

Web Development Fundamentals

March 2019

Today: Open your mind



Open your mind

The next few lectures are probably the most conceptually difficult in the entire quarter.

We are going to be exploring a few ways in which JavaScript is **very, very different** from other programming languages you know.

We will likely push on your understanding of how programming languages work!



Today's schedule

Today:

- Keyboard events
- Mobile events
- Simple CSS animations
- Functional Programming
- First-class functions
- Loading data from files
 - Fetch API
 - Promises - High-level!
 - JSON
-

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?

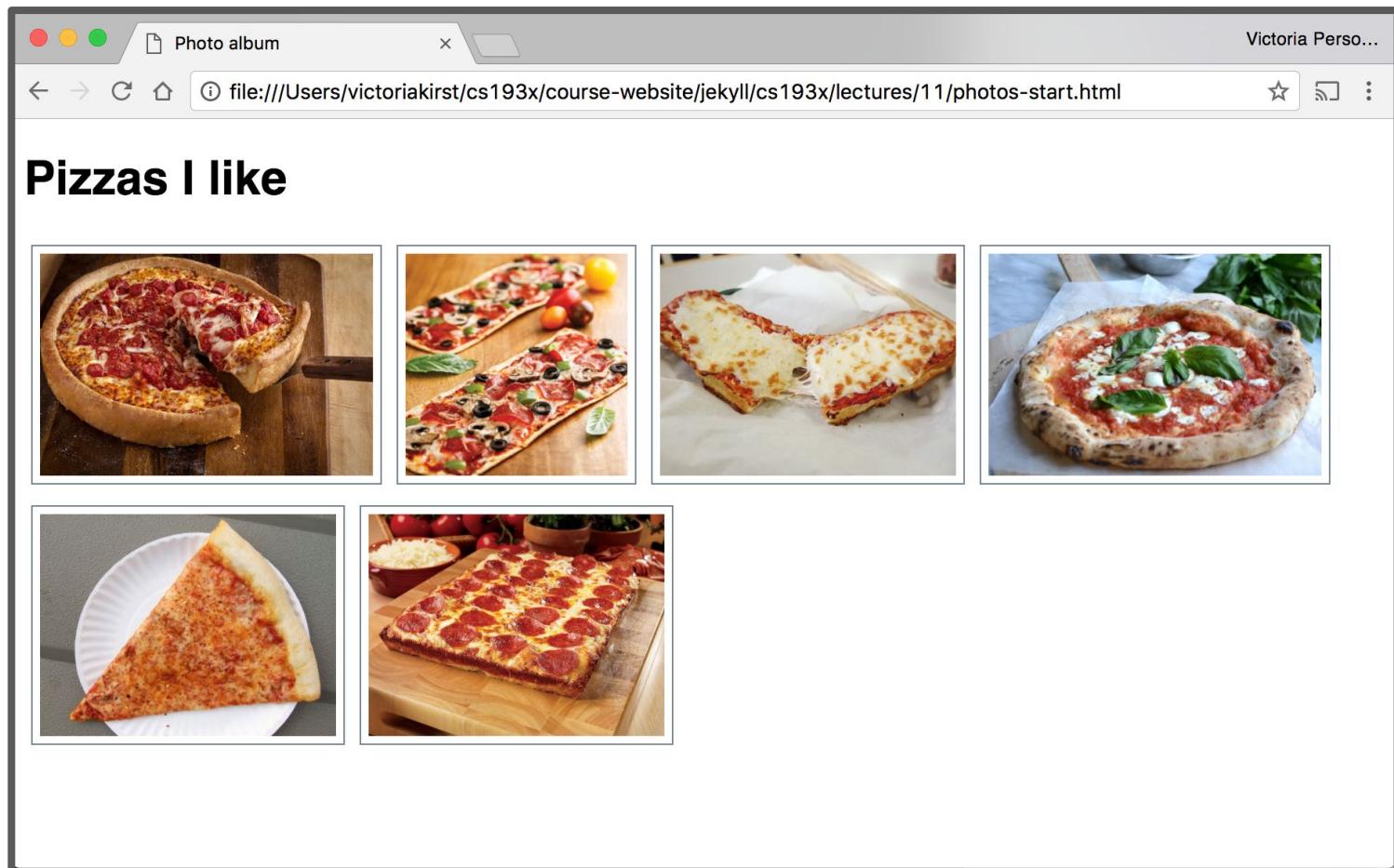
We've been doing a ton of JavaScript examples that involve **click** events...

Aren't there other types of events?

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

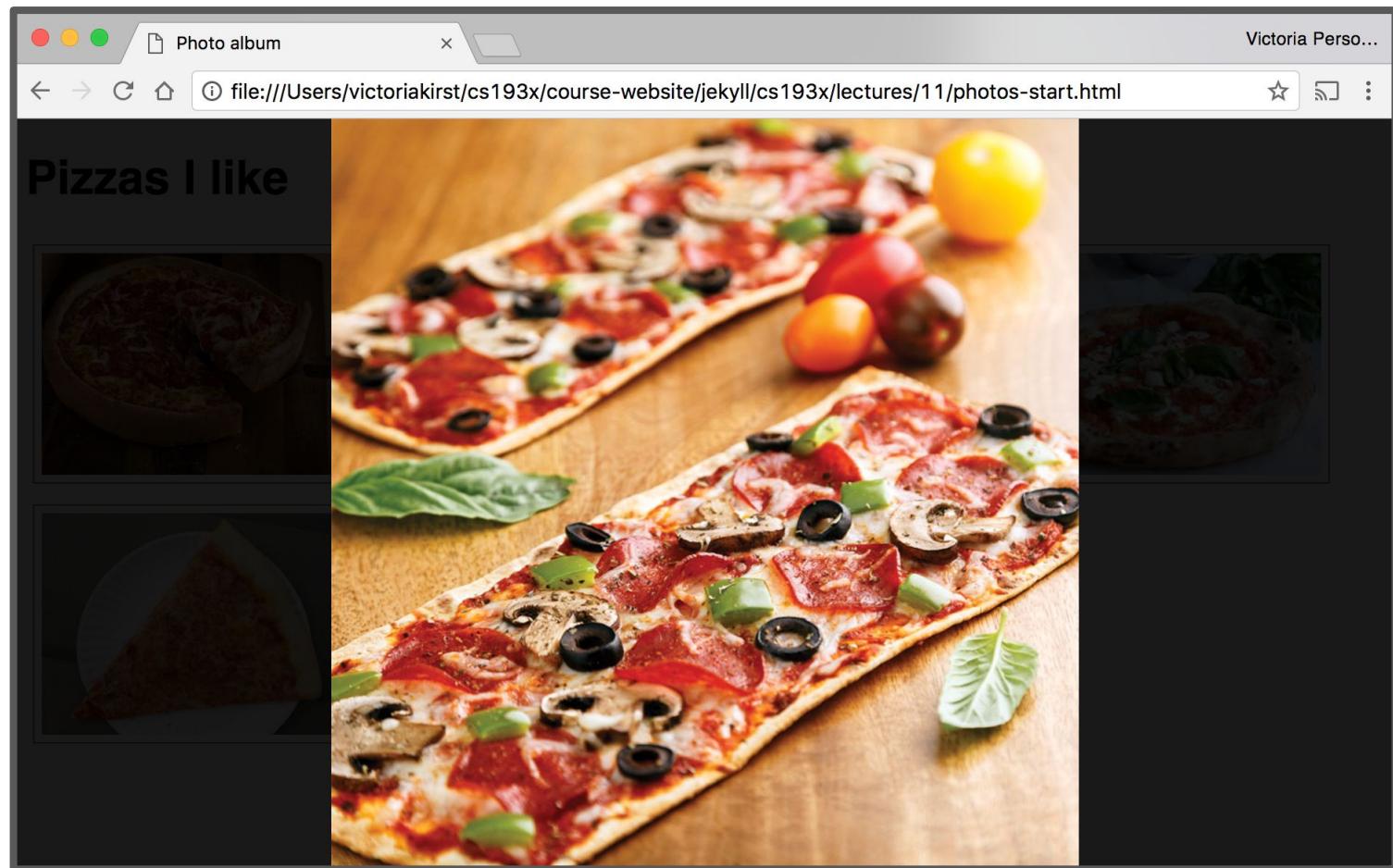
Example: Photo Album

We're going to add a few features to this photo album:



Example: Photo Album

We're going to add a few features to this photo album:



Code walkthrough:

[photo-start.html](#)

[photo.js](#)

[photo.css](#)

General setup

```
<body>
  <h1>Pizzas I like</h1>
  <section id="album-view">
    </section>

    <section id="modal-view" class="hidden">
      </section>
    </body>
```

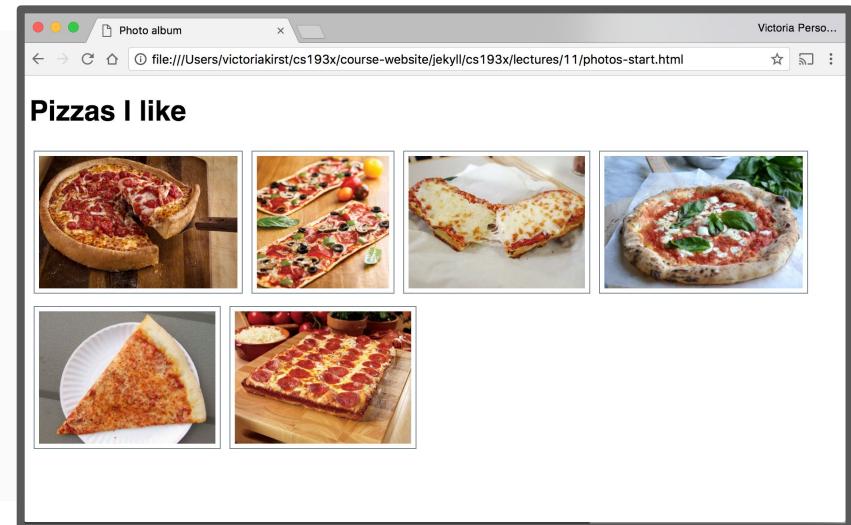
[photo.html](#) contains both "screens":

- The album view: Thumbnails of every photo
- The "[modal](#)" view: A single photo against a semi-transparent black background
 - Hidden by default

CSS: Album

photo.css: The album view CSS is pretty straightforward:

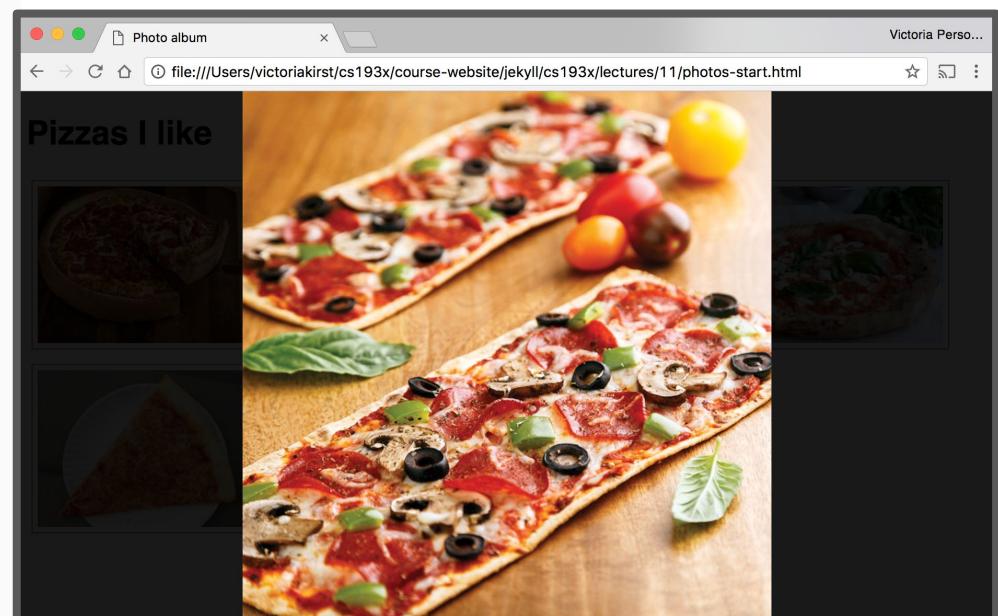
```
#album-view img {  
    border: 1px solid slategray;  
    margin: 5px;  
    padding: 5px;  
    height: 150px;  
}
```



CSS: Modal

Modal view is a little more involved, but all stuff we've learned:

```
#modal-view {  
    position: absolute;  
    top: 0;  
    left: 0;  
    height: 100vh;  
    width: 100vw;  
  
    background-color: rgba(0, 0, 0, 0.9);  
    z-index: 2;  
  
    display: flex;  
    justify-content: center;  
    align-items: center;  
}
```



CSS: Modal image

```
#modal-view img {  
    max-height: 100%;  
    max-width: 100%;  
}
```

