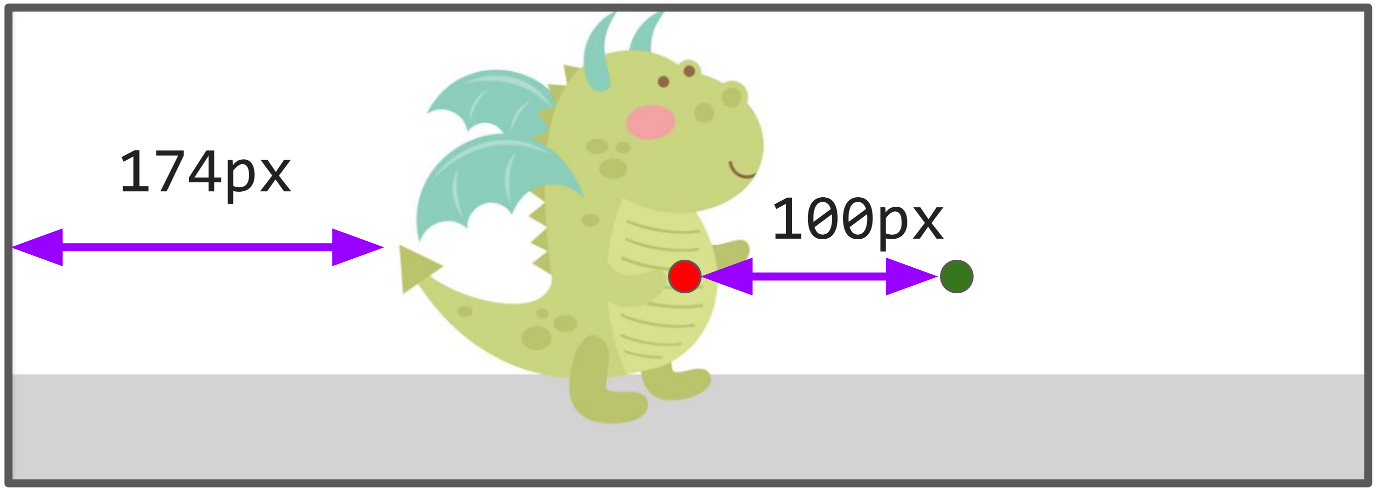
🟩 event.clientX

Dragon walk bug (buggy code)



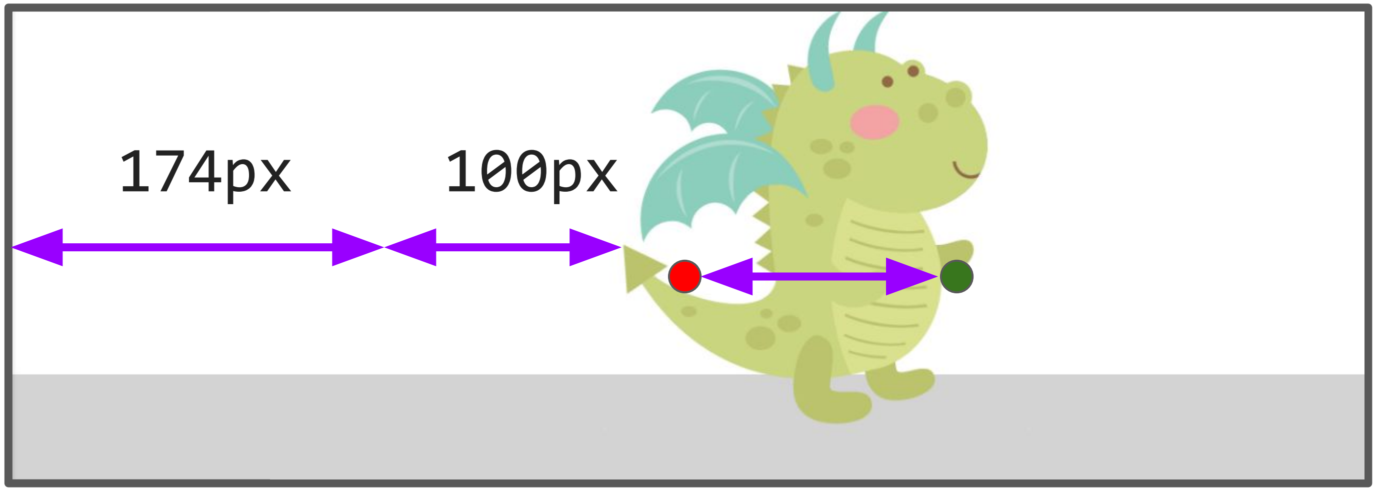
Our buggy code moves our dragon from where it originally started, rather than from its newly translated position

Dragon walk bug fix



What we actually want to do is move our dragon 100px from where it was last dragged.

Dragon walk bug fix



What we actually want to do is move our dragon 100px from where it was last dragged.

