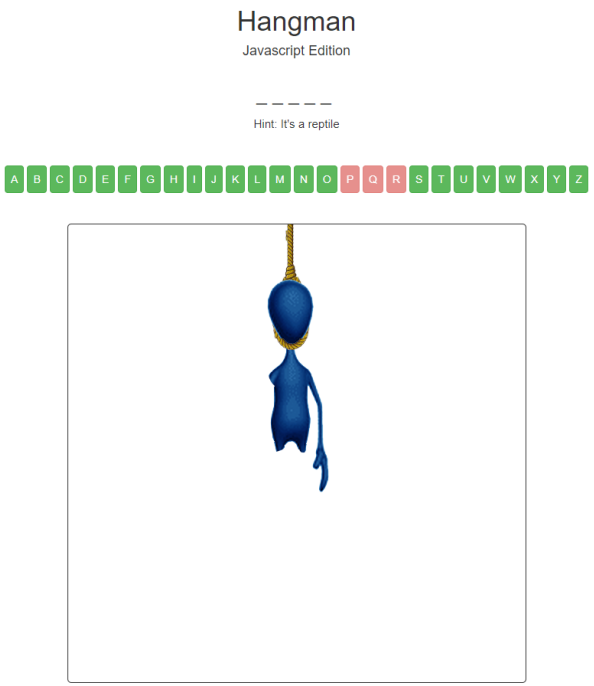
Hangman

In this tutorial we'll create Hangman using Javascript, jQuery, and Bootstrap. The player will be given a word length, a hint, and a series of buttons with each letter. As each letter is selected, the word is either filled in if they guessed correctly, or a piece of the blue alien is added to the picture. Letters that have already been guessed will be disabled.



Here is a completed version of the project: http://cst336.herokuapp.com/projects/hangman/

As we work in this project, you will practice using:

- JavaScript Loops
- JavaScript Functions

- JavaScript Arrays
- JavaScript Objects
- Accessing and Manipulating DOM
- Events and Handlers
- jQuery

Planning!

Take a moment to think about our goals for the project and how we would go about achieving them.

a. We need to provide input for the user, but how do we prevent them from selecting the same letter twice?
b. How do we store and randomly select our word and hint data?
c. How is game progress checked so that we know when to update the image and end the game?
d. Should the hangman visual be a single image or multiple images?

Optionally, you can also watch the video tutorials for this lab:
https://youtu.be/PLJ01HwigIo (Part 1)
https://youtu.be/S2n3FNdO-uA (Part 2)
https://youtu.be/l1fCMvICPtw (Part 3)

Lesson 0: Setup

0.1 Create a folder in your cloud9 workspace called **hangman**.

0.1 Download and extract the zip file located here: http://cst336.herokuapp.com/projects/hangman/hangman.zip

0.3 Upload **css**, **img**, and **js** folders as well as **index.html** to the hangman directory on cloud9.

0.4 Take a moment to look at the **index.html**.

Lesson 1: Add HTML and initialize the game

1.1 Begin by creating a <**div**> with the class **"container center-text"** inside the <**body**> of our page**.** This is where we will be putting all of our HTML. Bootstrap will recognize these classes and apply padding and center the text. Next, place the page headers inside **<header>** semantic tags.

We also need a couple empty <**div**> tags with an **id** of **"word"** for one and **"letters"** for the other. These are the containers for the word and letter buttons. We will be using Javascript to add content to these divs.

Finally, include a <**div**> with **id="man"** and put the first image of our hangman inside it. The <**img**> tag will need **id="hangImg"**.

```
<body>
    <div class='container text-center'>
        <header>
            <h1>Hangman</h1>
            <h4>Javascript Edition</h4>
        </header>

        <div id="word"></div>
        <div id="letters"></div>

        <div id="man">
            <img src="img/stick_0.png" id="hangImg">
        </div>
    </div>
</body>
```

1.2 With the HTML in place, we can start writing the logic of the game in Javascript. Inside the **<head>** tag of our page, open up a **<script>** tag.

Inside these tags we will define some variables using the **var** syntax. The **words** variable will be an array of words that we will randomly pull from at the start of a new game.

Another important variable is **board**, this will be an array of characters that represent the current state of the board, that is, the letters that have been guessed so far. We can initialize this to an empty array with **board = []**.

