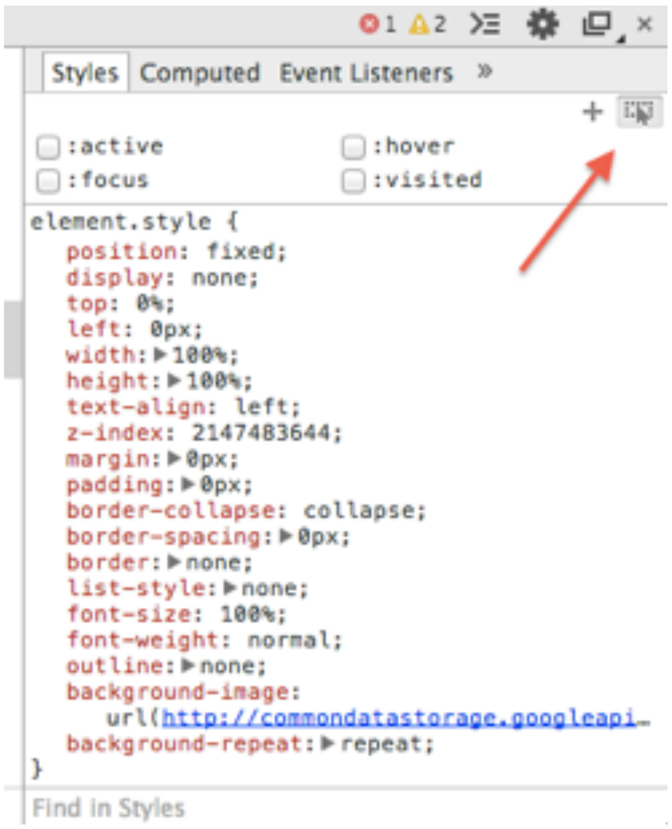
letter-spacing: -0.02em;
text-transform: uppercase;

DEV TOOLS / HACK THE TIMES:

You can drag and drop elements around in the HTML panel to test out different arrangements for the page.

You can play with the CSS of an element at a different state like "hover" or "active" by clicking on this little icon:



```
element.style {
   position: fixed;
   display: none;
   top: 0%;
   left: 0px;
   width: ▶100%;
   height: ▶100%;
   text-align: left;
   z-index: 2147483644;
   margin: ▶0px;
   padding: ▶0px;
   border-collapse: collapse;
   border-spacing: ▶0px;
   border: ▶none;
   list-style: ▶none;
   font-size: 100%;
   font-weight: normal;
   outline: ▶none;
   background-image:
      url(http://commondatastorage.googleapi…
   background-repeat: ▶repeat;
}
```

Many sites, including the New York Times, will automatically refresh at set intervals. This presents a problem because if you're working in DevTools and the page refreshes, you'll lose your modifications. Fortunately, you can deactivate auto-reloading. To do this, open the Chrome Developer Tools JavaScript console by clicking View --> Developer --> JavaScript Console. Then, at the cursor, enter the following command, followed by return: window.onbeforeunload = function(){return 'Reload?';}. This line will prompt you to confirm a reload before it happens.

FONT AWESOME

Font Awesome provides icons that can be used for just about anything. Scalable vector icons are an incredibly flexible way of adding nice icons consistently and quickly on any website you build.

To install Font Awesome locally, head over to http://fortawesome.github.io/Font-Awesome/ and click on the Download button. Once the zip file has been downloaded, unzip it and take a look at what is inside. You should see four directories, css, fonts, less, and scss. The two directories that you are concerned about are css and fonts. Inside the css directory are two files, these are the main styles for Font Awesome, one is minified and the other is the original. For now, you're going to use the minified version so copy the font-awesome.min.css file to the css folder of your project. The other key pieces are the various font files that Font Awesome uses. Take a look in the fonts directory and you should see 6 font files representing the different font types (EOT, SVG, TFF, OTF, and WOFF). You will need these files as well in order to use Font Awesome. In your project folder, create a new folder called fonts, this is where Font Awesome is expecting to find the font files. Next, copy all the font files into the fonts directory that you created.

Now that you've installed Font Awesome, the last thing to do is link your index.html file to it. Add the following line to your index.html:

```
<link href="//maxcdn.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css"
rel="stylesheet">
```

See how no protocol is supplied (http://, file://, git://, https://)? You're loading the files from a location called a CDN, in this case MaxCDN. This is a common technique of loading resources from a location that will have been used on multiple websites before, so most people visiting your website will have this file saved on their computer already; this speeds up the loading of the stylesheet. CDN links are not always the best choice, but for this project it's perfect.

There's a problem. When you open up the Chrome Developer Tools to check if this stylesheet has loaded in the Network tab, you'll see an error. This is because you're 'hosting' the file on your own computer, so when you open the file in the browser, the URL starts with 'file://'. Your browser will now block this font-awesome.min.css file from downloading. Later you will move this web page to a server hosted environment and the CDN will work... but for now, this won't do.

FAVICONS (Head/Title)

RESPONSIVE

Take a moment to explore the responsive website designs found here. Resize the browser window and see how the layout changes for narrower screens. If you can, view the page on your phone and tablet too. The content adapts to the available space—that is responsive web design!

Device Emulation in Google Chrome

If you don't have a mobile phone or tablet to test with, you can preview the site using the Device Mode Emulation in the Chrome developer tools, or with online testing tools like BrowserStack screenshots. Try them out!

Responsive design techniques
There are two key concepts that serve as the core for responsive design: fluid and adaptable. From these simple foundations, tons of techniques have appeared in recent years, but they're all aiming to achieve one goal—making the page content and layout fit the screen.

Fluid
By default, the web is fluid—a pure HTML page with no CSS markup will stretch and shrink to fill the browser window. However, by adding layout CSS, the size of all the elements becomes restricted. This causes problems when users' screen sizes don't match the developer's expectations. In the past, many websites used a 960px grid system. This worked well for desktop sites, but all mobile phones have much smaller screens than 960px—causing those users to have to scroll horizontally on websites designed this way.

Luckily, CSS has many ways to keep fluidity while allowing you to control your layouts. The simplest way to achieve fluidity is to avoid setting explicit pixel widths in your CSS. If you look back at your About Me project, you should see lots of percentage widths. This is the first step to fluidity. The most widely-used technique to accomplish fluidity is a responsive grid system. Many CSS frameworks like Bootstrap or Foundation implement responsive grids for you.

Other techniques include absolute positioning, or using JavaScript to control elements' attributes on the fly, but these two techniques are not as simple or effective as responsive grids.

Adaptable

Sometimes, content isn't appropriate for certain contexts. On a mobile phone's limited space, some information doesn't make the cut. In these cases, content is removed or hidden in order to free up screen space for more important content. Essentially, the page layout "adapts" to the screen context. These design decisions are based on more than just screen size—it's important to consider the content users need when they are out and about using a phone, compared with lounging at home with a tablet or working on their desktop PC.

Generally, adaptable designs are achieved using media queries. You'll learn more about media queries in the following assignment.

Other Concepts

Mobile First
One major technique for achieving responsive design is to think "mobile-first". It's self-explanatory: you design a site for a mobile phone first, and then adapt the design for progressively wider screens. Generally, this means going from a simple single-column layout on phones to more complex, grid-based layouts on tablets and desktops.
Accessibility
Accessibility is a parallel concern to responsive design. Both are concerned with making websites user-friendly in multiple contexts. However, accessibility relates to "non-conventional" browsers, including screen readers and text-only browsers. HTML5 has plenty of attributes that help with accessibility, including the alt attribute and aria roles among others.
Proportional Units
Pixels and points are static units—they refer to a specific, device-dependent unit of measure. With all of this emphasis on being device-agnostic, it makes sense that proportional units—like percentages, ems, and rems—are more attractive options.
Proportional units allow a page to retain fluidity while also controlling its layout.
In the next assignment, you'll revisit your About Me project and add a couple of media queries to make it adaptable.

RESPONSIVE PORTFOLIO

To wrap up Unit 1, you'll design and code a personal portfolio from scratch. Right now, you'll only have a few projects, but that will quickly change as you progress through the course. This portfolio will provide you the platform to display your work to potential employers or to friends and family members interested in your learning.

Portfolio Requirements:
Your project must:

Use Semantic HTML
Implement several well-designed fonts
Have a consistent color pattern
Prominently identify you as a "Web Developer"
Display your "About Me" page followed by a section titled "Projects" with screenshots of your NY Times hack and Karma landing page clone
Your project optionally could:

Have multiple sections/pages like an 'About Me' section, a 'Portfolio' section, and a 'Blog Post' section
Use CSS3 Animation
Be fully responsive
Use CSS grids

The version of your portfolio that will be submitted today will begin with your "About Me" page (created during lesson one) and be followed by a section titled "Projects" which will display screenshots of your products from "Hack the Times" and cloning the Karma landing page. Recall, that the "About Me" page was created in CodePen. In order to share this as part of your personal portfolio site, you will need to move your HTML and CSS into sublime text. Remember, your HTML sheet and your CSS sheet will need to be linked. If you do not remember how to complete this process you may need to reference the "Introducing HTML" and "Introducing CSS" lessons. Additionally, if you cannot remember how to add photos to your HTML you can find a tutorial in the lesson "Building the Navigation Bar."

After you have your code successfully transferred into sublime. You will need to create a new repository in GitHub. Name this new repository "firstname-lastname-portfolio." Once this repository is created, complete your initial commit.

To actually create a site that is viewable by others you will need to create a "gh-pages" branch. If you do not remember how to perform this command, you can find instructions in the lesson "FavIcons and Other Finishing Touches." Once you have completed this first version of your portfolio, submit the site's url to your mentor.

Always consider:

Deeply think about the user experience you want to create. Think about the site's audience and how to best capture their interest. Also, make sure to think about the main goal of the landing page, and how you can ensure that the goal will be met for viewers.
After you've thought about your users, start sketching and/or wireframing your site. As you're doing this, keep your UX decisions top of mind. Whenever you're designing a component, you should be able to explain why you think it's the best way of accomplishing your goal.
After you've created your wireframe, share it with your mentor and colleagues and ask for their feedback. Be sure to let them know the context for the project if you're working on your own business. If you get valuable feedback, do an additional iteration on your designs before moving on to code.

The importance of creating a wireframe before coding cannot be stressed enough. Being good at front end development is being able to deliberately code up a fully specified design. Once your design is set, it's time to code your site. This process will look like it did for the Karma clone. You'll want to create a new project folder, and you'll need an index.html and main.css file. Work on the overall layout first, and then go back and work on the small details (for instance, a gradient on a button). Finally, when your landing page is done, make it publicly available on GitHub Pages, and share a link to it with your fellow students and mentor.

STREET FIGHTER ALTERNATE HTML / CSS

In this assignment you will set up your HTML and CSS. You'll need to make the background for the body black and need a container div inside of the body with a white background that can display your Ryu-related content. You are going to need to use .hide() and .show(), so you will also need some classes and divs that will be hidden and shown. Ryu will be positioned within the container and the Hadouken will be positioned in relation to Ryu.

As you go through the steps below, you'll want to have main.html and main.css open in Sublime Text (and for this, you may find it useful to use View --> Layout --> Columns:2 so you can see both files side by side). You'll also want to open main.html in Chrome, so you can preview your page as you make changes.

Background Color
Setting the body background to black is easy enough. Inside of main.css, make that your first declaration, with

```
body {
  background-color: black;
}
```

Main Container
Create the outermost container div - the white block across the middle of the page. In main.html, between the body tags, add a new div with a class of "main". Then, in main.css, add the following declaration:

```
.main {
  background-color: white;
  margin-top: 110px;
  padding-left: 100px;
  min-width: 1200px;
  height: 500px;
  position: relative;
  z-index: 1;
}
```

These first three settings should make immediate sense: you want a white background for this div and you want it to be pushed down from the top of the page so the black background shows through. We set the left padding so that whatever is contained by this DIV will be pushed away from the left of the page. If you were trying to implement these margin and padding properties on your own, you'd use a trial and error process — ideally with Dev tools — to fine tune your margin-top.

The reasoning behind the min-width of 1200px is probably less obvious. Ultimately, this div is going to need to accommodate our Ryu images (which are all 659px wide) and our Hadouken, which will start out near the right of this image and travel to the right of the screen. Setting the min-width to 1200px should accommodate these requirements (even if it makes the experience suboptimal for mobile users). If the user's browser window is narrower than 1200px, they'll have to scroll.

The minimum height of this div is set to 500px. This means that even if we don't have content in it, it will still display.

Ryu Container
Add another container div inside of div.main to hold the Ryu images. You will alternate between showing and hiding these three images. All three will need to be in the HTML (you'll take care of this in the next step), so it makes sense to contain these related elements in a common div. Inside of div.main, create a new div with a class of "ryu". In main CSS, create a style rule for this class, and set the float property to left. Ultimately, you will want div.ryu and div.hadouken (which we'll create in a moment) to float next to one another within the main div. Setting the float property to left on both classes will achieve this.

Ryu Image Divs
Now create divs with classes for the Ryu still, Ryu ready, and Ryu "Hadoukening" images (hold off on doing anything with the Ryu looking cool image; you'll implement that in the final assignment of this lesson). Inside of div.ryu, create three new divs as follows:

```
<div class="ryu">
  <div class="ryu-still"></div>
  <div class="ryu-ready"></div>
  <div class="ryu-throwing"></div>
</div>
```
Then, in main.css, add the following style declarations:

```
.ryu {
    width: 659px;
    height: 494px;
}

.ryu-ready, .ryu-still, .ryu-throwing {
  width: 659px;
  height: 494px;
}

.ryu-ready, .ryu-throwing {
  display: none;
}

.ryu-ready {
  background-image: url('../images/ryu-ready-position.gif');
}

.ryu-still {
  background-image: url('../images/ryu-standing-still.png');
}

.ryu-throwing {
  background-image: url('../images/ryu-throwing-hadouken.png');
}
```

Note how you are avoiding repetition by assigning the width and height of 659 and 494 for all three classes at the same time. You then set .ryu-ready and .ryu-throwing to display:none because initially you want the div.ryu-still to display — you'll only display the other divs in response to events. You then set the background image for the three classes separately.

There's one additional background-image related property you need to set, not just for the Ryu classes, but also for the Hadouken class in the next step. To guarantee that the images are not repeated if the div they are in is bigger than their dimensions, you must set the background-repeat property. In main.css, add the following rule:

```
.hadouken, .ryu-ready, .ryu-still, .ryu-throwing {
  background-repeat: no-repeat;
}
```

Look at a Preview

Before moving on to position the Hadouken div, preview what you have so far in Chrome. Open main.html in your browser. You should see the black background, a white container in the middle of the page, and inside of that, the image of Ryu standing still, pushed away from the left of the page. You're eventually going to be hiding and showing the three Ryu images in response to events, so preview that behavior now. Open Developer Tools and go to the Elements panel. Select the div.ryu-still and in the Styles panel set this div's display property to none. Then select div.ryu-ready and in the Styles panel set its display property to block. You should see the Ryu ready animated gif replace Ryu standing still. Finally, hide div.ryu-ready and show div.ryu-throwing to preview that image.

Hadouken Div

Now insert the Hadouken div into the page and write your styles. Inside div.main, below div.ryu, add a new div with a class of "hadouken". You need to set the background image, width, height, and display properties for this div much like you did for the Ryu image divs. In main.css, add the following rule:

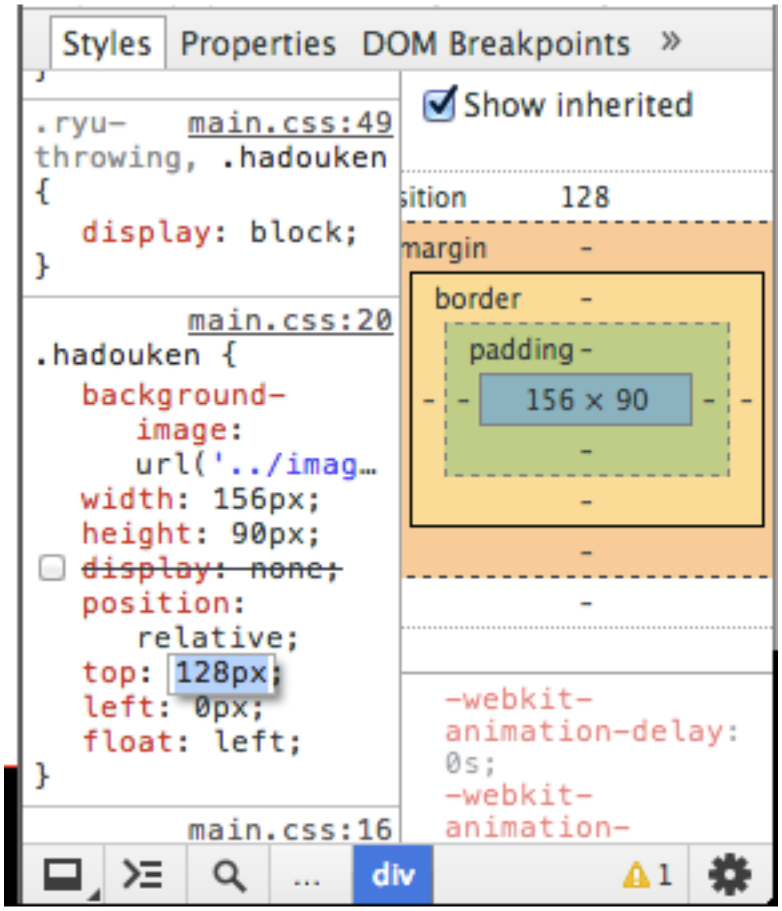
```
.hadouken {
  background-image: url('../images/hadouken.gif');
  width: 156px;
  height: 90px;
  display: none;
  float: left;
  position: absolute;
  z-index: 10;
}
```

Now figure out the settings needed to properly position this div. To better visualize how your elements are positioned, temporarily set the following style rule in our main.css:

```
* {
  outline: red 1px solid;
}
```

This will put a red outline around all of your DOM elements. Now open main.html in Chrome, then in Developer tools, show div.ryu-throwing, hide div.ryu-still, and show div.hadouken. You should see Ryu in his throwing position, with the Hadouken positioned immediately to the right of div.ryu and at the top.

