Lesson | Wednesday

# Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Branching

Text | Cheat sheet

Now that we've learned the basics of DOM manipulation and event handling, we're going to add some logic to our pages that will allow us to control the flow of our code based on certain conditions. This kind of logic is called **branching**.

As an example, we are going to create a website for a bar. We only want to show the drinks menu to the user if they are over 21. So, the condition is age. When we come to determining this condition in our code, it is like a fork in the road with two paths, or branches. If the user is over 21, they'll see the drinks menu. If not, they will get a message indicating that they are not old enough to view the content.

Here's the CSS, HTML, and the JS:

**css/styles.css**

```
.hidden {
  display: none;
}
```

## drinks.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <link rel="stylesheet" href="css/styles.css" type="text/c
ss">
  <script src="js/scripts.js"></script>
  <title>Drinks</title>
</head>
<body>
  <div>
    <form>
      <label for="age">We need to verify that you are over
21 years of age before we can show you the drinks menu. Wha
t is your age?</label>
      <input id="age" type="number">
      <button type="submit">Submit!</button>
    </form>

    <div id="drinks" class="hidden">
      <h1 id="birthday-wish"></h1>
      <h1>Drink menu</h1>

      <h2>Beer</h2>
      <ul>
        <li>The King of Beer</li>
        <li>The Queen of Beer</li>
        <li>Real cold beer</li>
      </ul>

      <h2>Wine</h2>
      <ul>
        <li>Red wine</li>
        <li>White wine</li>
        <li>Blue wine</li>
      </ul>
    </div>

    <div id="under-21" class="hidden">
      <h1>Sorry, we can't serve you.</h1>

      <p>It's illegal in the US. Try Mexico or Europe.</p>
```

```
    </div>
  </div>
</body>
</html>
```

### js/scripts.js

```
window.onload = function() {
  // we get the form element
  const form = document.querySelector("form");
  // we create an event handler for the form's submission
  form.onsubmit = function(event) {
    event.preventDefault();

    // we access two HTML elements, the drink menu and the
message to underaged folks,
    // and add the hidden class to those elements;
    // doing this clears results before displaying new one
s, which
    // allows the user to submit the form again and again,
and
    // see new results.
    let drinkMenu = document.getElementById("drinks");
    drinkMenu.setAttribute("class", "hidden");
    let under21Message = document.getElementById("under-2
1");
    under21Message.setAttribute("class", "hidden");

    // we gather the age value
    const age = parseInt(document.querySelector("input#ag
e").value);

    // we check if the age is greater than 21.
    if (age >= 21) {
      drinkMenu.removeAttribute("class");
    }
  };
};
```

First, we create a form submission event handler and call `event.preventDefault();` to prevent the page from refreshing (the default behavior of the 'submit' event).

Then, we hide the two possible results on the page, the drink menu or the message to underaged people. This may seem odd to do, since these results start off as hidden based on how we've configured our CSS. However, we do this to clear previous results before displaying new ones so that the user can submit the form over and over and always see new results.

Next, we gather the form input. Keep in mind that all form values are strings, no matter if a number is entered. Since we want our age to be compared to another number, we have to change it into a number using the `parseInt()` function.

Next, we add branching.

## Adding an `if` Branch

With the user's age stored in the `age` variable, we're ready to write a JavaScript `if` statement, also known as a **branch**:

```
const age = parseInt(document.querySelector("input#age").va
lue);

if (age >= 21) {
  drinkMenu.removeAttribute("class");
}
```

`If` statements consist of the keyword `if` followed by a **condition** in parentheses. The condition will always evaluate to a boolean value: either `true` or `false`. After the condition, we add the code that we want to execute if the condition is true. This code is set between two curly braces on its own lines.

When the condition evaluates to `true`, the code that follows between the curly braces is executed. If the condition is `false`, the code in the curly braces is not executed.

Also, take note of how we use semicolons here. It's similar to how we use semicolons with functions. Any statements inside of the branch should terminate with semicolons (such as `drinkMenu.removeAttribute("class");`). However, there shouldn't be semicolons on the lines with curly braces.

## How `age` Gets Evaluated

Let's work through a few more examples of our age condition. If our user inputs an `age` of 25, then the condition `age >= 21` becomes `25 >= 21`, which evaluates to `true`. The code in the curly braces is run and the div with an id of *drinks* is shown.

If `age` is 16, `age >= 21` becomes `16 >= 21`, which evaluates to `false`. In this case, the code in the curly braces is skipped and nothing is displayed.

## Adding an `else` Branch

Back to our code, let's show a different message if the condition is `false`, which is when the user is under 21. To do this we'll add an `else` branch:

**js/scripts.js**