Let's also output the first element of the words array to the browser's console using **console.log(words[0])**.

```
<script language="javascript">
    var selectedWord = "";
    var selectedHint = "";
    var board = [];
    var remainingGuesses = 6;
    var words = ["snake", "monkey", "beetle"];

    console.log(words[0]);
</script>
```

What value do you expect to see in the browser's console?

Notes:

Notice how we didn't need to specify types (int, string, etc) for our variables. Like PHP, Javascript is a **dynamically typed** language. Rather than the programmer providing a type for the variables at compile time, the interpreter determines the correct type at runtime. This means variables can change type over time.

This is perfectly valid, for example:
var myNumber = 5;

myNumber = "five";

This doesn't mean that Javascript doesn't have types, only that they can change over time. You can check the type of a variable with the typeof operator.

1.3 We want to select a random word from our array of available words. Thankfully, Javascript's **Math** object contains a function for getting a random number: **Math.random()**.

The random function returns a value between zero and 1. In order to get a number between 0 and the length of our array, we simply need to multiply it by the length of the array which we can access with **words.length**.

The number we would get from **Math.random() * words.length** would be within the range of numbers, but they're real numbers, not integers. This is where **Math.floor** comes in.

We can then use this random number to index **words** and assign it to **selectedWord.**

Add these two lines right before the closing </**script**> tag.

```
var randomInt = Math.floor(Math.random() * words.length);
selectedWord = words[randomInt];
```

1.4 We're going to use the variable **board** to represent the current state of the word as it would be drawn on a board. In hangman, it's illustrated as a mark under each letter, let's represent that with underscores.

Make a function using the keyword **function** called **initBoard** with no arguments. Inside the function we can use a **for loop** to iterate over the letters of our **selectedWord**. For each letter of our selected word, we want to push an underscore onto the board, to represent that letter. Javascript arrays include a **push** function that adds an element to the end of the array.

```
// Fill the board with underscores
function initBoard() {
    for (var letter in selectedWord) {
        board.push("_");
    }
}
```

Notes:
Using the keyword **"in"** in the for loop here assigns the variable **letter** to the current ***index*** of **selectedWord**. So if the word is "monkey", **letter** will have the values 0, 1, 2, 3, 4, 5.

Alternatively we can use the keyword **"of"** to assign letter to the ***value*** at that index. In the case of "monkey", **letter** would have the values "m", "o", "n", "k", "e", "y".

Also note, the single line comment. Javascript uses the same syntax as C and C++ for it's comments, meaning you can create:
/* multi-line comments this way */

1.5 We will use a for loop to access each letter of the **board**. Inside the loop, we want to append each letter of the board into div with **id="word"**. You can access this div's content with the **getElementById()** function on the document and by specifying the Id you want to find.

The html contained inside the div can be accessed with the attribute **innerHTML**. Let's access this and append each letter on our board (which is all underscores at the moment) along with a space so we can tell them apart.

```
initBoard();
for (var letter of board) {
    document.getElementById("word").innerHTML += letter + " ";
}
```

Uh oh, it's not working! The browser's console should provide hints on where things are going wrong.

1.6 There is nothing wrong with our code, only where it is located. Javascript is executed from top to bottom, line by line. Right now, our code is in the head section, so the javascript is being executed before the body, and thus the ** ** does not exist. To resolve this, move the code from the **<script>** tags to it's own **"hangman.js"** file in the **"js"** directory. Now we can include this file in our **index.html** at the bottom of the body tag. Notice that since the javascript is in it's own file, we do not need **<script>** tags surrounding our code.

js/hangman.js

```
var selectedWord = "";
var selectedHint = "";
var board = [];
var remainingGuesses = 6;
var words = ["snake", "monkey", "beetle"]

var randomInt = Math.floor(Math.random() * words.length);
selectedWord = words[randomInt];

// Fill the board with underscores
function initBoard() {
    for (var letter in selectedWord) {
        board.push("_");
    }
}

initBoard();
for (var letter of board) {
    document.getElementById("word").innerHTML += letter + " ";
}
```

Index.html

```
<body>
    <div class='container text-center'>
        <header>
            <h1>Hangman</h1>
            <h4>Javascript Edition</h4>
        </header>

        <div id="word"></div>
        <div id="letters"></div>

        <div id="man">
            <img src="img/stick_0.png" id="hangImg">
        </div>
    </div>

    <script src="js/hangman.js"></script>
</body>
```

1.7 Lets clean some things up by creating some more functions. First we'll create a function that updates the board, this is just our previous for loop, call it **updateBoard()**. Additionally, we should put our code to randomly select the word inside a function. We will call this function **pickWord()**. To kick off the game we can also create a function to call each of these: **startGame()**.