Fixed code: [CodePen]()

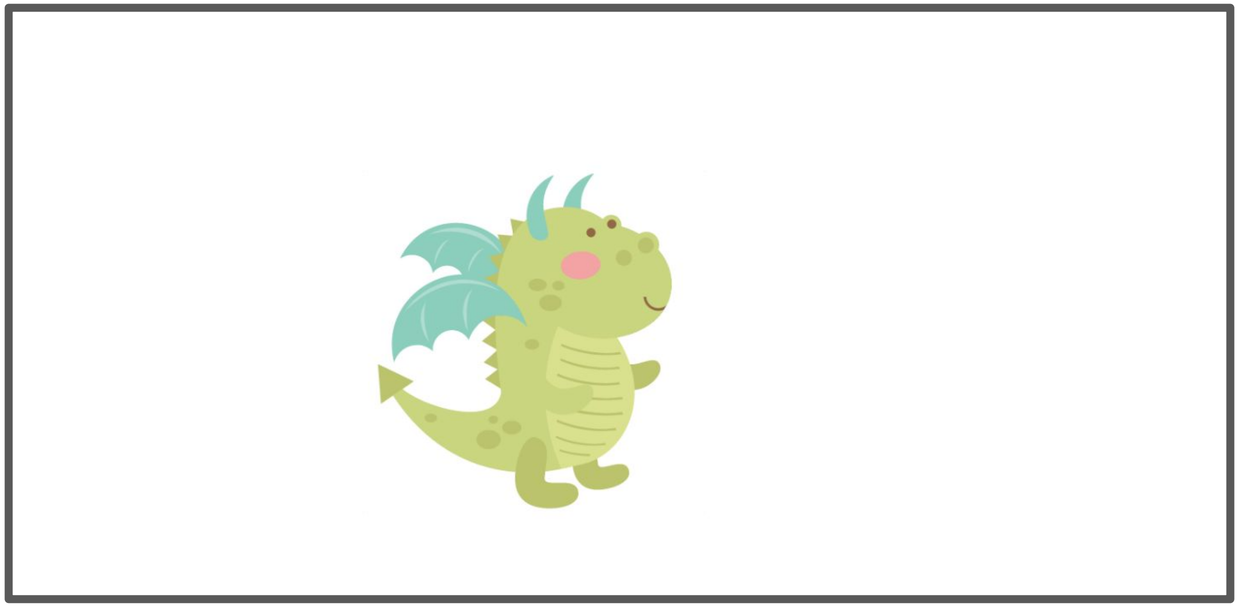
`setPointerCapture()`

To listen to pointer events that occur when the pointer goes offscreen, call `setPointerCapture` on the target you want to keep tracking:

```
event.currentTarget.setPointerCapture(event.pointerId);
```

2-D dragon walk

We can make our dragon move in both the X and Y direction using the same technique for the Y-direction:



Solved CodePen for 2-D walk

Back to our photo album example

`style` attribute

The `style` attribute has **higher precedence** than any CSS property.

To undo a style set via the `style` attribute, you can set it to the empty string:

```
element.style.transform = '';
```

Now the element will be styled according to any rules in the CSS file(s).

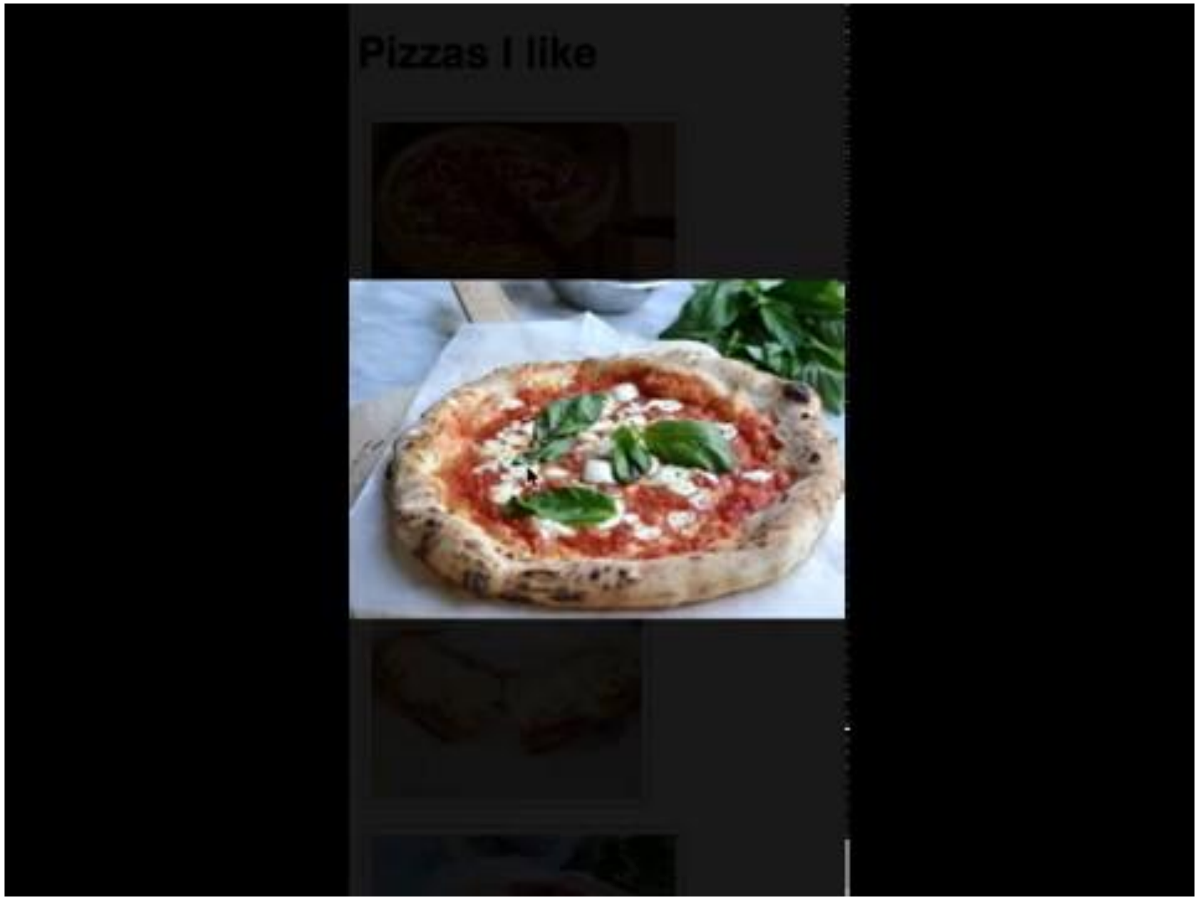
(`requestAnimationFrame`)

(We are missing one key piece of getting smooth dragging motion, which is: `requestAnimationFrame`

However, using `requestAnimationFrame` well requires us to know a little bit more about the JavaScript event loop. Functional programming also helps. We'll get there next week!)