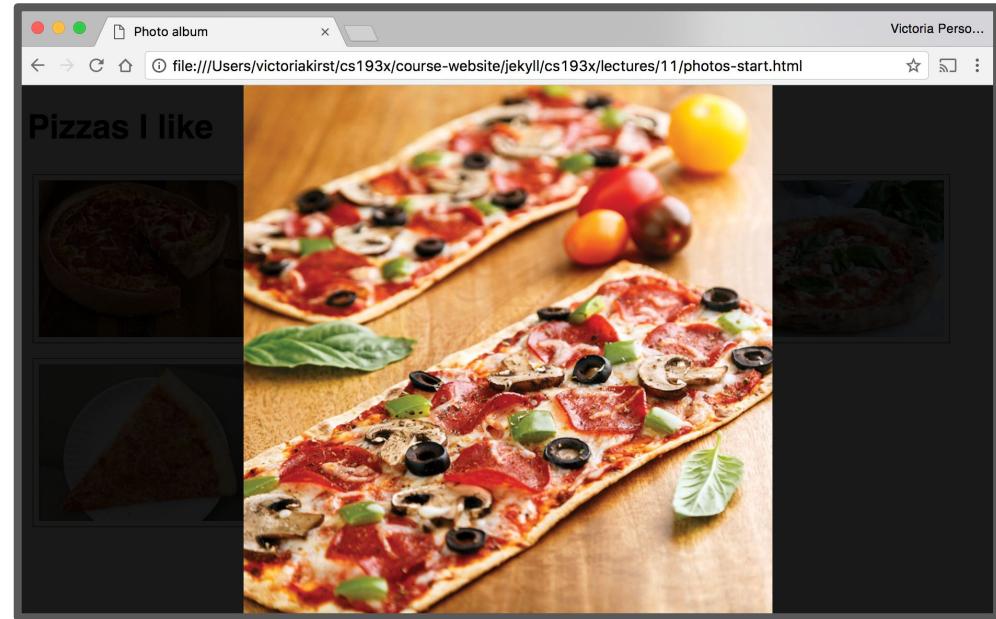


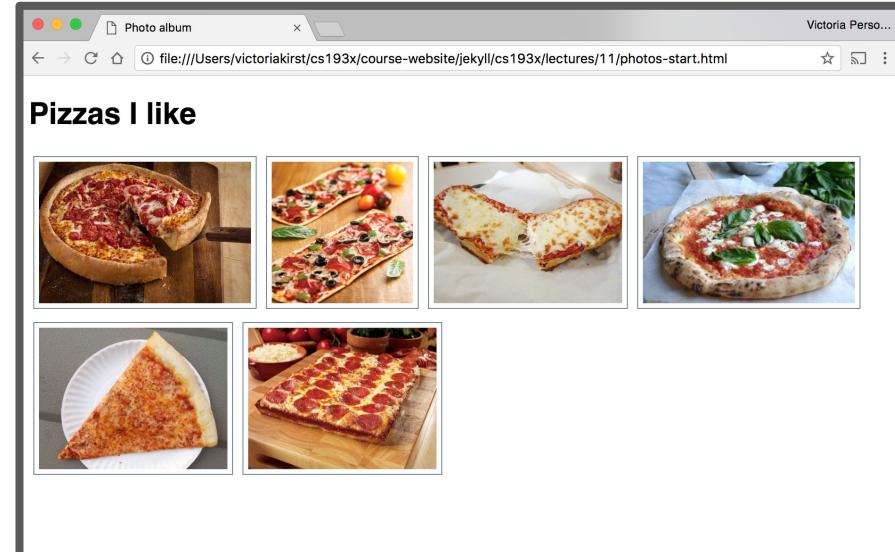
Image sizes are constrained to the height and width of the parent, #modal-view (whose height and width are set to the size of the viewport)

CSS: Hidden modal

```
<body>
  <h1>Pizzas I like</h1>
  <section id="album-view">
    </section>

    <section id="modal-view" class="hidden">
    </section>
</body>
```

```
#modal-view.hidden {
  display: none;
}
```



Even though both the album view and modal view are in the HTML, the model view is set to `display: none;` so it does not show up.

Global List of Photos

```
<head>
  <meta charset="utf-8">
  <title>Photo album</title>
  <link rel="stylesheet" href="css/photo.css">
  <script src="js/photo-list.js" defer></script>
  <script src="js/photo.js" defer></script>
</head>
```

```
const PHOTO_LIST = [
  'images/deepdish.jpg',
  'images/flatbread.jpg',
  'images/frenchbread.jpg',
  'images/neapolitan.jpg',
  'images/nypizza.jpg',
  'images/squarepan.jpg'
];
```

[photo-list.js](#): There is a global array with the list of string photo sources called PHOTO_LIST.

Photo thumbnails

```
function createImage(src) {  
  const image = document.createElement('img');  
  image.src = src;  
  return image;  
}
```

```
const albumView = document.querySelector('#album-view');  
for (let i = 0; i < PHOTO_LIST.length; i++) {  
  const photoSrc = PHOTO_LIST[i];  
  const image = createImage(photoSrc);  
  image.addEventListener('click', onThumbnailClick);  
  albumView.appendChild(image);  
}
```

[photo.js](#): We populate the initial album view by looping over PHOTO_LIST and appending s to the #album-view.

Clicking a photo

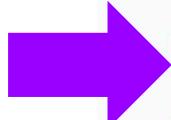
```
function onThumbnailClick(event) {  
  const image = createImage(event.currentTarget.src);  
  modalView.appendChild(image);  
  modalView.classList.remove('hidden');  
}
```

When the user clicks a thumbnail:

- We create another tag with the same src
- We append this new to the #modal-view
- We unhide the #modal-view

Positioning the modal

```
function onThumbnailClick(event) {  
  const image = createImage(event.currentTarget.src);  
  modalView.style.top = window.pageYOffset + 'px';  
  modalView.appendChild(image);  
  modalView.classList.remove('hidden');  
}
```



We'll add another line of JavaScript to anchor our modal dialog to the top of the viewport, not the top of the screen:

```
modalView.style.top = window.pageYOffset + 'px';
```

(See [window.pageYOffset mdn](#). It is the same as [window.scrollY](#).)

Aside: style attribute

Every HTMLElement has a style attribute that lets you set a style directly on the element:

```
element.style.top = window.pageYOffset + 'px';
```

Generally **you should not use the style property**, as adding and removing classes via classList is a better way to change the style of an element via JavaScript

But when we are setting a CSS property based on JavaScript values, we must set the **style** attribute directly.

No scroll on page

```
function onThumbnailClick(event) {  
    const image = createImage(event.currentTarget.src);  
    document.body.classList.add('no-scroll');  
    modalView.style.top = window.pageYOffset + 'px';  
    modalView.appendChild(image);  
    modalView.classList.remove('hidden');  
}  
  
.no-scroll {  
    overflow: hidden;  
}
```

And we'll also set `body { overflow: hidden; }` as a way to disable scroll on the page.

Closing the modal dialog

```
function onModalClick() {  
    document.body.classList.remove('no-scroll');  
    modalView.classList.add('hidden');  
    modalView.innerHTML = '';  
}
```

```
const modalView = document.querySelector('#modal-view');  
modalView.addEventListener('click', onModalClick);
```

When the user clicks the modal view:

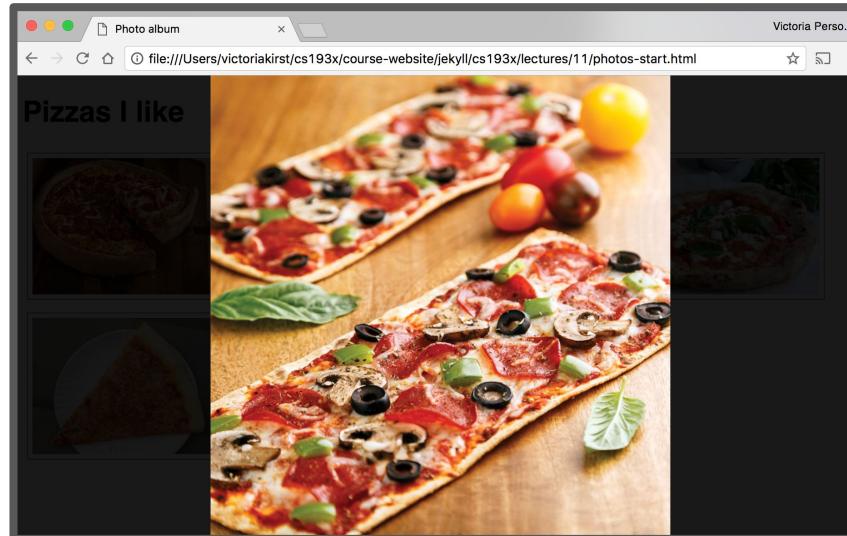
- We hide the modal view again
- We enable scroll on the page again
- We clear the image we appended to it by setting
`innerHTML = '';`

Adding keyboard navigation

Navigating photos

Let's add some keyboard events to navigate between photos in the Modal View:

- Left arrow: Show the " $i - 1$ "th picture
- Right arrow: Show the " $i + 1$ "th picture
- Escape key: Close dialog



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	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](#)

Let's finish the feature!

Finished result:
[photo-desktop-finished.html](#)

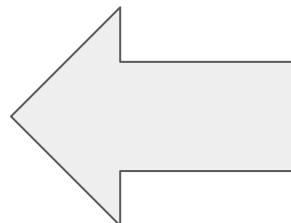
Mobile?

Keyboard events work well on desktop, but keyboard navigation doesn't work well for mobile.

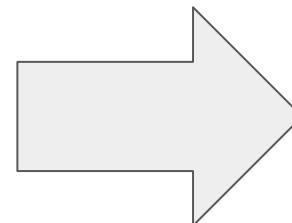


On your phone, you can usually navigate photo albums using **gestures**:

- **Left swipe** reveals the next photo
- **Right swipe** reveals the previous photo



Next



Previous

Mobile?

Keyboard events work well on desktop, but keyboard navigation doesn't work well for mobile.



On your phone, you can usually navigate photo albums using **gestures**:

- **Left swipe** reveals the next photo
- **Right swipe** reveals the previous photo

How do we implement the swipe gesture on the web?

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.

To do this, it's helpful to learn about a few more JS events:

- MouseEvent
- TouchEvent
- PointerEvent

MouseEvent

Event name	Description
<code>click</code>	Fired when you click and release (mdn)
<code>mousedown</code>	Fired when you click down (mdn)
<code>mouseup</code>	Fired when you release from clicking (mdn)
<code>mousemove</code>	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](#))

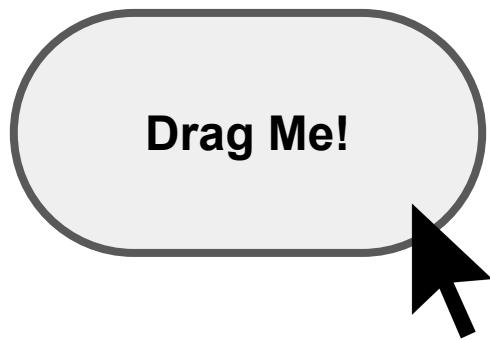
clientX and clientY

```
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

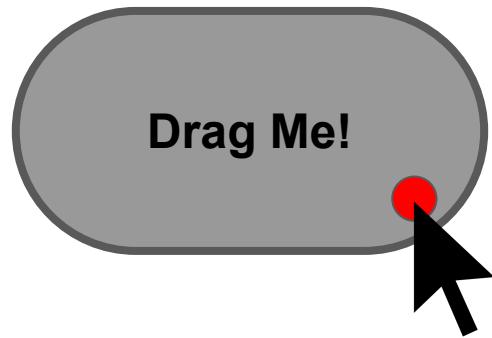
Implementing drag



When a user clicks down/touches
an element...

Implementing drag

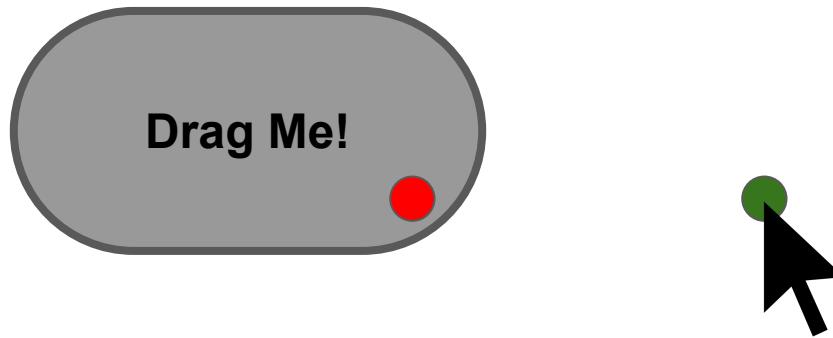
```
originX = 100;
```



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 add 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.

Moving an element

We are going to use the transform CSS property to move the element we are dragging from its original position:

```
originX = 100;  
newX = 150;  
delta = newX - originX;
```



```
element.style.transform = 'translateX(' + delta + 'px)';
```

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(<i>deg</i>)	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")

Finally, let's code!

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();
```

setPointerCapture()

To listen to pointer events that occur when the pointer goes offscreen, call [setPointerCapture](#) on the target you want to keep tracking:

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

style attribute

Every HTMLElement also has a style attribute that lets you set a style directly on the element:

```
element.style.transform =  
  'translateX(' + value + ')';
```

Generally **you should not use the style property**, as adding and removing classes via classList is a better way to change the style of an element via JavaScript

But when we are dynamically calculating the value of a CSS property, we have to use the style attribute.

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

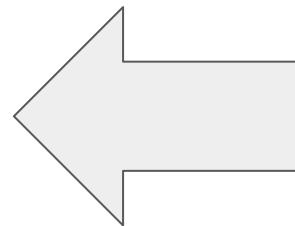
However, using `requestAnimationFrame` well requires us to know a little bit more about the JavaScript event loop. Functional programming also helps. We'll get there next week!)