We can ensure that the document is fully loaded before starting the game by assigning **Window.onload** to the **startGame()** function.

```
// VARIABLES
var selectedWord = "";
var selectedHint = "";
var board = [];
var remainingGuesses = 6;
var words = ["snake", "monkey", "beetle"]


// LISTENERS
window.onload = startGame();


// FUNCTIONS
function startGame() {
    pickWord();
    initBoard();
    updateBoard();
}

function initBoard() {
    for (var letter in selectedWord) {
        board.push("_");
    }
}

function pickWord() {
    var randomInt = Math.floor(Math.random() * words.length);
```

Notes:

Javascript should always be placed in a separate file as it follows the **separation of concerns**. The same way we separate out the style of the webpage from the content, the the scripts for a website should not be mixed with the content (when possible).

You will notice in the code above that we have organized our javascript file into sections. In order to be able to easily find things and make changes, we've put the **Variables, Functions, and the Listeners** (more on this later) into their respective sections.

Lesson 2: jQuery and user input

2.1 Now that the word is selected and the board is setup, we need a way for the users to select letters. On our html page, add a textbox inside the **div with id=letters** with the **id="letterBox"**. We also need a button to submit the letter, give that an **id="letterBtn"**.

```
<div id="letters">
    <input type="text" id="letterBox">
    <button id="letterBtn">Submit</button>
</div>
```

Important!

Notice that JavaScript uses **<button>** or **<input type="button">** but **DOES NOT** use
<input type="submit">. If you use "submit" as the type, the form will be submitted to the server and even though the JavaScript code gets executed right before the form is submitted to the server, it's so fast that you won't notice anything.

2.2 With the HTML in place to submit the letter, we need to detect when the user clicks on the Submit button. Instead of using plain Javascript which can be verbose, we're going to use a Javascript library call jQuery. The HTML file already includes a link to the jQuery CDN.

Let's grab the button in our script with **$("#letterBtn")** which returns a jquery object. On that object we will call the **click()** function to bind an event handler. You pass the function you want to handle the button click as an argument, lets just make this a simple anonymous function within our hangman.js file that prints the contents of the box to the console.

```
$("#letterBtn").click(function(){
    var boxVal = $("#letterBox").val();
    console.log("You pressed the button and it had the value: " + boxVal);
})
```

Notes:

In Javascript, functions are treated as **first-class objects**. What this means in practice is that they act just much like any other variable. We saw an example of this above where we passed a function into jQuery's click function.

In the case of our event handler, it was anonymous, meaning we didn't assign a name to it. If we wanted to use this function in another context, we could have done this:

```
var handler = function() { … }
$("#letterBtn").click(handler);
```

Using anonymous functions as arguments is very common in Javascript / jQuery, but it can look confusing at first. Hopefully the example above makes what is going on a little clearer.

Do you see any downsides to our approach to user input? How would you handle the player entering multiple characters or numeric characters? What prevents the user from selecting the same letter twice?

2.3 Instead of allowing the user to enter values into a textbox, which will involve parsing the input to see if it's valid, let's simplify this by using letter buttons instead. Remove the event handler we just created as well as the HTML textbox and button.

We could put 26 <**button**> tags into our HTML file to represent the buttons, but it would make the HTML file difficult to read and difficult to maintain. What if we wanted to change the class or id of the buttons? It would require changing all 26 tags! Let's leverage the power of jQuery for this task.

Create an array of all upper case letters at the top of the file, like this:

```
// Creating an array of available letters
var alphabet = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
                'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
                'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'];
```

2.4 Now create a function that will iterate through the **alphabet** array and append a button with the value of each letter into the **letters** div. Let's also include the letter in the id for the button so we can reference it later. Call the **createLetters** function within the **startGame()** function.

```
// Creates the letters inside the letters div
function createLetters() {
    for (var letter of alphabet) {
        $("#letters").append("<button class='letter' id='" + letter + "'>" + letter + "</button>");
    }
}
```

Notes:

Here we are using a jQuery selector **$("#letters")** to grab the html element with id letters. Then with the **append()** function, dynamically HTML elements.

