## Basic JavaScript: Comment Your JavaScript CodePassed

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.

You can make a multi-line comment beginning with /\* and ending with \*/:

/\* This is a

multi-line comment \*/

**Best Practice**  
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.

Try creating one of each type of comment.

//One line

/\*Two

lines\*/

## Basic JavaScript: Declare JavaScript VariablesPassed

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.

Use the var keyword to create a variable called myName.

**Hint**  
Look at the ourName example above if you get stuck.

var myName;

## Basic JavaScript: Storing Values with the Assignment OperatorPassed

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.

Assign the value 7 to variable a.

// Setup

var a;

// Only change code below this line

a = 7;

## Basic JavaScript: Assigning the Value of One Variable to AnotherPassed

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.

Assign the contents of a to variable b.

// Setup

var a;

a = 7;

var b;

// Only change code below this line

b = a;

## Basic JavaScript: Initializing Variables with the Assignment OperatorPassed

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.

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

var a = 9;

## Basic JavaScript: Understanding Uninitialized VariablesPassed

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 literal string of "undefined".

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 = 5;

b = 10;

c = "I am a";

a = a + 1;

b = b + 5;

c = c + " String!";

## Basic JavaScript: Understanding Case Sensitivity in VariablesPassed

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;

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;

## Basic JavaScript: Add Two Numbers with JavaScriptPassed

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:**

myVar = 5 + 10; // assigned 15

Change the 0 so that sum will equal 20.

var sum = 10 + 10;

## Basic JavaScript: Subtract One Number from Another with JavaScriptPassed

We can also subtract one number from another.

JavaScript uses the - symbol for subtraction.

**Example**

myVar = 12 - 6; // assigned 6

Change the 0 so the difference is 12.

var difference = 45 - 33;

## Basic JavaScript: Multiply Two Numbers with JavaScriptPassed

We can also multiply one number by another.

JavaScript uses the \* symbol for multiplication of two numbers.

**Example**

myVar = 13 \* 13; // assigned 169

Change the 0 so that product will equal 80

var product = 8 \* 10;

## Basic JavaScript: Divide One Number by Another with JavaScriptPassed

We can also divide one number by another.

JavaScript uses the / symbol for division.

**Example**

myVar = 16 / 2; // assigned 8

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

var quotient = 66 / 33;

## Basic JavaScript: Increment a Number with JavaScriptPassed

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.

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

var myVar = 87;

// Only change code below this line

myVar++;

## Basic JavaScript: Decrement a Number with JavaScriptPassed

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.

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

var myVar = 11;

// Only change code below this line

myVar--;

## Basic JavaScript: Create Decimal Numbers with JavaScriptPassed

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

**Note**  
Not all real numbers can accurately be represented in floating point. This can lead to rounding errors. [Details Here](https://en.wikipedia.org/wiki/Floating_point#Accuracy_problems).

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

var ourDecimal = 5.7;

// Only change code below this line

var myDecimal = 4.6;

## Basic JavaScript: Multiply Two Decimals with JavaScriptPassed

In JavaScript, you can also perform calculations with decimal numbers, just like whole numbers.

Let's multiply two decimals together to get their product.

Change the 0.0 so that product will equal 5.0.

var product = 2.0 \* 2.5;

## Basic JavaScript: Divide One Decimal by Another with JavaScriptPassed

Now let's divide one decimal by another.

Change the 0.0 so that quotient will equal to 2.2.

var quotient = 4.4 / 2.0; // Change this line

## Basic JavaScript: Finding a Remainder in JavaScriptPassed

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.

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

// Only change code below this line

var remainder;

remainder = 11%3;

## Basic JavaScript: Compound Assignment With Augmented AdditionPassed

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.

var myVar = 1;

myVar += 5;

console.log(myVar); // Returns 6

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

var a = 3;

var b = 17;

var c = 12;

// Only change code below this line

a += 12;

b += 9;

c += 7;

## Basic JavaScript: Compound Assignment With Augmented SubtractionPassed

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

myVar = myVar - 5;

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

myVar -= 5;

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

var a = 11;

var b = 9;

var c = 3;

// Only change code below this line

a -= 6;

b -= 15;

c -= 1;

## Basic JavaScript: Compound Assignment With Augmented MultiplicationPassed

The \*= operator multiplies a variable by a number.

myVar = myVar \* 5;

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

myVar \*= 5;

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

var a = 5;

var b = 12;

var c = 4.6;

// Only change code below this line

a \*= 5;

b \*= 3;

c \*= 10;

## Basic JavaScript: Compound Assignment With Augmented DivisionPassed

The /= operator divides a variable by another number.

myVar = myVar / 5;

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

myVar /= 5;

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

var a = 48;

var b = 108;

var c = 33;

// Only change code below this line

a /= 12;

b /= 4;

c /= 11;

## Basic JavaScript: Declare String VariablesPassed

Previously we have used the code

var myName = "your name";

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

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

var myFirstName = "Andres";

var myLastName = "Bucheli";

## Basic JavaScript: Escaping Literal Quotes in StringsPassed

When you are defining a string you must start and end with a single or double quote. What happens when you need a literal quote: " or ' inside of your string?