You need to position the Hadouken so it's next to Ryu's hands. To achieve this, the left and top properties seem ideal. You may recall that when you don't explicitly set the position property on an element, it defaults to static. However, the left and right properties only work with relatively or absolutely positioned elements. You'll set div.hadouken's position property to relative, and initially set the left and right properties to zero, then you'll use Developer tools to refine those settings. Inside of main.css change your .hadouken style so it looks like this:



```
.ryu-throwing, .hadouken {        main.css:49
    display: block;
}
.hadouken {                        main.css:20
    background-image: url('../imag…
    width: 156px;
    height: 90px;
    ~~display: none;~~
    position: relative;
    top: 128px;
    left: 0px;
    float: left;
}
                                   main.css:16
-webkit-animation-delay: 0s;
-webkit-animation-
```

```
.hadouken {
  background-image: url('../images/hadouken.gif');
  width: 156px;
  height: 90px;
  /*display: none;*/
  position: absolute;
  z-index: 10;
  top: 0px;
  left: 0px;
  float: left;
}
```

Note that you're temporarily commenting out display:none here, so it displays when the page loads. You'll want to uncomment that line after you figure out the top and left values.

Save these changes, and then reload main.html. Back in the elements panel, you'll want to set div.ryu-still to display: none, and div.ryu-throwing to display: block. Then, you should select div.hadouken, and in the styles panel click on the value for the top property. After you click, your Styles panel should look like this:

Chrome Dev Tools Picture

With the top value highlighted, you can use the up and down arrows on your keyboard to increase and decrease this value. To position the Hadouken, you'll probably need to iterate between adjusting the top and left properties. Setting top to 167px and left to 520px looks pretty good (but if you'd like the starting position of the Hadouken to be slightly different, feel free to go with different values). Now, go back to main.css and put in the top and left values for .hadouken that you just tested. Also uncomment the display: none line in .hadouken, and get rid of your * { outline: red 1px solid;} setting.

In the next assignment, you'll write the code for your events using jQuery and JavaScript. Before you move on, first commit your changes using your GitHub client. And if you want to check your code, you can see what main.html and main.css should look like at this point in this Gist.

If you push your changes to GitHub and setup GH Pages to view your project in action be sure that you specify main.html in the URL bar. This will be explained more at the end of 2.2.5 or feel free to ask your mentor about this.

STREET FIGHER EVENTS

In this assignment, you'll write the code to handle when the user hovers over Ryu. When this happens, you want Ryu to move to his "ready position". You'll use .show() and .hide() to accomplish this.

Open Your Files

Most of the this assignment will be in js/app.js, so open that file in Sublime Text. You'll also want to keep main.html open, so you can reference it by selecting elements for jQuery objects, so open that file, too. This is a good occasion to use View --> Layout --> Columns: 2 , like you did in the last assignment. Also open up main.html in Chrome so you can preview your page as you work.

Document Ready

Even before you start writing the code for the events, you know you want to wrap it in a $(document).ready(function () {}); block, which will prevent the jQuery from executing before the page has loaded, so go ahead and put that code in app.js. Inside of that file (which should be blank at this point), put the following code:

```
$(document).ready(function() {

});
```

You'll be putting your event-related code within this function.

Event Strategy

When main.html loads, your .ryu-still div appears, and although .ryu-ready and .ryu-throwing will load as part of the DOM since they're in the HTML, they won't be displayed because you have set their display property to none in main.css. You need a way to register when a user hovers over and clicks on Ryu, and for this, you're going to take advantage of the fact that all of your Ryu image divs are contained within a single .ryu container. You want to know when a user is interacting with this .ryu container div irrespective of which Ryu image happens to be displaying. That means you'll want to listen for events on that div in particular.

Ryu Ready 1

You will now write the code that will move Ryu into his ready position. For this, you'll listen for a mouseenter event on the .ryu div. So modify app.js so it looks like this:

```
$(document).ready(function() {
  $('.ryu').mouseenter(function() {
    alert('mouse entered .ryu div');
  });
});
```

You have added the alert method call, so you can confirm that your event is firing as expected. Before you bother writing the code that determines what happens when this event fires, you should ensure that the event happens. Alerts or console.logs are a good technique for this.

Save this change to app.js, then go to Chrome and reload the page. After the page loads, hover your mouse over Ryu, and you should get an alert with the message indicated.

Ryu Ready 2

Now that you know your event is firing, you just need to write the code that will hide the .ryu-still div and show the .ryu-ready div. You'll use the .show() and .hide() methods to achieve this. Inside of the mousenter callback function, replace the alert with the following code:

```
$('.ryu-still').hide();
$('.ryu-ready').show();
```

This code hides div.ryu-still and shows div.ryu-ready, whose initial state was display: none. Back in Chrome, refresh the page, then move your mouse over Ryu. He should move into his ready position. Note, however, that when you move the mouse back away from Ryu, he remains in the ready position, which makes sense, because you haven't written that code yet.

The code is currently listening for mouseenter events on the div.ryu, but you want it to listen for mouseleave events on this same object. In situations such as these, take advantage of method chaining. Like before, first confirm that you can get the mouseleave event to fire, then we'll write the code that will move Ryu back to his still position. Modify app.js so it looks like this:

```
$(document).ready(function() {
  $('.ryu').mouseenter(function() {
    $('.ryu-still').hide();
    $('.ryu-ready').show();
  })
  .mouseleave(function() {
    alert('mouse left');
  });
});
```

Note in the code the chained .mouseleave method call is on a new line — this makes the code more readable. Save this change, then go back to Chrome, refresh the page, hover your mouse over Ryu, then move it away. If you get the alert, the event is firing correctly.

Now replace that alert with code that will make Ryu reassume his still position. Replace the alert() with the following code:

```
$('.ryu-ready').hide();
$('.ryu-still').show();
```

Save those changes, then go back to Chrome and refresh your page (are you starting to notice a pattern here?). Try moving the mouse over and then away from Ryu. When the mouse leaves, Ryu should now return to his still position.

STREET FIGHTER ANIMATION

In this assignment, you'll write the code to handle when a user clicks on Ryu. When this happens, you want Ryu to throw a Hadouken (and yell "Hadouken!"). You will code the .hide() and .show() behavior first, then animate the Hadouken, and finally add the "Hadouken!" sound.

Ryu Throwing Position
You want Ryu to fire Hadoukens when users click on him. More specifically, when the mousedown event is detected, you want Ryu to lean forward. When the user releases the mouse, Ryu should return to his still position.

Follow the same procedure you did before by first proving that your events are firing. This time, though, instead of using alert(), use console.log(). Because you're testing mouseup and mousedown behavior, and having to click to dismiss an alert would interfere with this. Also, put in placeholder comments about the behavior you want in your event callback functions. Change app.js to look like this:

```
$(document).ready(function() {
  $('.ryu').mouseenter(function() {
    $('.ryu-still').hide();
    $('.ryu-ready').show();
  })
  .mouseleave(function() {
    $('.ryu-ready').hide();
    $('.ryu-still').show();
  })
  .mousedown(function() {
    console.log('mousedown');
    // play hadouken sound
    // show hadouken and animate it to the right of the screen
  })
  .mouseup(function() {
    console.log('mouseup');
    // ryu goes back to his ready position
  });
});
```

The console will now log both the mousedown and mouseup events. Save these changes, then return to Chrome. This time, before refreshing the page, open Developer Tools and click on the Console tab. Then, reload the page and try clicking on Ryu. You should see a message logged in the console for the mousedown and mouse up events.

Focus next on swapping out Ryu images on mouse down — animating the Hadouken and playing the Hadouken sound will come later. Hiding and displaying looks like it should work here. By definition, when you click on Ryu, the mouse already has to be over him, so that means you'll need to hide div.ryu-ready, not div.ryu-still. Then you'll want to show the Ryu throwing and Hadouken images. Alter your .mousedown event handler so it looks like this:

```
.mousedown(function() {
    // play hadouken sound
    $('.ryu-ready').hide();
    $('.ryu-throwing').show();
    $('.hadouken').show();
    // animate hadouken to the right of the screen
  })
```

Note that the comments in our code to remind us to implement the Hadouken sound and animation are retained. Save these changes, then go back to the browser, refresh the page and verify the behavior. If you release the mouse while still hovering over Ryu, he'll remain in his throwing position because we haven't implemented the mouseup event handler yet. Let's take care of that now. Modify .mouseup() so it looks like this:

```
.mouseup(function() {
  $('.ryu-throwing').hide();
  $('.ryu-ready').show();
});
```

Save these changes, then return to the browser to verify everything works. Note that when the mouse is released, although Ryu returns to his ready position, the Hadouken doesn't disappear. This is a good thing because you will animate across the screen, and then want it to disappear.

You've made great progress, so take a moment to add and commit your changes. Here's what app.js should look like at this point:

```
$(document).ready(function() {
  $('.ryu').mouseenter(function() {
    $('.ryu-still').hide();
    $('.ryu-ready').show();
  })
  .mouseleave(function() {
    $('.ryu-ready').hide();
    $('.ryu-still').show();
  })
  .mousedown(function() {
    // play hadouken sound
    $('.ryu-ready').hide();
    $('.ryu-throwing').show();
    $('.hadouken').show();
    // animate hadouken to the right of the screen
  })
  .mouseup(function() {
    $('.ryu-throwing').hide();
    $('.ryu-ready').show();
  });
```

Animating the Hadouken

To animate the Hadouken, use jQuery's animate() method. The animate method allows you to specify style properties and values, and jQuery will handle transitioning from the current settings to the new ones in a specified interval of time.

To animate the Hadouken's motion from Ryu's hands to the right of the screen, you will animate the left property on div.hadouken. When div.hadouken reaches its right most point, you'll use a callback function (that is, a function that gets executed after a prior function completes) to .hide() the div.

Inside of app.js, replace $('.hadouken').show(); and the placeholder comment in the mousedown method call about animating the Hadouken, with the following code:

```
$('.hadouken').show().animate(
  {'left': '1020px'},
  500,
  function() {
    $(this).hide();
    $(this).css('left', '520px');
  }
);
```
Before previewing the animation in the browser, go over each of the parameters supplied. If you glance back at main.css, you'll see that the .hadouken class' left property is set to 520px. The first parameter says to animate our jQuery object so its left value is 1020px (500px to the right). The 500 (milliseconds) is how long you want this animation to take. Finally, the anonymous function at the end will be called when the animation completes. It says to hide the element with the .hadouken class then set its left property back to its original value of 520px.

Make sure to save your changes and then preview the page in Chrome. When you click on Ryu, you should see the Hadouken fire. If you click on Ryu repeatedly, however, you'll notice that the Hadoukens no longer originate from his hands. This is because the previous animation hasn't completed when you're retriggering the lever.

There are different ways of dealing with this bug, but you'll use jQuery's .finish() method. .finish() completes all currently running animations on an element. Add the .finish() method immediately after the section of code where you select the .hadouken in the mousedown method. Modify your code so it looks like this:

```
$('.hadouken').finish().show()
  .animate(
    {'left': '1020px'},
    500,
    function() {
      $(this).hide();
      $(this).css('left', '520px');
    }
  );
```
Playing the Hadouken Sound

Your nearly finished coding the responses wanted when users click on Ryu. Now you need to trigger the sound. For this, you're going to write a named function that you'll call on the line with your comment about audio.

A dive into HTML5 audio is beyond the scope of this project, so some basic code will be provided. If you're curious about HTML5 audio, check out the Mozilla Developers Network's articles about the audio element and using HTML5 audio and video.

First, you need to embed an audio element to main.html. Inside of that file, right after the closing tag for div.main, add <audio id="hadouken-sound" src="sound/hadouken.mp3">. Save your changes to this file.

Now inside app.js, the bottom of the file, after the document ready function, add the following code:

```
function playHadouken () {
  $('#hadouken-sound')[0].volume = 0.5;
  $('#hadouken-sound')[0].load();
  $('#hadouken-sound')[0].play();
}
```

When this function gets run, it will load and play the sound file indicated in audio#hadouken-sound. Volume values can range from 0 to 1. Set the volume to 0.5, so it's audible but not blaring. The final step for your audio effect is to call playHadouken within the body of the .mousedown() code block. Modify the mousedown code so it looks like this:

```
.mousedown(function() {
  playHadouken();
  $('.ryu-ready').hide();
  $('.ryu-throwing').show();
  $('.hadouken').finish().show()
  .animate(
    {'left': '1020px'},
    500,
    function() {
      $(this).hide();
      $(this).css('left', '520px');
    }
  );
})
```

When mousedown fires, it will now call the playHadouken() function. Save this change, then refresh the page in your browser. Fire a Hadouken, and this time you should hear sound.

Take this opportunity to add and commit the changes you've made. If your code isn't working, note that this is what app.js should look like at this point:

```
$(document).ready(function(){
    $('.ryu').mouseenter(function(){
        $('.ryu-still').hide();
        $('.ryu-ready').show();
    })
    .mouseleave(function() {
        $('.ryu-still').show();
        $('.ryu-ready').hide();
    })
    .mousedown(function(){
        playHadouken();
        $('.ryu-ready').hide();
        $('.ryu-throwing').show();
        $('.hadouken').finish().show().animate(
            {'left': '1020px'},
            500,
            function(){
                $(this).hide();
                $(this).css('left', '520px');
            });
    })
    .mouseup(function(){
        $('.ryu-throwing').hide();
        $('.ryu-ready').show();
    });

});

function playHadouken() {
        $('#hadouken-sound')[0].volume = 0.5;
        $('#hadouken-sound')[0].load();
        $('#hadouken-sound')[0].play();
    }
```

You're almost done with this project. In the next assignment, you'll be responsible for implementing an event listener that makes Ryu "look cool" when the user holds down the "x" key.

STREET FIGHTER KEYDOWN CHALLENGE

In this assignment, you'll need to add code to app.js that will cause Ryu to switch to his "looking cool" pose when the user presses the "x" key. When the user releases "x", Ryu should go back to standing still.

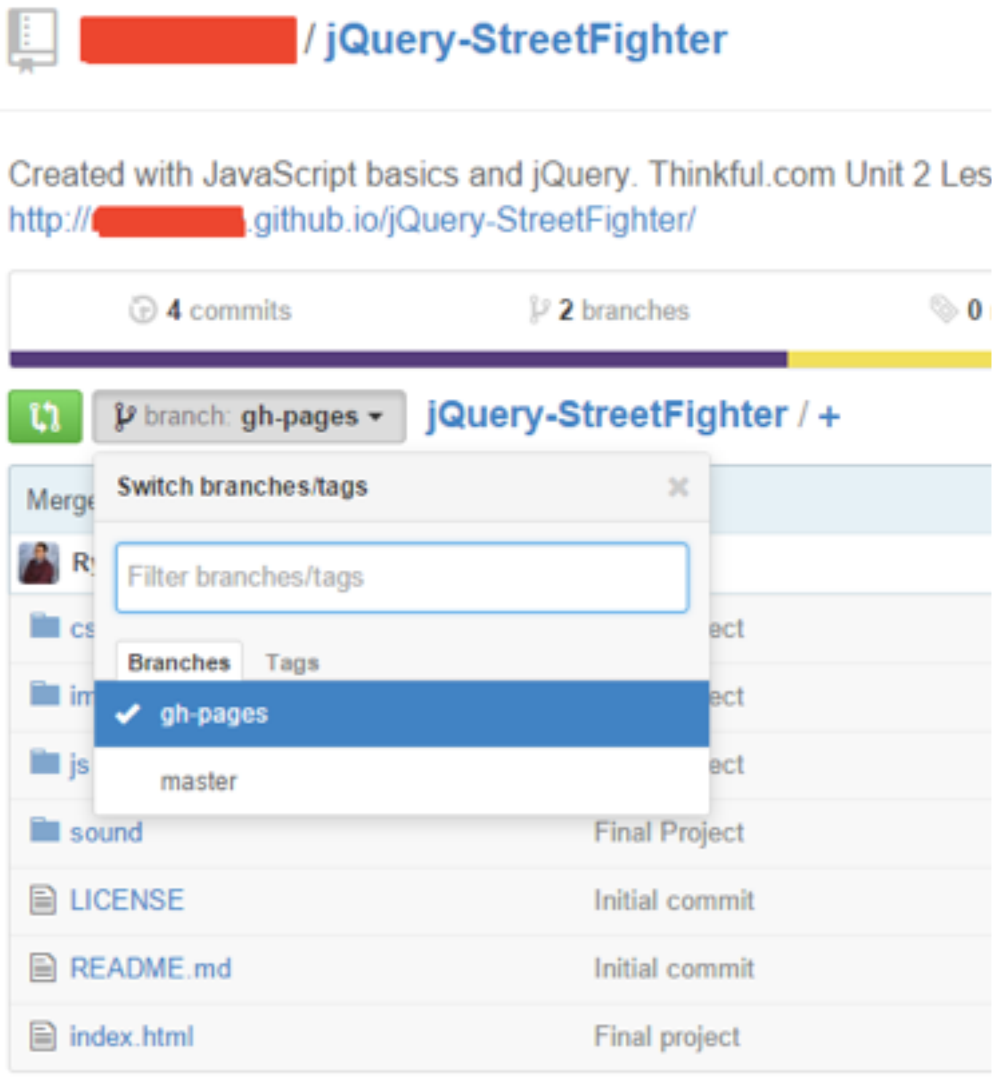
To complete this assignment, you'll need to use the .keydown() method from jQuery. The image you'll need for this is at images/ryu-cool.gif in the project's folder. Between looking at the API docs for keydown and consulting StackOverflow for additional advice, you will be able to glean how to use keydown. Some hints: when you use the keydown method, you need to check to see if the key that the user pressed down is the specific one you're looking for. The key code for "x" is 88.

