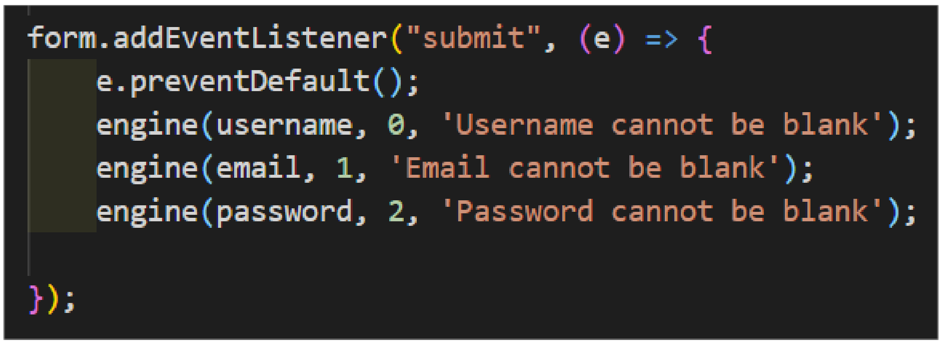
Student: Jorge Alvarez Grana (#249179033)
Class: WDD330
Term: Fall 2021
Subject: W04 Readings

Ch. 8: Forms

Interesting reading about forms and how to make them interactive. Nice start when saying that before in time these interactions used to be done using server languages (PHP or Ruby) but that now that information can be processed using JavaScript (JS). That led me to think why then it is Ruby so highly popular having wage averages of $90,000? And also how JS is becoming 'the' language of webdevelopement because it can be used both in the front and backend environments.

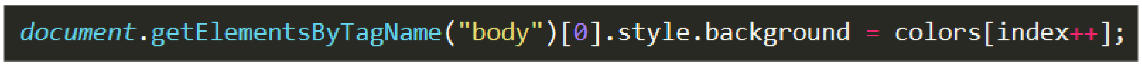
I could see how to have access to HTML elements using *document.getElementById* or similiar ways of referencing them. Also very interesting the idea of using *eventListeners* to see what the user is doing on the keyboard. I had one issue with the text because in one line it used *event.preventDefault()* and Visual Studio Code kept saying it was "deprecated" so I could not make it to work. Later on I found out that the alternative *e.preventDefault()* was the new way of using that method. In my case I used to display an error message and that the text was kept all the time in form unless the user made it right:



```
form.addEventListener("submit", (e) => {
    e.preventDefault();
    engine(username, 0, 'Username cannot be blank');
    engine(email, 1, 'Email cannot be blank');
    engine(password, 2, 'Password cannot be blank');

});
```

I could see this time more clearly that in HTML classes are good but to be used in CSS and ids are better to be used in JS. Of course that is not a rule, first because I used in my code one document.getElementByClassName but I think that making that distinction helped me see the code more organized and clear. As I said, it is not a rule becuase I've used to access the body tag to modify it using CSS the clause:



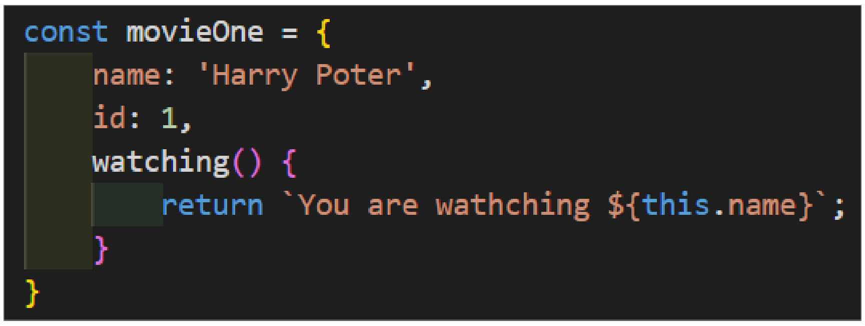
```
document.getElementsByTagName("body")[0].style.background = colors[index++];
```

