Week 2: Weekly Videos and Curriculum

1. Boolean Operators and Conditions/Conditionals

Boolean Operators

Using programming to automate tasks means the computer needs a way to make decisions. Decisions require comparing and evaluating information available and then deciding which way to proceed. **Boolean values** are essential in these comparisons. With each decision we need the computer to make, we eventually answer in a **yes** or **no** manner -- in JavaScript, **yes** or **no** is represented by: `true` or `false`

Boolean Expression -- Legally Old Enough To Drive Example:

Imagine that we want to know if someone is old enough to drive. To determine that, we need to compare that individuals age: `currentAge` to the age required to drive: `ageRequiredToDrive`

> If the person's `currentAge` is greater than or equal to (`>=`) `ageRequiredToDrive`, then the person can drive. If not, the person cannot legally drive. `true` and `false` are the only two options.

Here is an example of how this could be represented in JavaScript:

```
var ageRequiredToDrive = 16;

var currentAge = 14;

var canPersonDrive = currentAge >= ageRequiredToDrive;

console.log(canPersonDrive);
```

Some Observations:

- The code above will compare our two variables: `ageRequiredToDrive` and `currentAge`
- We use an `if` statement to determine if `currentAge` is greater than or equal to `ageRequiredToDrive`

- If the value stored in `currentAge` is greater than or equal to `ageRequiredToDrive` the result of the Boolean expression (the operation using the `>=` operator) based on the values assigned to those variables is `true`

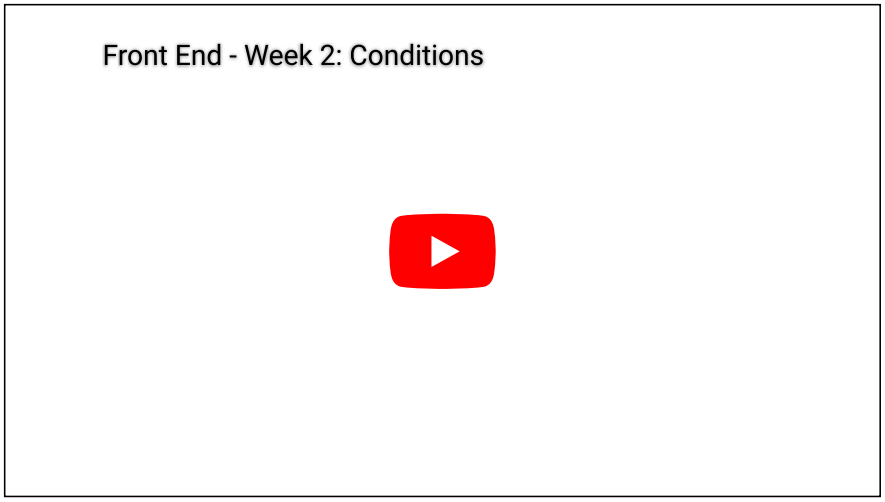
- If the value stored in `currentAge` was less than `ageRequiredToDrive` then the result would be `false`.

- The result of the Boolean expression is assigned to the variable `canPersonDrive` and then printed to the console. In this case, the Boolean expression evaluates to `false`.
- **Note**: If `currentAge` were 16, 17, or another higher number, it would evaluate to `true`

List of **Boolean operators**:

- Less than: `<`
- Greater than: `>`
- Less than or equal to: `<=`
- Greater than or equal to: `>=`
- Equal (type does not matter, i.e. `"3" == 3` is `true`): `==`
- Strictly equal (type matters): `===`

Conditions/Conditionals



Now, simply printing out whether or not a Boolean expression evaluates to `true` or `false` doesn't completely help the computer in making a decision. To make a decision the computer needs to be told that if a Boolean expression evaluates to `true`, then do something, otherwise do something else or even nothing at all. To do this, we use conditionals. The most common conditional is an `if` statement. `if` statements have the following syntax:

```
if (/*Boolean expression*/) {
  //code to run if Boolean expression in parentheses evaluates to true
}
```

`if` Statement -- Legally Old Enough To Drive Example:

The Boolean expression inside of the parentheses following the `if` statement evaluates first, and if it is `true`, then all the code in between the following opening and closing curly brace executes. If the Boolean expression evaluates to `false`, then the code in between the curly braces is skipped and does not execute. Using the previous example, we could do something like this:

```
var ageRequiredToDrive = 16;
```

```
var currentAge = 14;

var canPersonDrive = currentAge >= ageRequiredToDrive;

if (canPersonDrive) {

  console.log('This person can drive');

}
```

In this example, nothing will happen because `canPersonDrive` is `false`

Coding Challenge: Try increasing the `currentAge` to 16 or higher and run it again!

