Comment Your JavaScript Code

Comments are lines of code that JavaScript will intentionally ignore. Comments are a great way to leave notes to yourself and to other people who will later need to figure out what that code does.

There are two ways to write comments in JavaScript:

Using // will tell JavaScript to ignore the remainder of the text on the current line. This is an in-line comment:

```
// This is an in-line comment.
```

You can make a multi-line comment beginning with /* and ending with */. This is a multi-line comment:

```
/* This is a
multi-line comment */
```

NOTE: As you write code, you should regularly add comments to clarify the function of parts of your code. Good commenting can help communicate the intent of your code—both for others and for your future self.

You Try

Try creating one of each type of comment.

Declare JavaScript Variables

In computer science, data is anything that is meaningful to the computer. JavaScript provides eight different data types which are undefined, null, boolean, string, symbol, bigint, number, and object.

For example, computers distinguish between numbers, such as the number 12, and strings, such as "12", "dog", or "123 cats", which are collections of characters. Computers can perform mathematical operations on a number, but not on a string.

Variables allow computers to store and manipulate data in a dynamic fashion. They do this by using a "label" to point to the data rather than using the data itself. Any of the eight data types may be stored in a variable.

Variables are similar to the x and y variables you use in mathematics, which means they're a simple name to represent the data we want to refer to. Computer variables differ from mathematical variables in that they can store different values at different times.

We tell JavaScript to create or declare a variable by putting the keyword var in front of it, like so:

```
var ourName;
```

creates a variable called `ourName`. In JavaScript we end statements with semicolons. Variable names can be made up of numbers, letters, and $ or _, but may not contain spaces or start with a number.

You Try

Use the var keyword to create a variable called myName.

Storing Values with the Assignment Operator

In JavaScript, you can store a value in a variable with the assignment operator (=).

myVariable = 5;

This assigns the Number value 5 to myVariable.

If there are any calculations to the right of the = operator, those are performed before the value is assigned to the variable on the left of the operator.

```
var myVar;
myVar = 5;
```

First, this code creates a variable named myVar. Then, the code assigns 5 to myVar. Now, if myVar appears again in the code, the program will treat it as if it is 5.

Exercise

Assign the value 7 to variable a.```js

// Setup var a;

// Only change code below this line

```
# Assigning the Value of One Variable to Another

After a value is assigned to a variable using the assignment operator, you can
assign the value of that variable to another variable using the assignment
operator.
```

var myVar; myVar = 5; var myNum; myNum = myVar;

```
The above declares a myVar variable with no value, then assigns it the value
5. Next, a variable named myNum is declared with no value. Then, the contents
of myVar (which is 5) is assigned to the variable myNum. Now, myNum also has
the value of 5.

# Exercise
Assign the contents of a to variable b.
```js
// Setup
var a;
a = 7;
var b;

// Only change code below this line
```
```

Initializing Variables with the Assignment Operator

It is common to initialize a variable to an initial value in the same line as it is declared.

var myVar = 0;

Creates a new variable called myVar and assigns it an initial value of 0.

Exercise

Define a variable a with var and initialize it to a value of 9.

Declare String Variables

Previously you used the following code to declare a variable:

var myName;

But you can also declare a string variable like this:

var myName = "your name";

"your name" is called a string literal. A string literal, or string, is a series of zero or more characters enclosed in single or double quotes.

Exercise

Create two new string variables: `myFirstName` and `myLastName` and assign them the values of your first and last name, respectively.

Understanding Uninitialized Variables

When JavaScript variables are declared, they have an initial value of undefined. If you do a mathematical operation on an undefined variable your result will be NaN which means "Not a Number". If you concatenate a string with an undefined variable, you will get a string of undefined.

Exercise

Initialize the three variables a, b, and c with 5, 10, and "I am a" respectively so that they will not be undefined.

```
// Only change code below this line
var a;
var b;
var c;
// Only change code above this line

a = a + 1;
b = b + 5;
c = c + " String!";
```

Understanding Case Sensitivity in Variables

In JavaScript all variables and function names are case sensitive. This means that capitalization matters.

MYVAR is not the same as MyVar nor myvar. It is possible to have multiple distinct variables with the same name but different casing. It is strongly recommended that for the sake of clarity, you do not use this language feature.