This portion of the lesson is for you to apply the ideas you learned about showing and hiding divs (and making divs with background images) in the previous three assignments to implement the x-key functionality. Although you're still early in your career as a front end web developer, you've got enough experience at this point to feel comfortable enough to teach yourself new things, like the keydown() method for jQuery. Indeed, the ability to rapidly learn new programming techniques is a core skill for professional programmers.

In addition to the interactive features described above, you should provide some text in the .main container div that tells users how to control Ryu. Ideally, this text should be located within the white .main container.

When you've completed this assignment, be sure to add and commit your changes to git. You should also publish this page using GitHub Pages and be sure to share a link to your project with your fellow students and mentor.

To publish this project using GitHub pages first push your most recent changes to GitHub. Then open the repository in GitHub and create a new branch called gh-pages. Afterwards you should be able to view the project using the url http://yourusername.github.io/repositoryname/. If your main html file is still main.html you will need to add it to the end of the url. If your main html file is index.html you do not have to specify it at the end of the url. By default servers look for the index file by default if no file is specified in the url.



[redacted] / jQuery-StreetFighter

Created with JavaScript basics and jQuery. Thinkful.com Unit 2 Les
http://[redacted].github.io/jQuery-StreetFighter/

cs	ect
im	ect
js	ect
sound	Final Project
LICENSE	Initial commit
README.md	Initial commit
index.html	Final project

JQUERY SHOPPING LIST

In this lesson, you'll design and code a project from scratch: a shopping list app. The basic requirements for this app are:

Users should be able to enter items they need to purchase.
Users should be able to check and uncheck items on the list (you don't necessarily need to use checkboxes for this, but there should be a way of visually distinguishing between purchased and unpurchased items).
Users should also be able to permanently remove items from the list.

It will be up to you to come up with the look and feel of your app. In the next lesson, you will get a clear idea of what you want to build before you start building. Next, you will build an initial version of your app that demonstrates the layout and style, but does not have the interactive elements. Finally, in the third part of this project, you will use jQuery and JavaScript to write the code that will make your app interactive.

This project will be the most challenging yet because you'll be almost entirely on your own and bringing together everything you've learned so far. You'll get to practice designing and wireframing, HTML, styling with CSS, and using jQuery events to add interactive behaviors. To get a sense of what's possible, have a look at what previous Thinkful students have built (and while you can look under the hood of these projects to see strategies others have used - this is a key programming skill! - please refrain from copy-pasting code from these projects as that will not help you develop your own skills):

This project meets the basic requirements of the assignment.
http://www.katecbrown.com/shoplist/

This one lets you reorder your list items.
http://luketheterrible.github.io/ShoppingList/

This one lets you enter quantity and prices for your items.
http://mplamurphy1996.github.io/ShoppingCart/

Here's an example of an app with a particularly nice design.
http://kyreeswilliams.github.io/ShopList/

By completing this project, you'll solidify the skills you've learned so far in this course, giving you a solid foundation as you move to JavaScript in Unit 3.

SHOPPING LIST : DESIGN & WIREFRAME

http://webdesign.tutsplus.com/tutorials/a-beginners-guide-to-wireframing--webdesign-7399

https://balsamiq.com/products/mockups/

Spend up to an hour sketching your app and detailing the functionality. While this is an opportunity to express your creativity, make sure you're doing so through the eyes of the user: your app should ultimately be easy to use and that goal should guide your design decisions. At the very least, you'll need to design an interface for users to enter shopping list items (which means a text box) and a way for users to add and remove items (for instance, with add/remove buttons) as well as check and uncheck them. You should also think about how items will look when they're checked off vs. not checked off.

When you've completed your designs, if you'd like feedback from the community, post a picture or pdf of your sketches. Ideally, design is an iterative process where you get feedback from users and other stakeholders (for instance, clients) on your ideas that you then take into account before you've sunk time into coding up your app.

Steps:
Create a detailed wireframe (either by hand or with a wireframing tool such as Balsamiq) of a web page that will achieve the desired user experience.
Share the wireframe with your mentor and fellow students. To do this, you will need to take a picture or screenshot of your wireframe and post it to a photo sharing site, such as Imgur, and submit the URL below. Be sure to give a brief description explaining the user experience of your app to your mentor and peers so they can assess whether or not they think your app design works.

SHOPPING LIST:(More Mastering HTML / CSS) + Finish

In this assignment, you'll need to create a non-interactive version of your app by hard coding in a list of items ("hard coding" here simply means putting your shopping list items directly in the HTML, rather than using text inputs and add buttons to get them from the user). You'll focus on writing the HTML and CSS that will position and style your app so it looks like the design you created in the previous assignment.

You'll want to write styles for each of the states that list items can be in. For example, you might have a hard coded list of 5 shopping list items, with the top three not checked off and the bottom two checked off. Although you should go ahead and write the HTML and styles for your text input, you don't need to do anything with text inputs at this point — you'll tackle that issue in the next assignment.

Use this assignment as a way to improve your knowledge of positioning. This is probably the hardest HTML/CSS topic to master, and it takes practice. By creating designs first, and then coding them up, you'll be forced to deal with positioning, which is a good thing (even if it causes some productive frustration). And be sure to check that your layout behaves as expected when you resize the browser. Try it at very small and very large sizes.

Do make sure to initialize a git repo for this new project, and when you've completed the non-interactive version of your app, be sure to add and commit your files and then share the url for mentor and peer feedback.

In this assignment, you'll complete your shopping list app. You'll need to write JavaScript code that handles appending new items to the shopping list and allows users to check off, uncheck and remove items. You will be completing this project independently.

Tips:
-You'll likely need to use the .val() and .prepend() or .append() methods.

Adding an event listener to items that don't exist on the page yet can be achieved by including the optional selector parameter to the .on() function.

```
$('ul').on('click', 'li', function(event){
  // fires when any LIs are clicked on
  // including LIs that aren't on the page when it is initially loaded
});
```

-Additionally, you will likely need to use jQuery's this.
-Work on one feature at a time, breaking it up into its smallest parts, and don't start in on a new feature until you've got a working version of the current one.

When you've completed this project, be sure to add and commit your changes on GitHub. You should also publish your app with GitHub Pages and share a link with your fellow students and mentor.

JAVASCRIPT GOALS

Understand the concept of control flow
Know how to implement control flow via conditionals and loops
Understand what "truthy" and "falsy" values are in JavaScript
Work with a new data structure: arrays

Eloquent JavaScript http://eloquentjavascript.net/00_intro.html

Note that you should only read up to the paragraph "Code and What to Do with It". At that point, the chapter begins to describe the layout of the broader book, which (although it's a great resource!) you won't be looking at again in this course.

NUMBERS AND ARITHMETIC OPERATORS AND COMPARATORS + STRINGS

In this assignment, you'll get an introduction to numbers, and arithmetic operators and comparators in JavaScript. Arithmetic operators are things like: +, -, *, and /. They're used to compute values. Comparators are things like < and >= (that is, less than, and greater than or equal to). Comparators are used to compare values and they return either true or false depending on whether or not the expression is true or false.

Numbers
Numbers are one of the data types defined in JavaScript. You need numbers to perform arithmetic operations. There are five arithmetic operators in JavaScript, each of which is represented by operators:

multiplication (*)
division (/)
addition (+)
substraction (-)
modulus (%)
Operator precedence
Each JavaScript operator has an operator precedence which controls the order of execution when two or more operators are used in a single expression. "PEMDAS" is used to remember the order of operations. It stands for "Parenthesis, Exponents, Multiplication/Division, then Addition/Subtraction". This tells you the order in which operations should be performed: Parentheses being first and addition/subtraction last.

Parantheses (round brackets) can also change the order of evaluation. Consider the expression 16 / 2 + 3 * 2 . The numbers you perform operations with are called operands. By applying PEMDAS rules the result would be 14, but the order of operations can be changed simply by adding paranthesis ((16 / 2) + 3) * 2 with the outcome 22.

Modulus
The modulus operator perfoms an implied division and returns the remainder. For example 35 % 9 produces a result of 8 which is the remainder after dividing 35 by 9.

Comparators
The comparison operators are available for comparing any two data types:

> greater than
< less than
== equality (converts the operands if they are not of the same type, then applies strict comparison)
=== strict equality (if two operands are strictly equal without type conversion)
!= not equals
!== strict not equals (if two operands are strictly not equal without type conversion)
>= greater or equal
<= less or equal
Try it!
Using the comparators above, write an expression in the JavaScript console that evaluates to "true." Afterward write an expression that evaluates to "false."

What is the % operator called and what does it do?

The % operator is the modulus operator in JavaScript and it returns the remainder from dividing the two arguments.
What is the order of arithmetic operations in JavaScript (hint: PEMDAS)?
Parenthesis, Exponents, Multiplication, Division, Addition, then Subtraction. To see a more detailed ordered list visit https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence.
Make sure you understand what the following comparators do in JavaScript:
< (less than), > (greater than), === (equality without type coersion. In other words, if using the triple equals, the values must be equal in type as well), <= (less than or equal to), >= (greater than or equal to), !== (not equal).

Comparisons
Checking for matching strings is very easy if you apply == ("double equals") . != ("Not equals") returns true if there is a mismatch (case counts). As an example, "Times Literary Supplement" != "times Literary Supplement" will evaluate to true.

Special characters
A special character is a sequence of characters that cannot be typed directly, and must first be cleared with a backslash "\". For example: "\"I'm happy to go through the environmental assessment with the planner,\" volunteered Martin". The backslash in this example is used to clear, or escape, the double quotes so they aren't interpreted as being the end of the string. The backslash is also often referred to as an escape character.

You can review other common special characters below:

\t tab ("Today\\Tomorrow\t: late afternoon summer storms")
\n new line ('Time will say nothing but I told you so\n\r' + "Time only knows the price we have to pay\n")
\r carriage return (a carriage return is a type of new line)
\' start/end of string
\\ backslash
\uHHHH where HHHH is a 4-hexadecimal code to embed a unicode string

VARIABLES

When creating variables, you have to comply to naming rules:

You can use numbers inside variable names, but the name cannot begin with a number (legal example: key2success, illegal example: 4you)

You cannot have mathematical or logical operators inside variable name (illegal example: cats+dogs)

You cannot have punctuation marks of any kind other than underscore _ (legal example: $price , alice_in_wonderland ; illegal example: car#number)

You cannot have spaces in a variable name (illegal example: Star Trek)

Variable names must not be JavaScript keywords, however they can contain them (illegal example: var , for ; legal example: forThe)

CONDITIONALS AND TRUTHINESS

```
$(document).keydown(function(e) {
  if (e.keyCode == 88) {
    $('.ryu-action').hide();
    $('.ryu-cool').show();
  }
})
```

You may not have realized it at the time, but you were using control flow. The .keydown() method detects any and all keydown events, and you used an "if" statement to say, “Do the Ryu looking cool behavior only if it was the x key that was pressed down.” Implicitly, you were also saying, if it’s a key other than “x”, don’t do anything.

else if

```
if (score >= 90) {
        console.log('You got an A.');
} else if (score >= 80) {
        console.log('You got a B.');
} else if (score >= 70) {
        console.log('You got a C.');
} else if (score >= 60) {
        console.log('You got a D.');
} else {
        console.log('You got an F.');
}
```

A list of falsy values can be seen below:

Boolean value false
the empty string ""
the number 0 or -0
null
undefined
NaN (not a number)

Other values which are not included in the list above are truthy. Therefore, the values will be translated to true when ran in a logical context. This includes empty arrays, the Boolean true and string representations of values. Examples of truthy values in JavaScript follow:

```
if (true) {
  console.log("truthy");
}

if ({}) {
  console.log("truthy");
}

if ("false") {
  console.log("truthy");
}
```

Try It!
Create an if statement that checks that the length of your name is more than 10. If it is true, print the message "More than 10 letters!" Otherwise print, "Less than 10 letters!" Create a variable to store your name.

Solution:

```
var name = "Jane Doe";
if (name > 10 ) {
  console.log("More than 10 letters!");
  }
else {
  console.log("Less than 10 letters!");
}
```

Try It!
Correct the mistakes in the following code:

```
if (12 = 12 ); {
  console.log("More than 10 letters!")
  }
else {
  console.log("Less than 10 letters!");
}
```

Solution:

```
if (12 === 12 ) {
  console.log("More than 10 letters!");
  }
else {
  console.log("Less than 10 letters!");
}
```

Comprehension Check:
Make sure you understand and memorize which values are falsy in JavaScript.

```
// Outputs: "Falsy."
logTruthiness(false);

// Outputs: "Falsy."
logTruthiness(null);

// Outputs: “Falsy."
logTruthiness(undefined);

// Outputs: “Falsy."
logTruthiness(NaN);

// Outputs: "Falsy."
logTruthiness(0);

// Outputs: "Falsy."
logTruthiness(“");
```

LOOPS AND ARRAYS

Much of programming includes working with collections of objects. Frequently, you’ll need to perform a routine on each member of a collection. This is where looping (also known as iterating) comes in. Looping allows you to say things like “For each item in this collection, do X”.

In this lesson you’ll learn all about two kinds of looping in JavaScript: for and while loops, and you’ll also learn about a new data structure: arrays (note that the term “data structure” is used advisedly here, arrays actually fall into the object category and are a particular kind of object in JavaScript). Arrays are used to store lists of items and it’s common to loop over an array of items, which is why it is being covered.

Loops are a very important part of any programming language. Programs need to repeat a sequence a number of times until a condition is fullfilled or when a value has changed. To address this need, JavaScript introduces the while , do and for keywords.

The for loop

You will use the for loop when you know how many times you want a code to execute. The syntax for the for loop is as follows:

```
for ([initialization]; [condition]; [final-expression])
  statement
```

There are variations, such as:

```
for ( variable = initialValue; variable < finalValue; variable = variable + increment) {
  // code goes here
}
```

OR

```
for ( variable = initialValue; variable >= finalValue; variable = variable - increment) {
  // code goes here
}
```

The sections between parantheses are separated by semicolons, which are mandatory (even if any of the section is empty). First section is a variable of your choice, second is an expression that evaluates to true or false. If it evaluates to true, the code executes. The third section is the loop modification section where you can increment (increase) or decrement (decrease) the loop variable. It only executes after the loop code runs for each loop.

Try it!
In the JavaScript console, run the following code:

```
for (var i = 1; i < 9; i++) {
   console.log(i);
}
```

Solution:

This for loop logs the numbers 1 to 8, starting at 1 (i = 1), incrementing by 1 (i++), and stopping before 9 (i < 9).

The while loop

The while loop is used when you do not know how many times you want to execute the code. You only know that you want to execute the code as many times as a condition is true. When that condition is false, you will stop executing the code. The syntax can be seen next:

```
while ( condition ) {
  //code goes here
}
```

Try it!
In the JavaScript console, run the following code:

```
var appleCount = 10;
while ( appleCount > 0 ) {
  console.log(appleCount);
  appleCount--;
}
```

Solution:

The while loop above will continue to iterate while the appleCount is greater than 0, the result should log the numbers 10 through 1. With each iteration (loop) the appleCount is decremented (decreased) by 1. On the 10th iteration, appleCount will become set to 0 and the while condition of appleCount > 0 is no longer true. This stops the execution of the while loop. It's important to note that if you add an expression that always evaluates to true, you will create an infinite loop, which means that the loop will execute until you force close the program.

Arrays

An array is a collection object that has a sequence of items that you can access and modify. You can assign the array to a variable then access the array values by an index, which is noted in square brackets. Array items are assigned a number based on the order they are listed within an array. This number is known as an indexer and can be referenced in your code. The first item in an array is assigned an index of "[0]".

