Exercise: Working With Variables

Working with Variables Step 1

Using the `console.log`

Before you start working with variables, you'll learn about `console.log`. The console.log() command prints out whatever you put inside the parentheses to the `console`, which you can access in your developer tools. This is useful to find the outcome of a variable, function, or object definition, as well as debug errors in your code. You have watched the faculty use this command in the videos.

In this activity, you will run the `console.log()` command, which is already written for you. This will help you see the changes you make to variables.

For example:

```
console.log('Good day!');
```

Declaring Variables with Var, Let, and Const.

Javascript has three ways to declare variables:

- var: declares a variable that is available from anywhere in the program, and optionally giving it an initial value. This initial value can be changed afterwards.
- let: declares a variable that is only available within a defined scope.
- const: declares a constant. It is read-only and cannot be reassigned a value.

The differences between each variable declaration will gradually make sense as you use them.

Variable Names

Variable names must be unique. You will not be able to declare two variables using the same name.

```
var city = "Sacramento"
let bank_account_balance = 350.00
const API_URL = "https://openweathermap.org/api"
```

Using Variables To Substitute Values In a String

One simple way to begin working with variables is to use them to update content in a string. For instance, let's imagine that a website you're building needs to display a welcome message to each user when they log in with their name in the message like:

`Welcome back, Jeff!`

Without using variables, you would have to write code to create a message for every user on the system and then figure out which one you need to display! Often, this is called "hard coding" when a value that should be dynamic (change easily) is written to never change. So instead, you would use a variable to make the string dynamic. There are two methods to do this:

You can add the variable to the string using +

```
var userName = "Jeff";

"Welcome back, " + userName + "!" // "Welcome back, Jeff!"
```

Or, you can use back-ticks () around the string and then add the variable wherever you want by wrapping it with ${}

```
var userName = "Jeff";

‘Welcome back, ${userName}!‘ // "Welcome back, Jeff!"
```

Note: the back-tick (‘) can be found on the same key as the tilde on a US keyboard.

Task Instructions

In this activity, you will define a variable within a JavaScript function. You'll learn more about functions later. For now, we'll focus on variables.

To accomplish this task, you need to do the following:

Open the Variables-01 folder and add code to the variables01.js file to do the following:

- Create a variable called myFriend using one of the variable declarations described above.
- Set the variable you created to contain your friend's name.
- Inside the greetings function, return the string: "Greetings [your-friend's-name]."

Note: The following line of code `console.log('results: ', greetings());` will print the result of your code to the console. You will learn more about how this works later on in the course.

When you are done, refresh the browser window and open the console to see the results.

Task

- [ ] Declare a variable containing your friend's name, then use that variable to create a greeting message.

Working with Variables Step 2

Scope

In JavaScript `scope` refers to the visibility of variables. Some variables are visible from anywhere in the program, and some are only visible within certain parts of the program.

While writing code, it is very useful to limit a variable's scope. If you set a variable's scope within a function, meaning that variable is not visible from outside the function, then you can make sure that changes within that function will not affect the rest of your program.

Declaring Variables with `let`

`let` allows you to declares a variable that is only available within a defined scope. This is unlike var, which defines a variable globally.

Let's examine the following example:

In this example, the variable name is declared using var. This makes it available both inside and outside the scope defined by the curly braces in the printName function (a function is a block of organized, reusable code that is used to perform a single action. More on this later.)

```
function printName() {
  var name = 'Klopp';
  {
    // This is a new scope defined by the curly braces
    var name = 'Akira';
    console.log(name); // output from inside the scope: Akira
  }
  console.log(name); // output from outside the scope: Akira
}

//this code will output the following:
//Akira
//Akira
```

Notice how when we declared a second variable `name` inside the scope defined by the curly braces, JavaScript just assigned the new value "Akira" to the original variable `name` that was declared outside the scope. This is how `var` works. Once you declare a variable using `var`, that variable will be accessible from anywhere in the program. Changing the value of it anywhere in the program means that will change its value everywhere else.

Now, let's take a look at the same example using `let` this time:

```
function printName() {
  let name = 'Klopp';
  {
    // This is a new scope defined by the curly braces
    let name = 'Akira';
    console.log(name); // output from inside the scope: Akira
  }
  console.log(name); // output from outside the scope: Klopp
}
//They are not the same temp variable, it will log
//Akira
//Klopp
```

As you can see, declaring a new `name` variable inside the scope and changing its value to "Akira" did not affect the original `name` variable value of "Klopp". This is how `let` works. Once you declare a variable using `let`, that variable will only be accessible within the scope in which you declared it. Changing the variable value within a scope will not affect the value of that variable globally.

Task Instructions

Your task in this activity is to declare a variable named `currentYear` using `let`. This variable should have the current year as a value(ex. 1991).

Note: This line of code

```
console.log('current year result: ', currentYear);
```

will print the value of your variable to the console.

Task

- [ ] Create the variable currentYear using let and assign the current year as the value.

Working with Variables Step 3

Declare variables with `const`

`const` declares a constant. It is read-only and cannot be reassigned a value. These constants are defined within a specific scope (much like variables declared using `let`.)

Let's examine the following code example:

```
const person = "Akira";

person = "Klopp"; // This is not allowed.
```

This will cause an error in your program. This code will not run successfully. The error here is that this code is trying to assign a new value, "Klopp", to the constant `person`. A constant is read-only and cannot be assigned new values after being created.

Initialization Constants declared using `const` must be initialized. This means that you must assign an initial value to every `const` you declare.

Here's an example of `const` initialization:

```
const firstName = "Ray";
```

Task Instruction

In this activity, you will declare variables with `let` and `const` keywords. The function `testBestFlavor` needs to print the message: "Ana says, lacroix orange is a better flavor than lacroix grapefruit."

To make this function complete, you need to declare two constants:

- `lacroix1` : This needs to be declared outside the scope of the function `testBestFlavor` using const and it needs to have the value "grapefruit".
- `lacroix2` : This needs to be declared inside the scope of the function `testBestFlavor` using `const` and it needs to have the value "orange".

After you're done with your code changes, check the console to see if the printed message is correct. Then, submit the task.

Task

- [ ] Given the function `testBestFlavor`, return a string with the word 'orange' in the sentence.