CSS animations

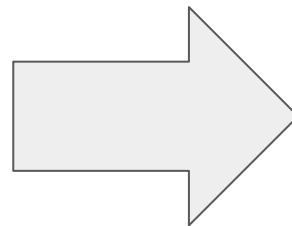
Softening the edges

Our photo album feels a little jerky still. 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 syntax

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

Examples

Then set the following CSS property:

animation: *animation-name duration*;

Easier example: Fade in

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

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

Finished result:
[photo-mobile-finished.html](#)

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:

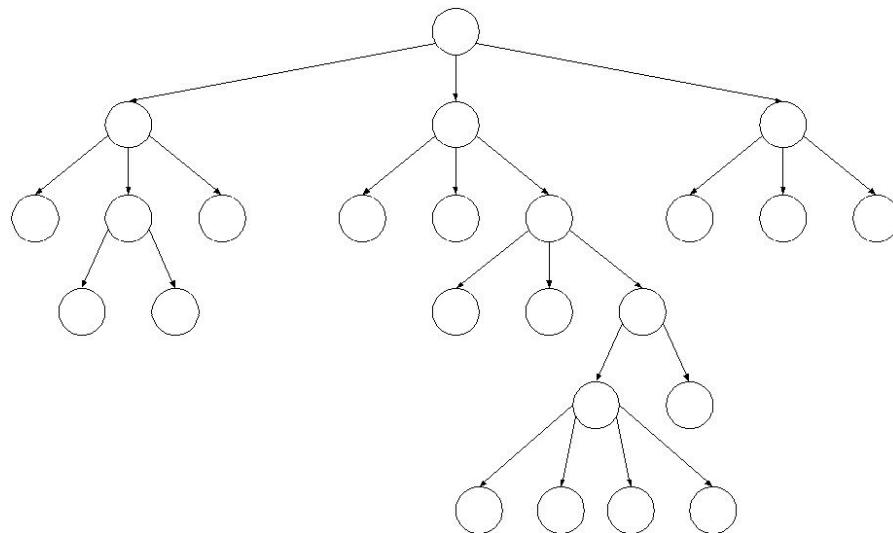
```
document.dispatchEvent(  
  new CustomEvent('button-click'));
```

Understanding the DOM

DOM Nodes

If the DOM is a tree composed of Nodes...

Q: Does that mean a Node in the DOM has child pointers like the trees we learned about in 106B?

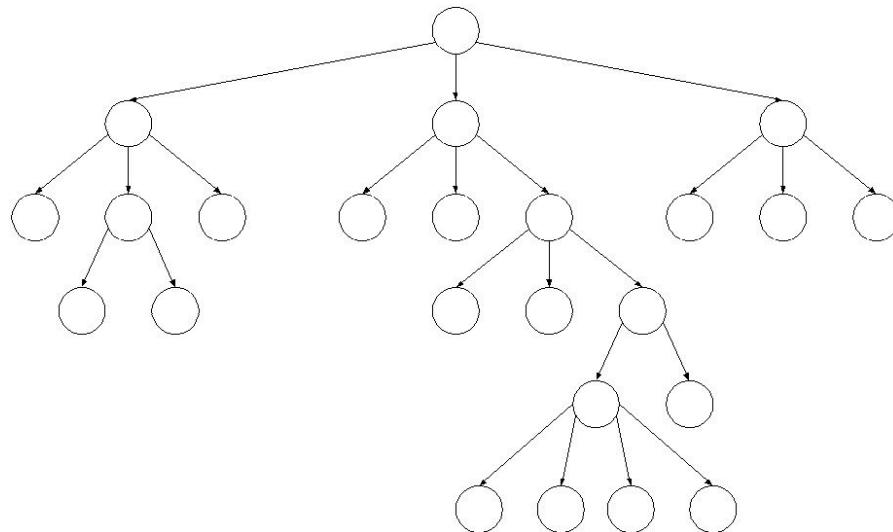


DOM Nodes

If the DOM is a tree composed of Nodes...

Q: Does that mean a Node in the DOM has child pointers like the trees we learned about in 106B?

A: Yes!



Node properties

Property	Description
<u>textContent</u>	The text content of a node and its descendants. (This property is writeable)
<u>childNodes</u>	An array of this node's children (empty if a leaf)
<u>parentNode</u>	A reference to this node's parent Node

```
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```

What's the **parentNode** of
<section>?

parentNode

```
> section = document.querySelector('section');
<- ►<section>...</section>
> section.parentNode
<- ►<body>...</body>
```

```
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```

The **parentNode** of
<section> is **<body>**.

What are the **childNodes**
of **<section>**?

childNodes

```
> section = document.querySelector('section');
< ◀<section>...</section>
> section.childNodes
< ◀ [text, p, text, p, text]
> section.childNodes.length
< 5
```

???

```
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```

Why does **section** have 5 children, not 2?!

TextNode

In addition to [Element](#) nodes, the DOM also contains [Text](#) nodes. All text present in the HTML, **including whitespace**, is contained in a text node:

```
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```

TextNode

All text present in the HTML, **including whitespace**, is contained in a Text node:

```
<body>
  <h1>My favorites</h1>
  <section>
    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```

DOM and Text nodes

The DOM is composed of [Nodes](#), and there are several subtypes of [Node](#).

- [Element](#): HTML (or SVG) elements in the DOM
- [Text](#): Text content in the DOM, including whitespace
 - [Text](#) nodes cannot contain children (are always leafs)
- [Comment](#): HTML comments
- ([more](#) / [DOM visualizer](#))

The type of a node is stored in the [nodeType](#) property

Traversing the DOM

Q: How would we print out all nodes in the DOM?

Traversing the DOM

Q: How would we print out all nodes in the DOM?

A: Recursively walk the DOM tree:

```
function walkTree(root, level) {  
  if (root.nodeType == Node.TEXT_NODE) {  
    console.log(level + 'text:' + root.textContent);  
  } else {  
    console.log(level + root.nodeName);  
  }  
  for (const child of root.childNodes) {  
    walkTree(child, level + "    ");  
  }  
}  
walkTree(document.querySelector('html'), "");
```

What's the point?

- If we have `document.querySelector` that lets us get elements in the DOM...
- And if we can change the HTML as necessary to add classes/ids/elements/etc to select the right things...

Q: When would we ever want to traverse the DOM?

What's the point?

- If we have `document.querySelector` that lets us get elements in the DOM...
- And if we can change the HTML as necessary to add classes/ids/elements/etc to select the right things...

Q: When would we ever want to traverse the DOM?

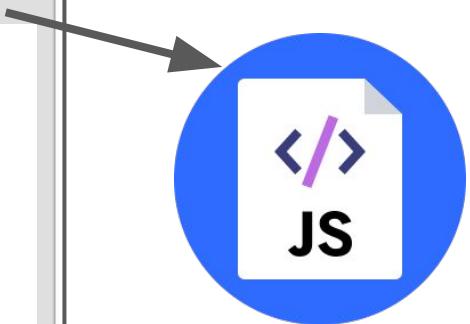
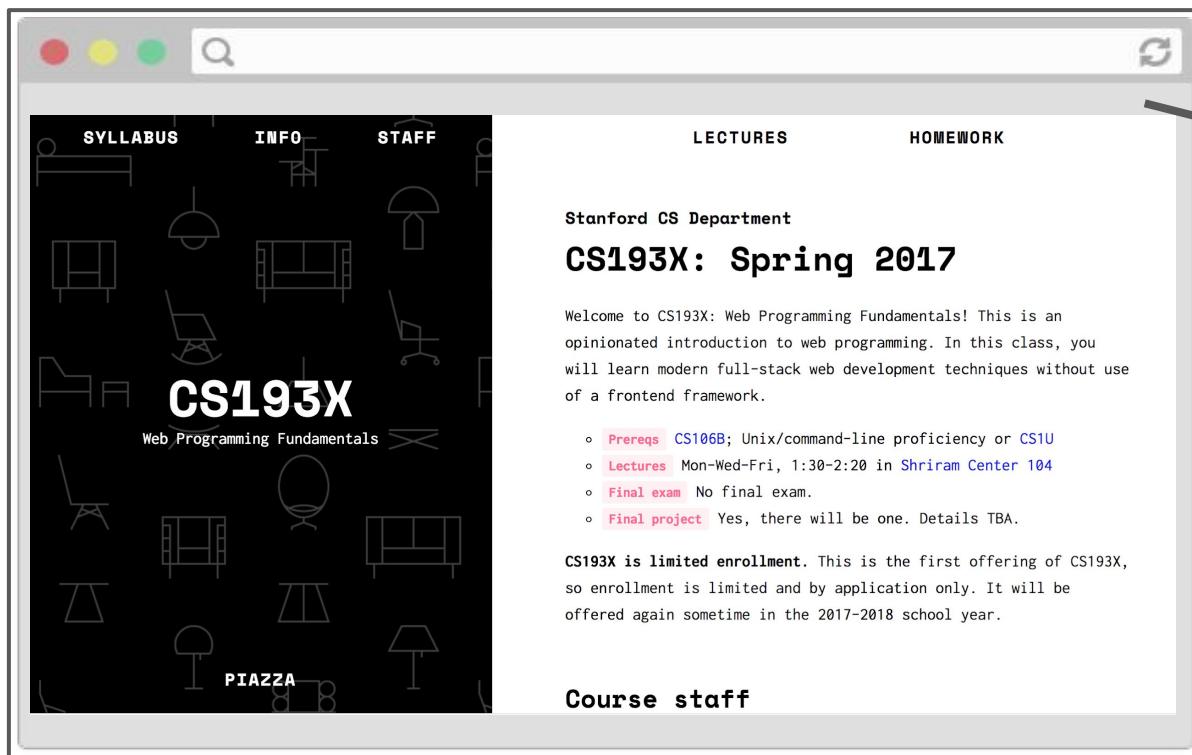
A: Pretty much only in browser extensions

or the Web Console

(i.e. manipulating someone else's page)

Browser extensions

- Add-on that extends the functionality of the browser
- A piece of JavaScript that is injected into the webpage before or after it has loaded



Content scripts

- A type of extension that runs in the context of a web browser, meaning it can access the DOM of the page on which it's loaded
 - Concept is the same in [Chrome](#), [Firefox](#), probably other browsers

Usually composed of:

- `manifest.json`: Contains title, settings, etc for the script
- `page.js`: The extension file

We are using a **content script** in HW2, Part 2

Example manifest.json

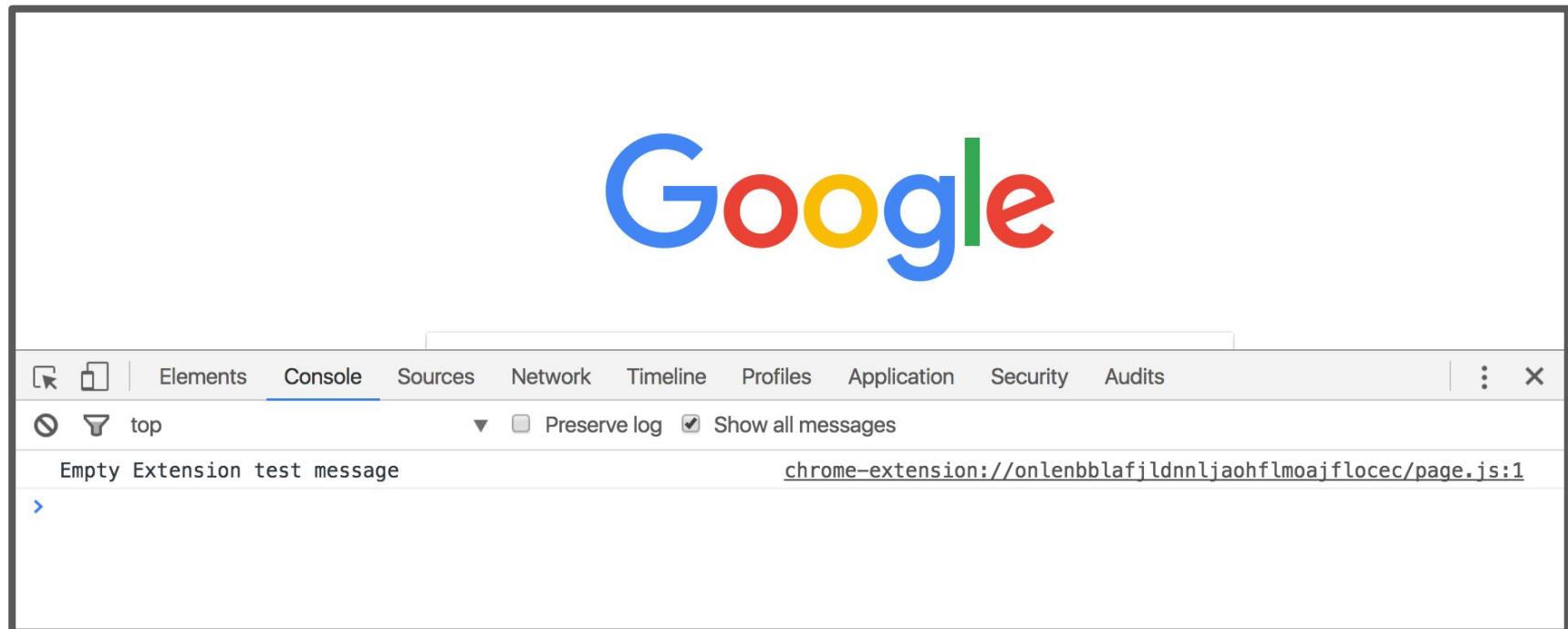
```
{  
  "name": "Empty Chrome Extension",  
  "version": "1.0",  
  "description": "",  
  "content_scripts": [  
    {  
      "matches": ["<all_urls>"],  
      "js": ["page.js"]  
    }  
  ],  
  "icons": {  
    "16": "pizza.png",  
    "48": "pizza.png",  
    "128": "pizza.png"  
  },  
  "manifest_version": 2  
}
```

Example page.js

