

CS 130(0) JavaScript Lab

Lab Setup

The presentation slides can be found at the following link: [Presentation Slides](#)

The only software that you'll need in order to complete this lab is access to the Google Chrome web browser and access to a text editing tool (such as Sublime).


What is JavaScript?

JavaScript is an object-oriented scripting language¹ used on the client side of web browsers. JavaScript helps to make webpage content both dynamic and interactive, leading to the creation of friendlier interfaces for users.

This lab aims to provide you with a simple introduction to JavaScript syntax. You will learn how to declare variables, write functions, and create classes. Understanding these three topics will give you a good baseline for JavaScript programming, and will help expedite your understanding of topics/assignments introduced later in the semester.

Printing

You will begin this lab by opening your Chrome console on this web page.

Press on the button that looks like this:  on the upper right-hand corner, then navigate to More Tools and open Developer Tools. Alternatively, right clicking your webpage and selecting "Inspect" will also navigate you to Chrome's Developer Tools.

You will use this console to write JavaScript code that will directly modify and interact with this web page. You will type your code into the console, then press Enter to run it.

To try this out, enter the following code into Chrome's console:

¹ A scripting language is a programming language the supports scripts: a series of commands within a file that is capable of being executed without being compiled.

```
console.log("Hello World!");
```

This will print out the line "Hello World!" directly beneath the code you just ran. `console.log()` is the JavaScript command for printing statements, and is equivalent to `System.out.println()` in Java.

Note that beneath the "Hello World!" line there is one other line that gets printed: a light grey line stating "undefined". After executing your print statement, Chrome attempts to return some sort of value for your program. The "undefined" keyword signals that nothing has been returned; it is the JavaScript equivalent of null.

Variables

JavaScript stores data in variables. You can declare variables in JavaScript using the keywords "const" and "let". You should use "const" when the variable's value will never be reassigned, and "let" when the variable's value might be reassigned.

Note that "let" does not only indicate that the variable's value may change, but also that the variable will only be used in the block that it's defined in. This means that a variable declared with "let" may not always be accessible by the entirety of your code.

It's standard practice to use "const" to declare variables whenever applicable. We recommend you use "const" as often as your own code allows.

With this in mind, there is one other way that you can declare variables in JavaScript: with the keyword "var". However, we do not recommend the use of "var", as this keyword offers the least amount of information about the data stored in the variable. It has confusing behavior, and using this keyword is generally seen as bad practice.

Note that JavaScript doesn't care what type of data the variable is, and you therefore do not need to explicitly declare a variable's type when declaring the variable.

JavaScript will automatically detect a variable's type depending on the value associated with that variable. You can check this by running "typeof [VARIABLE NAME];" into Chrome's console.

In order to get comfortable using and declaring variables, we're going to run a couple of tests within Chrome's console. Run the following lines:

```
const name = "Jeff";  
const token = "Top hat";  
let money = 1500;
```

These variables are saved locally within the console. To test this, try running `console.log(name)`. The printed line should display "Jeff".

Now try running the following code:

```
name = "Jeff Huang";
```

The console should return an error stating "Uncaught TypeError: Assignment to constant variable". This is because we tried to reassign the value of a variable we declared with the keyword "const".

Now try running the following code:

```
money -= 500;  
console.log(money);
```

The console should not return an error, as we declared money with "let". Instead, the console should return a value of 1000.

In addition to the primitive data types we've been declaring thus far, JavaScript also has variable objects. Objects are variables that contain information in the form of key-value pairs. JavaScript objects are very similar to hashmaps, except they're able to store variables of differing data types.

Try running the following code:

```
const player = {  
  name: "Jeff",  
  token: "Top hat",  
  money: 1500  
};
```

This creates an object called `player`. You can directly change the values in this object by running the following lines of code:

```
player.money -= 500;  
player.token = "Cat";  
console.log(player.money);  
console.log(player.token);
```

`console.log(player.money)` should return 1000, just as `console.log(player.token)` should return "Cat".

Note that although we declared `player` with `"const"`, we were still able to change the values contained within `player`. In this case, `player` itself is the variable that we cannot reassign to a new object.

As a final note on variables, it's important to know that that you should use `"==="` as opposed to `"=="` in order to compare two variables in JavaScript. `"=="` implicitly casts the variables being compared to the same type, while `"==="` does not. `"==="` will evaluate to false if the inputs are of different types.

Try typing the following code into the console:

```
1 == "1";
```

Although the first input is a number and the second input is a string, the comparator `"=="` will cast both variables to be the same type before checking for equality.

Therefore, `1 == "1"` will evaluate to true. `1 === "1"` will, on the other hand, evaluate to false, simply because `"==="` takes type into account when making a comparison.

In short, `"==="` is better practice for comparing equality in variables, as it simultaneously functions as a type-checker.

Functions

Functions in JavaScript are essentially objects. You declare them like this:

```
function goToJail(player) {.../*function body*/...};
```

Depending on whether a function needs access to data from another source, a function may or may not have arguments. In the example above, the function named "goToJail" takes in a single argument named "player".