2.5 With the letter buttons in place, we can add a handler to monitor when one is clicked. For that, we can use the jQuery **click()** function. First use the jQuery selector for class **$(".letter")** and call click, passing in an anonymous function. Inside that function, lets just print out the **id** of the button to the console to make sure it's working.

```
$(".letter").click(function(){
    console.log($(this).attr("id"));
});
```

Notes:

The keyword **this,** in the function above, is a reference to the button that called it. It is then made into a jQuery object so we can get the button's id.

Lesson 3: Changing / updating game state

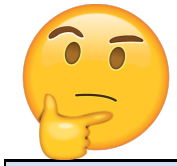
3.1 Now that we are able to get input from the user, let's implement the game mechanics themselves. Create a function called **checkLetter** that accepts a letter and checks to find that letter's position in **selectedWord.** We can do this by making a new array to hold the positions, or the index of the for loop as we iterate through **selectedWord**.

```
// Checks to see if the selected letter exists in the selectedWord
function checkLetter(letter) {
    var positions = new Array();

    // Put all the positions the letter exists in an array
    for (var i = 0; i < selectedWord.length; i++) {
        console.log(selectedWord)
        if (letter == selectedWord[i]) {
            positions.push(i);
        }
    }
}
```

Since our array of characters is uppercase, it won't be equal to the lower case letters of our **selectedWord**. We can fix that by using the **toUpperCase** function when selecting the word.

```
function pickWord() {
    var randomInt = Math.floor(Math.random() * words.length);
    selectedWord = words[randomInt].toUpperCase();
}
```



With the positions array containing the positions of the selected letter in the current word, how could you determine if the player guessed correctly?

If the player did guess correctly, how would you update the board to reflect their guess?

3.2 With our array of positions for letters in a word, we can determine if the player guessed correctly by checking the length of **positions.** If there is at least one element in the positions array, the player's guess is in the word. Let's create a function called **updateWord()** to handle updating the board appropriately

But first, if the player does not guess correctly, reduce the available guesses by 1.

```
// Checks to see if the selected letter exists in the selectedWord
function checkLetter(letter) {
    var positions = new Array();

    // Put all the positions the letter exists in an array
    for (var i = 0; i < selectedWord.length; i++) {
        if (letter == selectedWord[i]) {
            positions.push(i);
        }
    }

    if (positions.length > 0) {
        updateWord(positions, letter);
    } else {
        remainingGuesses -= 1;
    }
}
```

3.3 The **updateWord** function needs to handle updating the current word and updating the board to reflect the changes made. Since the board is a global variable we only need to send it the positions that need to be replaced as well as the letter selected. Lets iterate over all the positions with a **for** loop and replace those positions in the board array with the selected letter.

```
// Update the current word then calls for a board update
function updateWord(positions, letter) {
    for (var pos of positions) {
        board[pos] = letter;
    }

    updateBoard();
}
```

If you try the game now, it mostly works. The problem is that letters seem to be added each time we guess a letter correctly.

Take a minute to try and debug the problem before moving forward!

3.4 Once the board has been modified to include the letter selected by the player (assuming they guessed correct), we need to redraw it. The **updateBoard()** function already does this, but before we only called it once. Now that we will be calling it over and over again, we need to clear out the div before redrawing the letters. Let's use the jQuery **empty()** function to empty the **#word div** before appending the updated board letters.

```
function updateBoard() {
    $("#word").empty();

    for (var letter of board) {
        document.getElementById("word").innerHTML += letter + " ";
    }
}
```

Lesson 4: Updating score and changing image

4.1 The game is playable, but the incorrect guesses from the player don't draw more of our hangman. We already have a variable to keep track of how many guesses the player has remaining: **remainingGuesses**. We just need to update the image and stop the game once the **remainingGuesses = 0**.

The images of our hangman are numbered 0 to 6 which is why our guesses variable is set to six when the page loads. We can take advantage of these facts to calculate the correct image. By doing the subtraction **(6 - remainingGuesses)** we can determine what image should be displayed.

```
// Calculates and updates the image for our stick man
function updateMan() {
    $("#hangImg").attr("src", "img/stick_" + (6 - remainingGuesses) + ".png");
}
```

4.2 When the man is completely drawn, we need to prevent the player from continuing to guess letters. Lets add a function to hide the letters div once they run out of guesses. We should also display a message to the user. We can put these messages in divs and then hide them until the game is over.

index.html

```
<div id="won">
    <h2>You Won!</h2>
    <button class="replayBtn btn btn-success">Play Again</button>
</div>
<div id="lost">
    <h2>You Lost!</h2>
    <button class="replayBtn btn btn-warning">Play Again</button>
</div>
```

hangman.js

```
// Ends the game by hiding game divs and displaying the win or loss divs
function endGame(win) {
    $("#letters").hide();

    if (win) {
        $('#won').show();
    } else {
        $('#lost').show();
    }
}
```