Best Practice

Write variable names in JavaScript in camelCase. In camelCase, multi-word variable names have the first word in lowercase and the first letter of each subsequent word is capitalized.

Examples:

```
var someVariable;
var anotherVariableName;
var thisVariableNameIsSoLong;
```

Exercise

Modify the existing declarations and assignments so their names use camelCase.

Do not create any new variables.

```
// Variable declarations
var StUdLyCapVaR;
var properCamelCase;
var TitleCaseOver;

// Variable assignments
STUDLYCAPVAR = 10;
PRoperCAmelCAse = "A String";
tITLEcASEoVER = 9000;
```

Explore Differences Between the var and let Keywords

One of the biggest problems with declaring variables with the var keyword is that you can easily overwrite variable declarations:

```
var camper = "James";
var camper = "David";
console.log(camper);
```

In the code above, the camper variable is originally declared as James, and is then overridden to be David. The console then displays the string David.

In a small application, you might not run into this type of problem. But as your codebase becomes larger, you might accidentally overwrite a variable that you did not intend to. Because this behavior does not throw an error, searching for and fixing bugs becomes more difficult.

A keyword called let was introduced in ES6, a major update to JavaScript, to solve this potential issue with the var keyword. You'll learn about other ES6 features in later challenges.

If you replace var with let in the code above, it results in an error:

```
let camper = "James";
let camper = "David";
```

The error can be seen in your browser console.

So unlike var, when you use let, a variable with the same name can only be declared once.

Exercise

Update the code so it only uses the let keyword.

```
var catName = "Oliver";
var catSound = "Meow!";
```

Declare a Read-Only Variable with the const Keyword

The keyword let is not the only new way to declare variables. In ES6, you can also declare variables using the const keyword.

const has all the awesome features that let has, with the added bonus that variables declared using const are read-only. They are a constant value, which means that once a variable is assigned with const, it cannot be reassigned:

```
const FAV_PET = "Cats";
FAV_PET = "Dogs";
```

The console will display an error due to reassigning the value of FAV_PET.

You should always name variables you don't want to reassign using the const keyword. This helps when you accidentally attempt to reassign a variable that is meant to stay constant.

Note: It is common for developers to use uppercase variable identifiers for immutable values and lowercase or camelCase for mutable values (objects and arrays). You will learn more about objects, arrays, and immutable and mutable values in later challenges. Also in later challenges, you will see examples of uppercase, lowercase, or camelCase variable identifiers.

Exercise

Change the code so that all variables are declared using let or const. Use let when you want the variable to change, and const when you want the variable to remain constant. Also, rename variables declared with const to conform to common practices. Do not change the strings assigned to the variables.

```
var fCC = "freeCodeCamp"; // Change this line
var fact = "is cool!"; // Change this line
fact = "is awesome!";
console.log(fCC, fact); // Change this line
```

Add Two Numbers with JavaScript

Number is a data type in JavaScript which represents numeric data.

Now let's try to add two numbers using JavaScript.

JavaScript uses the + symbol as an addition operator when placed between two numbers.

Example:

const myVar = 5 + 10;

myVar now has the value 15.

Exercise

Change the 0 so that sum will equal 20.

```
const sum = 10 + 0;
```

Subtract One Number from Another with JavaScript

We can also subtract one number from another.

JavaScript uses the - symbol for subtraction.

Example

const myVar = 12 - 6;

myVar would have the value 6.

Change the 0 so the difference is 12.

```
const difference = 45 - 0;
```

Multiply Two Numbers with JavaScript

We can also multiply one number by another.

JavaScript uses the * symbol for multiplication of two numbers.

Example

const myVar = 13 * 13;

myVar would have the value 169.

Exercise

Change the 0 so that product will equal 80.

```
const product = 8 * 0;
```

Divide One Number by Another with JavaScript

We can also divide one number by another.

JavaScript uses the / symbol for division.

Example

```
const myVar = 16 / 2;
```

myVar now has the value 8.

Exercise

Change the 0 so that the quotient is equal to 2.

```
const quotient = 66 / 0;
```

Increment a Number with JavaScript

You can easily increment or add one to a variable with the ++ operator.

```
i++;
```

is the equivalent of

```
i = i + 1;
```

Note: The entire line becomes i++;, eliminating the need for the equal sign.