There are various ways to create and populate an array, you will see the most common ones below:

1) An array can be initialized using the new keyword with no initial values, then add values later on. In the following example, an array is initialized with the new keyword which creates an instance of the Array object. Values are then added to the array.

```
var flowers = new Array();
flowers[0] = "rose";
flowers[1] = "lily";
flowers[2] = “tulip";
```

2) An array can be initialized with a known set of values. In this example, the array is initialized with the new keyword but this time the values are passed into the Array constructor.

```
var flowers = new Array("rose" , "lily" , “tulip");
```

3) An array can also be created by supplying the item list which is a shorthand for the above option.

```
var flowers = ["rose" , "lily", “tulip"];
```

To access the items of an array, an indexer must be used. Take note that JavaScript indexes start at 0, for example to get the third element of an array you would need to use an index of 2

(0 = 1st element, 1 = 2nd element, 2 = 3rd element, etc...).

```
var flowers = ["rose" , "lily", "tulip"];
console.log(flowers[2]);

//result: tulip
```

You can also modify the items in an array like so: flowers[2] = "wood lily";

To perform operations on array, you can use array methods. The Array object has the following methods which you will find useful:

concat Joins one or more arrays, as we can see in the example below:

```
var flowers = ["rose" , "lily", "tulip"];
var colors = ["red" , "blue", "yellow"];
var colorFlowers = flowers.concat(colors);

//result: ["rose", "lily", "tulip", "red", "blue", “yellow"]
```

indexOf Returns the index of an item in an array or -1 if the item was not found.

```
var flowers = ["rose" , "lily", "tulip"];
console.log(flowers.indexOf(“lily"));

//result: 1

console.log(flowers.indexOf(“azalea"));

//result: -1
```

lastIndexOf Searches from the end of an array for the last item in the array that meets the requirement and returns its index or -1 if the item was not found.

```
var flowers = ["rose" , "lily", "tulip",  "lily", "azalea"];
console.log(flowers.lastIndexOf(“lily"));

//result: 3

console.log(flowers.lastIndexOf(“daisy"));

//result: -1
```

pop Removes and returns the last element of the array

```
var flowers = ["rose" , "lily", "tulip",  "lily", “azalea"];
var lastFlower = flowers.pop();
console.log(lastFlower);
//result: azalea

console.log(flowers);
//result: ["rose", "lily", "tulip", “lily"]
```

push Adds a new item to the end of an array

```
var flowers = ["rose", "lily", "tulip", "lily"];
flowers.push('azalea');
console.log(flowers);
//result: ["rose", "lily", "tulip", "lily", “azalea"]
```

shift Removes and returns the first item in the array

```
var flowers = ["rose" , "lily", "tulip",  "lily", "azalea"];
var firstFlower = flowers.shift();
console.log(firstFlower);
//result: rose

console.log(flowers);
//result: ["lily", "tulip", "lily", “azalea"]
```

unshift Adds a new item to the beginning of an array and returns the new length

```
var flowers = ["lily", "tulip", "lily", "azalea"];
flowers.unshift("rose");
console.log(flowers);
//result: ["rose", "lily", "tulip", "lily", “azalea"]
```

reverse Reverses the order of items in an array and returns a reference to the reversed array

```
var flowers = ["rose", "lily", "tulip", "lily", "azalea"];
flowers.reverse();
console.log(flowers);
//result: ["azalea", "lily", "tulip", "lily", “rose"]
```

slice Returns a new array that is part of the existing array

```
var flowers = ["rose", "lily", "tulip", "lily", "azalea"];
var part = flowers.slice(1, 3);
// 1 is the starting index
// 3 is indicates the index to top at, but not include
console.log(part);
//result: ["lily", “tulip"]
```

sort Sorts the items in an array and returns the reference to the array

```
var flowers = ["rose", "lily", "tulip", "lily", "azalea"];
flowers.sort();
console.log(flowers);
//result: ["azalea", "lily", "lily", "rose", “tulip"]
```

splice Adds and removes items from an array and returns the removed items

```
var flowers = ["rose", "lily", "tulip", "lily", "azalea"];
// Remove 0 items from index 2 and insert 'daisy' at index 2
var removed = flowers.splice(2, 0, 'daisy');
console.log(flowers);
//result: ["rose", "lily", "daisy", "tulip", "lily", “azalea"]

console.log(removed);
//result:[]

// Remove 1 item from index 3
removed = flowers.splice(3, 1);
console.log(flowers);
//result: ["rose", "lily", "daisy", "lily", “azalea"]

console.log(removed);
//result: ["tulip"]

// Remove 2 items from index 0 and insert 'tulip', 'carnation', and 'iris' at index 0
removed = flowers.splice(0, 2, 'tulip', 'carnation', 'iris');
console.log(flowers);
//result: ["tulip", "carnation", "iris", "daisy", "lily", “azalea"]

console.log(removed);
//result: ["rose", “lily"]
```

toString Creates a string from the items in the array

```
var flowers = ["rose" , "lily", "tulip"];
console.log(flowers.toString());
//result: “rose,lily,tulip"
```

valueOf Returns the primitive values of the array as comma-delimited string

```
var flowers = ["rose" , "lily", "tulip"];
console.log(flowers.valueOf());
//result: ["rose", "lily", “tulip"]
```

Memorize the syntax "for" loops, and make sure you can explain in layperson’s terms how a "for" loop works.

A for loop starts with a counter, modifies the counter every time it runs, and repeats itself until it’s counter exceeds the condition.

Memorize the syntax "for while" loops.

html while (condition) { code block to be executed }
How would you write a for loop that decrements (counts down)?
for (var i = 10; i > 0; i--) { }

What is an array?

An array is an ordered collection of JavaScript values. A JavaScript array can hold values of any type, from Number to String to Function.

What is the syntax for creating an array?

[item1, item2, … , item n]

If you have var myArray = [1, ‘a’, ‘b’, 4], how would you access the second element?
Array indexing starts from 0, so the second element is at myArray[1] .

FIZZBUZZ CHALLENGE

In this assignment, you’ll build a FizzBuzz app from scratch. In the original game of Fizz Buzz, you count from 1 to a given number (let’s say 100). For each number if it’s not divisible by 3 or 5, you simply say the number. If it’s divisible by 3, instead of the number you say “fizz”. If it’s divisible by 5, you say “buzz”. And finally if it’s divisible by both 3 and 5, you say “fizz buzz”. So, counting from 1 to 15, you’d say: “1, 2, fizz, 4, buzz, fizz, 7, 8, fizz, buzz, 11, fizz, 13, 14, fizz buzz”.

This is admittedly not a very useful program, but as it turns out it’s a great small project to do, for two reasons. First, it brings together all of the programming fundamentals we’ve learned so far. To complete this project, you’ll need to use loops, conditionals and the modulo operator (and that’s about all the help you’ll get from us about implementing the JavaScript). Second, and as a consequence of the first point, FizzBuzz is a popular problem that interviewees will be asked to solve in technical interviews. If your goal is to become a professional developer, you definitely should know how to code up FizzBuzz. According to some commentators, the ability to code up FizzBuzz falls flat amongst the top half percent of programmers who actually know how to program!

The requirements for this app are as follows:

DO NOT spend any time designing this app or making it look pretty. We don’t want you to consider UX at all here, since the point is to build up your programming fundamentals.

At a minimum, you’ll need an HTML file and a JavaScript file. The HTML file should link to jQuery and to your application JavaScript file. The program should append each number (or its “fizz”/”buzz”/”fizzbuzz” substitution) to the body element of the DOM (On the page).

The program should print out each number (not 'one' for 1, but 1 for 1), from 1 to 100, replacing numbers divisible by both 3 and 5 with “fizz buzz”, those divisible by 3 with “fizz”, and those divisible by 5 with “buzz”.

You should use GitHub for this project, and when it’s done, publish it with GitHub pages and submit the link to your mentor below.

Because FizzBuzz is a common interview question, it’s easy enough to find ready made solutions with a quick Google search. You’re on the honor system here and you should keep in mind that copying and pasting a solution won’t help you to really master the skills that FizzBuzz requires.

INTRO TO FUNCTIONS

Functions store behavior that can be used to easily reuse code. Functions in JavaScript are very similar to functions in math: they take a value, called an "argument" (sometimes also called a "parameter" or "input"), and return a value as an output. Functions are useful for re-using code; instead of typing some block of code over and over again, you can write it once as a function and then call it whenever you need to. Proper use of functions allows you to write programs in the "DRY" style: Don't Repeat Yourself!

This assignment will introduce you to the concept of a function, and teach you how to write functions of your own. You'll learn the syntax and style to create well-formed functions.

What is a function?
Imagine you wrote the following math program. It's a calculator that tells you how much your living expenses are going to be for this month: it factors in the heating bill, the water bill, the rent, the food, some going-out fun money, and a 30% cushion, just in case.

```
var rent    = 1000,
    heating = 30,
    water   = 60,
    food    = 300,
    fun     = 200,
    season  = 'summer',
    income  = 3000,
    cushion = 0.3;

// Add some seasonal dependencies
if ( season == 'winter' ) {
    heating = heating * 7; /* I live in Rochester NY, this is no joke! */
}

// Add up your total expenses
var expenses = rent + heating + water + food;

// Only add the funmoney if you can still afford it!
if ((expenses + fun) < income) {
    expenses = expenses % fun;
}

// Add the 30% cushion to your expenses, and then
// subtract from your income.
var net = income - (expenses * (1 + cushion));

//
```

However, this code doesn't work. The function is executed the output is clearly wrong, so you need to debug. Track some variables through the program flow to see what's happening to it. Do this by outputing the value of the variables to the console with the console.log() function:

```
var rent    = 1000,
    heating = 30,
    water   = 60,
    food    = 300,
    fun     = 200,
    season  = 'summer',
    income  = 3000,
    cushion = 0.3;

// Add some seasonal dependencies
if ( season == 'winter' ) {
    heating = heating * 7;
    console.log("Heating: " + heating);
}


// Add up your total expenses
var expenses = rent + heating + water + food;
console.log("Expenses: " + expenses);

// Only add the funmoney if you can still afford it!
if ((expenses + fun) < income) {
    expenses = expenses % fun;
    console.log("Expenses with funmoney: " + expenses);
}

console.log("Expenses with cushion: " + expenses*(1+cushion))

// Add the 30% cushion to your expenses, and then
// subtract from your income.
var net = income - (expenses * (1 + cushion));
```

The code above added console.log() throughout the program so you can see what's happening to it. When you run the program in the browser (for example, in codepen.io, you see the following output in the console:

```
Expenses: 1390
Expenses with funmoney: 190
Expenses with cushion: 247
```

Now you can see what's wrong. Somewhere between the first expenses calculation and the addition of funmoney, something is incorrect. On line 24, expenses % fun should be expenses + fun. Change that line to make the program work as intended.

Notice that the same lines of code are present multiple times: console.log(). Additionally, this code will always run, even if you don't need to do debugging. You could turn it off by going through and commenting out each console.log() line, but in a large program that will quickly become tedius. What if you could have a "debug mode" that only executes console.log() if you want it to?

This is where functions come in handy. Consider the following code:

```
var DEBUG_MODE = false;

var debug = function(msg) {
   if (DEBUG_MODE == true) {
      console.log("DEBUG:", msg);
   }
}
```

This code creates two variables, DEBUG_MODE and debug. The first is in all caps to signify that it is an important global variable, meaning is intended to be used in any area of the program. This is just a convention for making code more readable by humans, the JavaScript interpreter doesn't actually care if your code is in all-caps.

The second variable, debug, actually points to a function that takes one argument, msg. The function checks to see if DEBUG_MODE is true, and if it is, it prints a helpful debugging message to the console. If you add this debug function to your code, it will look like this:

```
var rent    = 1000,
   heating = 30,
   water   = 60,
   food    = 300,
   fun     = 200,
   season  = 'summer',
   income  = 3000,
   cushion = 0.3;

var DEBUG_MODE = true;
var debug = function(msg) {
   if (DEBUG_MODE == true) {
      console.log("DEBUG:", msg);
   }
}

// Add some seasonal dependencies
if ( season == 'winter' ) {
   heating = heating * 7;
   debug("Heating: " + heating);
}


// Add up your total expenses
```

```
var expenses = rent + heating + water + food;
debug("Expenses: " + expenses);

// Only add the funmoney if you can still afford it!
if ((expenses + fun) < income) {
   expenses = expenses + fun;
   debug("Expenses with funmoney: " + expenses);
}

debug("Expenses with cushion: " + expenses*(1+cushion))

// Add the 30% cushion to your expenses, and then
// subtract from your income.
var net = income - (expenses * (1 + cushion));
```

And the output will look like this:

```
DEBUG: Expenses: 1390
DEBUG: Expenses with funmoney: 1590
DEBUG: Expenses with cushion: 2067
```

You know that the program works, so turn off the debugging messages with DEBUG_MODE = false;.

Notice that towards the end of the program there is the same bit of code twice. During the debug message and when you calculate the net variable, you use the code expenses * (1 + cushion). This can easily be refactored into a function:

```
function addCushion(x, cush) {
   if(cush == null){
      cush = 0.3;
   }
   return x * (1 + cush);
}

// Replace the aforementioned lines with the following:
debug("Expenses with cushion: " + addCushion(expenses, cushion));

var net = income - addCushion(expenses, cushion);
```

This example shows a slightly different way to create functions. Here, the function addCushion() is defined at parse-time, while debug() is defined at run-time. Parse-time definition is useful because you can define your function at the end of the file, and yet you can call the function at the top of the file. This is possible because the JavaScript VM first parses the file and stores any functions and variables it needs into memory, and after everything is parsed, then it runs the program. (If this seems confusing, you can read a more lenghty explanation at StackOverflow. For most cases in this course, the distinction won't matter, but it's important to know of the difference.)

The addCushion() function also shows you how to set default values for arguments. Here, cush is set to a default of 30%. If you only give the function one argument, like addCushion(expenses), then it would use 0.3 for the value of cush.

Finally, this example shows you an important step in the programming process: refactoring! Not every programmer will admit it, but when writing code, they tend to frantically mash on the keyboard until the program works. The code is rarely "pretty" when first written, but after it works, you can go back, do some code review, and see how to better organize the code.

Conclusion
Now you should have a good understanding of what functions are and how to use them.

Code review and refactoring is incredibly important. The next section will teach you more about functions and what you can do with them.

Explain why it's said that functions store behavior.

Because you can call a function later, allowing you to run whatever behavior (code) was stored in the function.

How do you invoke a function?

Using parenthesis after the function's name. Alternatively, you can use the .call() or .apply() methods on the function object.

What is an argument to a function?

An argument is a variable that is passed in to a function, allowing it to operate on a set of inputs.

Why do functions make your code more maintainable?

They package up code, allowing it to be handled in smaller, more manageable chunks. Also, it allows you to avoid duplicating code.

How do you assign a function to a variable?

var func = function() { };

What is the block in a function definition?

The block is the code enclosed in the curly braces. { }

What do functions that don't have a return statement evaluate to?

undefined

How do you return a value from a function?
return "value";

PRACTICE FUNCTIONS

Now that you know how to write basic functions, here are some exercises to get you geared up and writing more complex functions. It's important that you read each piece of code and try to figure out what it does, and even try typing it into CodePen or a JavaScript console.

FizzBuzz variants

First, you'll look at some different implementations of fizzbuzz and learn how they work.

If you're typing this into the console to practice it, replace cprint with console.log for now. If you're trying this in CodePen, scroll down to the "Emulate the Console in CodePen" to see the cprint function.

```
function fizzbuzz(max) {
  for ( var i = 1; i <= max; i++ ) {
    if (i % 5 == 0 && i % 3 == 0) {
      cprint("FizzBuzz");
    } else if ( i % 3 == 0 ) {
      cprint("Fizz");
    } else if (i % 5 == 0) {
      cprint("Buzz");
    } else {
      cprint(i);
    }
  }
}
```

This code loops fizzbuzz from 1 to max. Inside the function block, it creates a local loop with a local variable called i. These are called "local" because they are not accessible to code outside of the function block. This is called "scope" and what it does is denote where variables and functions are available for use. Here's another one:

```
function fizzbuzzer(max) {

  function check(n) {
    var msg = '';
    if ( n % 3 == 0 ) { msg += "Fizz" };
    if ( n % 5 == 0 ) { msg += "Buzz" };
    return msg || n;
  }

  for (var i = 1; i <= max; i++) {
    cprint(check(i));
  }

}
```

fizzbuzzer uses a locally-scoped function called check to see if the function should return "Fizz" or "Buzz". Being locally-scoped, you cannot call check outside of the fizzbuzzer block.

check uses a different fizzbuzz algorithm than you might have seen before. It builds up a string by appending "Fizz" and/or "Buzz". For n = 15, both if-statements will evaluate to true, and they will append both "Fizz" and "Buzz" onto msg. Thus, return msg || n (read: "return msg or n") will return "FizzBuzz". For n = 7, neither of the if-statements will evaluate to true, and msg will just be an empty string. In JavaScript, empty strings is a "falsey" value, so return msg || n will return n as it checks if msg is truthy, if it is return msg otherwise return n.

The second part of fizzbuzzer is a locally-scoped for-loop that outputs the result of check from 0 to max, quite like fizzbuzz above.

Try it!

Try moving the line var msg = ''; from inside the check function to just above it, but still inside fizzbuzzer. Notice how that changes the output. Can you explain why?

Closures

A closure is just a function that closes over another function. In the above example, fizzbuzzer is one kind of a closure. Here is another:

```
function adder(x) {
   return function(y) {
      return x + y;
   }
}

var add3 = adder(3);
var add5 = adder(5);

cprint(add3(5)); // 8
cprint(add5(5)); // 10
```

adder is a different kind of closure than fizzbuzzer because it actually returns a function, not a value. Each new adder function takes a variable x and returns a new function that takes a variable y, and then adds x to y. This is kinda like a "template" for creating functions, but you can think of many other ways to use closures in addition to the two listed here. If you're familiar with other programming lanuages, closures can be used to emulate "classes". Closures can be confusing, but that is often because they are trumped up with lingo and technical terminology. At their core, closures are just functions that close over other functions. They are very simple, but also very powerful.

Emulate the Console in CodePen

If you've been following along closely, you should have a pretty strong grasp of functions and their capability. Now, you are going to use the knowledge you acquired over the past two lessons to build a quasi-console, like the browser console, except your console will only be able to print code, it won't have an input box; the input functionality will be provided by CodePen.