```
page.js  
1 console.log('Empty Extension test message');  
2
```

Result

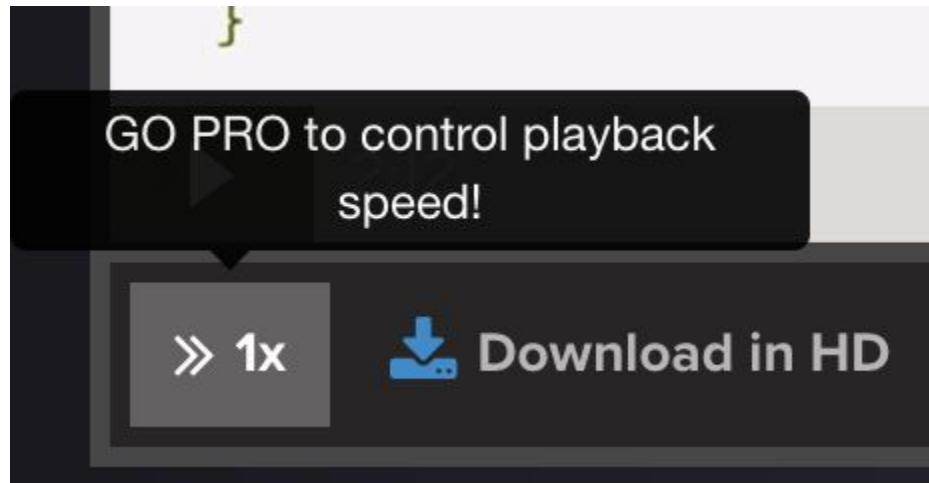
When this extension is loaded, the console message appears in the Web Inspector for every page:



Hacks and Mischief

Example: Egghead

[Dan Abramov's Redux videos](#)



\$ **199** 99
PER YEAR

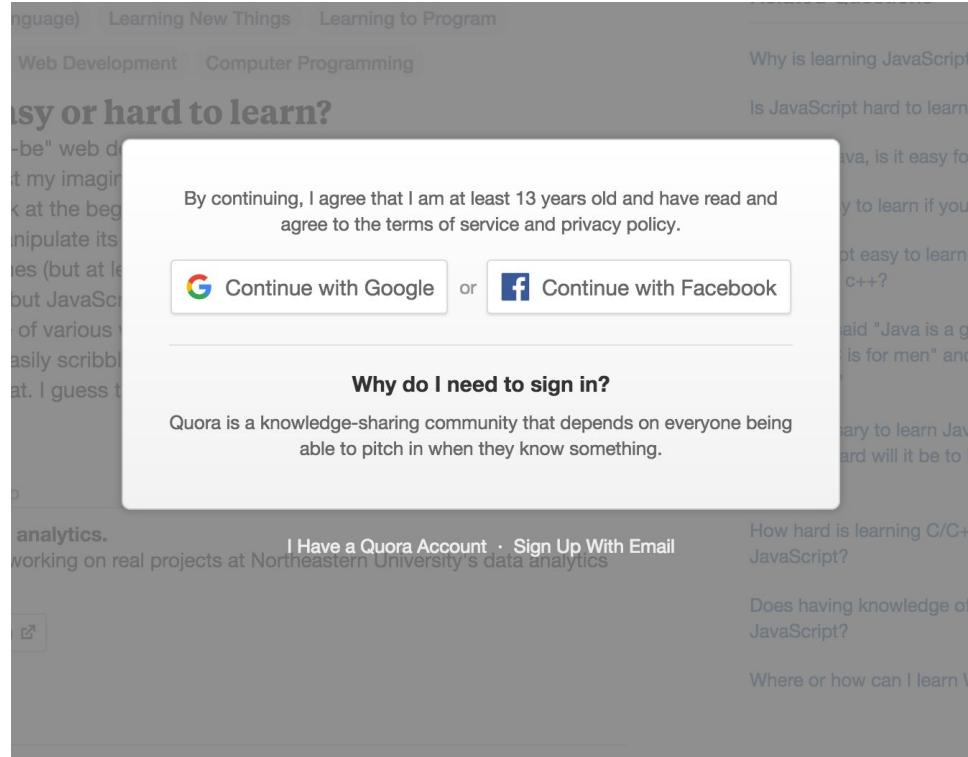
Example: Egghead

```
> document.querySelector('video')
< ><video id="wistia_simple_video_25" crossorigin="anonymous" preload="me
  "background: transparent; display: block; height: 100%; position: static
> document.querySelector('video').playbackRate = 2
< 2
> document.querySelector('video').playbackRate = 1.5
< 1.5
>
```

We can access the DOM via `querySelector` in the Web Inspector. Useful for debugging, and for speeding up videos.
([MDN for `playbackRate`](#))

Example: Quora signin wall

<https://www.quora.com/Why-is-learning-JavaScript-so-hard>



Learning JavaScript Beginning Computer Programming Advice

JavaScript (programming language) Learning New Things Learn

Programming Languages Web Development Computer Program

By continuing, I agree that I am at least 13 years old and have read and
agree to the terms of service and privacy policy.

 Continue with Google

or  Continue with Facebook

Why do I need to sign in?

Quora is a knowledge-sharing community that depends on everyone being
able to pitch in when they know something.

Sponsored by Vetter

I Have a Quora Account · Sign Up With Email

New dev job, no hassle.

The job search process is awful. Vetter makes it better. Apply to 1

Sign Up at vetter.com ↗

18 Answers

```
<!DOCTYPE html>
<html lang="en" class="js-wf-loaded">
  <head>...
<body class="web_page logged_out lang_en gating-feed_desktop_modal_skinny-off feed_desktop_modal_skinny_off signup_wall_prevent_scroll" style="padding-right: 0px;"> ==
  <script charset="utf-8" src="https://tch512495.tch.quora.com/up/chan31-8888/updates?min_seq=629585...">
  <script type="text/javascript" id="settings_js">
  </script>
  <div class="InteractionModeBanner fade_out" id="__w2_mJbghVi_banner" style="display: none;">...</div>
  <div class="ErrorBanner" id="__w2_aezubdt_banner">...</div>
  <div id="__w2_modal_container_">...</div>
  <div class="content_page_feed_offset">...</div>
  <div class="SimpleToggle LoggedOutContentPageFeed ToggleContentPageFeed hidden" id="__w2_P6plUEA_truncated">...</div>
  <div class="SimpleToggle LoggedOutContentPageFeed ToggleContentPageFeed" id="__w2_P6plUEA_expanded">...</div>
  <div class="LargeFooter">...</div>
  <div id="WNzSrD">...</div>
<div id="__w2_AJH0u6H_signup_wall_wrapper">
  <div class="vertical_alignment_wrapper" id="__w2_KDwkuQ8_background_wrapper">
    <div class="modal_signup_background new_web_signup_wall_design" id="__w2_KDwkuQ8_background">...</div>
  <div class="vertical_alignment_wrapper">
    <div class="dialog modal_signup_dialog" id="__w2_KDwkuQ8_outer_form">...</div>
  </div>
</div>
...
<body> ...
  <div>...
    <div>...
      <div>...
        <div>...
          <div>...
            <div>...
              <div>...
                <div>...
                  <div>...
                    <div>...
                      <div>...
                        <div>...
                          <div>...
                            <div>...
                              <div>...
                                <div>...
                                  <div>...
                                    <div>...
                                      <div>...
                                        <div>...
                                          <div>...
                                            <div>...
                                              <div>...
                                                <div>...
                                                  <div>...
                                                    <div>...
                                                      <div>...
                                                        <div>...
                                                          <div>...
                                                            <div>...
                                                              <div>...
                                                                <div>...
                                                                  <div>...
                                                                    <div>...
                                                                      <div>...
                                                                        <div>...
                                                                          <div>...
                                                                            <div>...
                                                                              <div>...
                                                                                <div>...
                                                                                  <div>...
                                                                                    <div>...
                                                                                      <div>...
...
```

```
><div class="LargeFooter">...</div>
<div id="WNzSrD"></div>
... <div id="__w2_AJH0u6H_signup_wall_wrapper" style="display: none; "> == $0
| <div class="vertical_alignment_wrapper" id="__w2_KDwku08_background_wrapper">
```

Filter

```
element.style {
    display: none;
}
div {
    display: block;
```



Search for questions, people, and topics

Sign In

Learning JavaScript Beginning Computer Programming Advice

JavaScript (programming language) Learning New Things Learning to Program

Programming Languages Web Development Computer Programming

Is JavaScript easy or hard to learn?

I'm a 15 year old "want-to-be" web developer and I have dreams of using a plethora of web languages to manifest my imagination onto the Internet. They're big dreams for me, but unfortunately I'm stuck at the beginning mulling over the semantic structure of JavaScript and how to manipulate its use for my whim. I've found it at sometimes frustrating and at sometimes (but at less times) absolutely magical. I've gotten over HTML quickly, and CSS as well; but JavaScript is by far obviously the hardest of all three. I have looked at the source code of various websites and become boggled at how developers for that specific website easily scribble down all that code. I tell myself sometimes that I will never be able to do that. I guess the essence of my question is: Is JavaScript easy or hard to learn?

Sponsored by Vetter

New dev job, no hassle.

The job search process is awful. Vetter makes it better. Apply to 1,000+ top tech startups now.

[Sign Up at vetter.com](#)

18 Answers



Clay Murphy IS Engineer@Google (no longer)

Related Questions

[Why is learning JavaScript so hard?](#)

[Is JavaScript hard to learn?](#)

[Is Javascript easy to learn when I already can program in c++?](#)

[If I know Java, is it easy for me to learn JavaScript?](#)

[Is PHP easy to learn if you're used to JavaScript?](#)

[My friend said "Java is a girl's programming language and C++/C is for men" and I feel offended. How can I get over it?](#)

[Is it necessary to learn JavaScript if I know Ruby? And how hard will it be to learn JavaScript if I know Ruby?](#)

[How hard is learning C/C++ considering you know JavaScript?](#)

[What is the best approach to learn Hybris?](#)

[Does having knowledge of Java make it easy to learn JavaScript?](#)

Quora Chrome Extension

You can make a Chrome extension to automate this:

manifest.json

```
"matches": ["*://www.quora.com/*"],  
"js": ["page.js"]
```

page.js

```
document.body.classList.remove('signup_wall_prevent_scroll');  
const nagScreen = document.querySelector('.vertical_alignment_wrapper');  
if (nagScreen) {  
  nagScreen.style.display = 'none';  
}
```

Aside: style attribute

Every [HTML Element](#) also has a [style](#) attribute that lets you set a style directly on the element:

```
nagScreen.style.display = 'none';
```

Generally **you should not use this property**, as adding and removing classes via [classList](#) is a better way to change the style of an element via JavaScript

(But for extensions/hacking, [style](#) can be useful!)

Example: Adblock block

<http://www.on-demandkorea.com/kpop-star-season-6-seoul-qualifier.html>

The screenshot shows a streaming platform's homepage. At the top, there is a navigation bar with categories: Pay-Per-View, News, Drama, Variety (which is highlighted in green), Documentary, Life, Kids, Sports, Education, and Religion. Below the navigation bar is a promotional image featuring a young woman with long brown hair smiling. In the top left corner of this image, there is handwritten Korean text: '유인아' (Yoo In-ah) above 'Layit'. Overlaid on the bottom left of the promotional image is a large red octagonal sign with a white hand icon, indicating that the content is blocked by an ad-blocker. To the right of this icon, the text 'Are you using AdBlock?' is displayed in bold. Below this, a message reads: 'OnDemandKorea provides free legal streaming service to our users through ad revenue. Instead of using AdBlock, sign up for ODK PLUS membership to watch all contents ad-free!' At the bottom of the page, there is a green button with the text 'Sign up for ODK PLUS to stream contents ad-free' and a link 'Need help disabling AdBlock?'

View-source, search for "adblock"...

