Week 2: Weekly Videos and Curriculum

3. User Input

Note: In this video, we use a **Template Literal**, which you have not seen up to this point, and will be discussed in **Week 4.**

Week 4: ECMAScript6 introduces a concept called **Template Literals**. A **Template Literal** uses something called a *backtick* instead of using single or double quotes; and when used creates a string, evaluating any variables contained inside the backticks which are placed in a construct like this:

```
`Any random text ${variableName}`
```

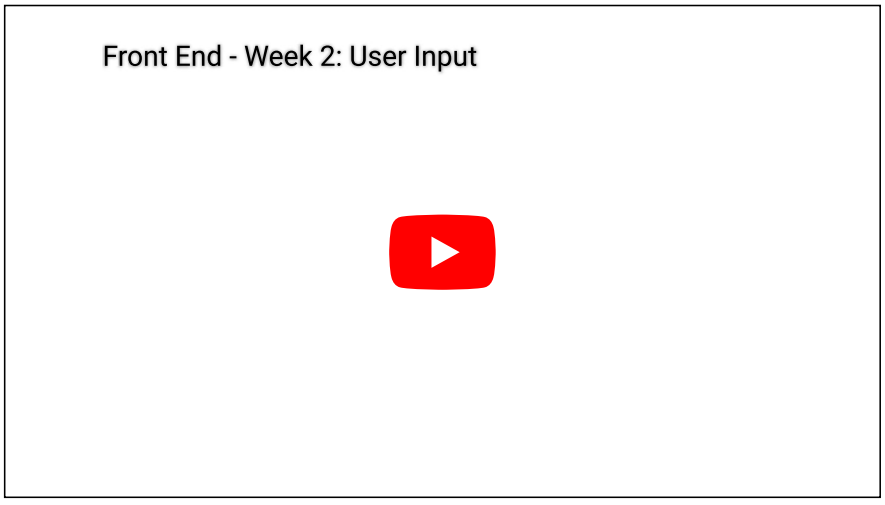
The **User Input** video uses the following **Template Literal** (**bold** in this example using backticks):

```
welcome.alert(`Welcome, ${name}`);
```

The **Template Litera**l creates a string that will print out the exact same result as this code which uses **string concatenation** (with single quotes):

```
welcome.alert('Welcome, ' + name);
```

Template literals are extremely useful, and you will learn much more about them in **Week 4.** Specifically in the **Week 4: Weekly Videos and Curriculum** Tile under **2. ES6 Template Literals**



We need data to tell our programs what to make decisions with; and up to this point, we've been hard coding data into variables to explore coding constructs. However, the original source for most data is **user input**. In order to make decisions based on responses or data entered from a user, let's take a look at one way we can prompt a user to enter some data and then store the data in a variable to use in our code.

Note: this method of interacting with a user is a temporary method for the purpose of being able to receive user input and is not the recommended way in live, production code to interact with users. We will learn additional ways later on.

To receive user input, we can use the `window.prompt()` function. This function takes a **String** as an argument and displays a popup box displaying the **String** and an input text box to receive user input. The function then returns the user input so it can be stored in a variable.

For example:

```
var name = window.prompt('What is your name?');

window.alert('Welcome, ' + name);
```

The above code will display a popup allowing the user to enter their name. It will then store the entered value in the `name` variable. The next line will also display a popup, but without a text input field.

We can quickly see how use input will make our code more dynamic since the values can change depending on what users enter. Let's try the following:

```
var username = prompt('Username:');

var password = prompt('Password:');

if (username == 'samy123' && password == '12345') {

    alert('Welcome back, ' + username);

} else {

    alert('Inaccurate credentials!');

}
```

The above code will prompt a user for a `username` and a `password` and if the `username` and `password` match `samy123` and `12345` relatively, the user will be welcomed. If not, the user will see a message saying their credentials are not correct.

Notice how we don't have to use `window.prompt()` or `window.alert()` - that is because most browsers allow us to omit the window object and call it's methods/functions directly.

With the above example, we have to refresh the browser each time to run it again. So, if a user puts in the incorrect credentials, they see the message and that's it until they refresh the page and try again. To have a better user experience, we would most likely want to prompt the user for their credentials again after a failed login attempt. We can do this with a loop.

```
var loggedIn = false;

while (!loggedIn) {

    var username = prompt('Username:');

    var password = prompt('Password:');

    if (username == 'samy123' && password == '12345') {

        alert('Welcome back, ' + username);

        loggedIn = true;

    } else {
```

```
            alert('Inaccurate credentials!');

        }

    }
```

In the above code, we use a Boolean variable as a flag to determine whether the user is logged in or not. If the user enters the wrong credentials, nothing happens to the `loggedIn` variable, so it remains `false` causing the loop to iterate again and again until the user enters the correct credentials, at which point we update the `loggedIn` variable to `true`, which will cause the loop to stop iterating.

Coding Challenge:

> We could also add a login attempt count that would enable the user to only enter the incorrect credentials a certain number of times before displaying a message like "You are locked out" and ending the loop.
>
> > **Challenge**: See if you can figure out how to implement this feature, using the above code as a starting point.