First, go to CodePen.io and create a new pen. You'll need a bit of boilerplate HTML and CSS first. HTML:

```
<div id="console"></div>
```

This will simply act as a bucket to accept our console's output. And the CSS:

```
pre {
  border-bottom: 1px solid lightgrey;
  padding: 0px 5px;
  margin 0;
  height: 1em;
  line-height: 0em;
  font-size: 1em;
  vertical-align: middle;
}

#console, html, body {
  margin: 0; padding: 0;
}
```

This will make the console look presentable. Feel free to add other styles to make it even more stylish.

Now, for the JavaScript. First, you'll need to create a Printer function that will handle all of the printing functionality. The first thing it will need to do is count the lines outputted, so you'll need a lineCount variable.

```
function Printer() {
    var lineCount = 1; // start counting at 1
}
```

Next, it will have to actually be able to print a line, so we need a printLine function:

```
function Printer() {
  var lineCount = 1;

  this.printLine = function(m) {
    var message = lineCount + ': ' + m; // prepend lineCount to our message
    lineCount++; // increase our counter
    return message; // spit out the message
  }
}
```

You'll notice that the printLine function is stored as a variable called this.printLine. You need to do this because JavaScript is an object-oriented language. When you wrote function Printer(), you actually created a template for a new object. The variable lineCount is called a property of

Printer. The function printLine is called a method of Printer. You'll learn more about objects later, but for now you need to understand that you need this. to signify that printLine is attached to the Printer object. You can access printLine by writing new Printer().printLine().

This function looks pretty good. Each time you call printLine, it will prepend the lineCount to the message, increase the lineCount, and then return the message. But how do you actually get this to display in the console div?

The answer is that you need to actually create a new element and append it to the div each time you call printLine.

```
function Printer() {
  var lineCount = 1;

  this.printLine = function(m) {
    var message = lineCount + ': ' + m;

    var n = document.createElement("pre"),    // create a new 'pre' element
        t = document.createTextNode(message); // create a text node to hold our message

    n.appendChild(t); // append your text to the pre element
    document.getElementById("console").appendChild(n); // append your element to the
#console element

    lineCount++;
  }
}

new Printer().printLine('hi');
```

Now, the function will print out the message to the console div.

What if you want to print out multiple things at once? For example, what if you want to write new Printer().printLine("Hello", name, "how are you?"); where name is the name of the user? You can do this my creating a function with multiple arity, that is a function that can take multiple arguments.

As an object-oriented language, every function in JavaScript is actually an object. The objects come with certain pre-established properties. One of those properties is called arguments which stores all of the arguments that you pass into the function as an array. You can do whatever you want with these arguments, but here you are going to loop through them, and build up a string like you did in the fizzbuzzer function called check.

```
function Printer() {
  var lineCount = 1;

  // remove the m variable from the top of the function, and use it to replace
  // the message variable
  this.printLine = function() {
    var m = lineCount + ': ';
    // start the loop
    for (var i = 0; i < arguments.length; i++) {
      // for each argument, add it to the variable m plus a space afterwards
      m += arguments[i] + ' ';
    }

    var n = document.createElement("pre"),
        t = document.createTextNode(m);

    n.appendChild(t);
    document.getElementById('console').appendChild(n);

    lineCount++;
  }
}
```

Now you can print out as much stuff as you want in a single call to printLine simply by passing multiple arguments into the function.

You are almost finished with the console - You just have one more use-case. What if you want to have multiple consoles on the page? Right now, you can only have one because the line that says getElementById('console') binds my Printer to the console div. But what if you want two different printers, each printing a different version of fizzbuzz?

You can actually add an argument to Printer that does just that, like so:

```
// add the new argument divID here
function Printer(divId) {
  var lineCount = 1;

  this.printLine = function() {
    var m = lineCount + ': ';

    for (var i = 0; i < arguments.length; i++) {
      m += arguments[i] + ' ';
    }

    var n = document.createElement("pre"),
        t = document.createTextNode(m);

    n.appendChild(t);
    document.getElementById(divId).appendChild(n); // and use it here

    lineCount++;
  }
}
```

Now, you can create new Printers whenever you want:

var consolePrinter = new Printer("console");
var cprint = consolePrinter.printLine;
This is the same cprint function that you used for the above examples. When you create a new Printer object with the new keyword, it's called "instantiating" and object because you are making a new "instance" of the object called Printer. Get it?

Try it!

Create multiple printers on a single page. If you get stuck, take a look at this CodePen to help you.

Should you put a semicolon after each line of code in a function block (i.e., between the curly braces)?

Yes.

What does it mean when it is said that the var keyword creates a variable in the current scope?

The var keyword creates a variable that only exists within the current executing function. When that function exits, you no longer have access to the variable - Except for closures, which are a more complicated case.

Describe the idea of scope, in your own words.

Scope is the collection of valid “names” or variables defined for the current executing context.
Why is it a bad idea to omit the var keyword when defining a variable in a function?
It defines the variable on the global scope, which is available to all functions.

Does JavaScript have block level scope?

No, it does not.

FIZZBUZZ REFACTOR CHALLENGE

Earlier in this unit you built a FizzBuzz app. As per our instructions, your app was required to list out the numbers from 1 to 100, using the FizzBuzz rules for substituting in multiples of 3, 5 and both 3 and 5 (a.k.a. 15). For this assignment, take your original FizzBuzz app and refactor it so it uses a function and meets the following requirements:

When the page loads, the user should be prompted to supply a number. The easiest way to do this would be with the prompt() function, but you should also feel free to create a simple text input with text telling the user to input a number.

You'll need to write one named function that takes an integer as argument, and then counts from 1 to the argument value, substituting "fizz", "buzz", and "fizzbuzz" accordingly.

You'll need to convert the value the user supplies from a string to a number. Remember that the value that you get from prompt() or the val() on your form will by default be a string. To convert this to an integer you can use the + operator to convert a string to an integer. For instance, if you had the string "22", you could convert that to a number with +"22".

You will need to validate user input to ensure the user is submitting a number. You can use the parseInt() function in javascript to convert a string into an integer. For example, parseInt('597a') will return 597 where parseInt('hello') will return NaN. Read more about parseInt() on MDN.

Optionally, you can also write code to ensure that the user has not supplied a decimal value. For an easy way to do this, check out this answer on Stack Overflow. http://stackoverflow.com/questions/2304052/check-if-a-number-has-a-decimal-place-is-a-whole-number/2304062#2304062

DO NOT spend any time thinking about design for this app. Like the first version, the goal here is not to create a pretty app, but instead to hone your JavaScript skills.

As usual, use Git to store your changes. When you've completed this assignment, push it up to GitHub pages, and share a link with your mentor and fellow students.

HOT OR COLD APP CHALLENGE

In this lesson, you've learned about the syntax of functions, how you can use them to avoid repetitious code, and the peculiarities of scope in JavaScript. In the final assignment for this lesson, you're going use your knowledge of functions and jQuery to build a "Hot or Cold" game. In this game, the computer randomly selects a number between 1 and 100, and the player then tries to guess the number. The player gets feedback for each guess – "hot" if their guess was close, and "cold" if their guess was far. When the user guesses the secret number, the app lets them know, and they'll have the option to start a new game.

For this project, you will be supplyed with HTML and CSS files for the app, and you'll only be responsible for writing the JavaScript that brings these static files to life. There are two reasons for this approach. First, learning to work with functions and JavaScript (alongside jQuery) is challenging enough, and you should focus on JavaScript rather than on design and layout.

Second, this approach simulates what it's like to collaboratively work on a code base. Oftentimes it's the case that you won't be starting from scratch, and instead inherit existing code to which you need to add features. Being able to take existing code, figure out how it works, and write new code that integrates is a key skill, and it takes practice to get good at this.

In the past, students coded this project from scratch. The starter files you're using for this project are a modified version of Thinkful student Jeya Karthika's Hot or Cold app. She has a good eye for design, and a nice simple layout that will be a good starting point for this project. http://thinkful-fewd.github.io/hot-or-cold-starter/

The mandatory requirements for this app are as follows:

You must use the HTML and CSS supplied. Once you've completed the project, you may choose to alter the layout and styles, but stick with the templates supplied initially. Note that the index.html file already links to the CSS files, app.js file and jQuery. You should write your JavaScript code in app.js. Also, note that there is a small amount of code in app.js - there's a $ (document).ready() block with code that handles displaying and hiding the instructions for the game.

When the page loads, JavaScript should start a new game. Since you'll need to be able to start a new game when the user clicks the "New Game" button, you'll want to create a newGame function that does everything necessary to start a new game.

When a new game starts, a secret number between 1 and 100 should be generated that the user will have to guess. You should write a named function that takes care of this. You should try to start a new game without refreshing or reloading the page.

The user should get feedback about each guess – if it was too low, too high, or just right. This means that you'll need to write a named function that takes a user guess and determines which feedback to provide.

Initially, you shouldn't worry about telling users if they're getting "hotter" or "colder" relative to their previous guess. Instead, you can use absolute values. For instance, you might decide that if a user is 50 or further away from the secret number, they are told they are "Ice cold", if they are between 30 and 50 they are "cold", if they are between 20 and 30 they are warm, between 10 and 20 hot, and between 1 and 10 "very hot". You can choose what the ranges are and what feedback you provide.

Feedback about the guess should appear in div#feedback. By default, when the page loads, the text in this field is set to "Make Your Guess!"

The game should track how many guess the user has made. Feedback about this should appear in span#count (which defaults to 0, when the page loads).

The game should also supply users with a list of the numbers they have guessed so far. The CSS for this game is set up in such a way that you can simply add each guessed number as an <li> to ul#guessList.

You’ll need to ensure that users provide valid inputs. Note that the guess text input field has the HTML 5 required flag set, so you won’t have to worry about blank guesses being submitted (if the user submits a blank guess, they'll be prompted to supply an input).

However, you will need to write code that ensures that the user has supplied a numeric input between 1 and 100.

The starter template already contains a button in the upper right hand corner for starting a new game, however, this button does not currently do anything. You’ll need to write code that allows users to start a new game without making additional calls to the server. Clicking “New Game” should trigger the JavaScript function that starts a new game.

These are the minimal requirements. If you’re able to meet these basic requirements and want to build in more advanced features, you can write code that provides users with feedback about their most recent guess in relation to the previous one. If the most recent guess is closer to the secret number, you would tell the user they are “warmer”, and if it’s further, you’d tell them they are “colder”. Note that for the first guess, you’d still need to provide absolute feedback, since they’re won’t be a previous guess to compare to.

STEPS

Fork the Repository: Make sure you’re signed into Github.com, then visit our repo and click on the “Fork” button in the top right. After your fork has been created, you’ll want to clone it to your local computer. https://github.com/thinkful-fewd/hot-or-cold-starter

Familiarize Yourself with The Layout and CSS: You’ll need to use jQuery in the project to listen for when users submit guesses. You should spend a few minutes looking over the HTML and CSS files and using Developer tools to inspect page element.

Break the App Logic Down Into Steps and Write Functions: You’ll need a newGame() function that does everything necessary to start a new game. This function will itself need to call other functions to take care of specific tasks—for instance, setting the randomly generated secret number. You should break the application logic down into discrete steps, then work on one step at a time.

Share When You’re Done: You should use Git/GitHub for version control on this project. Make sure to add and commit your changes as you work on the project, and once it’s complete, publish a copy using GitHub pages, and share a link to the live page and your repo with your mentor and fellow students.

OBJECTS AND OBJECT ORIENTED PROGRAMMING

In this lesson, you'll work with objects. Objects are composite because they can combine the simple data types you’ve studied so far: strings, numbers, and functions. Recall that a variable is simply a name that points at a value. That value can be a string, a number, or a function. You’ve also learned about scope. Scope refers to the level in a hierarchy at which a set of variable names refer to a respective set of values. These two concepts are brought together in objects. An object combines a set of properties and methods (the term we use for functions defined on an object) into a common namespace. That means if you define the property “foo” on some object—object1 - to be “bar”, and and the property “foo” on another object—object2—to be “manchu”, object1.foo’s value will be “bar” while object2.foo's will be "manchu". Each object separately binds the variable “foo” to a unique name space.

Another key characteristic of objects is that they allow for classes and inheritance. You can think of a class as a generic type of object, such as a car. You can then have individual instances of cars (which themselves might be grouped by class, such as convertible, station wagon, etc.). In JavaScript, one class can inherit from another. Imagine that you have a Quadruped (that is, 4-legged creature) class. All members of this class share the property of having four legs, and share the methods of being able to walk and run. Now imagine you want to create another class for dogs. Because they’re a type of quadruped, dogs should also have 4 legs and be able to walk and run. You could make the dog class inherit from quadruped, so that it automatically gets 4 legs and walk and run methods, without having to explicitly write code that sets these characteristics. You could then add methods and properties that are specific to dogs to this class, such as the ability to bark. If you wanted to create a cat class, it could also inherit from Quadruped.

Objects turn out to be incredibly important for modeling real world entities, whether they be things like people or companies, or processes like customer acquisition or particle physics. Object oriented programming (that is, using objects to model things in our applications) can be challenging to first learn, but it’s a powerful programming technique that is critical to learn if you want to become a professional developer.

In this lesson, you'll get a basic introduction to objects and object oriented programming in JavaScript. These skills will give you the ability to create complex, interactive web pages. By the end of this unit, you’ll be able to use objects and object-oriented programming principles to create a quiz app that walks users through a set of questions and lets them know which questions they’ve answered correctly and incorrectly. To keep you motivated and inspired, take a moment to look at a Quiz App from a few of our past Front End students:

http://keithlamar.github.io/videogamequiz/

http://mreiben.github.io/projects_page/coffeequiz/index.html

http://case-dubs.github.io/realitytvquiz/

Goals

Understand how to work with objects
Understand what prototypical inheritance is, and how to use it
Be able to use objects to model real world processes and things
Know how to use Chrome Developer tools to debug JavaScript in the console

OBJECT BASICS

The JavaScript object data type include functions and arrays, however when we speak of objects in JavaScript we are usually referring to object literals which look like this:

```
var obj = {}
```

The curly braces of the object literal are used to contain data as a series of key-value pairs. Look at the code below for an example.

```
//_______________A real world example

var user = {
  name:"Brendan Eich",
  profession:"programmer"
}

user.name // Brendan Eich
user.profession // programmer
```

The key value pairs expressed in the example above are what are called properties of the object. These act identically to variables the exception being that they are hitched to the object. Just like variables they can be changed on the fly.

```
//_______________Example of changing values of properties on the fly.

var user = {
name:"Brendan Eich",
profession:"programmer"
}

user.profession = "public speaker"

user.profession // "public speaker"
```

Just as properties are similar to variables, you can also add methods to your objects which are the equivalent of functions. Here's an example.

```
var user = {
name:"Brendan Eich",
profession:"programmer",
sayNameAndProfession:function(){
    return user.name + " is a "+ user.profession
    }
}

user.sayNameAndProfession(); // Brendan Eich is a programmer
```

Objects can be created and modified using a few different syntax styles. The two most common are bracket and dot notation which is what we used in our previous examples. Below is an explicit example of both styles.

```
//_________________Dot notation
var user = {};
user.name  = "Brendan";
user.profession = "programmer";

//_________________Bracket notation

var user = {
  name:"Brendan",
  profession:"programmer"
}
```

This is the basics of object literals. You might be noticing the similarities between object literals and arrays, hence wonder when to use which. The rule of thumb is that if you need a data structure that requires a consecutive order then use arrays. Beyond this basic rule of thumb there are established programming patterns that rely exclusively on objects which we will explore later.

How to determine if an object contains a particular property
You can determine if an object contains a specific property by using the in operator combined with a conditional statement. An example is provided in the code below. Keep in mind that the property on the left side of the in operator needs to be expressed as a string.

```
var user ={
  name:"Brendan Eich",
  profession:"programmer",
  invented:"JavaScript"
}

if("name" in user){   //  'name' needs to be expressed as a string.
  console.log(true)
}
```

Looping through objects
You can loop through an objects keys or values using a for-in loop.The syntax looks like this:

```
var user ={
  name:"Brendan Eich",
  profession:"programmer",
  invented:"JavaScript"
}

//___________________________Loop through keys

for(var prop in user){
   console.log(prop)//_______name: profession: invented:
}

//___________________________Loop through values
```

```
for(var prop in user){
  console.log( user[prop])//_Brendan Eich programmer JavaScript
}
```

Exercise

Add a method called reset to the object literal below that resets all the keys to undefined not including method called "reset". In other words you should be able to run the user.reset() methods multiple times because that method is itself excluded from the reset.

```
var user = {
   name: "mr smith",
   profession: "dandy fella",
   age: 100000,
   //___________________________<---add reset() method here


}


//___________________________________The result should look like this:

user.reset();

user.name; //________________________undefined
user.profession; //__________________undefined
user.age; //_________________________undefined

user.test = "yo";

user.reset();//______________________no error

user.test; //________________________undefined
```

What is an object?

An object is a key-value store for both values and functions.

What is the syntax for defining an object?
{ property: “value” }

What is a property?

A property is the name of a value stored in an object. Basically, think of it as a variable that is attached to an object.

How would you access the property "job" on the following object? html var fred = { company: 'Thinkful', job: 'Programmer', salary: 50000 }

fred.job

What's the difference between a method and a function?

A method is a function that is associated with an object and executes within that context.