```
1132    });
1133
1134    var view_seconds = 0;
1135    var last_watched_position = 0;
1136    var view_timer = setInterval(function() {
1137        if(jwplayer().getState() == 'playing')
1138        {
1139            if(jwplayer().getPosition() != last_watched_position)
1140            {
1141                view_seconds += 1;
1142                if(view_seconds > 0 && view_seconds%300 == 0)
1143                {
1144                    $p("send","watched_5_min");
1145                    ga('set', 'dimension5', 'sbs_sub');
1146                    ga('send', 'event', 'Video/variety', 'ViewhourLog', 'kpop-star-season-6-e20161120', 300);
1147                }
1148            }
1149        }
1150        last_watched_position = jwplayer().getPosition();
1151    }, 1000);
1152
1153        jwplayer().on('adBlock',function(event) {
1154            jwplayer().remove();
1155            $("#odk_player").html('<a id="adblock_signup_in_player" href="https://www.ondemandkorea.com/odk-plus-benefits" target="_blank" style="position: absolute; margin-top: 291px; margin-left: 262px; width: 406px; height: 38px;"></a><a href="http://www.ondemandkorea.com/blog/?p=2085" target="_blank" style="position: absolute; margin-top: 341px; margin-left: 497px; width: 172px; height: 20px;"></a>');
1156            });
1157            if(offset != false)
1158            {
1159                jwplayer().on('firstFrame',function(){
1160                    jwplayer().seek(offset);
1161                });
1162            }
1163            var overlay_shown = true;
1164            var overlayShowingNow = false;
1165            var ad_playing = false;
1166
1167            jwplayer("odk_player").on('time',function(e){
1168                if(overlay_shown == false && ad_playing == false)
1169                {
1170                    if(overlayShowingNow == false && (e.position > 0.5 && e.position < 4.5))
1171                    {
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2597
2598
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2697
2698
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2797
2798
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2897
2898
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2997
2998
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3097
3098
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3197
3198
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3288
3289
3289
3290
3291
3292
3293
3294
3295
3296
3297
3297
3298
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
332
```

Lol global scope

```
jwplayer().on('adBlock', function(event) {  
    jwplayer().remove();  
    $('#odk_player').html('<a id="adblock_signup_in_player"  
href="https://www.on-demandkorea.com/odk-plus-benefits" target="_blank"  
style="position: absolute; margin-top: 291px; margin-left: 262px; width:  
406px; height: 38px;"></a><a  
href="http://www.on-demandkorea.com/blog/?p=2085" target="_blank"  
style="position: absolute; margin-top: 341px; margin-left: 497px; width:  
172px; height: 20px;"></a>' );  
});
```

Note that this is using the [jQuery library](#), which I **do not** recommend you use

jQuery on?

Again, I **do not** recommend you use jQuery. But if we're trying to understand someone else's jQuery code...

.on()

Categories: [Events](#) > [Event Handler Attachment](#)

.on(events [, selector] [, data], handler)

Description: *Attach an event handler function for one or more events to the selected elements.*

We see that .on() is essentially addEventListener

jQuery off

.off()

Categories: [Events](#) > [Event Handler Attachment](#)

.off(events [, selector] [, handler])

Description: Remove an event handler.

And .off() is essentially removeEventListener

Lol global scope

What if we try this...

page.js

```
1 const child = document.createElement('script');
2 child.textContent += 'jwplayer().off(\'adBlock\');");
3 document.body.appendChild(child);
4
```

Aside: inline <script>

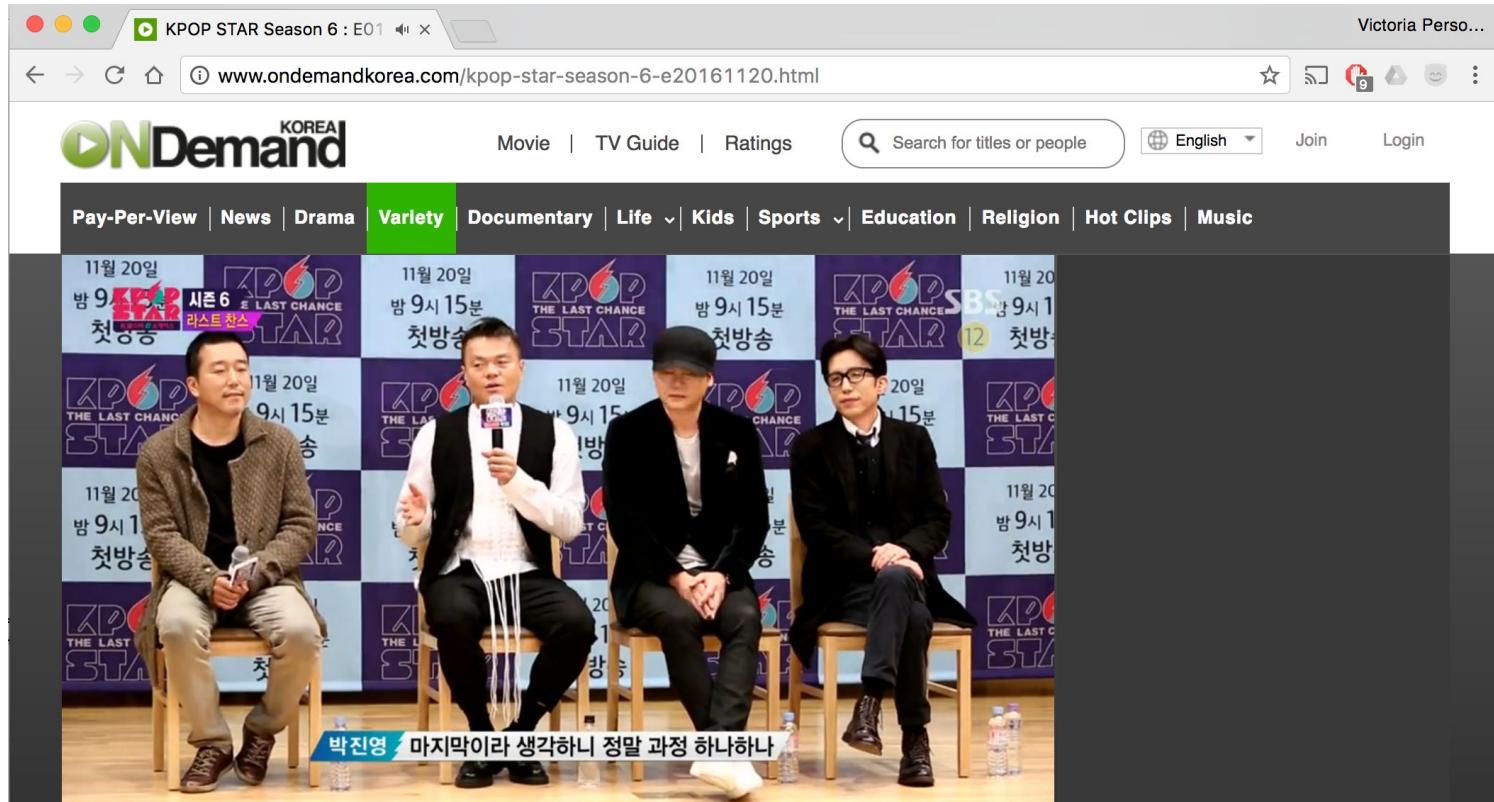
You can put JavaScript directly in your HTML by adding it to the body of a <script> tag:

```
<script>console.log('hello');</script>
```

Generally **you should not use do this**, as it's a violation of Separation of Concerns, mixing behavior (JS) in content (HTML)

(But for extensions/hacking, adding JavaScript directly in a <script> tag can be useful!)

Success!



KPOP STAR Season 6 : E01 - Part 1 - 11/20/2016

Share 0

Callbacks

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.

First-class functions

Recall: addEventListener

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

```
dragon.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"?



Be prepared to let go of some assumptions
you had about programming languages.

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:

x

15

```
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:

x 15

y true

```
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;
```

x 15

y true

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

greeting ...



"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);
```

This example is really contrived!

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

[CodePen](#)

Anonymous functions

We do not need to give an identifier to functions.

When we define a function without an identifier, we call it
an anonymous function

- Also known as a **function literal**, or a **lambda function**

```
function makeHelloFunction(name) {  
  return function() {  
    console.log('Hello, ' + name);  
  };  
}
```

[CodePen](#)

Practical Functional JavaScript

Functional programming

We are going to cover some topics that are fundamental to a programming paradigm called **functional programming**.

Pure functional programming is pretty extreme:

- Everything in your code is either a function or an expression
- There are no statements
- There is no state:
 - No variables, fields, objects, etc

Comes from the idea of treating a computer program as a mathematical function

Functional programming

This is a code snippet from [Scheme](#), a functional programming language:

```
(define (sum row)
  (let loop ((row row) (result '()))
    (if (= (length row) 1)
        (reverse result)
        (loop (cdr row)
              (cons (+ (first row) (second row))
                    result)))))
```

Everything is a function or the result of a function call.

Practical FP in JS

Most software is **not** built using a pure functional programming paradigm, so we won't be covering it.

But there are some ideas from functional programming that are immensely useful:

- First-class functions (functions as objects)
- **Currying**
- **Closures**
- **Anonymous functions / lambdas / function literals**

Why FP matters

Why should we learn about this other programming paradigm?

- There are **ideas you can express more clearly** and concisely with functional programming.
- There are **problems you can solve much more easily** with functional programming.
- (*very practically*) You will see JavaScript code in the wild that uses functional programming and the code will be indecipherable if you don't learn it.
- (*very practically*) Functional programming is trendy and so useful that C++ and Java added support for a few critical FP concepts (lambdas/closures) in the past few years.

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.)

Recall: Functions as parameters

We know that we can pass functions as parameters to other functions. We've already done this multiple times:

- The event handler parameter to `addEventListener`
- As a parameter for a constructor of a new object

Array objects also have several methods that take functions as parameters.

Closure & Currying

Loading data from files

Loading data from a file

What if you had a list of images in a text file that you wanted to load in your web page?

```
1 https://media1.giphy.com/media/xNT2CcLjhbI0U/200.gif
2 https://media2.giphy.com/media/3o7btM3VVVNtssGReo/200.gif
3 https://media1.giphy.com/media/l3q2uxEzLIE8cWMq4/200.gif
4 https://media2.giphy.com/media/LDwL3ao61wfHa/200.gif
5 https://media1.giphy.com/media/3o7TKMt1VVNkHV2PaE/200.gif
6 https://media3.giphy.com/media/DNQFjMJbbsNmU/200.gif
7 https://media1.giphy.com/media/26FKTsKMktUSomuNq/200.gif
8 https://media1.giphy.com/media/xThuW5Hf2N8idJHFVS/200.gif
9 https://media1.giphy.com/media/XlFFSD0CiyGLC/200.gif
10 https://media3.giphy.com/media/ZaBHSbiLQTmFi/200.gif
11 https://media3.giphy.com/media/JPbZwjMcXJYic/200.gif
12 https://media1.giphy.com/media/FArgGzk7K014k/200.gif
13 https://media1.giphy.com/media/UFoLN1EyKjLbi/200.gif
14 https://media1.giphy.com/media/11zXBCAb9soCQM/200.gif
15 https://media4.giphy.com/media/xUPGcHeIeZMmTcDQJy/200.gif
16 https://media2.giphy.com/media/apZwWJInOBvos/200.gif
17 https://media2.giphy.com/media/sB4nvt5xIiNiq/200.gif
18 https://media0.giphy.com/media/Y8Bi9lC0zXRkY/200.gif
19 https://media1.giphy.com/media/12wUXjm6f8Hhcc/200.gif
20 https://media4.giphy.com/media/26gsuVyk5fKB1YAAE/200.gif
21 https://media3.giphy.com/media/l2SpMU9sWIvT2nrCo/200.gif
22 https://media2.giphy.com/media/kR1vWazNc7972/200.gif
23 https://media4.giphy.com/media/Tv3m2GAAl2Re8/200.gif
24 https://media2.giphy.com/media/9nujydsBLz2dq/200.gif
25 https://media3.giphy.com/media/AG39l0rHgkRLa/200.gif
```

Intuition: loadFromFile

If we wanted to have an API to load external files in JavaScript, it might look something like this:

```
// FAKE HYPOTHETICAL API.  
// This is not real a JavaScript function!  
const contents = loadFromFile('images.txt');
```

Intuition: loadFromFile

```
// FAKE HYPOTHETICAL API.  
// This is not real a JavaScript function!  
const contents = loadFromFile('images.txt');
```

A few problems with this hypothetical fake API:

- We want to load the file **asynchronously**: the JavaScript should not block while we're loading the file
- There's no way to check the status of the request. What if the resource didn't exist? What if we're not allowed to access the resource?

Intuition: loadFromFile

An asynchronous version of this API might look like this:

```
// FAKE HYPOTHETICAL API.  
// This is not real a JavaScript function!  
function onSuccess(response) {  
    const body = response.text;  
    ...  
}  
loadFromFile('images.txt', onSuccess, onFail);
```

Where `onSuccess` and `onFail` are callback functions that should fire if the request succeeded or failed, respectively.

Fetch API

Fetch API: `fetch()`

The [Fetch API](#) is the API to use to load external resources (text, JSON, etc) in the browser.

The Fetch API is made up of one function, and its syntax is concise and easy to use:

```
fetch('images.txt');
```

Note: [XMLHttpRequest](#) ("XHR") is the old API for loading resources from the browser. XHR still works, but is clunky and harder to use.

Fetch API: fetch()

The [Fetch API](#) is the API to use to load external resources (text, JSON, etc) in the browser.

The Fetch API is made up of one function, and its syntax is concise and easy to use:

```
fetch('images.txt');
```

- The `fetch()` method takes the string path to the resource you want to fetch as a parameter
- It returns a Promise

Fetch API: `fetch()`

The [Fetch API](#) is the API to use to load external resources (text, JSON, etc) in the browser.

The Fetch API is made up of one function, and its syntax is concise and easy to use:

```
fetch('images.txt');
```

- The `fetch()` method takes the string path to the resource you want to fetch as a parameter
- It returns a Promise
 - **What the heck is a Promise?**

Promises: Another conceptual odyssey

Promises and .then()

A Promise:

- An object used to manage asynchronous results
- Has a `then()` method that lets you attach functions to execute `onSuccess` or `onError`
- Allows you to build **chains** of asynchronous results.

Promises are easier to **use** than to **define**...

Simple example: getUserMedia

There is an API called `getUserMedia` that allows you get the media stream from your webcam.

There are two versions of `getUserMedia`:

- `navigator.getUserMedia (deprecated)`
 - Uses callbacks
- `navigator.mediaDevices.getUserMedia`
 - Returns a Promise

getUserMedia with callbacks

```
const video = document.querySelector('video');

function onCameraOpen(stream) {
  video.srcObject = stream;
}

function onError(error) {
  console.log(error);
}

navigator.getUserMedia({ video: true },
  onCameraOpen, onError);
```

[CodePen](#)

getUserMedia with Promise

```
const video = document.querySelector('video');

function onCameraOpen(stream) {
  video.srcObject = stream;
}

function onError(error) {
  console.log(error);
}

navigator.mediaDevices.getUserMedia({ video: true })
  .then(onCameraOpen, onError);
```

[CodePen](#)

Hypothetical Fetch API

```
// FAKE HYPOTHETICAL API.  
// This is not how fetch is called!  
function onSuccess(response) {  
    ...  
}  
function onFail(response) {  
    ...  
}  
fetch('images.txt', onSuccess, onFail);
```

Real Fetch API

```
function onSuccess(response) {  
    ...  
}  
function onFail(response) {  
    ...  
}  
fetch('images.txt').then(onSuccess, onFail);
```

Promise syntax

Q: How does this syntax work?

```
fetch('images.txt').then(onSuccess, onFailure);
```

Promise syntax

Q: How does this syntax work?

```
fetch('images.txt').then(onSuccess, onFailure);
```

The syntax above is the same as:

```
const promise = fetch('images.txt');
promise.then(onSuccess, onFailure);
```

Promise syntax

```
const promise = fetch('images.txt');
promise.then(onSuccess, onFail);
```

The object `fetch` returns is of type [Promise](#).

