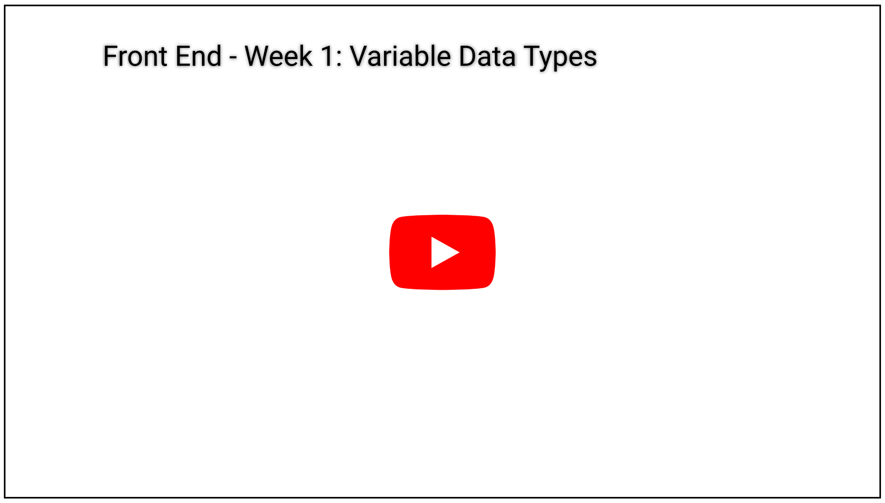
Week 1: Weekly Videos and Curriculum

3. Variables, Data Types, and Operations

Variables and Data Types



Variables:

Since programming is moving, manipulating, and displaying data, we need a way to know what data we are working with. We need a way to **assign names to data**. To do this, we use something called **variables**.

Imagine that we have a piece of data: `25.74` How do we know what that data is? What it represents? It could represent the price for an item, the balance of a bank account, the distance from one place to another. It could represent any number of things. **Variables** allow us to identify what specific data is so that we can refer to the data by its variable name and write instructions telling the computer what to do with it.

Data Types:

Variables can refer to **different types of data**. Different types of data are typically used in different ways. For example, alphanumeric/textual data is typically used to label something (think anything with text), while numeric data deals more with values and math.

Below are some data types that **JavaScript** uses:

- **Boolean** - true or false
- **Number** - numeric values
- **String** - alphanumeric/text values

Variables and Data Types:

In **JavaScript** we don't have to tell a variable what type of data it will hold, the data type is determined implicitly based on the data assigned to the variable. This is called **dynamic typing**. **JavaScript** is also loosely typed, meaning that the data type of a **variable** can change (i.e. a **variable** could be pointing to a **String** value and then changed to hold a **Number** value).

To declare **variables** in **JavaScript**, we start with the keyword `var`. This lets **JavaScript** know we are creating a new **variable**. We then give the **variable** a name, or identifier. This is how we can refer to the variable after it's been declared.

String Example:

For example, if we have customer data we are working with, we may have a **variable** called `customerFirstName` that holds String data representing the first name of a customer. After declaring the variable name/identifier, we use an assignment operator (the equal sign: `=`) followed by the value we want to assign to the variable. Finally, we end the statement (the line of code) with a semicolon. The result is as follows:

```
var customerFirstName = "Sam";
```

That is how we declare a variable. Anywhere we refer to `customerFirstName` name after that line, the computer will substitute the value "Sam".

Number Example:

Below are some examples of variables that have Number values assigned to them.

```
var bankAccountBalance = 100.54;

var numberOfFriends = 10;
```

Notice how there are no quotes around the **Number** values. **Strings** are denoted with single or double quotes, but **Numbers** do not use quotes.

Boolean Example:

Below are some examples of **Boolean** values assigned to variables.

```
var isHotOutside = false;

var isHappy = true;
```

Notice that the Boolean values `true` and `false` also do not have quotes around them.