We can also place the Boolean expression directly inside the parentheses instead of creating a variable to hold the value, if we want. This code produces the same result as the previous version.

```
var ageRequiredToDrive = 16;

var currentAge = 14;

if (currentAge >= ageRequiredToDrive) {

  console.log('This person can drive');

}
```

`if/else` Statement -- Legally Old Enough To Drive Example:

What if we want to do something else if the Boolean expression evaluates to `false` rather than simply doing nothing? Then we can use an `else` statment. An `else` statement follows an `if` statement and will execute only if the preceding `if` statement's Boolean expression evaluates to `false`

```
var ageRequiredToDrive = 16;

var currentAge = 14;

if (currentAge >= ageRequiredToDrive) {

  console.log('This person can drive');

} else {

  console.log('This person cannot legally drive');

}
```

If the `currentAge` is greater than or equal to `ageRequiredToDrive` then the code in the first block will execute and 'This person can drive' will be printed. However, if the expression evaluates to `false` (as it will in this case since `currentAge` is only 14), 'This person cannot legally drive' will be printed. Thus we've enabled the computer to make a decision based on comparing data.

`if/else if/else` Statement -- How Many Eggs Example:

Sometimes, there are more than two options in a decision. For example, what if the decision to be made was how many eggs to purchase based on how much each dozen costs?

1. If a dozen of eggs costs $3 or more, we may only want to purchase one dozen.
2. If they are less than $3 but greater than $2 per dozen, we may buy 2 dozen.
3. If they are less than $2, we may buy 3 dozen.
4. And finally, If they are less than a dollar, we want to buy 4 dozen.

To do this, we can add some `else if` statements to our decision:

- `else if` statements work similarly to `if` statements in that they contain a set of parentheses with a Boolean expression and will only execute if that expression evaluates to `true`.
- However, they also function like an `else` statement in that they will not run if the previous `if`, or `else if` Boolean expression is `true`.

Once one of the Boolean expressions evaluates to `true`, that code block will run and the rest will be skipped. If none evaluate to `true`, the final `else` statement is the default code that will run. For example:

`if/else if/else` Example:

```
var costOfEggs = 2.12;

var numberOfDozensOfEggsToPurchase = 0;

if (costOfEggs > 3) {

    numberOfDozensOfEggsToPurchase = 1;

} else if (costOfEggs > 2) {

    numberOfDozensOfEggsToPurchase = 2;

} else if (costOfEggs > 1) {

    numberOfDozensOfEggsToPurchase = 3;

} else {

      numberOfDozensOfEggsToPurchase = 4;

}

console.log('I will buy ' + numberOfDozensOfEggsToPurchase + ' dozen eggs.');
```

If we have a logical decision flow that has many paths, we could use a bunch of `else if` statements, with a single `else` statement at the very end that defines the default code to execute if all of the previous Boolean expressions in the `if` and `else if` statements evaluate to `false`.

`switch` Statement -- Grade Range Example:

There is also another programming construct we can use to create logical paths with multiple options in a similar fashion. This construct is called a `switch` statement and is used to evaluate a variable and then provide multiple different code blocks that could be executed based on the value of the variable.

`switch` Grade Range Example:

```
var grade = 'D';
switch (grade) {
    case 'A':
        console.log('90-100');
        break;
    case 'B':
        console.log('80-89');
        break;
    case 'C':
        console.log('70-79');
        break;
    case 'D':
        console.log('60-69');
        break;
    default:
        console.log('0-59');
}
```