A promise is in one of three states:

- **pending**: initial state, not fulfilled or rejected.
- **fulfilled**: the operation completed successfully.
- **rejected**: the operation failed.

You attach handlers to the promise via `.then()`

Promise syntax

```
const promise = fetch('images.txt');  
promise.+
```

The object

(We'll think about this more
deeply in a later lecture.)

A promise

- pending
- fulfill
- reject

(Right now we will just use
Promises.)

You attach handlers to the promise via .then()

Using Fetch

```
function onSuccess(response) {  
    console.log(response.status);  
}  
fetch('images.txt').then(onSuccess);
```

The success function for Fetch gets a response parameter:

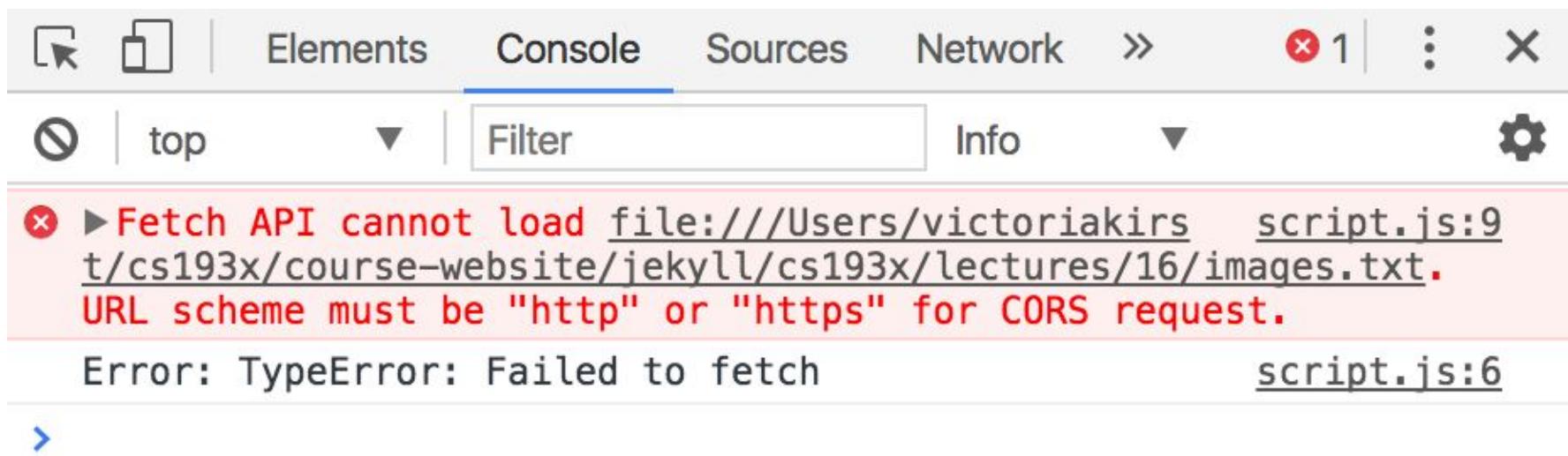
- **response.status**: Contains the status code for the request, e.g. 200 for HTTP success
 - [HTTP status codes](#)

Fetch attempt

```
function onSuccess(response) {  
    console.log(response.status);  
}  
  
function onError(error) {  
    console.log('Error: ' + error);  
}  
  
fetch('images.txt')  
    .then(onSuccess, onError);
```

Fetch error

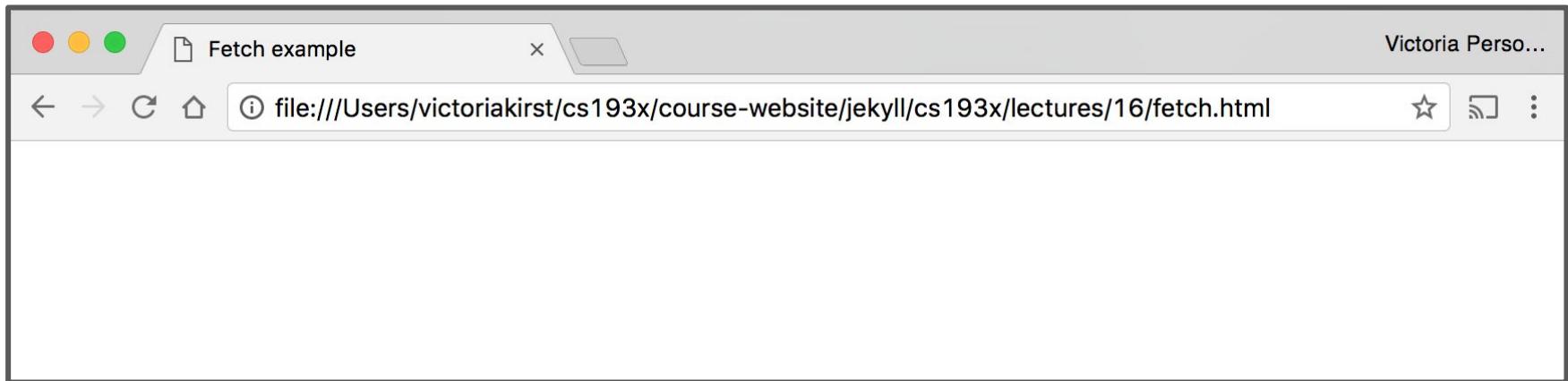
If we try to load this in the browser, we get the following JavaScript error:



Notice that our `onError` function was also called.

Local files

When we load a web page in the browser that is saved on our computer, it is served via `file://` protocol:



We are **not allowed** to load files in JavaScript from the `file://` protocol, which is why we got the error.

Serve over HTTP

We can run a program to serve our local files over HTTP:

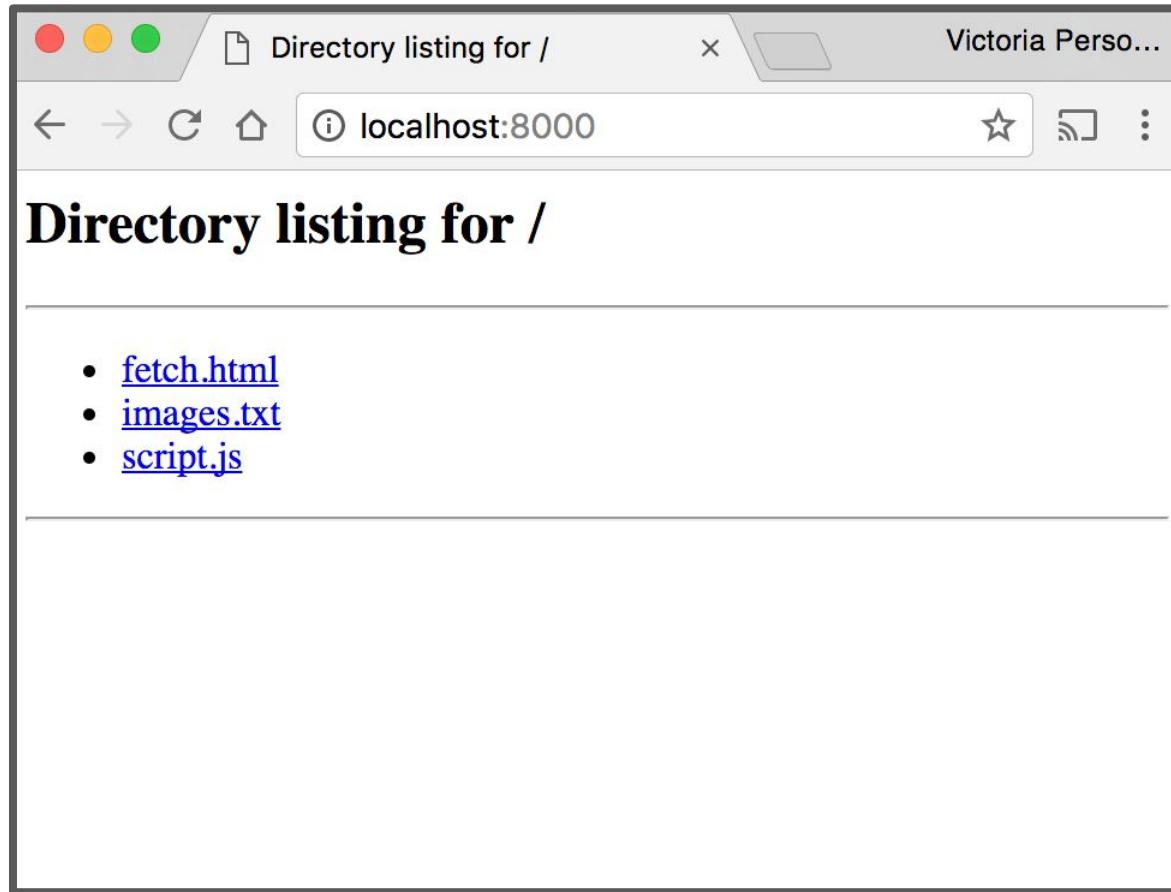
```
$ python -m SimpleHTTPServer  
Serving HTTP on 0.0.0.0 port 8000 ...
```

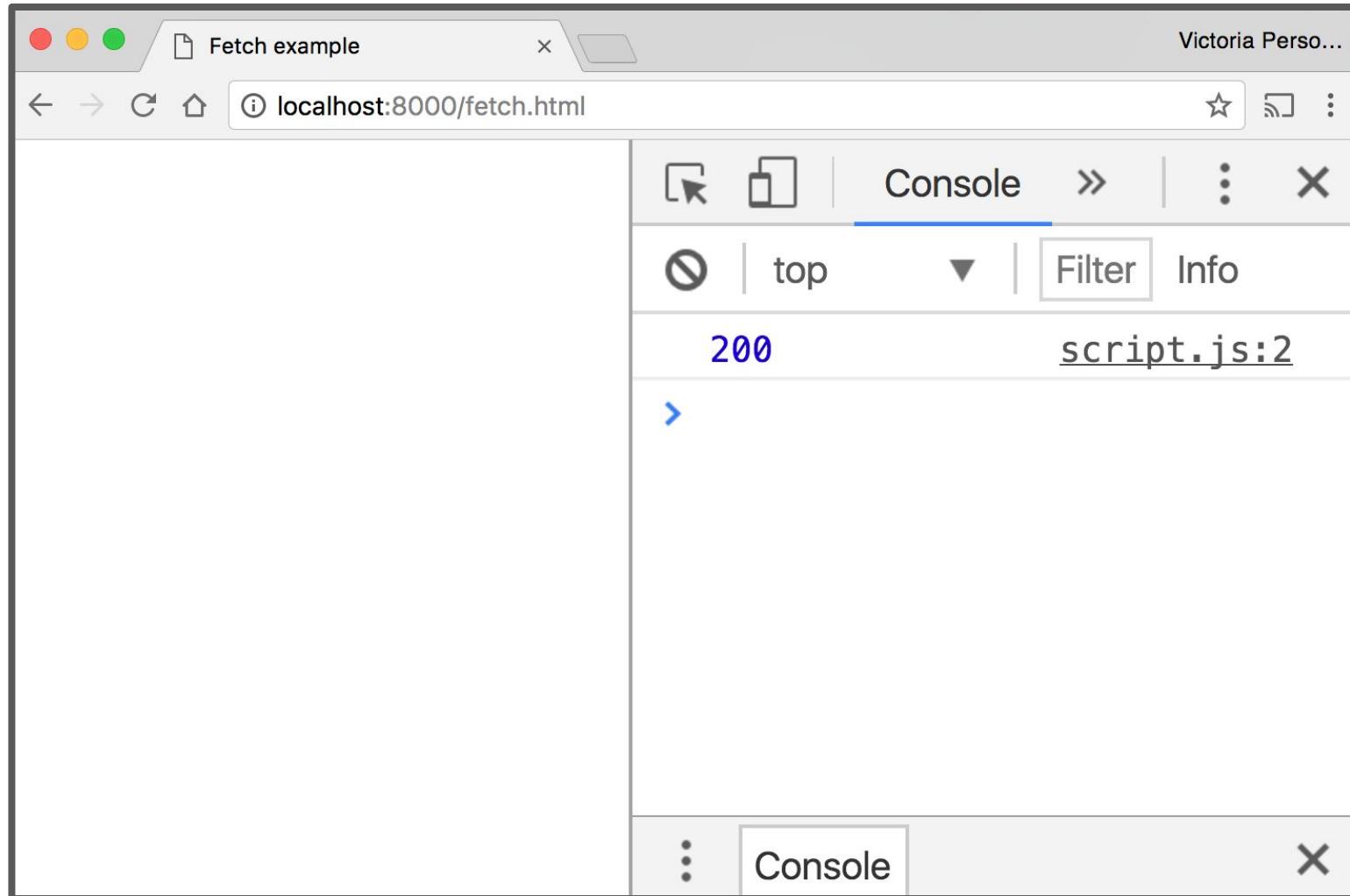
This now starts up a **server** that can load the files in the current directory over HTTP.

- We can access this server by navigating to:

<http://localhost:8000/>

```
$ python -m SimpleHTTPServer  
Serving HTTP on 0.0.0.0 port 8000 ...
```





We got HTTP response 200, which is success! ([codes](#))

How do we get the data from `fetch()`?