Camel Case:

When we write variables, notice how the first letter of the first word is lowercase and then each first letter of each following word is uppercase. This is a naming convention called **Camel Case**, or **camelCase** to follow its own convention, and is the naming style we use for variables in **JavaScript**.

Connecting a JavaScript File to an HTML File:

In order to test much of the code we are going to write, we need to run it in a web browser. To do this, we create folder that will house our **JavaScript** files and an **HTML** file that we will open in the browser. We will link our **JavaScript** file to our **HTML** file so that when the **HTML** file loads it runs the **JavaScript**.

To do this, we need to add a script tag to the head element in our **HTML**.

Below is an example, but the *index.js* would be replaced with whatever the name of our **JavaScript** file is.

Note: **JavaScript** files need to end in a **.js** extension and **HTML** files end in a **.html** extension, meaning that a proper **JavaScript** file name would look like: *myFile.js* and a proper **HTML** file name would look like: *myFile.html.*

Example: *index.html*

```
<html>
    <head>
        <script src="index.js"></script>
    </head>
</html>
```

When we open this **HTML** file in the browser, the **JavaScript** file mentioned in the script tag will run.

Operations

Operations:

Knowing that there are different types of data is great, but what is data good for if we don't use it in some way? For example, point of sales systems (the software used when we purchase something from a store, restaurant, etc) have to add up line items and then apply a tax to them. That means we have to perform actions on data (**addition** and **multiplication** in this case). In programming, these actions are called **operations**.

An **operation** consists of one or more pieces of data, known as **operands**, and an **operator**, and performs a calculation or action on the **operands** thus resulting in a new value. One **operator** we are already familiar with is the **assignment operator** (the equals sign `=`), which assigns the data on the right-hand side to the variable name/identifier on it's left. The data on the right and the variable on the left are the **operands** in this **operation**.

```
var name = "Sam";
```

While **operations** take one or more **operands**, most **operations** take exactly **two operands**. For example, our arithmetic operations (**additions**, **subtraction**, **multiplication**, and **division**) take two **Number** operands and perform their related math operation on them.

To see how some of these work, we can use the `console.log()` function, which prints values to the console when executed. To test this out, we need to make sure we are writing our code in a **JavaScript** file that is linked to an **HTML** that we will open in our web browser.

```
console.log(5 + 3);

console.log(4 - 2);

console.log(6 * 7);

console.log(8 / 2);
```

The above **operations** provide examples of **addition**, **subtraction**, **multiplication**, and **division** in that order. The **operation** consists of the **numbers** and **operator** inside the `console.log()` function. The `console.log()` function is not part of the **operation**, rather a statement to print out the results of the operation to the console where we can see them.

Operations also work with **variables**, not just hard-coded values. For example:

```
var bankAccountBalance = 100.24;

bankAccountBalance = bankAccountBalance - 5;

console.log(bankAccountBalance);
```

In the above example, we initialize the `bankAccountBalance` variable with the value `100.24` using an **assignment operator**. On the next line, we use the **assignment operator** again to reassign the value of another **operation** `bankAccountBalance - 5`, which subtracts 5 from the original value and updates the **variable**. Note that there are **two operations** happening on that line.

Short-hand Notation:

As developers, we love making things short. Another way to write the second line in the example above is this:

```
bankAccountBalance -= 5;
```

This is the same thing as saying `bankAccountBalance = bankAccountBalance - 5` There is also the `+=` operator for adding to a variable. Short-hand makes it easier!

Addition: Numbers vs. Strings:

The `+` **operator** can do different things depending on whether the **operands** are **Numbers** or **Strings**. If the **operands** are **Numbers**, then the `+` **operator** performs **addition**. If any of the **operands** are **Strings**, it performs something called **concatenation**, which basically adds the **Strings** together by combining them.

For example:

1. Addition:

```
console.log(5 + 3); //logs out 8
```

2. Concatenation:

```
console.log("Hello" + "World"); //logs out HelloWorld
```

The `+` **operator adds** the values in the first example (**addition** for **Numbers**) and **concatenates** them in the second example (**concatenation** for **Strings**).