We need the win and loss messages to be hidden from the moment the page loads. Lets disable their visibility using CSS instead of javascript.

styles.css

```
#won, #lost {
    display: none;
}
```

Finally, don't forget to call the **endGame()** function when the player wins or loses. We can do that in **checkLetter()**

```
if (positions.length > 0) {
    updateWord(positions, letter);

     // Check to see if this is a winning guess
    if (!board.includes('_')) {
        endGame(true);
    }

} else {
    remainingGuesses -= 1;
    updateMan();
}

if (remainingGuesses <= 0) {
    endGame(false);
}
```

4.3 The **#won** and **#lost** divs contain buttons that should reload the page when clicked. They have the class **replayButton**. Lets also add that handler.

```
$(".replayBtn").on("click", function() {
    location.reload();
});
```

Notes:

The location object in in javascript contains information about the current URL. In addition to being a way to access the hostname or href (URL), it contains a **reload()** function that we can use to reload the page.

Lesson 5: Polishing and styles

5.1 Games of hangman often provide hints to the player so that they're not just guessing numbers at random. Lets incorporate this feature by changing our array of words to an array of objects that contain both a word and a hint.

```
var words = [{ word: "snake", hint: "It's a reptile" },
             { word: "monkey", hint: "It's a mammal" },
             { word: "beetle", hint: "It's an insect" }];
```

We need to change the pickWord function to to access the **word** property of the array element instead of just the array. We should also assign the **selectedHint** here.

```
function pickWord() {
    var randomInt = Math.floor(Math.random() * words.length);
    selectedWord = words[randomInt].word.toUpperCase();
    selectedHint = words[randomInt].hint;
}
```

To display the hint, we need to add it to the function that displays the board. Lets use jQuery's **append()** to update the word **div** this time.

```
function updateBoard() {
    $("#word").empty();

    for (var i=0; i < board.length; i++) {
        $("#word").append(board[i] + " ");
    }

    $("#word").append("<br />");
    $("#word").append("<span class='hint'>Hint: " + selectedHint + "</span>");
}
```

Notes:

Javascript objects are similar to objects in other languages. They can encapsulate data and functions and work well for storing information like you would in a python dictionary (or associative array in php). They are created with **{ field: value }** syntax.

Javascript also supports adding properties to existing objects through Object Prototypes. You can learn more about that here: https://www.w3schools.com/js/js_object_prototypes.asp

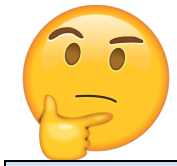
5.2 We can improve the look and of the letter buttons by applying the bootstrap button styles. Add the class **'btn'** and **'btn-success'** to the existing **'letter'** class on the buttons.

Another improvement for the buttons would be to disable the functionality of a button that has already been selected. You can change properties on HTML elements in jQuery with the **prop()** function.

```
// Disables the button and changes the style to tell the user it's disabled
function disableButton(btn) {
    btn.prop("disabled",true);
    btn.attr("class", "btn btn-danger")
}
```

We now need to call this function when the button is pressed. Thankfully, we already have a handler setup for that. We just need to pass the button as a jQuery object to our disable function.

```
$(".letter").click(function(){
    checkLetter($(this).attr("id"));
    disableButton($(this));
});
```



Spend some time changing font sizes, padding, and margins to reduce the clutter of the page and make it more readable.

5.3 Additional CSS styling is needed to make things more usable. This aspect is highly subjective, but if you want to match the project example, here is the CSS:

```css
#word, #letters {
    padding: 20px 0px;
}

#man {
    margin: 20px auto;
    width: 600px;
    height: 600px;
    border: solid;
    border-radius: 5px;
    border-width: 1px;
}

#word {
    font-size: 1.8em;
}

.hint {
    font-size: 0.6em !important;
}

.btn {
    padding: 7px 7px !important;
    margin: 0px 2px;
}
```

Save and run the file. The symbols should be displayed in the right place now!

It's Your Turn!

1. Hide the hint and just display a "Hint" button.
   The hint should be displayed **after** clicking on the button.
   The hint should cost a body part.
2) Display the list of words that the player has guessed below the image. For instance, if the player guessed the words "snake" and "monkey", both words should be listed.