Using Fetch

```
function onSuccess(response) {  
    ...  
}  
fetch('images.txt').then(onSuccess);
```

- `response.status`: Status code for the request
- `response.text()`:
 - Asynchronously reads the response stream
 - **Returns a Promise** that resolves with the string containing the response stream data.

text() Promise

Q: How do we change the following code to print out the response body?

```
function onSuccess(response) {  
    console.log(response.status);  
}  
  
function onError(error) {  
    console.log('Error: ' + error);  
}  
  
fetch('images.txt')  
    .then(onSuccess, onError);
```

```
function onStreamProcessed(text) {  
    console.log(text);  
}  
  
function onResponse(response) {  
    console.log(response.status);  
    response.text().then(onStreamProcessed);  
}  
  
function onError(error) {  
    console.log('Error: ' + error);  
}  
  
fetch('images.txt').then(onResponse, onError);
```

Chaining Promises

We want the following asynchronous actions to be completed in this order:

1. When the `fetch` completes, run `onResponse`
2. When `response.text()` completes, run `onStreamProcessed`

```
function onStreamProcessed(text) { ... }
function onResponse(response) {
  response.text().then(onStreamProcessed);
}
fetch('images.txt').then(onResponse, onError);
```

We can rewrite this:

```
function onStreamProcessed(text) {  
    console.log(text);  
}  
  
function onResponse(response) {  
    response.text().then(onStreamProcessed);  
}  
  
function onError(error) {  
    console.log('Error: ' + error);  
}  
  
fetch('images.txt').then(onResponse, onError);
```

We can rewrite this:

```
function onStreamProcessed(text) {  
    console.log(text);  
}
```

```
function onResponse(response) {  
    return response.text();  
}
```

```
function onError(error) {  
    console.log('Error: ' + error);  
}
```

```
fetch('images.txt')  
    .then(onResponse, onError)  
    .then(onStreamProcessed);
```

Chaining Promises

```
function onStreamProcessed(text) {  
    console.log(text);  
}  
  
function onResponse(response) {  
    return response.text();  
}  
  
fetch('images.txt')  
    .then(onResponse, onError)  
    .then(onStreamProcessed);
```

Chaining Promises

```
function onStreamProcessed(text) {  
    console.log(text);  
}  
  
function onResponse(response) {  
    return response.text();  
}
```

```
const responsePromise = fetch('images.txt')  
    .then(onResponse, onError)  
responsePromise.then(onStreamProcessed);
```

The Promise returned by `onResponse` is effectively* the Promise returned by `fetch`. (*Not actually what's happening, but that's how we'll think about it for right now.)

Chaining Promises

```
function onStreamProcessed(text) {  
    console.log(text);  
}  
  
function onResponse(response) {  
    return response.text();  
}  
  
fetch('images.txt')  
    .then(onResponse, onError)  
    .then(onStreamProcessed);
```

If we don't think
about it too hard, the
syntax is fairly
intuitive.
We'll think about this more
deeply later!

Completed example

```
function onStreamProcessed(text) {  
    const urls = text.split('\n');  
    for (const url of urls) {  
        const image = document.createElement('img');  
        image.src = url;  
        document.body.append(image);  
    }  
}  
function onSuccess(response) {  
    response.text().then(onStreamProcessed)  
}  
function onError(error) {  
    console.log('Error: ' + error);  
}  
  
fetch('images.txt').then(onSuccess, onError);
```

JSON

JavaScript Object Notation

JSON: Stands for **JavaScript Object Notation**

- Created by Douglas Crockford
- Defines a way of **serializing** JavaScript objects
 - **to serialize:** to turn an object into a string that can be deserialized
 - **to deserialize:** to turn a serialized string into an object
- `JSON.stringify(object)` returns a string representing ***object*** serialized in JSON format
- `JSON.parse(jsonString)` returns a JS object from the ***jsonString*** serialized in JSON format

JSON.stringify()

We can use the `JSON.stringify()` function to serialize a JavaScript object:

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

```
const serializedBear = JSON.stringify(bear);  
console.log(serializedBear);
```

[CodePen](#)

JSON.parse()

We can use the `JSON.parse()` function to deserialize a JavaScript object:

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

```
const bear = JSON.parse(bearString);
console.log(bear);
```

[CodePen](#)

Why JSON?

JSON is a useful format for storing data that we can load into a JavaScript API via `fetch()`.

Let's say we had a list of Songs and Titles.

- If we stored it as a text file, we would have to know how we are separating song name vs title, etc
- If we stored it as a JSON file, we can just deserialize the object.

JSON

```
songs.json
```

```
1  {
2    "cranes": {
3      "fileName": "solange-cranes-kaytranada.mp3",
4      "artist": "Solange",
5      "title": "Cranes in the Sky [KAYTRANADA Remix]"
6    },
7    "timeless": {
8      "fileName": "james-blake-timeless.mp3",
9      "artist": "James Blake",
10     "title": "Timeless"
11   },
12   "knock": {
13     "fileName": "knockknock.mp4",
14     "artist": "Twice",
15     "title": "Knock Knock"
16   },
17   "deep": {
18     "fileName": "janet-jackson-go-deep.mp3",
19     "artist": "Janet Jackson",
20     "title": "Go Deep [Alesia Remix]"
21   },
22   "discretion": {
23     "fileName": "mitis-innocent-discretion.mp3",
24     "artist": "MitiS",
25     "title": "Innocent Discretion"
26   },
27   "spear": {
28     "fileName": "toby-fox-spear-of-justice.mp3",
29     "artist": "Toby Fox",
30     "title": "Spear of Justice"
31   }
32 }
```

Fetch API and JSON

The Fetch API also has built-in support for json:

```
function onStreamProcessed(json) {  
    console.log(json);  
}  
  
function onResponse(response) {  
    return response.json();  
}  
  
fetch('songs.json')  
    .then(onResponse, onError)  
    .then(onStreamProcessed);
```

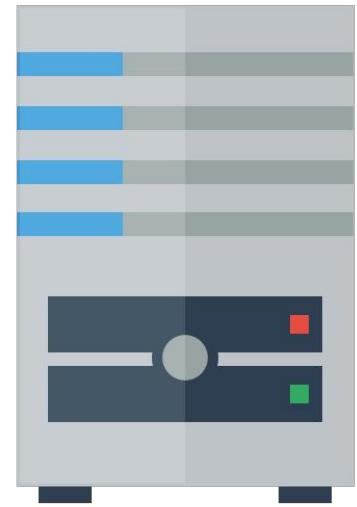
Querying REST APIs

First: Servers again

Servers

Sometimes when you type a URL in your browser,
the URL is a **path to a file** on the internet:

- Your browser connects to the host address
and requests the given file over **HTTP**
- The web server software (e.g. Apache) grabs
that file from the server's local file system,
and sends back its contents to you



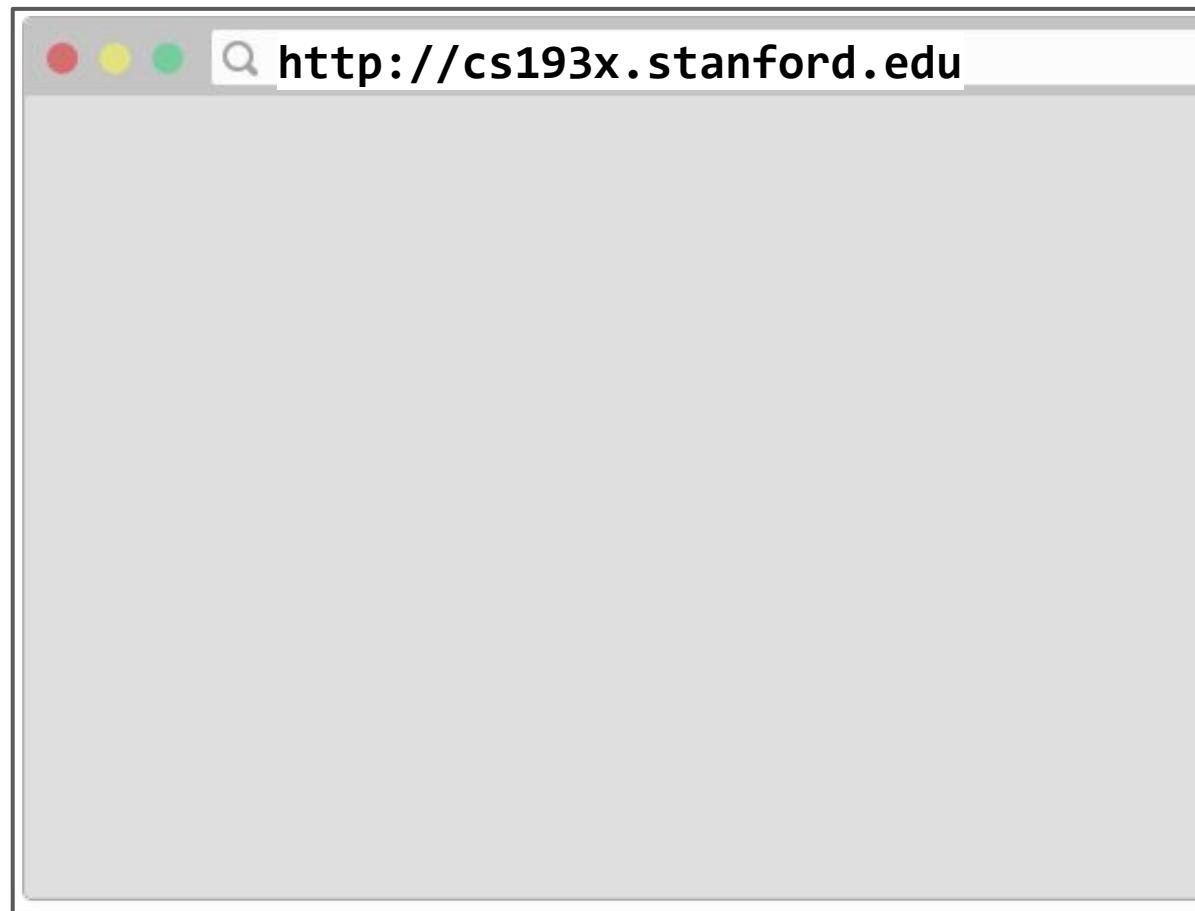
HTTP: Hypertext Transfer Protocol, the protocol for sending files and messages through the web

HTTP methods

HTTP Methods: the set of commands understood by a web server and sent from a browser

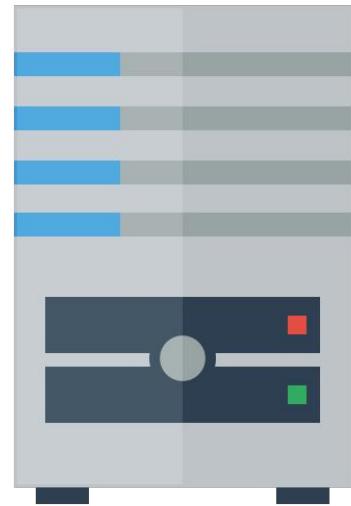
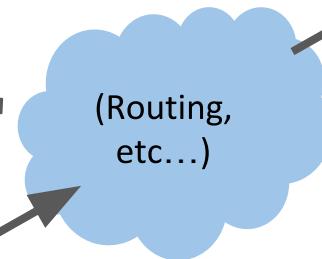
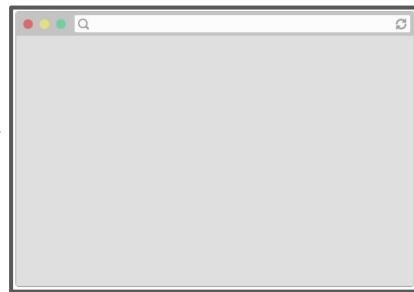
- **GET:** request/retrieve data
This is request sent by the browser automatically whenever you navigate to a URL!
- **POST:** send/submit data
- **PUT:** upload file
- **PATCH:** updates data
- **DELETE:** delete data
- [More HTTP methods](#)

You type a URL in
the address bar and
hit "enter"