Photo album jerkiness

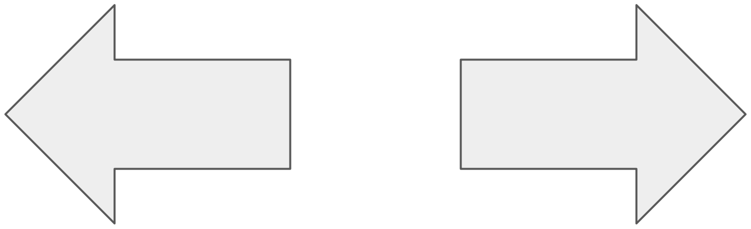
It feels a little jerky when we swipe through photos:



Softening the edges

This is mostly a perception issue. We can make the UI **feel** a little smoother if we added some animations.

- The image should **slide in from the left** if we are going to the previous picture
- The image should **slide in from the right** if we are going to the next picture


Next Previous

CSS animations

CSS animations syntax

```
@keyframes animation-name {
   from {
      CSS styles
   }
   to {
      CSS styles
   }
}
```

Examples

Then set the following CSS property:

```
animation: animation-name duration;
```

Example: Fade in

```css
#album-view img {
  animation: fadein 0.5s;
}

@keyframes fadein {
  from {
    opacity: 0;
  }
  to {
    opacity: 1;
  }
}
```

CSS animations events

You can listen to animation events (mdn):

- `animationstart`: fires at the beginning of the animation
- `animationend`: fires at the end of the animation

```
const image = document.querySelector('img');
image.addEventListener('animationstart', onStart);
image.addEventListener('animationend', onEnd);

image.classList.add('fade-grow');
```

CodePen example

CSS animations

There are all kinds of customizations (mdn):

- Set multiple keyframes
- Set keyframes by percentage
- Make animations repeat
- Make animations alternate
- Change the timing function

Also note that not all CSS is animatable: see list

Fancy CodePen example

(credit CSS tricks -- check out their article for more details)

CSS transitions

You can also set a **CSS transition** on an element, which controls the animation speed of a changing CSS property (mdn)

```
transition: Ns;
```

CodePen example

Finished result:
photo-mobile-finished.html

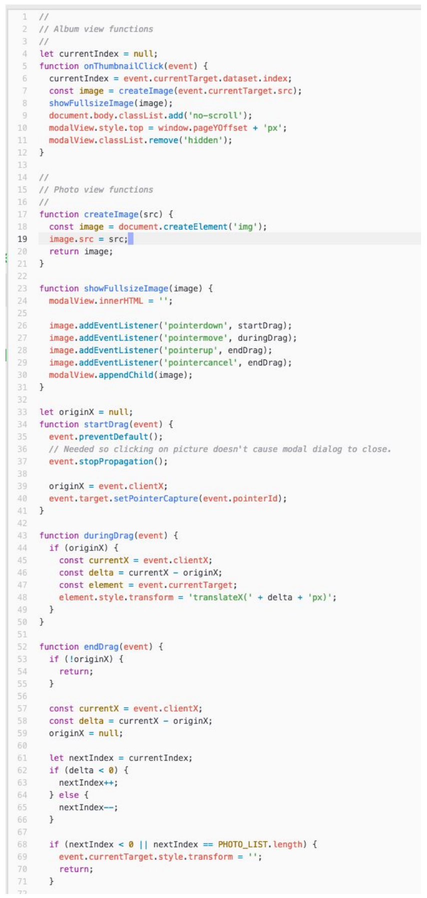
Classes in JavaScript

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 (comm

that makes ES decisions)

Functional approach: next week!

Today:

- We will check out ES6 classes.

Next week:

- We will explore "functional JavaScript," allowing us to understand a way to create object factories without classes.

In this class:

- We will use ES6 classes because the syntax is significantly simpler.