Ch12: Object Oriented Programming

This is one of the areas where I find more clear that at some point I will need to pick up more programming theory. I don't know if my coding will be much better (because I am aware that I also need much coding practice to arrive to a fluent level), but I am of the opinion that knowing more how computers "think", will make you a better programmer. Having said that, Object Oriented Programming (OOP) is a style of programming that centers more on objects than in functions. In the same manner that I tend to multiplicate *if-else* statements on my code, when a good programmer would just create a function. The same goes with objects if because the alternative is having functions all over the place. That's why OOP is a <u>style of programming.</u> And popular frameworks like Angular was created having OOP in mind.

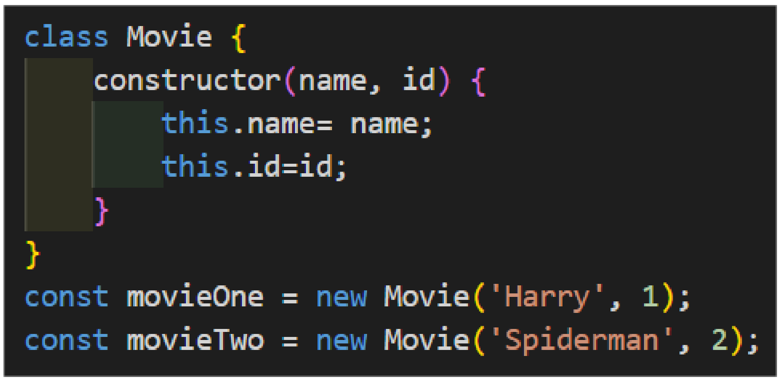
This programming style or concept stands on 4 foundations or techniques: 1) Encapsulation; 2) Abstraction, 3) Inheritance and 4) Polymorphism. *Encapsulation* refers to combining a group of related variables and functions in a unit called *object*, where the variables are *properties* and the functions are called *methods*. *Abstraction* helps functions end up with less parameters, making the code easier to maintain because it helps reduce complexity. *Inheritance* allows reducing redundant code, because different objects can have access and use similar properties. And *polymorphism* allows reducing those long if-else or switch statements that at times inexperienced programmers like me tend to create.

As example, let's image we are making a website for Netflix. We can have a literal object in the following form. But when expanding to several thousan movie, using this paradimg we will have to to create these same lines for each entry (movie), making the code very big:



```
const movieOne = {
    name: 'Harry Poter',
    id: 1,
    watching() {
        return `You are wathching ${this.name}`;
    }
}
```

Using OOP we can use a class where it will work as a template, so to speak. In that class we will create an object using the reseved word *constructor.* Now, the code is greately simplified.



```
class Movie {
    constructor(name, id) {
        this.name= name;
        this.id=id;
    }
}
const movieOne = new Movie('Harry', 1);
const movieTwo = new Movie('Spiderman', 2);
```

Ch15: Modern Javascript

There are thounsands of libraries to help programm in JS, and the most popular according to the book is JSQuery. I've read some place that is not that important nowadays as it was in the past, especially having mpw React, VueJS and Angular (see here and here). But to me it was interesting to see how the simplification transformed code into *$("h1")* coming from the original *document.querySelector("h1")*. Those things are very usefull for programmers and I wonder if going back "raw" again is a real possibility again after having used those libraries.

Interesting also was the idea of MVC, a design pattern that divides an application into three parts: Models, Views and Controllers. The idea behind it all is that an user makes a request and that is handled by the controller. The model will handle the data logic and View will be centered on making the presentation of that data. So in a real life application:

Controller	if (success) View table
Model	SELECT * FROM table
View	<body><h1>Table</h1>...</body>

Lastly, I was interested by Package Managers. Packages can be either stand alone programs or external code libraries, so they are also called at times *dependencies. T*hey are external libraries that our applications depend on. And to manage those files we have the Package Manangers to install, update, erase and list those files. Linux has apt or yum and JS uses npm that comes with NodeJS. So using console we can have access to those libraries easily.