Server at
<http://cs193x.stanford.edu>

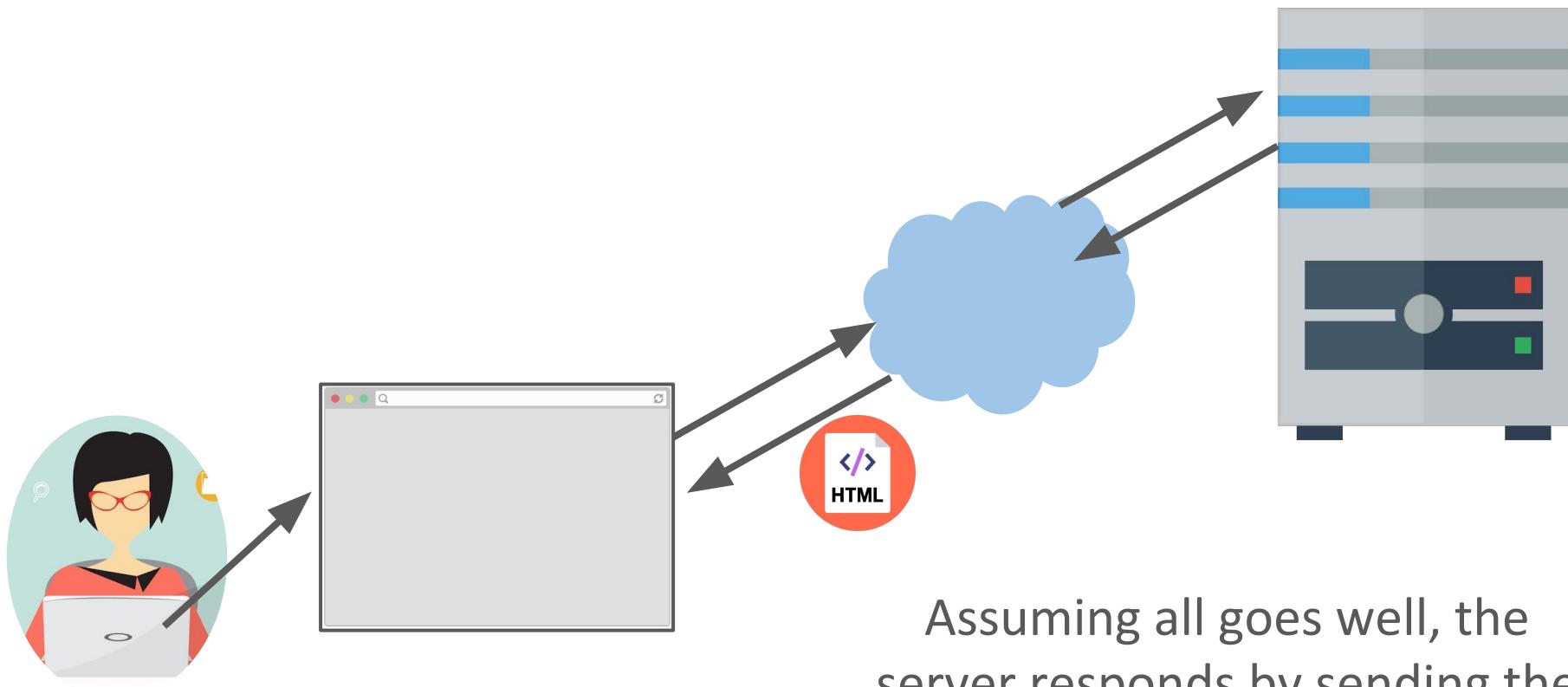
Browser sends an HTTP GET
request saying "Please GET me
the index.html file at
[http://cs193x.stanford.edu"](http://cs193x.stanford.edu)"



(Warning: Somewhat inaccurate,
massive hand-waving begins now.

See [this Quora answer](#) for slightly more detailed/accurate handwaving)

Server at
<http://cs193x.stanford.edu>

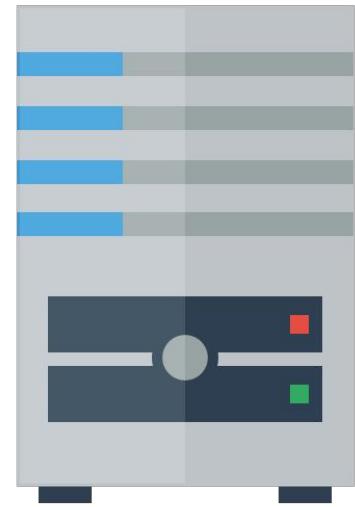


Assuming all goes well, the server responds by sending the HTML file through the internet back to the browser to display.

Servers

Sometimes when you type a URL in your browser,
the URL is a **path to a file** on the internet:

- Your browser connects to the host address
and requests the given file over **HTTP**
- The web server software (e.g. Apache) grabs
that file from the server's local file system,
and sends back its contents to you



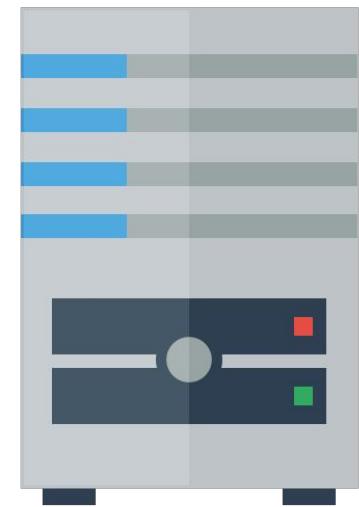
But that's not always the case.

Web Services

Other times when you type a URL into your browser, the URL represents **an API endpoint**, and not a path to a file.

That is:

- The web server does **not** grab a file from the local file system, and the URL is **not** specifying where a file is located.
- Rather, the URL represents a **parameterized request**, and the web server dynamically generates a response to that request.



API endpoint example

Look at the URL for this [Google slide deck](#):

https://docs.google.com/presentation/d/1Rim3-IXt6yN7yny_SBv7B5NMBiYbaQEiRMUD5s66uN8

API endpoint example

Look at the URL for this [Google slide deck](#):

`https://docs.google.com/presentation/d/1Rim3-IXt6yN7yny_SBv7B5NMBiYbaQEiRMUD5s66uN8`

- **presentation**: Tells the server that we are requesting a doc of type "presentation"
- **d/1Rim3-IXt6yN7yny_SBv7B5NMBiYbaQEiRMUD5s66uN8**: Tells the server to request a doc ("d") with the document id of "1Rim3-IXt6yN7yny_SBv7B5NMBiYbaQEiRMUD5s66uN8"

RESTful API

RESTful API: URL-based API that has these properties:

- Requests are sent as an **HTTP request**:
 - **HTTP Methods**: GET, PUT, POST, DELETE, etc
- Requests are sent to **base URL**, also known as an "**API Endpoint**"
- Requests are sent with a specified **MIME/content type**, such as HTML, CSS, JavaScript, plaintext, JSON, etc.

RESTful API

Almost every website on the internet uses RESTful URLs / RESTful APIs to handle requests to its servers.

Notable alternatives to REST:

- [GraphQL](#)
 - Used by Facebook since 2012
 - Open-sourced by Facebook since 2015
 - Still early but some big clients: GitHub, Pinterest
- [Falcor](#)?
 - Netflix's REST alternative, introduced ~2015
 - Probably cool but never hear of anyone using it
 - Doesn't even have a Wikipedia page

Using REST APIs

3rd-Party APIs

Many websites expose REST APIs to outside developers.

These are often called "**3rd-party APIs**" or "**Developer APIs**"

Examples:

- Spotify
- Giphy
- GitHub
- Hoards of Google APIs
- Facebook
- Instagram
- Twitter
- etc...

Try Googling
"<product name> API"
to see if one exists for
a given company!

Example: Spotify

Spotify has a [REST API](#) that external developers (i.e. people who aren't Spotify employees) can query:

Our Web API endpoints give external applications access to Spotify catalog and user data.

Web API Base URL: <https://api.spotify.com>

[User Guide](#) | [Tutorial](#) | [Code Examples](#)

Search:

METHOD	ENDPOINT	USAGE	RETURNS
GET	/v1/albums/{id}	Get an album	album
GET	/v1/albums?ids={ids}	Get several albums	albums
GET	/v1/albums/{id}/tracks	Get an album's tracks	tracks*
GET	/v1/artists/{id}	Get an artist	artist
GET	/v1/artists?ids={ids}	Get several artists	artists
GET	/v1/artists/{id}/albums	Get an artist's albums	albums*

Example: Spotify

REST API structure ([details](#)):

- The **Base URL** is `https://api.spotify.com`
- The **HTTP method** is GET
- The **API endpoint** to query is:
`https://api.spotify.com/v1/albums/<spotify_id>`
- It returns **JSON data** about the album that's requested

Web API Base URL: `https://api.spotify.com`

METHOD ENDPOINT

GET `/v1/albums/{id}`

Example: Spotify

If we had Spotify Album ID 7aDBFWp72Pz4NZEtVBANi9, how would we make a GET request for the album information?

REST API structure ([details](#)):

- The **Base URL** is `https://api.spotify.com`
- The **HTTP method** is **GET**
- The **API endpoint** to query is:
`https://api.spotify.com/v1/albums/<spotify_id>`
- It returns **JSON data** about the album that's requested

GET request: Browse to URL

Loading a URL in a browser issues an HTTP GET request for that resource.

So if we just piece together this URL:

- **API Endpoint:**

`https://api.spotify.com/v1/albums/<spotify_id>`

- **Album ID:** 7aDBFWp72Pz4NZEtvBANi9

- **Request URL:**

<https://api.spotify.com/v1/albums/7aDBFWp72Pz4NZEtvBANi9>

If you click on the link, you see it returns a JSON object.

GET request: fetch()

Actually, the `fetch()` API also issues an HTTP GET request by default.

So if we do:

```
fetch('https://api.spotify.com/v1/albums/7aDBFWp72Pz4  
NZEtvBANi9')  
  .then(onResponse)  
  .then(onTextReady);
```

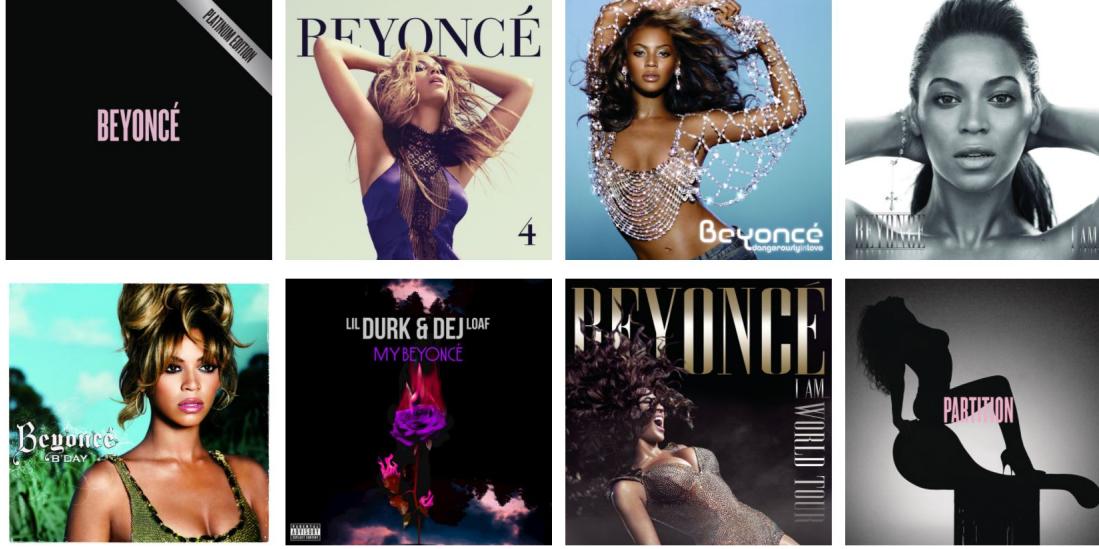
...we can load the JSON data as a JavaScript object, as we did with our `.json` files!

([CodePen](#) / [demo](#))

Album example

Let's write a web page that asks the user to enter an artist's name, then displays the albums of that artist, as provided by the [Spotify Search API](#). ([live demo](#))

Enter an artist:



Spotify search API

Spotify Search URL:

<https://api.spotify.com/v1/search?type=album&q=query>

E.g.

<https://api.spotify.com/v1/search?type=album&q=beyonce>

Q: Hey, what's that at the end of the URL?

- ?type=album&q=beyonce

Query parameters

You can pass parameters to HTTP GET requests by adding **query parameters** to the URL:

?**type=album&q=beyonce**

- Defined as key-value pairs
 - ***param=value***
- The first query parameter starts with a **?**
- Subsequent query parameters start with **&**

Reminder: HTML elements

Single-line text input:

```
<input type="text" />
```

hello

In JavaScript, you can read and set the input text via
`inputElement.value`

Some other input types:

- [Select](#)
- [Textarea](#)
- [Checkbox](#)

Form submit

Beyonce

Submit

Q: What if you want the form to submit after you click "enter"?

Form submit

1. Wrap your input elements in a <form>

```
<form>
  <input type="text" id="artist-text" />
  <input type="submit" />
</form>
```

You should also use <input type="submit"> instead of <button> for the reason on the next slide...

Form submit

2. Listen for the 'submit' event on the form element:

```
const form = document.querySelector('form');
form.addEventListener('submit', this._onSubmit);
```

This is why you want to use `<input type="submit">` instead of `<button>` -- the 'submit' event will fire on click for but not `<button>`.

Form submit

3. Prevent the default action before handling the event through `event.preventDefault()`:

```
_onSubmit(event) {
  event.preventDefault();
  const textInput = document.querySelector('#artist-text');
  const query = encodeURIComponent(textInput.value);

  this.albumUrls = [];
  fetch(SPOTIFY_PATH + query)
    .then(this._onResponse)
    .then(this._onJsonReady);
}
```

The page will refresh on submit unless you explicitly prevent it.

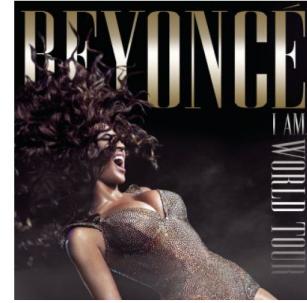
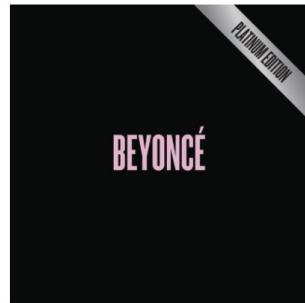
Album example

Solution: [GitHub](#) / [Demo](#)

Enter an artist:

Beyonce

Submit



Chuck Norris API

The Cats API

Musixmatch lyrics API

<https://api.chucknorris.io/jokes/random>

Chuck Norris API

The Cats API

Musixmatch lyrics API

Chuck Norris API

The Cats API

Musixmatch lyrics API

Recap

Review: Functional JavaScript

- Functions in JavaScript are **first-class citizens**:
 - Objects that can be passed as parameters
 - Can be created within functions:
 - Inner functions are called **closures**
 - Can be created without being saved to a variable
 - These are called **anonymous functions**, or function literals, or lambdas
 - Can be created and returned from functions
 - Constructing a new function that references part of the outer function's parameters is called **currying**