How do you call an object method?

object.methodName();

DEEPER INTO OBJECTS

In our last lesson you learned the basics of object literals. We are now going to venture into the world of "Object orientated programming". The conceptual idea behind object orientated programming is one where we write a blueprint for an object and then we can create many objects that contain methods and properties that reflect our blueprint. In traditional object orientated languages this "blueprint" would be called a "class" and would usually be a data type. However, technically JavaScript doesn't have classes. Regardless of this perceived limitation, JavaScript has a few different mechanisms to create a similar result in a much more flexible manner that many programmers find more enjoyable. If you are confused by the use of the word "class" don't be. The important thing is to understand what is happening in the examples below.

Three ways to create pseudo "class" functionality is by:

Using Object.create()
Using factory functions
Using constructor functions
We are going to go over all three of these.

Cloning objects using Object.create()

You can clone JavaScript objects by using a built-in method called Object.create. Here's an example:

```
var human = {
   planet: earth,
   living:yes
}

var alien = Object.create(human); //__clone human and make it an 'alien'

alien.power = "zapper";          //__give alien new properties
alien.planet = "another dimension";

alien.power  // zapper
alien.planet // another dimension
alien.living // yes  (This was 'inherited' from its parent)
```

If you noticed in the above example we not only copied the object called 'human' to an 'alien' object but we also replaced a property and added a property to our 'alien' object. This did not affect the 'human' object which remains perfectly intact. The property called 'living' was not replaced in our 'alien' object hence it inherited this from our human object.

We can demonstrate this inheritance chain even further in the below example.Below we cloned another object called superAlien and replaced its power property with an array of items.

```
var human = {
   planet: earth,
   living:yes,
}

var alien = Object.create(human);

alien.power = "zapper";
alien.planet = "another dimension";

alien.power  // zapper
alien.planet // another dimension
alien.living // yes ....This was 'inherited' from it's parent

var superAlien = Object.create(alien);

superAlien.power = ['zapper','lazer','force shield'];


superAlien.power  // zapper, lazer, force shield
superAlien.planet // another dimension ( This was 'inherited' from its parent )
superAlien.living // yes ( )This was 'inherited' from it's parents parent )
```

Prototypal inheritance

This inheritance "chain" where a child object looks up the chain of all or any of its parent objects and uses any methods or properties that are available is called prototypal inheritance. This is a very important concept to grasp when understanding JavaScript. The concept can be succinctly expressed in the following statement

The ability of a child object to use a property or method that is connected to any of its parental ancestors is commonly referred to as prototypal inheritance in JavaScript.

Objects and the 'this' keyword

Reading the previous examples you should now have a visual understanding of how objects inherit properties from other objects. We did not use any methods in our example but the same concept applies. In the code below, our method uses the this keyword. The purpose of this is to be used as a placeholder which points to the current object. If we run the method on the alien object this.name and this.power will point to the alien. If we run it on the superHero object this.name and this.power will point to the superHero object

```
var alien ={
   name:"Martian",
   power:'flying',
   fly:function(){
      return this.name +" is now " + this.power
   }
}

alien.fly()//___"Martian is now flying"

var superHero = Object.create(alien);

superHero.name = "Superman";

console.log( superHero.fly() ) //____"Superman is now flying"
```

Creating and accessing private data with Object.create();

When using Object.create we often want to keep some of our data private. This simply means we don't want users to be able to directly see it or modify it the way they can with basic object properties. We can achieve this by creating a method and placing a variable inside of it. If we then want to allow the user to access this data but not to change it we can make it available via another method inside the harboring method. If we do want to give the user tha abillity to change this data we can control what happens as if it was any other function. An example of this is shown in the below code. The main thing to understand is that in the code below the variable hiddenNumber can only be accessed by the getHiddenNumber() method. The code also demonstrates how you can use the a harboring method (in this case submitNumber()) to modify the hidden data which in this case is the hiddenNumber variable.

```
var proto = {
   submitNumber: function(yourNumber) {
      var hiddenNumber = yourNumber || 123;
      proto.getHiddenNumber = function() {
         return "The number is " + hiddenNumber;
      };

   },

   prop: "defaultvalue",
}

//_____Doesn't modify private data

var submitNothing = Object.create(proto);

submitNothing.submitNumber();
submitNothing.getHiddenNumber();


//_____Modifies private data

var submitSomething = Object.create(proto);

submitSomething.submitNumber(100)

console.log(submitSomething.getHiddenNumber())
```

Factory functions

Another way to replicate a "pseudo class style" of programming in JavaScript is to use what are called factory functions. Factory functions are functions that return an object. Below is an example of a factory function that returns an object set to a variable named "user".

```
function makeUser(name, skills) {
   var user = {};
   user.name = name;
   user.skills = skills;
   user.getSkills = function() {
      return this.name + " has the following skills " + this.skills
   }

   return user;

}

var brendan = makeUser("Brendan Eich", ['programmer', 'public speaker', 'author'])

console.log(brendan.getSkills())

//____Brendan Eich has the following skills programmer,public speaker,author
```

In the above code we create a function and inside this function we create an object. We then create properties and methods for this object and set the properties of our object to the parameter values of our harboring factory function. We then return the object. When we invoke the function we place our desired property values as arguments, and then assign the returned object to a variable. If we then wanted to clone the brendon object using Object.create, we could do this as well.

Creating private data in factory functions

To create private data in factory functions you can simply place a variable inside it.

```
function makeUser(name) {
   var somePrivateData = "QWERTY"   // Private data
   var user = {};
   user.name = name;
```

```
    return user;

}

var micky = makeUser("micky")
console.log(micky.somePrivateData) // undefined
```

To manipulate private data in factory functions you can create user accessible methods that have access to said data.

```
function makeUser(name) {
    var somePrivateData = "QWERTY";
    var user = {};
    user.name = name;
    user.changePrivateData = function(newData) {
        somePrivateData = newData || somePrivateData;
        return "somePrivateData is " + somePrivateData
    }
    return user;

}



var micky = makeUser("micky");

micky.changePrivateData(); // somePrivateData is QWERTY

micky.changePrivateData('BLAH'); // somePrivateData is BLAH
```

Constructor functions

The previous two versions of object creation should be simple to understand. However there is a third way in which we can replicate 'class' style object creation and it is by far the more complicated of the bunch for new programmers to grasp. These are called constructor functions.

The easiest way to visualize a constructor function is as a factory function in which the creation of the resulting object happens 'behind the scenes'. So let's take a look at some code to demonstrate what I mean.

```
function Person(name,profession){
    this.name = name;
    this.profession = profession;
}

var brendan = new Person('Brendan Eich','programmer');

brendan.name // Brendan Eich
brendan.name //
```

In the code above we created a constructor function. You probably noticed that it was capitalized. This is because it is considered good JavaScript etiquette to always capitalize your constructors. The next thing that happens is we set a series of parameter value to the object(s) we will soon create from this constructor by setting them to the property name prefaced with the this key word. The next thing we do is to invoke the constructor using the new keyword. The new keyword is what tells the JavaScript interpreter that you want to use your function as a constructor and are intending to return an object from it.

Now so far this isn't too complicated, so let's add some methods to our constructor.

```
function Person(name,profession){
   this.name = name;
   this.profession = profession;
}

//________BEGIN method

Person.prototype.sayNameAndProfession = function(){
   return this.name +" is a "+ this.profession;
}

//________END method

var brendan = new Person('Brendan Eich','programmer');

brendan.sayNameAndProfession(); //____Brendan Eich is a programmer
```

In the above code you are probably a bit bewildered as to what this weird syntax is for adding methods. So let me attempt to explain. Every time you create a function in JavaScript a hidden object is created in the backdrop that is associated with that function. This hidden object is not viewable to you. If you want a visual representation of what this might look like here's an example:

```
// {}  <--- This lurking object sits in the backdrop of the below function

function doSmething(){};


/* Each function gets their own hidden object so
{}  <---- This lurking object is associate with the below function
*/

function doSomethingElse(){}
```

This hidden object will never be used for anything except if you decide to use your function as a constructor. If you never use your function as a constructor you will never have to worry about it. Now if you do decide to use your function as a constructor this hidden object will be accessible through a property called prototype.

```
Person.prototype.sayNameAndProfession = function(){
   return this.name +" is a "+ this.profession;
}
```

This hidden object represents an object that is 'higher up' in the prototype chain and any methods you attach to it will be accessible to any objects you create from its respective constructor.

Creating hidden data using constructor functions

If you want to create private data in a Constructor you simply need to create a variable and assign the data you want to it

```
function Person(name,profession){
   var privateData = "QWERTY";   // private data
   this.name = name;
   this.profession = profession;
}
```

If you want to let users modify this private data you can create methods that allow for this. Below we put the private data inside the method which is attached to the prototype property.

```
function Person(name) {
   this.name = name;

}

Person.prototype.modifyPrivateData = function(newData) {
   var somePrivateData = "QWERTY";
   somePrivateData = newData || somePrivateData;
   return "somePrivateData is " + somePrivateData
}



var micky = new Person("Micky");

console.log(micky.modifyPrivateData()); // somePrivateData is QWERTY

console.log(micky.modifyPrivateData("BLAH")); // somePrivateData is BLAH
```

So you might be wondering why we use constructors when the other 2 styles of pseudo "class" creation work just fine. The answer is because when JavaScript was first invented the people funding the project at the time wanted the language to look like Java with the hopes of attracting Java developers to JavaScript. As a result this pattern of constructors-as-classes has become the defacto way of replicating class-style programming in JavaScript. Additionally programmers from other languages are not only using the constructor pattern to create pseudo-classes, but have adopted "Java-looking" techniques to create 'sub-classes' out of constructors. In

JavaScript none of this is necessary. If you want a derivitive of an object, simply clone the object and add additional methods and properties to it.

As a side note there is one benefit to using the constructor pattern and that is performance. Technically using constructors as opposed to Object.create() or factory functions is faster, but some established programmers in the JavaScript community argue that since it's not 1995 anymore and computers are much faster the performance gains are negligible at best. https://www.youtube.com/watch?v=lKCCZTUx0sI

Regardless, all three of these patterns are used and prevalent in the world of JavaScript; hence you should strive to know and recognize them.

Further resources

Kyle Simpson: New Rules For JS Object talk https://youtu.be/S4cvuuq3OKY?t=37m14s

Eric Elliot Classical Inheritance is Obsolete: How to Think in Prototypal OO https://www.youtube.com/watch?v=lKCCZTUx0sI

Exercise

Create a pseudo class that creates "house" objects. The objects the pseudo class creates should have a customizable number of rooms, a color and size. You should also have a method that returns a sentence that says "Your house has x number of rooms is x color and x size". To do this you can use any of the three coding patterns mentioned in this tutorial. Bonus points to do it using all three!

Let's say you have a variable called myThing. How could you determine ifmyThing is an object?

typeof myThing == "object"

How can you tell if an object has a given property?

object.hasOwnProperty("propertyName")

How can you use a for in loop to get all the properties on an object?

html for (var propertyName in myObject) { // Do stuff }

How can you use a for in loop to get all the values on an object?

html for (var propertyName in myObject) { var value = myObject[propertyName]; }

What is a class?

A class is a kind of object. It is the "blueprint" of an object, and describes how it is created and what properties and methods it supports.

What is a prototype?

A prototype is a property that all Javascript objects have, that stores all methods that objects of that class share.

What happens to objects inheriting from a prototype if you add a new method or property to the prototype?

They all gain access to that property or method right away.

What does it mean to look up the prototype chain?

Because an object’s prototype is also an object, that means that the prototype has a prototype. Sort of like russian nesting dolls, there’s a chain of prototype objects all the way back to the basic Object().

How might you leverage console.log() in your application code to debug or inspect items?
Use console.log to output the results of functions to check their correctness, and to verify the contents of the objects in your code.

DEBUGGING JAVASCRIPT WITH DEVELOPER TOOLS

Earlier in this unit, you worked with the Dev Tools JavaScript console when you first learned about strings, numbers, and variables. Using the console to interactively try out code is definitely useful, but there's a lot more you can do. In this assignment, you will learn more about working with and debugging JavaScript in the console. As you start to build apps with more complexity, you'll inevitably encounter unexpected errors and bugs--all programmers do. Now you'll learn to wield the Developer Tools effectively, so you can squash any and all bugs that you encounter. You'll be able to leverage this knowledge as you go on to build your quiz app later in this lesson, and more generally in your career as a web developer.

The JavaScript console

The console is a powerful tool that is ubiquitous among JavaScript engineers. Every browser provides a similar JavaScript console, but this lesson will focus on the Google Chrome console.

Open the console by clicking View > Developer > JavaScript Console. Here you have a REPL (read-eval-print loop) which lets you write code and execute it live in the browser. You can try some math to get a feel for how it works, and then you can really dive in to by typing console.log(console). This will output the console's object to the console (so meta!). You can click on the arrows to inspect every element of the console object.

Another one you can try is $, this is called the "bling" function, and it's a console-only shortcut for document.querySelector(). If you try evaluating $('title') and document.querySelector('title') you'll see that they return the same thing. This function is also used in jQuery, and since most modern webapps use jQuery, you'll be able to find the jQuery bling function almost everywhere. This brings up an important point: the JavaScript console is running in the website you are viewing. If you want to have some fun navigate to Stack Overflow, open the console and run this code $('body').css('backgroundColor', 'red');. (To get rid of that awful background color, change red to white and run it again.)

Debugging Tricks

Here are some examples of useful debugging tricks. These are the sort of things you would eventually come up with yourself given enough time spent debugging JavaScript.

Expand errors to see the stack trace

In the console, try evaluating console.assert(1 == 2);. Obviously this is false, and the console will return an error. You can click on the arrow to the left of the failure message to see a stack trace of the error. Since we just ran the code in the console, the stack trace isn't too useful, but you'll se at the right that you have some links with numbers. If there was an error in the webapp code, those links would link to the file and line number of the problematic code. Clicking on them will go straight to the error. This is a highly useful bit of functionality that will save you a ton of time looking for errors.

Modify code on the fly

Clicking on one of those links will actually take you to an editable file. The file is stored in the browser's memory, so you're not editing the real code, but you can still test out some edits and see how it changes your app. After you have everything working, you can copy the edits over to your real code. Remember to commit the changes using git.

Pause your program during an error

Tough bugs call for tough tools, and the Chrome developer tools certainly are tough. They can even pause the execution of your code so you can inspect the objects and see exactly what's wrong. You can play around with this by clicking on the "pause" icon under the "Sources" tab on the upper-right of the console. There are a lot of moving parts in most programs, so pausing duing execution is extremely useful. If you can't find your way around the pause functionality, take a look at this video which gives you a visual introduction to the functionality.

Local Storage

Nowadays, browsers ship with a local, in-browser database called "local storage". If you're familiar with cookies, you can think of local storage as a modern replacement for cookies, sort of.

To view your local storage, click the "Resources" tab in the developer tools, and then click on the "Local Storage" sub-items on the left sidebar. If there are values in the database, you can click on them to edit them. At HTML5Demos there is a nifty little tool for playing with local storage. Head over there and open the local storage developer tool to get a feel for how local storage works. If you're curious to see how the JavaScript interfaces with local storage, you can peek at the source code for the demo here.

Conclusion

You should now have a working familiarity with the developer tools. These tools are ubiquitous among front-end programmers, and if you aren't fluent in them by now, you will be after working just one or two projects. If you're interested in all the available options at the console, check out the Chrome Developer Console API reference. It's not necessary to have a complete understanding of how each aspect of the tools work, but you definitely need to know that they are available. Because there will come a time when you are stuck and can't figure out what's causing a bug in your code. Being able to properly debug and inspect your code with the developer tools will often make the difference between giving up and pressing on to finish your project.

How might you leverage console.log() in your application code to debug or inspect items?

Use console.log to output the results of functions to check their correctness, and to verify the contents of the objects in your code.

What does the assert method do?

The console.assert method allows you to test a condition and output an error to the console if it evaluates to false.

You can use this to set up tests within your code about the state of certain variables. How can you view info about errors in the console?

Click the arrow on the left to display details about the error.

Can you use the $() to select HTML elements in the console, even if you haven't loaded jQuery?

No

What does the inspect() method do in the console?

The inspect method prints out the value and contents of the variable you pass to it.

How do you make the console pause on exceptions, and how can you use this to inspect the state of variables?

In the sources panel, click the button that looks like a pause button in a stop sign, on the right-hand side. Then, click the drop-down checkbox button “Pause on Caught Exceptions”

How do you pause on uncaught exceptions only, and why would you want to do this?

Same as 191, but uncheck the checkbox button “Pause on Caught Exceptions”. You would want to do this because uncaught exceptions are places where your code does not handle all exceptional cases. You should place the offending code in a try/catch block.

What is a debugger breakpoint?

A debugger breakpoint allows a developer to pause execution of running code to inspect the values of all variables and objects.

How do you resume and step over after a breakpoint?

The resume button looks like the play arrow for media playback controls, and the step-over button looks like an arrow-over-a-dot.

DESIGN AND WIREFRAME YOUR QUIZ APP

In the remaining three assignments in this lesson, you'll work on a single project: designing and coding an interactive quiz app from scratch. Like you did with your shopping list app in Unit 2, you'll tackle this assignment in three steps. First, in this assignment, you'll review the requirements for the app, and then you'll need to design and wireframe your app. Next you'll create a static version of the app, so you can focus on HTML and CSS. Finally, you'll write the JavaScript code you need to bring the app to life.

If you didn't already have a look at the examples shared in the intro to this lesson, make sure to do so now. It's up to you to come up with the topic you want to quiz users on and how the app will look and feel. The minimum requirements are as follows:

The app must lead the user through a set of questions. The user should only see one question at a time, with new answers presented only after they have answered the current one.
Users should be able to input their answer to each question, with radio buttons or some other appropriate interface.