To call this function, simply use their name and include the appropriate arguments. In this case, we pass in a variable named "player1", containing the string value "Valerie".

```
const player1 = "Valerie";  
goToJail(player1);
```

In JavaScript, functions may return one of three things: a variable, another function, or the value "undefined". Examples of all three of these types of functions can be found below:

```
function pickRiggedChanceCard() {  
    let chance = "Advance to GO!";  
    return chance;  
};
```

The above function returns the variable "chance", which contains a string value.

```
function counter() {  
    let count = 0;  
    return function() {  
        count++;  
        return count;  
    }  
}
```

This function returns a function named "function", which itself returns a variable. However, running the function "counter" will return the function itself, instead of the variable:

```
f () {  
    count++;
```

```
    return count;
}
```

The final function, displayed below, will return a value of “undefined”. This is due to the fact that there is no return value specified.

```
function goToJail(player) {
    console.log(player + " has been sent to jail!");
}
```

Classes

JavaScript classes are syntactical sugar, providing clear and simple syntax for declaring classes and using inheritance. They are designed to be easy to use, and are useful in approaching JavaScript from a more object-oriented perspective.

To declare a class, use the “class” keyword with the desired name of the class. In this example, we will create a class named “Player”.

```
class Player {
    constructor(name, token, money) {
        this.name = name;
        this.token = token;
        this.money = money;
        this.turn_order = 4;
    }
}
```

Note that you can add additional variables and set them within a class’ constructor, even if the value of those variables are not passed in as arguments.

You can create new instances of the Player class by running the below code:

```
const jeff = new Player("Jeff", "Top hat", 1500);
console.log(jeff.name + " is a " + jeff.token + ", with " + jeff.money + "
```

```
dollars.");
```

Learning about JavaScript classes and their associated syntax will help greatly when we start the Development assignment later on in the semester, as JavaScript classes are very closely related to certain aspects of React - the JavaScript library that we will be using in Development. Specifically, JavaScript classes are closely related to something that React calls “components”.

For those interested: React components are just JavaScript functions. They accept arbitrary inputs and return React elements describing what should be appear on the screen. These are high level concepts, and understanding anything regarding how React works is not necessary for the completion of this lab. However, if you have any questions, feel free to ask any of the TAs!

Stencil Code

Stencil code for the following exercises can be found at this link: [Stencil Code](#)

Please download the code as a .zip or clone the project in order to begin working.

Exercise I

The first exercise aims to give a better introduction to JavaScript variable syntax.

The two files you will need for this exercise are properties.js and properties.html. However, you will only need to work in the properties.js file.

Open properties.js in a text editor (such as Sublime), and open properties.html in Chrome. Note that you will be refreshing this page in order to test your code.

In this exercise, you will create four different object variables: a player and three properties. Each of these variables will have associated key-value pairs that are defined within the variable’s declaration. These variables and their associated key-value pairs will be displayed on the properties.html webpage.

You will be naming the variables and key-value pairs based on the JavaScript code already provided within the stencil code. In order for the program to run properly, you

should name your variables according to the variable calls made towards the bottom of `properties.js`.

You will see a series of lines that follow a syntax similar to the below code.

```
document.getElementById("player").innerHTML = "<p>Alainey</p>";
```

This line changes the HTML code associated with the HTML element named "player." All of the HTML code for this exercise has been provided for you, and we advise you do not change the JavaScript code at the bottom of `properties.js`. However, you will want to reference this code in order to properly define your variables.

If you have any questions about these lines (or any of the HTML code provided) while trying to create your variables, please don't hesitate to ask the TAs!

Exercise II

The second exercise aims to give a better introduction to JavaScript function syntax.

The two files you will need for this exercise are `jail.js` and `jail.html`. However, you will only need to work in the `jail.js` file.

Open `jail.js` in a text editor (such as Sublime), and open `jail.html` in Chrome.

In this exercise, you are attempting to roll your way out of jail. You will be able to "roll" a pair of dice by pressing a button on the `jail.html` webpage. If the two values of the rolled dice are equal, then you will be released from jail! If not, you'll have to continue rolling dice until you roll doubles. However, by the rules of Monopoly, you should also be released from jail after five consecutive fails. Thus, you will need to keep track of how many times you've rolled the dice in order to release yourself from jail after five attempted rolls.

You will be filling out a function named "roll" that runs whenever the button is pressed on the webpage. This function will use an additional function (called "diceRoller") which will be declared and defined by you.

Note that the following code will provide you with a random number from 1 to 6, which will be necessary for the "diceRoller" function.


```
Math.floor((Math.random() * 6) + 1);
```

Additionally, you will want to use the `document.getElementById().innerHTML` call that you saw in the prior exercise in order to display information to the webpage. There are three HTML elements (named “`rolls_text`” “`result_text`” and “`jail_text`”) that are predefined for you. You will be able to change these elements throughout the body of the “roll” function, and we recommend you use the `document.getElementById().innerHTML` in order to do so.

Exercise III

The third exercise aims to give a better introduction to JavaScript class syntax.

The two files you will need for this exercise are `hotels.js` and `hotels.html`. However, you will only need to work in the `hotels.js` file.

Open `hotels.js` in a text editor (such as Sublime), and open `hotels.html` in Chrome.

In this exercise, you will create a `Property` class, and will apply all of the other concepts you've learned from the prior exercises. Within the `Property` class, you will write a constructor that allows you to keep track of the name of a `Property` instance, and keep track of its hotel and house counts. You will also create `buildHouse()` and `buildHotel()` functions, which will enable you to purchase houses and hotels.

Note that you can use the following code in order to assign variables to a class:

```
this.[variable name] = [variable value];
```

Thus, if you wanted to create a variable named “`money`” with a value of 1500, you would use the below code to add it into that class.

```
this.money = 1500;
```

Keep in mind that this would typically happen within the class’ constructor.

Furthermore, you can also use the keyword "this" in order to access a class' variables within that class' functions. In order to access the "money" variable somewhere other than the constructor, you would simply call "this.money".

Make your way through the to-do's, and please ask the TAs if you have any questions. Once you're done with the lab, call a TA over to get checked off!