Exercise

Change the code to use the ++ operator on myVar.

```
let myVar = 87;

// Only change code below this line
myVar = myVar + 1;
```

Decrement a Number with JavaScript

You can easily decrement or decrease a variable by one with the -- operator.

```
i--;
```

is the equivalent of

```
i = i - 1;
```

Note: The entire line becomes i--;, eliminating the need for the equal sign.

Exercise

Change the code to use the -- operator on myVar.

```
let myVar = 11;

// Only change code below this line
myVar = myVar - 1;
```

Create Decimal Numbers with JavaScript

We can store decimal numbers in variables too. Decimal numbers are sometimes referred to as floating point numbers or floats.

Note: when you compute numbers, they are computed with finite precision. Operations using floating points may lead to different results than the desired outcome.

Exercise

Create a variable `myDecimal` and give it a decimal value with a fractional part (e.g. 5.7).

```
const ourDecimal = 5.7;

// Only change code below this line
```

Multiply Two Numbers with JavaScript

We can also multiply one number by another.

JavaScript uses the * symbol for multiplication of two numbers.

Example

const myVar = 13 * 13;

myVar would have the value 169.

Exercise

Change the 0 so that product will equal 80.

```
const product = 8 * 0;
```

Divide One Decimal by Another with JavaScript

Now let's divide one decimal by another.

Exercise

Change the 0.0 so that quotient will equal to 2.2.

```
const quotient = 0.0 / 2.0; // Change this line
```

Finding a Remainder in JavaScript

The remainder operator % gives the remainder of the division of two numbers.

Example

```
5 % 2 = 1 because
Math.floor(5 / 2) = 2 (Quotient)
2 * 2 = 4
5 - 4 = 1 (Remainder)
```

Usage In mathematics, a number can be checked to be even or odd by checking the remainder of the division of the number by 2.

```
17 % 2 = 1 (17 is Odd)
48 % 2 = 0 (48 is Even)
```

Note: The remainder operator is sometimes incorrectly referred to as the modulus operator. It is very similar to modulus, but does not work properly with negative numbers.

Exercise

Set remainder equal to the remainder of 11 divided by 3 using the remainder (%) operator.

```
const remainder = 0;
```

Compound Assignment With Augmented Addition

In programming, it is common to use assignments to modify the contents of a variable. Remember that everything to the right of the equals sign is evaluated first, so we can say:

```
myVar = myVar + 5;
```

to add 5 to myVar. Since this is such a common pattern, there are operators which do both a mathematical operation and assignment in one step.

One such operator is the += operator.

```
let myVar = 1;
myVar += 5;
console.log(myVar);
```

6 would be displayed in the console.

Exercise

Convert the assignments for a, b, and c to use the += operator.

```
let a = 3;
let b = 17;
let c = 12;

// Only change code below this line
a = a + 12;
b = 9 + b;
c = c + 7;
```

Compound Assignment With Augmented Subtraction

Like the += operator, -= subtracts a number from a variable.

```
myVar = myVar - 5;
```

will subtract 5 from myVar. This can be rewritten as:

```
myVar -= 5;
```

Exercise

Convert the assignments for a, b, and c to use the -= operator.

```
let a = 11;
let b = 9;
let c = 3;

// Only change code below this line
a = a - 6;
b = b - 15;
c = c - 1;
```

Compound Assignment With Augmented Multiplication

The *= operator multiplies a variable by a number.

```
myVar = myVar * 5;
```

will multiply myVar by 5. This can be rewritten as:

```
myVar *= 5;
```

Exercise

Convert the assignments for a, b, and c to use the *= operator.

```
let a = 5;
let b = 12;
let c = 4.6;

// Only change code below this line
a = a * 5;
b = 3 * b;
c = c * 10;
```

Compound Assignment With Augmented Division

The /= operator divides a variable by another number.

```
myVar = myVar / 5;
```

Will divide myVar by 5. This can be rewritten as:

```
myVar /= 5;
```

Exercise

Convert the assignments for a, b, and c to use the /= operator.

```
let a = 48;
let b = 108;
let c = 33;

// Only change code below this line
a = a / 12;
b = b / 4;
c = c / 11;
```