When the user answers the last question, the app should show his or her overall score.
Each question should be stored as a JavaScript object, and you'll probably want to store your list of questions in an array.

If you are feeling particularly ambitious, consider implementing the following features:

Let the user know where in the quiz they are at each step (i.e. "Question 3 of 5")
Let the user know their score so far
Let the user know if their previous response was correct

Steps

1) Spend some time thinking about the topic you want to quiz users about it. Pick a topic that you find interesting.

2) Next spend some time brainstorming how the app you should look and feel. On the one hand, you should consider how to best achieve the functional requirements outlined above. How can you make it easy and intuitive for users to start the quiz, answer questions, and get feedback on their responses? On the other hand, you may choose to let the theme of your quiz affect your design decisions. For instance, if you were creating a quiz app about fishing, you might use use a majestic background of a fisherman casting into a lake.

3) After you know in general terms what you want to create, it's time to sketch your app. You're free to use pen and paper, Photoshop, or a mockup tool like Balsamiq to create your designs.

4) Try to spend about 1 to 2 hours total on this assignment. When you've completed your designs, submit your mockup assets to your mentor by submitting a link to a Google Drive folder containing the files.

CREATE A NON-INTERACTIVE VERSION OF YOUR QUIZ APP

In this assignment, you'll need to create a non-interactive version of your quiz app. The easiest way to do this is by hard coding in a question and answers, so you can preview what that view will look like. You'll also need to write the HTML and CSS for what the initial and end state of the app looks like. You can use the approach we used for the jQuery Streetfighter project: create a single HTML page that holds all the elements your app will need, and using either Dev Tools or temporarily changing your CSS, display and hide divs as needed to style and position each element and view.

Try to focus on coding up the design you created, and as much as possible don't stray from it. This will give you more practice with deliberately coding up designs, and will test your positioning skills in particular. If you get stuck for too long, reach out to the community and your mentor.

Make sure to initialize a Git repo for this new project, and when you've completed the non-interactive version of your app, be sure to add and commit your files. Submit a link to your GH page or repository below for mentor review.

MAKE YOUR QUIZ APP INTERACTIVE

In this assignment, you'll complete your quiz app. You'll need to write JavaScript code that handles moving users through the quiz and ultimately giving them feedback on how they did. Although this assignment will definitely be challenging, you've already learned all the tools you'll need to implement the behavior.

When you've completed this project, be sure to add and commit your changes on GitHub. You should also publish your app with GitHub Pages, and share a link with your fellow students and mentor.

AJAX AND ADVANCED JQUERY

So far in this course, you've learned to design and code complex static web pages. You've used JavaScript (especially jQuery) to make UI components interactive (for instance, responding to click events, mouseovers, and inputted text) and to lead users through an interactive app, as you did in the Quiz Game.

With these skills, you can create compelling user experiences; however, the pages you have created so far all make a single call to a single server. This limits the degree to which you can respond to user behaviors. You can't make new requests for information based on how the user interacts with the page, and you can't pull in data from multiple sources (arriving at different times).

To build the data-driven apps that users increasingly expect and take for granted, you need the ability to make numerous server calls after a template webpage has loaded. For example, if you visit Facebook, you'll see that the framework for the page loads, then the news feed content loads, and if you scroll down to the bottom of what's initially available, Facebook makes an additional call to the server for more content to display in your news feed. Had it asked for more content upfront, the pageload would have been slower, so Facebook strikes a balance, requesting a small amount of content upfront, and requesting more as the user needs it.

In this unit, you will learn how to create similar web experiences using AJAX. AJAX stands for asynchronous JavaScript and XML. AJAX allows you to build web pages which make multiple calls to multiple servers without loading a new page. In this unit, you will use AJAX to create applications that request data from servers and then make that data accessible in the DOM.

WORKING WITH AJAX

In this lesson, you will begin working with AJAX. Once again, AJAX is simply the technique used to exchange data with servers, allowing parts of the page to update without having to reload the entire page.

In order to exchange data with servers besides the one serving your website, you will use APIs (application programming interfaces). APIs allow you to use AJAX to request and display content from other databases, such as those of YouTube and Instagram.

You will begin this lesson with an introduction to a movie database API. Next, you will use your newly acquired skills to build a page with a search form allowing the user to find YouTube videos.

INTRO TO APIs

An API is a Application Programming Interface. A surprising number of organizations provide access to their online databases which can be accessed using an API. This allows a web developer to use the data on their own websites. For example, you could use an API to create a website that generates graphs based on census data or generate maps based on real estate data from the New York Times.

Task

IMDB.com is a very popular movie database website. Though IMDB does not offer an API, a developer built OMDB (Open Movie Database) to make IMDB data accessible through an API.

Go to omdbapi.com. In the URL bar replace omdbapi.com with the following omdbapi.com/?s=Star Wars&r=json. What did you find? If you can parse through the formatting, you'll find a set of information about Star Wars. OMDB offers a similar set of information for each movie. If you look closely, you'll see that all the movies are enclosed in brackets. This is simply an array of movies, that you received from a get request to a server.

When looking at omdbapi.com/?s=Star Wars&r=json the results are really hard to read. There is a useful extension called JSON VIEW for Chrome that will clean up the results and make it easier to read.

Get

People say that they "go" to a website or are "on" a website, but really they are requesting that a server gets them content. Usually that content is in the form of HTML with links to images and styles, just with different URLs. When a browser gets HTML data it builds a webpage based on the HTML code. But this movie data is not written in HTML, it's in a format called JSON. Because the browser doesn't know what to do with JSON, it just shows us the raw data.

What is JSON?

JSON stands for JavaScript Object Notation, which organizes data so that it can be read by computers. It might be more clear if I write it like this:

```
{"Search":
  [{"Title":"Star Wars: Episode IV - A New Hope",
  "Year":"1977",
  "imdbID":"tt0076759",
  "Type":"movie"},
  ...
  ]
}
```

A JavaScript object is contained in curly braces, and is a series of name-value pairs. In this case, the array of movies is a value assigned to the name "Search". The names and the values (if they are strings) must be enclosed in double quotes.

JSON is not the only other format. In the URL, replace the json with xml. This shows the data in XML format, which is another way of handling the data. In this project you will work with JSON, but know that XML exists.

Go back to that URL to where it says "Star%20Wars". Don't worry about the %20 (that's simply the way to indicate a space in a URL) for now, just type in other movie names and see what happens. Try it with actors, too. Each time you do this you are making a get request to the server, asking it to search its database for the term provided. You can put a movie title or actor name after s= and the server will find the data and put it in your browser in JSON format.

The OMDB URL is called an endpoint. Endpoints are special URLs that you can be used to access the API. Most APIs have multiple endpoints for the specific types of data you want to access.

The Query String

You now know omdbapi.com is an endpoint, but what's ?s=Star%20Wars&r=json? ?s=Star%20Wars&r=json is called a query string. Query strings allow you to send requests to servers to get information. In this case the "s" stands for "search" and the "r" stands for "request type." These names (s and r) were defined on the OMDB page. There is no standardization across databases for these name-value pairs. The question mark at the beginning tells the server that it's about to receive a query string. The '&' separates the name-value pairs. Go to Amazon.com, click on a product and look up at the URL. You will be able to spot the query string and even pick out the name-value pairs.

CODING WITH APIs

n this assignment, you will learn how to use API data on a web page.

Task

Create an HTML page.
Give it a simple form with just one text entry field and a submit button.
Give the input element an ID of query.
Give the form an ID of `search-term'.
Make an empty <div> with an ID of search-results.
Install and link to jQuery
Make an "apps" directory, and put an empty app.js file inside.

Open your web page in a browser and open up the console in Development Tools to make sure you don't get any errors. Type a $ in the console window to make sure you installed jQuery correctly.

getJSON

In the app.js file the first thing you would usually put in is $(document).ready(function(){});. There is a short-hand version of this code that we would like to introduce you to now. There is no performance benefit from using this version, it is just quicker to type and requires less characters. If you prefer the old method go ahead and use the full version.

```
$(function(){});
```

Go ahead and add this short-hand version of $(document).ready(function(){}); to your app.js file unless you prefer the full version.

Now you are ready to write code. First, perform a get request on some JSON data, using the $.getJSON() function. The next step is to pass two parameters: a URL and a function that will be called when the data has been received from our get request (a callback function). The URL is the same one from the URL bar 'www.omdbapi.com/?s=Star%20Wars&r=json'. For the function, perform a console.log. Your app.js may look like this:

```
$(function(){

  $.getJSON('http://www.omdbapi.com/?s=Star%20Wars&r=json', function(){
    console.log(data);
  })
})
```

To get this to work, you still need to state where the data in the console.log(data) came from. When you run the $.get() function, you will received a package of data from the OMDBapi server. You must put this data in the callback function. Make sure your code looks like this:

```
$(function(){

  $.getJSON('http://www.omdbapi.com/?s=Star%20Wars&r=json', function(data){
    console.log(data);
  })
})
```

"Data" is the variable name you're using for the response data from the get request (you could name it anything). The request is passed to the function as a parameter, then passed to the console.log, causing it to be logged to the console.

Another way
You could also use jQuery's $.get() function, and specify that you are expecting the response to be JSON-formatted. It looks like this:

```
$.get('http://www.omdbapi.com/?s=Star%20Wars&r=json', function(data){
    console.log(data);
  }, ‘json')
```

The 'json' on the end is just another parameter that is passed in the $.get() function. The first parameter is the URL, the second is the function, and the third is "json". If you put "xml" instead of "json", in the URL and the third parameter, the data would be XML-formatted.

JSON data can be complex, but will always be arrays and name-value pairs. If you remember your array handling functions, you will be able to manage any JSON data.

EXPLORING THE API DATA

After performing the get request, you should see an object in the console – 'Object {Search: Array[10]} '. In this assignment you will learn how to access it and use it on the page.

Task

Click on the triangle to the left of the word "Object". This shows you the object is made of "Search" and "proto". Open objects and arrays until you find a movie.

The Plan

You will now get a list of movie titles to appear on the screen. You will need to use JavaScript to parse the data, and then append the titles to an HTML element.

Here's a quick tip to get you started:

```
  $.getJSON('http://www.omdbapi.com/?s=Star%20Wars&r=json', function(data){
    myData = data.Search;
    console.log(myData);
  })
```

Try it!

That Object in the console contained a Search and a proto. The Search was an array of more Objects. The data.Search isolates just the Search array and sends it to the console. Play with this line myData = data.Search; and try to get one of the titles to appear in the console.

Solution:

```
  $.getJSON('http://www.omdbapi.com/?s=Star%20Wars&r=json', function(data){
    myData = data.Search[0].Title;
    console.log(myData);
  })
```

Try it!

Try to iterate through the Search array and show all the titles in the console.

Solution:

```
$(function(){

  $.getJSON('http://www.omdbapi.com/?s=Star%20Wars&r=json', function(data){
    myData = data.Search;
    $.each(myData, function(index,value){
      console.log(value.Title);
    });
  });
});
```

SHOWING THE API DATA

Before continuing, separate the get data part of the app from the show data part of the app.

Try it!

Make a new function called showResults(). Have the $.getJSON function call showResults() and pass it the array in (data.Search). The showResults() function should parse the data and send the titles to the console.log().

Solution:

```
$(function(){
  $.getJSON('http://www.omdbapi.com/?s=Star%20Wars&r=json', function(data){
    showResults(data.Search);
  });
});

function showResults(results){
  $.each(results, function(index,value){
    console.log(value.Title);
  });
}
```

Try it!

Website visitors will not have the console open. Add a separate function to display the data on the page, by making your showResults function append the titles to the empty <div> on the page.

Solution:

The code should look like this. You can keep the console.log in the code while making changes to make sure that the code is still getting the data you want.

```
  function showResults(results){
    var html = "";
    $.each(results, function(index,value){
      html += '<p>' + value.Title + '</p>';
      console.log(value.Title);
    });
    $('#search-results').html(html);
  }
```

Try it!

Thus far, the site only gets a list of Star Wars movies. To be able to access other data you need to make a submit event handler that grabs the value of the form, then calls the $.getJSON, and puts the search term in the URL. Hint: Don't forget your preventDefault.

Solution:

```
$(function(){
  $('#search-term').submit(function(event){
    event.preventDefault();
    var searchTerm = $('#query').val();
    $.getJSON('http://www.omdbapi.com/?s=' + searchTerm + '&r=json', function(data){
      showResults(data.Search);
    });
  });
})

function showResults(results){
  var html = "";
  $.each(results, function(index,value){
    html += '<p>' + value.Title + '</p>';
    console.log(value.Title);
  });
  $('#search-results').html(html);
}
```

Refactor

This works, but to handle the $.getJSON call more professionally, get it into its own function called getRequest().

```
$(function(){
  $('#search-term').submit(function(event){
    event.preventDefault();
    var searchTerm = $('#query').val();
    getRequest(searchTerm);
  });
});

function getRequest(searchTerm){
  $.getJSON('http://www.omdbapi.com/?s=' + searchTerm + '&r=json', function(data){
    showResults(data.Search);
  });
}

function showResults(results){
  var html = "";
  $.each(results, function(index,value){
    html += '<p>' + value.Title + '</p>';
    console.log(value.Title);
  });
  $('#search-results').html(html);
}
```

This change allows you to pass $.getJSON() a base URL and an object of parameters, making the code neater and easier to change.

Instead of

```
function getRequest(searchTerm){
  $.getJSON('http://www.omdbapi.com/?s=' + searchTerm + '&r=json', function(data){
    showResults(data);
  });
}
```

do

```
function getRequest(searchTerm){
  var params = {
    s: searchTerm,
    r: 'json'
  };
  url = 'http://www.omdbapi.com';

  $.getJSON(url, params, function(data){
    showResults(data.Search);
  });
}
```

Some APIs will require different parameters; this is an easier way to keep track of them.

Where did the query string go?

The query string is still there, just not in the URL field. To find it for debugging purposes, go back to your web page, open Dev Tools, and click on the Network button. If you perform a movie search in the form, the query string will appear at the field. Clicking it will open a panel that shows you everything in the HTTP header. The Request URL shows the whole URL including the query string. If you scroll down, you will see Query String Parameters and the name-values.

In building this page, you have made a request to a server for a package of data. You parsed the data and put it on a page. Most APIs are more complex than OMDB, but follow the same steps. Next, you will learn how to get an API key for one of the most famous sites on the Internet, and then get data from it.

THINKFUL TUBE CHALLENGE

Project

For this project, you will build a page with a simple search form that allows the user to search YouTube videos. Based on the search results the page will display thumbnail images of videos that match the search.

Accessing the YouTube database works almost exactly the same way as accessing the OMDB database. One difference is that YouTube requires that you obtain a developer key. The developer key allows them to know who is accessing their API and prevent hacking.

API security

While all APIs send data in response to requests, each works differently and has different documentation. These differences mean it can take hours for even an experienced developer to understand how to get the desired response. One crucial difference between APIs is authentication. OMDB requires none. Sites with privacy concerns, such as Twitter, Facebook, and Flickr, require a multi-step authentication process (they document this process on their websites). For this project you will need a YouTube developer API key, which is free and easy to get. https://www.flickr.com/services/api/auth.oauth.html

Getting a YouTube API Key

Log into the Google Developer's Console. You need a Google account.
https://console.developers.google.com/
Click the Create Project button.
Fill out the form, picking any project name you want.
Click 'API's & auth' and then credentials on the left side.
Under "Public API Access" click create key.
You should see a popup. Click browser key.

On "Create Browser Key and Configure Allowed Referers" you can leave the box blank and click Create. You will see your API key on the right.

Enabling the API
Select APIs underneath APIs & auth from the left navigation menu.
Search for and select YouTube Data API v3.
Toggle the button near the top left to On.
Using the API key - Task

Build the YouTube search app described above. Sketch exactly what you want to build, and try describing in English each feature you'd like to build. It should be similar to the OMDB app you made in the earlier assignment. The key differences are the endpoint URLs and the parameters that you send.

The endpoint is "https://www.googleapis.com/youtube/v3/search" You will need to pass the following in the params object:

part: ‘snippet'

key: your API key as a string

q: put the search term here in the form of a string

The data you get back is more complicated, but like we did with OMDB, put the data in the console and explore. All developers go through this exploration phase with a new API.

Most API's will have pages going over how to use the endpoints and what kind of responses they generally return. If needed you can search for the Google Youtube API that goes with the 'search list' end point that we are trying to use.

Try to find the thumbnail image URL yourself (there are several). Give it some time. If you get stuck, you will find a reference below.

Bonus task:

Once you get the thumbnails to appear, make them clickable. Each thumbnail should link to the YouTube page that plays that video.

Hints:

If you can't find the thumbnails, here is a hint: item.snippet.thumbnails.medium.url.
The URL to the video is not in the data, but you are given the ID. Go to the YouTube site and see if you can find how the ID is used. -Google Youtube API documentation for 'search list':

https://developers.google.com/youtube/v3/docs/search/list

STACKOVERFLOW REPUTATION BUILDER

In the previous lesson, you built simple IMDB and YouTube apps, and learned how to work with jQuery’s AJAX API. In this lesson, you will gain more experience making AJAX calls and updating the DOM based on the response data you get.

This lesson includes two assignments that instruct you to build an interface where users can enter a coding topic they are interested in and see unanswered questions for that topic on StackOverflow. The first assignment has you clone a set of starter files and gain experience reading and understanding code written by another developer, and the second assignment has you build an additional feature on top of the existing functionality.