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();
```

Example: Present

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

Starter / 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

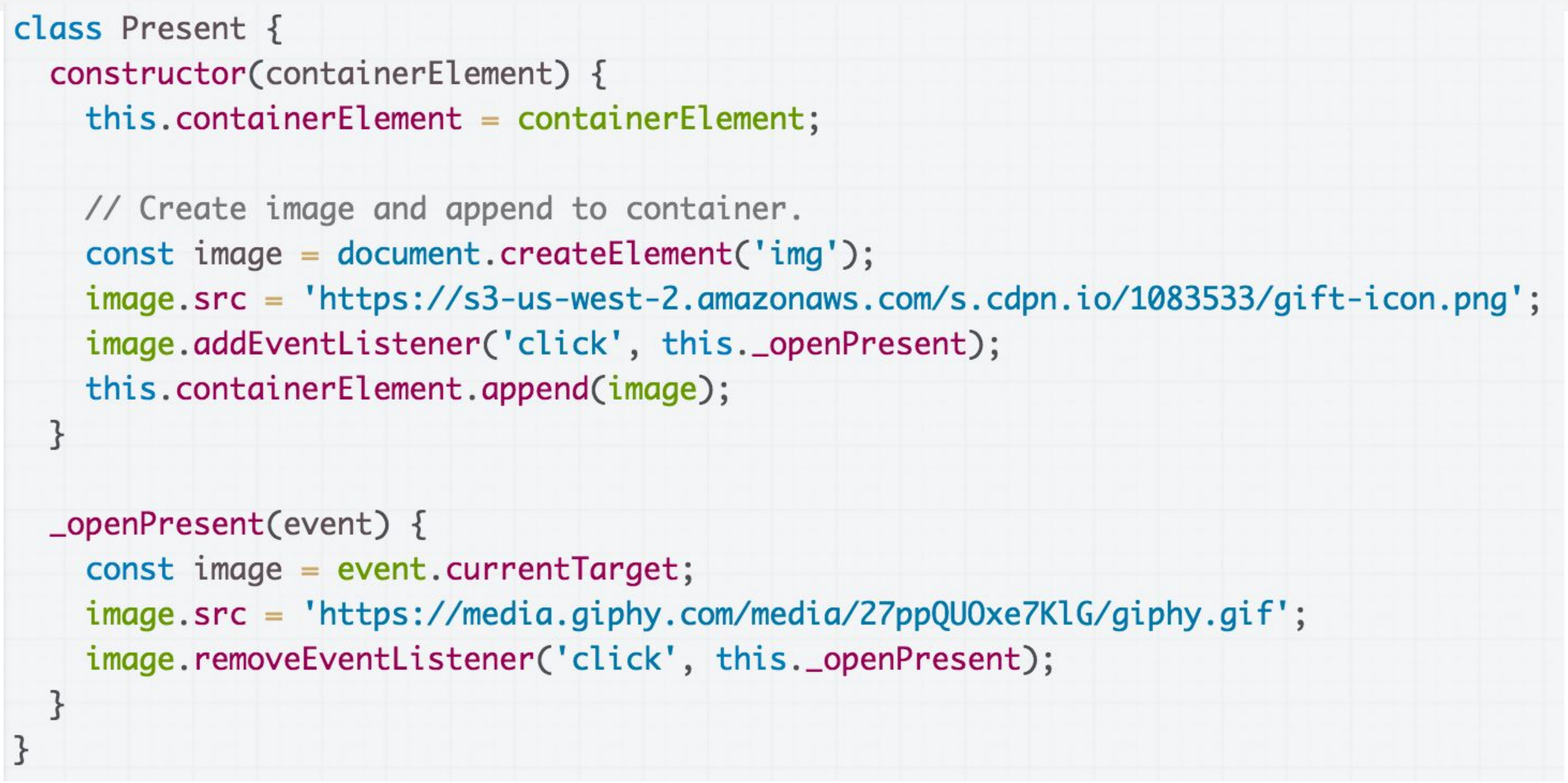
main.js