In JavaScript, you can escape a quote from considering it as an end of string quote by placing a backslash (\) in front of the quote.

var sampleStr = "Alan said, \"Peter is learning JavaScript\".";

This signals to JavaScript that the following quote is not the end of the string, but should instead appear inside the string. So if you were to print this to the console, you would get:

Alan said, "Peter is learning JavaScript".

Use backslashes to assign a string to the myStr variable so that if you were to print it to the console, you would see:

I am a "double quoted" string inside "double quotes".

var myStr = "I am a \"double quoted\" string inside \"double quotes\"."; // Change this line

## Basic JavaScript: Quoting Strings with Single Quotes

*String* values in JavaScript may be written with single or double quotes, as long as you start and end with the same type of quote. Unlike some other programming languages, single and double quotes work the same in JavaScript.

doubleQuoteStr = "This is a string";

singleQuoteStr = 'This is also a string';

The reason why you might want to use one type of quote over the other is if you want to use both in a string. This might happen if you want to save a conversation in a string and have the conversation in quotes. Another use for it would be saving an <a> tag with various attributes in quotes, all within a string.

conversation = 'Finn exclaims to Jake, "Algebraic!"';

However, this becomes a problem if you need to use the outermost quotes within it. Remember, a string has the same kind of quote at the beginning and end. But if you have that same quote somewhere in the middle, the string will stop early and throw an error.

goodStr = 'Jake asks Finn, "Hey, let\'s go on an adventure?"';

badStr = 'Finn responds, "Let's go!"'; // Throws an error

In the *goodStr* above, you can use both quotes safely by using the backslash \ as an escape character. **Note**  
The backslash \ should not be confused with the forward slash /. They do not do the same thing.

Change the provided string to a string with single quotes at the beginning and end and no escape characters.

Right now, the <a> tag in the string uses double quotes everywhere. You will need to change the outer quotes to single quotes so you can remove the escape characters.

var myStr = '<a href="http://www.example.com" target="\_blank">Link</a>';

**Basic JavaScript: Escape Sequences in StringsPassed**

Quotes are not the only characters that can be *escaped* inside a string. There are two reasons to use escaping characters:

1. To allow you to use characters you may not otherwise be able to type out, such as a carriage return.
2. To allow you to represent multiple quotes in a string without JavaScript misinterpreting what you mean.

We learned this in the previous challenge.

| **Code** | **Output** |
| --- | --- |
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \n | newline |
| \r | carriage return |
| \t | tab |
| \b | word boundary |
| \f | form feed |

*Note that the backslash itself must be escaped in order to display as a backslash.*

Assign the following three lines of text into the single variable myStr using escape sequences.

FirstLine  
    \SecondLine  
ThirdLine

You will need to use escape sequences to insert special characters correctly. You will also need to follow the spacing as it looks above, with no spaces between escape sequences or words.

Here is the text with the escape sequences written out.

FirstLinenewlinetabbackslashSecondLinenewlineThirdLine

var myStr = "FirstLine\n\t\\SecondLine\nThirdLine"; // Change this line

## Basic JavaScript: Concatenating Strings with Plus OperatorPassed

In JavaScript, when the + operator is used with a String value, it is called the concatenation operator. You can build a new string out of other strings by concatenating them together.

**Example**

'My name is Alan,' + ' I concatenate.'

**Note**  
Watch out for spaces. Concatenation does not add spaces between concatenated strings, so you'll need to add them yourself.

Example:

var ourStr = "I come first. " + "I come second.";

// ourStr is "I come first. I come second."

Build myStr from the strings "This is the start. " and "This is the end." using the + operator.

var myStr ="This is the start. "+"This is the end."; // Change this line

## Basic JavaScript: Concatenating Strings with the Plus Equals OperatorPassed

We can also use the += operator to concatenate a string onto the end of an existing string variable. This can be very helpful to break a long string over several lines.

**Note**  
Watch out for spaces. Concatenation does not add spaces between concatenated strings, so you'll need to add them yourself.

Example:

var ourStr = "I come first. ";

ourStr += "I come second.";

// ourStr is now "I come first. I come second."

Build myStr over several lines by concatenating these two strings: "This is the first sentence. " and "This is the second sentence." using the += operator. Use the += operator similar to how it is shown in the editor. Start by assigning the first string to myStr, then add on the second string.

// Only change code below this line

var myStr = "This is the first sentence. ";

myStr += "This is the second sentence."