```
window.onload = function() {
  const form = document.querySelector("form");
  form.onsubmit = function(event) {
    event.preventDefault();

    let drinkMenu = document.getElementById("drinks");
    drinkMenu.setAttribute("class", "hidden");
    let under21Message = document.getElementById("under-2
1");
    under21Message.setAttribute("class", "hidden");
    const age = parseInt(document.querySelector("input#ag
e").value);

    if (age >= 21) {
      drinkMenu.removeAttribute("class");
    } else {
      under21Message.removeAttribute("class");
    }
  };
};
```

Here, just like before, when the condition in parentheses evaluates to `true`, the code in the first set of curly braces is run. But when the condition evaluates to `false`, the code in the second set of curly braces, after the `else` keyword, is run.

An `else` statement does not need a condition because the code in its curly braces is what will be run when the `if` condition is `false`.

Generally, your code is executed from top to bottom, like a car driving straight down the road. When your car encounters an `if` statement, it is like encountering a fork in the road. You can't take both paths — if the condition is true, the `if` code runs, otherwise the `else` code runs. The car in this metaphor is our `age` variable.

Also, note that the `else` condition doesn't use semicolons, either. Any statements inside the `else` statement should terminate with a semicolon, though.

## Adding an `else if` Statement

What if we need to have more conditions? What if we want to display a special third message if the user is exactly 21? Let's pop up an alert that says "Have some fun, you're just 21!" if the age is equal to `===` 21. To do this, we'll update our `if` statement to check whether `age` is greater than 21, but not equal to it. Then, our `else if` statement will check whether `age` is exactly 21.

**js/scripts.js**

```
window.onload = function() {
  const form = document.querySelector("form");
  form.onsubmit = function(event) {
    event.preventDefault();

    let drinkMenu = document.getElementById("drinks");
    drinkMenu.setAttribute("class", "hidden");
    let under21Message = document.getElementById("under-2
1");
    under21Message.setAttribute("class", "hidden");

    const age = parseInt(document.querySelector("input#ag
e").value);

    if (age > 21) {  // updated to check if age is greater
than 21
      drinkMenu.removeAttribute("class");
    } else if (age === 21) { // new else if statement
      window.alert("Have some fun, you're just 21!");
      drinkMenu.removeAttribute("class");
    } else {
      under21Message.removeAttribute("class");
    }
  };
};
```

`else if` is just like an alternate `if`. It also has a condition to be checked in parentheses next to it, and if that condition is true then the code in curly braces following it is run. But `else if`'s condition is only checked when the `if` condition has already been found to be false. The curly braces with `else if` also don't use semicolons, but any statements inside of them do.

In real life, we use this idea of `if`, `else if`, and `else` all the time. Say you're going to the beach with your friends over the weekend. You all decide that the easiest way to get there would be to drive, but only your friend Lindsey has a car. So, if Lindsey is free, you can all drive with her. But, she never picks up her phone. So, you all decide that as an alternate plan you will take the bus — but only if Lindsey is busy. Then one of your friends remembers that this is a holiday weekend, so the bus might not be running. You roll your eyes at your friend and say "Fine, if we can't find Lindsey, and the buses aren't running, we can all just go to my house and watch movies." That is an `if` statement!

Let's re-interpret this discussion in pseudocode:

```
if (lindsey === "available to drive") { // if Lindsey can d
rive...
  driveToBeach();                       // call the functio
n to drive to the beach.
} else if (busesAreRunning === true) {  // if Lindsey can't
drive and the buses are running...
  busToBeach();                         // take the bus to
the beach.
} else {                                // if all the other
options fail...
  watchMovies();                        // watch movies ins
tead.
}
```

The triple equals sign `===` is easy to mix up with the single equals sign `=`. Remember that the single equals sign ( `=` ) is used to assign a value to a variable. The triple equals sign( `===` ) is used in conditions to check if the values on its left and right sides are equal.

When you are writing an `if` statement, you can have as many `else if` s in it as you like, but you can only have one `if` and one `else`. `if` must be the first statement, and `else` must be the last statement.

When JavaScript tries to figure out if the condition is true, it's looking for a **boolean**. You saw in the "Assignment, Comparison, and Equality Operators" lesson that comparison and equality operators return a boolean — either `true` or `false`. Check out what's going on with this code that we've put into the DevTools console:

```
> const age = 22;
> age > 21;
true
```

We're getting a boolean. So, when we put in this same code `age > 21` into our `if` statement, we're giving it code that specifically evaluates to `true` or `false`. We can also give our `if` statements variables to evaluate that are already set to `true` or `false`:

```
const age = 22;
const ageBool = age > 21; // ageBool will be set to true
if (ageBool) {
  drinkMenu.removeAttribute("class");
} else {
  under21Message.removeAttribute("class");
}
```

# Summary

In this lesson we introduced ourselves to **branching** with `if` statements. Branching determines the flow of your code based on certain conditions.

- There are a few ways to make branching logic. In this lesson we learned about `if` statements.
- We use comparison and equality operators to create conditions for `if` / `else if` statements to evaluate to `true` or `false`:
  - `if (4 > 5)`
  - `if (typeof "hello" === "string")`
  - `if (variableName === "add")`
- `=` sets a variable; `===` compares two things. Don't use `==`.
- `if` statements are made up of `if`, `else if`, and `else` statements. They must include one `if` statement, but it's not required that you use an `else if` or `else` statement in your branching. Somethings to note:
  - You can have an `if` statement all by itself (without an `else if` or an `else`).
  - You can have an `if` ... `else if` without an `else` statement.
  - You can have an `if` ... `else` without an `else if` statement.
  - You can only have one single `if` and `else` statement. You can have as many `else if` statements as you need.
- `if` and `else if` statements require conditions to be evaluated that are listed in parentheses.
- There is no condition (and no parentheses) for `else` statements, since they designate what to do if an `if` or `else if` condition has not evaluated to true.

Previous (/introduction-to-programming/javascript-and-web-browsers/practical-tips-for-researching-web-apis)
Next (/introduction-to-programming/javascript-and-web-browsers/practice-branching)



disable dark mode


(http://www.epicodus.com)