```
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



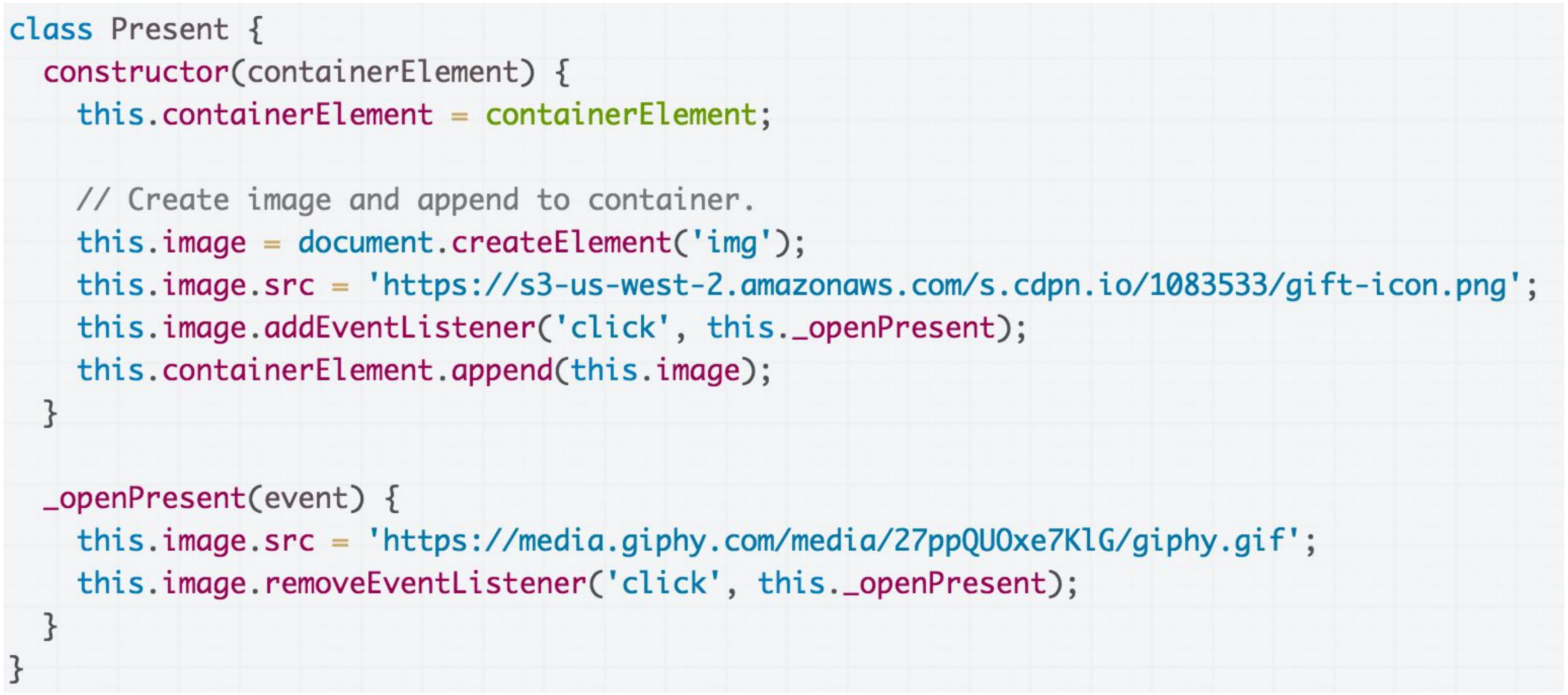
```
class Present {
  constructor(containerElement) {
    this.containerElement = containerElement;

    // 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



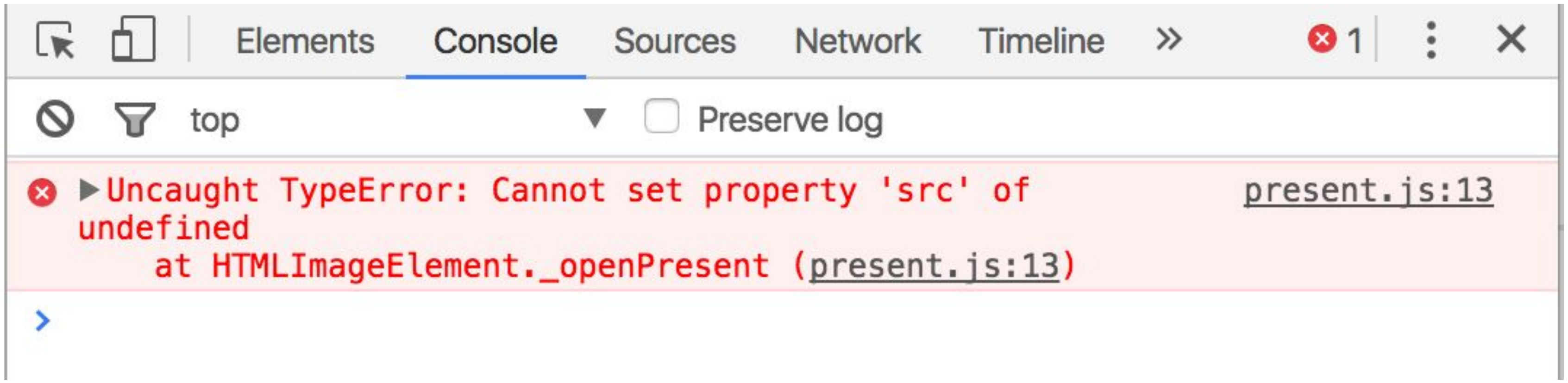
```
class Present {
  constructor(containerElement) {
    this.containerElement = containerElement;

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

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

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

this in event handler



```
Uncaught TypeError: Cannot set property 'src' of undefined          present.js:13
    at HTMLImageElement._openPresent (present.js:13)
```

Error message!

CodePen / Debug

What's going on?

JavaScript `this`

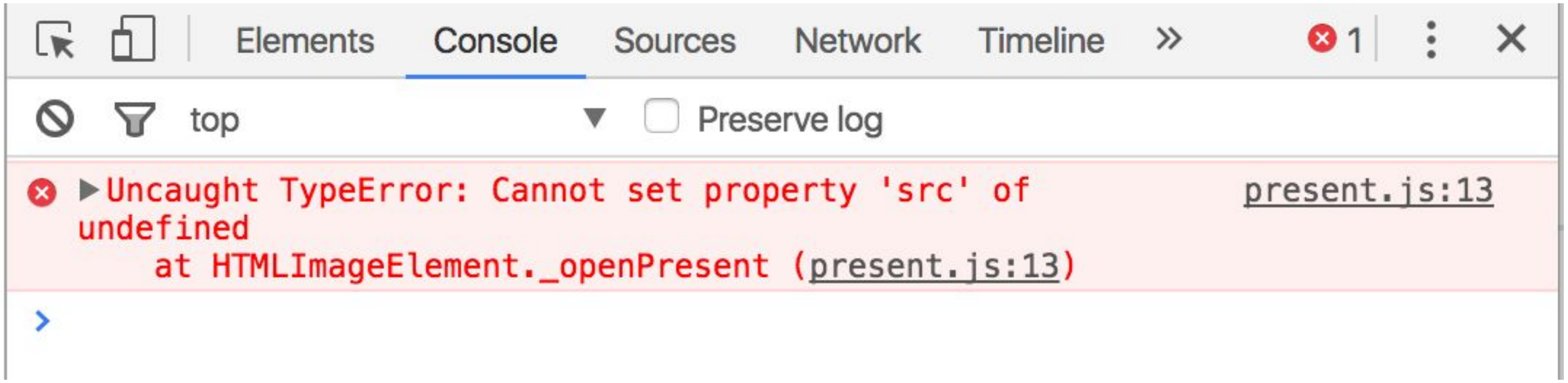
The `this` keyword in JavaScript is **dynamically assigned**, or in other words: `this` means different things in different contexts (mdn list)

- 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...