You've probably come across it in your learning already: StackOverflow is the central hub for programming Q&A, and it’s a great place to build your reputation by giving back to the community. StackOverflow's reptutation system gives you points for asking questions others find valuable and writing good answers to questions. As you build reputation points, you unlock new privileges. It can be addicting.

As a newcomer to StackOverflow, it can be rather difficult to start building reputation points. You don’t want to write low quality questions, because you’ll get downvoted. And because most questions are answered quickly, even if you can find a question you can answer, it’s often the case that it has already been answered.

The app you build will solve part of that problem. You will be creating a simple app that works with the StackExchange API (StackExchange is the umbrella organization of which StackOverflow is a part). This will make it easy for users to immediately find open questions about topics they’re familiar with. The app will help you and your fellow learners pounce on new questions before anyone else can. Specifically, it will let us see how recent the question is, the title, and the reputation points of the person who asked it. As a beginner, you’re probably best suited to answer new questions by other beginners (who will also have low reputation).

First you will walk through a tutorial using AJAX to build that functionality to this point. Then you will use the same design pattern to give the app the ability to query for the top answerers for tags you’re interested in.

Note that the emphasis in the assignment will be on implementing the basic functionality for your app, not design or user experience. Even if you had full designs for this app, you typically want to focus first on “connecting the thread” when you’re coding a web app from scratch. This means you write the minimal code to set up all basic components and get them talking to each other. After you’ve connected the thread, then drill down on perfecting the UI.

Note that it is possible to get the same information you are providing in this app through the StackOverflow site itself; however, it's not easy to do through that interface. There are often opportunities to take a precise use case – in this case, finding unanswered questions for a specific topic – and build a simpler interface that focuses just on that use case and doesn't worry about any others. By building for one user, you can do a better job serving that user than the StackOverflow team can because they have to worry about so many types of users.

Goals

Gain experience building an app that updates DOM with data retrieved through AJAX requests.

Get to know the jQuery AJAX API better

Learn the process of “connecting the thread”

REPUTATION BUILDER WALKTHROUGH

In this assignment, you will build the first feature for StackOverflow Reputation builder. The code will allow users to submit a topic they’re interested in and then display results of these queries. You will use AJAX to interact with the StackExchange API, and learn one way to manipulate the DOM using get request data.

Steps:

Fork Starter Files: For this project, use your GitHub client to fork an existing repository that has starter files for this project. The repository you need to clone is available here. After cloning the files, feel free to open index.html inside of your browser to verify that everything works. To be clear, when you run this page locally, you should have the same experience as you get on this live demo. https://github.com/thinkful-fewd/stackerAJAX DEMO http://thinkful-fewd.github.io/stackerAJAX/

Tour the HTML and CSS Code: Briefly look over index.html. You can do this either in Sublime Text or using the Elements panel in Developer Tools, whichever you prefer. In particular, note that:

Inside of the head of index.html, there are links to jQuery, app.js, and main.css

In the body, there is a div.container, which holds all of the visible elements on the page. In main.css, there are styles that set a width for this div, horizontally centering it.

Below div.container, there is a separate div with the templates and hidden class. If you take a look at main.css, you will see a display property set to none for .hidden. This means the user will not see the DOM elements inside of this div. This is an advantage in the JavaScript code, which clones .question and .error divs as necessary, inserting data received from AJAX calls into these clones.

Note that inside of div.container, there is a results container containing two children divs. You will be putting information about searches (e.g., how many questions were returned) in div.search-results and we’ll be putting information about the individual questions in div.results.

Inside of div.container, you have one with the class .unanswered-getter and another with the class .inspiration-getter. The code you've already cloned supplies the functionality for the first form, but it will be up to you, in the next assignment, to write the code that supplies the functionality for the second form.

The first form has the HTML5 autofocus flag set. This will cause the browser to apply its default focus styles to this input field upon page load. You have the required flag on both forms, which will cause the browser to prompt the user to enter information in the input if their submission is blank.

Read Up on API Endpoints: When you're planning on using a third party API to get data, you usually will need to spend some time getting to know the endpoints the API exposes and the various settings you need to include in your request in order to get back the right data. Take 5 minutes and read over Stack Exchange's API docs on the /unanswered-questions endpoint. http://api.stackexchange.com/docs/unanswered-questions

Note that API documentation is frequently terse and technical, but a key skill of the professional developer is being able to quickly read up on a REST API and be able to write code that interacts with it. It takes time and practice to build this skill, so the sooner you start reading API documentation, the better (at this point, you've probably already read at least parts of the jQuery API documentation).

Tour the JavaScript: Now inspect app.js. Look at it in SublimeText. At the top of this file, there is a document.ready function. As you can see, submission events are listened for on the '.unanswered-getter' element. When a user submits, jQuery's .html('') method, with '' inside of it, is used to clear the .results container in case there have been any prior searches. Next the variable tags initialize, setting its value to the value the user submitted. Finally, the getUnanswered function is run, which is defined further down in app.js.

Jump down to the bottom of the file where getUnanswered is defined. This function calls the other functions in the middle of the file. getUnanswered first creates an object that contains the parameters which will be passed in the GET request to the StackExchange API.

Next, a variable is created whose value is a deferred object (this is the underlying object returned by the $.ajax() method). In the call to .ajax(), for the url setting, the endpoint from step 3 gets passed. For the data key, you pass in the requestParams object you created above. Since you want a client-side cross-domain request to this API, you must specify "jsonp" for the dataType setting. Finally, set the method to "GET" since this is the HTTP method used.

The logic in the .done() and .fail() blocks is similar. Remember that the .done() block will only execute when and if the AJAX returns successfully. The .fail() block will only execute when and if the AJAX request fails. In both cases, the strategy (looking at the code for the showQuestion, showSearchResults and showError functions) is to .clone() the appropriate hidden DOM element, insert new text and HTML into the clone based on our response data, then insert this cloned element into the visible .container div.

This is a simple design pattern that you can use again and again when you're prototyping a AJAX-based web page. If you were building a full-fledged application, you would probably want to start thinking about storing your templates as partials that you could also load using AJAX.

This would make your app more modular. We won't go into using partials or templating in this course as it is considered an advanced topic. Below are some resources that go over the topic if you're interested.

Templates with Mustache.js http://coenraets.org/blog/2011/12/tutorial-html-templates-with-mustache-js/

MDN Javascript templates https://developer.mozilla.org/en-US/docs/JavaScript_templates

handlebars.js http://handlebarsjs.com/

There are some additional implementation details in the showQuestion, showSearchResults, and showError functions that you should review on your own. In the next assignment, you will take the principles demonstrated here and apply them to building a second feature for this web page.

GET INSPIRED FEATURE

The previous assignment was hopefully valuable in showing one way to go about building a simple AJAX app.

In this assignment, you will write the code that will bring the second feature online. The second feature for this page will give users the ability to search for the top StackOverflow answerers for particular topics. You already have the code for the HTML form, but in this assignment you'll need to add code to app.js that handles user requests for inspiration.

To clarify, the requirements for this feature are:

Users can submit a topic they want to find top answerers for on Stack Overflow.

The app makes an AJAX call to the appropriate endpoint on the StackExchange API (there are a few hints below).

The DOM is updated with information about top answerers (if any) after the response is returned.

Steps:

Read Up on Endpoints: Read up on the top-answerers-on-tags endpoint here. Keep in mind that this is your source of information about the URL that provides the resource you want, as well as the parameters you'll need to include in your request. http://api.stackexchange.com/docs/top-answerers-on-tags\

Decide how to display this data in the DOM: You will need to spend a few minutes thinking about how you want to display your top answerer data in the DOM. Presumably, you want to use the same .results div that were used to display results from the get-unanswered-questions form. You also want to create a template (and put it in the .templates div) that you can clone in app.js. Depending on how you want to display the results for the inspiration-getter, you may also need to add some styles to main.css.

Write Code, Preview in Browser, Debug, Repeat: You should know your strategy at this point, and now it's just a matter of writing the JavaScript code that makes the AJAX calls and updates the DOM with response data. Chances are, it will take you a few iterations to get your code working right. Use the same approach you learned in this course: first, outline the code so you have a strategy before you start building. Next, build the simplest version of what you're trying to do and preview your page in your browser to see if it's working. If it doesn't work, or if there's a setting you need to tweak, use Dev Tools to inspect and debug. When you zero in on a bug in Dev Tools, go back to Sublime Text and make necessary revisions, then back to the browser for another round of preview and debug. Repeat until your page works as required and expected.
Git Commit and Share: Once you've completed this project, be sure to use Git to commit all the changes you've made. Create a gh-pages branch and push it up to the remote server so you can share the app you created with others and submit it to your mentor.

API HACK

The application must make use of a third-party API and use the results to change the DOM.You will need to choose the API you wish to use and how you will use the data.

You must plan your app in detail before you start building. Send the sketch to your mentor so he or she can flag potential problems with it – such as data limits or technology requirements – before you start building.

Try making your project responsive using media queries, as you learned in unit 1.

CHOOSING AN API

Choosing an API

Before starting your project, you need to decide what you want to build and choose an API. You should specifically look for an API that offers response data in the JSONP format. Some browsers restrict you from making AJAX calls to servers in a different domain because of the Same Origin Policy; JSONP was created to bypass that. If you select a JSONP API, your project will work in every browser.

Here's a list of some favorite JSONP APIs along with a brief description of the functionality for each. For a much more complete list, check out this list of 258 JSONP APIs. http://www.programmableweb.com/news/258-jsonp-apis-get-your-json-response-anywhere/2011/10/07

The Etsy API lets you access products and sellers. https://www.etsy.com/developers/documentation/getting_started/jsonp
The Klout API gives you "social influence" scores for people based on their social media activity http://klout.com/s/developers/v2
The Instagram API lets you access a stream of public photos. http://instagram.com/developer/

The Tastekid API lets you integrate their recommendation service for music, movies, shows, and books (takes the input of one artist or book and outputs related one you might enjoy). http://www.tastekid.com/page/api
The Foursquare API gives you access to data about venues and checkins. Note the instructions on the response page for using JSONP instead of JSON. https://developer.foursquare.com/
The Census Bureau API has census data! http://www.census.gov/developers/

Stuck?

If you're looking for inspiration, here are some project ideas:

Use official US Government census data to build an app that lets users find the number of people who fit a particular set of criteria, such as age, ethnicity, state, or county. There's plenty of room to be creative here – you could make it into a guessing game ("How many people live in California between the ages of 15-24?") or a census version of Price Is Right.

Choose a theme of images, and create a creative layout for the corresponding image stream from Instagram. It could display one image at a time, or multiple images in one layout.
Use the Spotify API to create an app used to search for an artist's most popular song. Like the examples above, there are many ways you could get creative with the data offered from Spotify.
For a more advanced hack, you could use the Google Maps API in conjunction with the Meetup API to get Meetup.com search result locations to populate on Google Maps.

http://www.programmableweb.com/api/spotify-web
http://www.programmableweb.com/api/google-maps
http://www.programmableweb.com/api/meetup

SKETCHING YOUR PROJECT

Now that you have a sense for what you'd like to build, it's time to plan your project in detail before writing code.

Create Your Wireframe

The goal of this assignment is to devise a wireframe for an application that you think will deliver the user experience you've defined in the previous assignment. Keep your audience in mind as you make decisions about how the interface works. Try to find ways to make it intuitive. If you design it well, users should find it easy to use.

Steps

Create a detailed wireframe (either by hand or with a wireframing tool such as Balsamiq) of a web page that will achieve the user experience you defined in assignment 1.
Share the wireframe with your mentor and fellow students. To do this, you will need to take a picture or screenshot of your wireframe and post it to a photo sharing site, such as Imgur, and submit the URL below. Be sure to give a brief description explaining the user experience of your

app to your mentor and peers so they can assess whether or not they think your app design works.

PROTOTYPE YOUR APP

Give yourself approximately three hours to build a static (no JavaScript) version of your page. This version of your app should be synchronous, meaning it should only make a single round of server calls. You'll load an interface, and that's it. For now, that means you'll use hard-coded data (for example, a single set of Instagram images) to focus on building the visual interface.

Based on what you learn in the API documentation for the endpoint your app utilizes, you can create mock JSON objects in a script that act as stand-ins for the ones that will eventually be received from the server. You could, for instance, iterate over an array of mock StackOverflow answers. Just make sure that the hard coded data you create is in harmony with the resource the API docs promise the endpoint will return.

Be sure to use GitHub frequently. Get into the habit of committing once you've solved a specific problem, or a subset of a specific problem that can be described in a succinct, descriptive commit message. Even though the page will continue to be under construction through the end of the next assignment, you should also be sure to publish it using gh-pages and submit it to your mentor for feedback.

INTEGRATE AN API

Now that you've got a working interface, it's time to write the code that will retrieve the resource from the server at the right time and update the DOM as defined by your designs.

Give yourself approximately 3 hours to spend coding the AJAX component of the app. Be sure to use Git to commit your changes and store them remotely on Github.com to ensure everything's backed up, now that you've completed your project.

When you're done, be sure to update your github.io page for the project. Share it with your friends, family, and prospective employers. You've spent a lot of time on this project, and hopefully it's something you're proud of! If you're on Twitter, tell your friends about it (and be sure to let @Thinkful know!). Congratulations on completing your API Hack!

POLISHING YOUR PORTFOLIO

http://roseemmons.github.io/
http://raddevon.com/
http://obsoleter.com/portfolio/
http://www.bighuman.com/#/work/
http://marcorosella.com/
http://www.rorymcilroy.com/
http://bryanconnor.com/

Before moving on to the assignments, take a look over what has been covered in the course, as these are skills that you will want showcased in your portfolio.

Skills:

-Use HTML to describe and structure content -Use CSS to control the appearance of your content -Implement a number of HTML elements and CSS style properties -Use jQuery to dynamically manipulate HTML and CSS in response to user events -Program with JavaScript and jQuery -Work with AJAX to build web apps that talk to APIs

Goals

Design, code up, and iterate on a portfolio page to showcase your front end skills
Review ideas for further projects
Review topics for further study

If you've been following along with the lessons, you already have a basic portfolio started; however, now that you have a number of new skills in your web developer toolbox, you will want to review your portfolio that you built at the beginning of the course to make sure it reflects your new skillset.

The minimal requirements for your profile site are:

It should house all of your current projects and accommodate any future projects you want to add.

Since this site is about you, be sure to include some way for visitors to contact you (via Twitter, email, a form, etc).

Think about your audience and how to best capture their interest. Are you trying to impress prospective employers? Express your personality to a general audience? Show off to friends? Your design decisions should be informed by this goal.

WHAT TO LEARN NEXT

The most common question we hear from students who complete this course is “What should I do next?” We’d like to address that question here, giving you some project ideas and next technologies to learn.

Projects

The way you get better at programming is by building projects that push your skills further. If you’re looking to move into full time front end web development work, expanding your portfolio will make you more attractive to potential employers. Our main piece of “next steps” advice then is to tackle new projects. Here are some ideas:

Revisit your Small Business Landing Page: You completed this project at the end of Unit 1, before you had learned how to use jQuery and JavaScript. Now that you’re able to create interactive effects, consider revisiting this project. First revisit your designs for the site to see if adding interactive effects would improve the experience and then code up any changes you settle on.

Revisit your Hot or Cold app: As you'll recall, you worked with existing HTML and CSS code for this project. One quick win to improve your portfolio would be to revisit this app, implementing your own designs.

Get Small Freelance Gigs: The skills you've gained in this course equip you to create interactive, compelling static web pages. This means you're well suited for doing small freelance gigs building static web pages. We now offer a course that helps you find and manage freelance work to start earning income as you build your portfolio; email us at support@thinkful.com if you'd like to learn more.

Learn New Technologies

This course has covered HTML, CSS, jQuery, and programming fundamentals in JavaScript. We've also touched on design and developer tools Git and the command line. There's a lot you can do with these technologies alone, but part of being an employable developer is learning new skills and staying abreast of technology trends. With that in mind, here are some front end technologies that could be good ones to look into next:

Twitter Bootstrap: Bootstrap is a popular front end framework for rapidly building nice looking web pages using reusable components. In a nutshell, Bootstrap is a collection of CSS and JavaScript that you can incorporate into your projects. Out of the box, Bootstrap gives you attractive UI components such as buttons and navbars, a grid system, responsive design, and several other features. Bootstrap is quick to learn, easy to customize, and widely used.
Learn a CSS Preprocessor: A CSS preprocessor—like SASS or LESS, which are two of the most popular—makes writing CSS easier and smarter. Preprocessors allow you use variables and nested selectors in your style rules. You can read about CSS preprocessors in this article from Smashing Magazine. http://www.smashingmagazine.com/2011/09/09/an-introduction-to-less-and-comparison-to-sass/

Command Line and Git: In this course, you learned the basics of version control using a GUI client for Git and working with Github. As you begin to use Git to collaborate with others and work on larger scale projects, you'll want to learn more advanced Git concepts and know how to work with Git from the command line. This document will get you started learning the command line. Although it's admittedly a dry read, the freely available Pro Git book is a great resource for learning command line Git. Finally, we'd also recommend the Git Immersion tutorial, which walks you through different Git concepts.

https://docs.google.com/a/thinkful.com/document/d/1uVLOGfGu--uqHppaS-5pCp45AECMHPzgvEnLMXLsJSc/edit#heading=h.phxbb3clqyja

http://git-scm.com/book

http://gitimmersion.com/