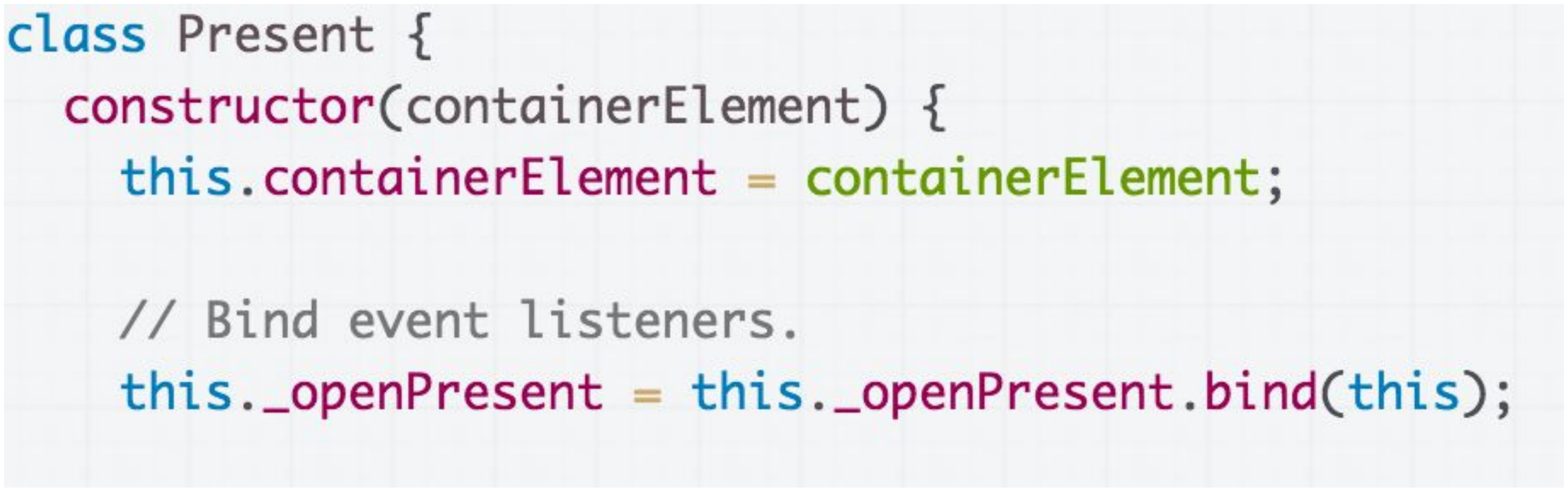
```
Uncaught TypeError: Cannot set property 'src' of          present.js:13
undefined
    at HTMLImageElement._openPresent (present.js:13)
```

...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);
```

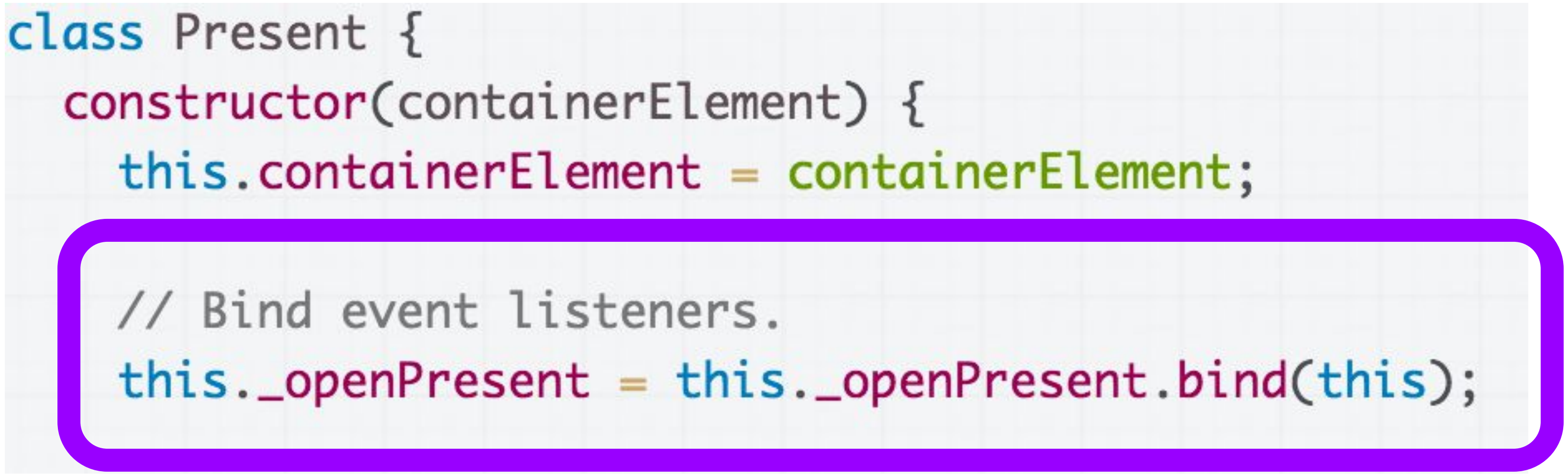
Solution: bind

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);
```

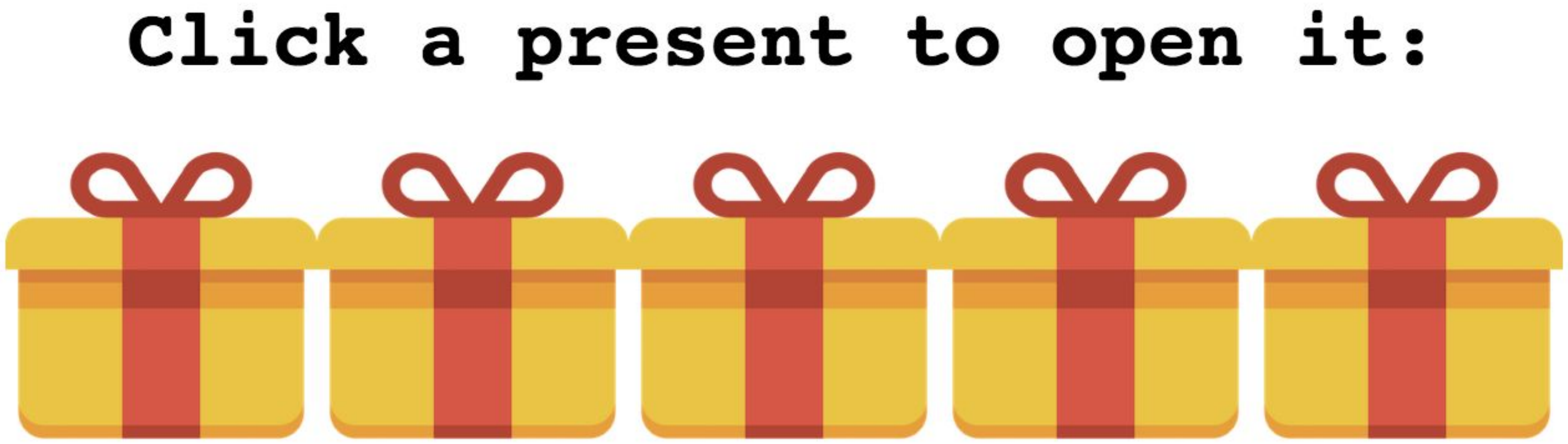
One more time:

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

Communicating between classes

Multiple classes

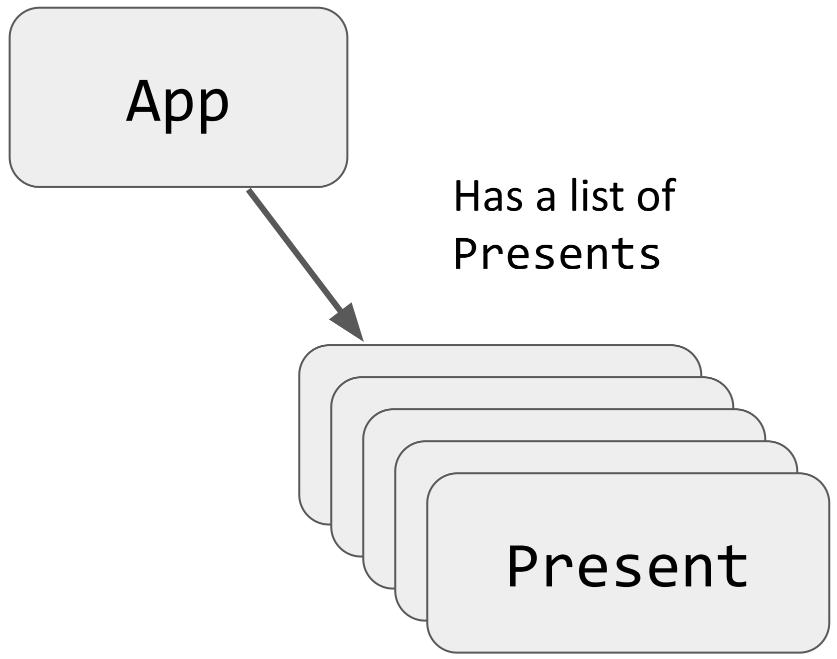
Let's say that we have multiple presents now (CodePen):


Click a present to open it:

Multiple classes

And we have implemented this with two classes:

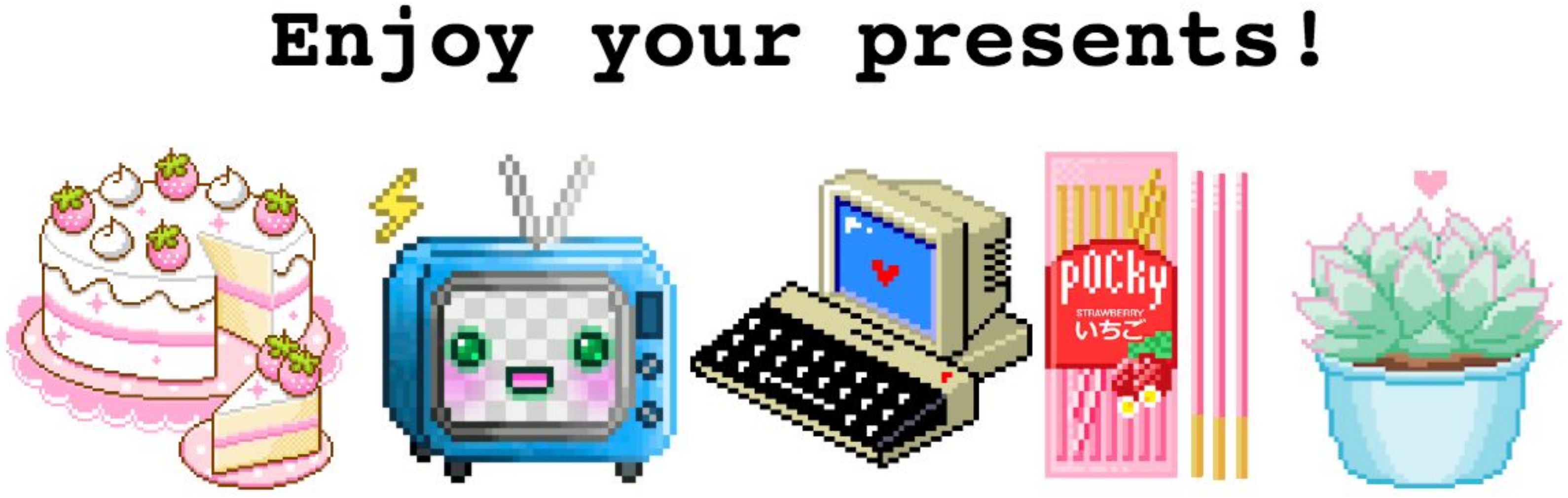
- `App`: Represents the entire page
  - `Present`: Represents a single present



[CodePen]

Communicating btwn classes

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


Enjoy your presents!

Communicating btwn classes

You have three general approaches:

1. Move reference to App, static counter?? to Photo

   DON'T go this route
2. Fire a custom event

   OK (don't forget to bind)
3. Add onOpened "callback function" to Present

   OK (don't foget to bind)

Custom Events

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

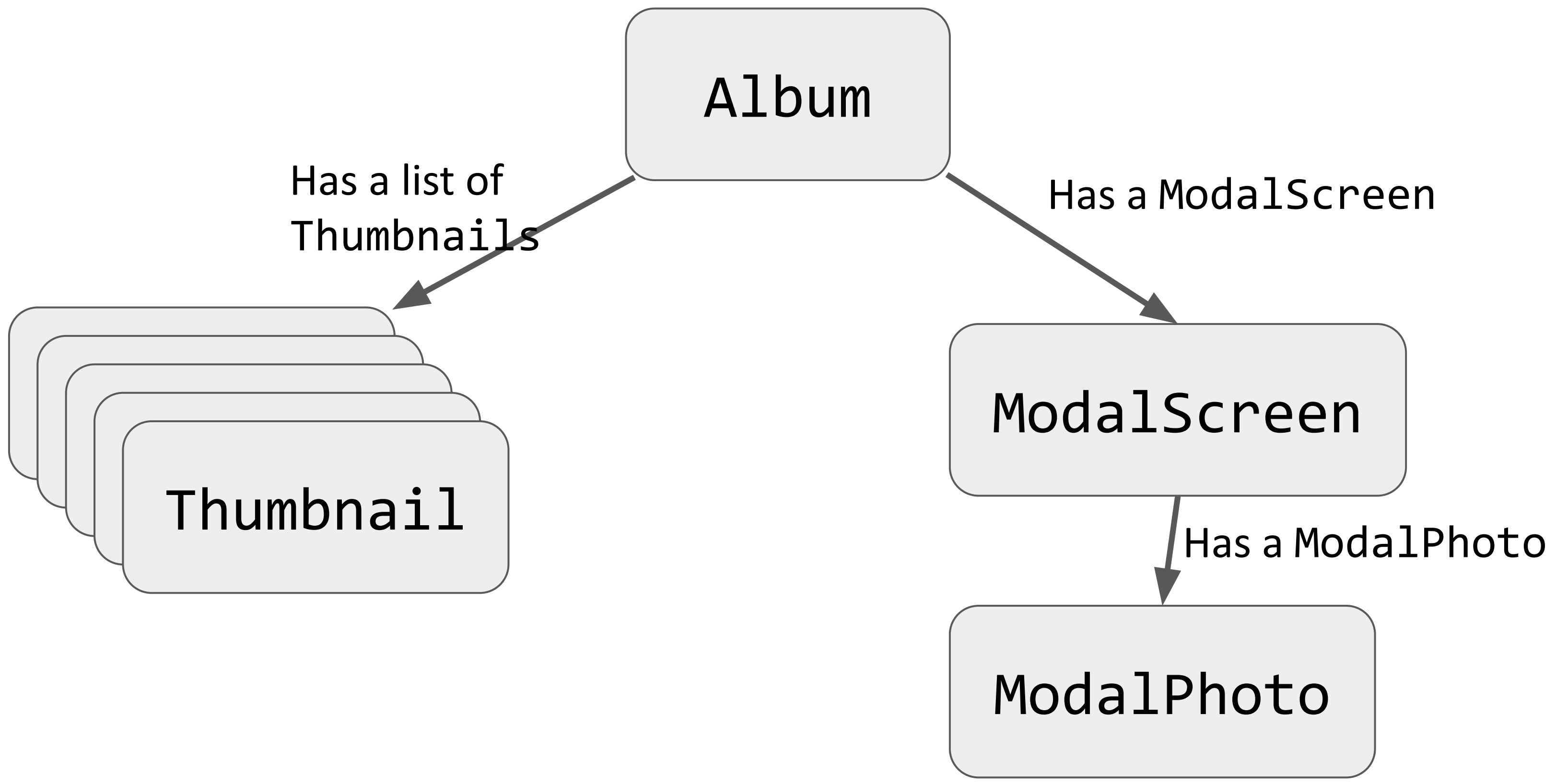
```
const event = new CustomEvent(
    eventNameString, optionalParameterObject);
element.addEventListener(eventNameString);
element.dispatchEvent(eventNameString);
```

CodePen solution

Object-oriented photo album

Let's look at an object-oriented version of the photo album:

CodePen / Debug


Album
Has a list of Thumbnails
Has a ModalScreen
Thumbnail
ModalScreen
Has a ModalPhoto
ModalPhoto