## Basic JavaScript: Constructing Strings with VariablesPassed

Sometimes you will need to build a string, [Mad Libs](https://en.wikipedia.org/wiki/Mad_Libs) style. By using the concatenation operator (+), you can insert one or more variables into a string you're building.

Example:

var ourName = "freeCodeCamp";

var ourStr = "Hello, our name is " + ourName + ", how are you?";

// ourStr is now "Hello, our name is freeCodeCamp, how are you?"

Set myName to a string equal to your name and build myStr with myName between the strings "My name is " and " and I am well!"

// Only change code below this line

var myName = "Andres";

var myStr = "My name is "+myName+" and I am well!";

## Basic JavaScript: Appending Variables to StringsPassed

Just as we can build a string over multiple lines out of string literals, we can also append variables to a string using the plus equals (+=) operator.

Example:

var anAdjective = "awesome!";

var ourStr = "freeCodeCamp is ";

ourStr += anAdjective;

// ourStr is now "freeCodeCamp is awesome!"

Set someAdjective to a string of at least 3 characters and append it to myStr using the += operator.

// Change code below this line

var someAdjective = "really cool";

var myStr = "Learning to code is ";

myStr += someAdjective;

## Basic JavaScript: Find the Length of a StringPassed

You can find the length of a String value by writing .length after the string variable or string literal.

"Alan Peter".length; // 10

For example, if we created a variable var firstName = "Charles", we could find out how long the string "Charles" is by using the firstName.length property.

Use the .length property to count the number of characters in the lastName variable and assign it to lastNameLength.

// Setup

var lastNameLength = 0;

var lastName = "Lovelace";

// Only change code below this line

lastNameLength = lastName.length;

## Basic JavaScript: Use Bracket Notation to Find the First Character in a StringPassed

Bracket notation is a way to get a character at a specific index within a string.

Most modern programming languages, like JavaScript, don't start counting at 1 like humans do. They start at 0. This is referred to as Zero-based indexing.

For example, the character at index 0 in the word "Charles" is "C". So if var firstName = "Charles", you can get the value of the first letter of the string by using firstName[0].

Example:

var firstName = "Charles";

var firstLetter = firstName[0]; // firstLetter is "C"

Use bracket notation to find the first character in the lastName variable and assign it to firstLetterOfLastName.

**Hint:** Try looking at the example above if you get stuck.

// Setup

var firstLetterOfLastName = "";

var lastName = "Lovelace";

// Only change code below this line

firstLetterOfLastName = lastName[0]; // Change this line

## Basic JavaScript: Understand String ImmutabilityPassed

In JavaScript, String values are immutable, which means that they cannot be altered once created.

For example, the following code:

var myStr = "Bob";

myStr[0] = "J";

cannot change the value of myStr to "Job", because the contents of myStr cannot be altered. Note that this does not mean that myStr cannot be changed, just that the individual characters of a string literal cannot be changed. The only way to change myStr would be to assign it with a new string, like this:

var myStr = "Bob";

myStr = "Job";

Correct the assignment to myStr so it contains the string value of Hello World using the approach shown in the example above.

// Setup

var myStr = "Jello World";

// Only change code below this line

myStr = "Hello World"; // Change this line

// Only change code above this line

## Basic JavaScript: Use Bracket Notation to Find the Nth Character in a StringPassed

You can also use bracket notation to get the character at other positions within a string.

Remember that computers start counting at 0, so the first character is actually the zeroth character.

Example:

var firstName = "Ada";

var secondLetterOfFirstName = firstName[1]; // secondLetterOfFirstName is "d"

Let's try to set thirdLetterOfLastName to equal the third letter of the lastName variable using bracket notation.

**Hint:** Try looking at the example above if you get stuck.

// Setup

var lastName = "Lovelace";

// Only change code below this line

var thirdLetterOfLastName = lastName[2]; // Change this line

## Basic JavaScript: Use Bracket Notation to Find the Last Character in a StringPassed

In order to get the last letter of a string, you can subtract one from the string's length.

For example, if var firstName = "Charles", you can get the value of the last letter of the string by using firstName[firstName.length - 1].

Example:

var firstName = "Charles";

var lastLetter = firstName[firstName.length - 1]; // lastLetter is "s"

Use bracket notation to find the last character in the lastName variable.

**Hint:** Try looking at the example above if you get stuck.

// Setup

var lastName = "Lovelace";

// Only change code below this line

var lastLetterOfLastName = lastName[lastName.length-1]; // Change this line

## Basic JavaScript: Use Bracket Notation to Find the Nth-to-Last Character in a StringPassed

You can use the same principle we just used to retrieve the last character in a string to retrieve the Nth-to-last character.

For example, you can get the value of the third-to-last letter of the var firstName = "Charles" string by using firstName[firstName.length - 3]

Example:

var firstName = "Charles";

var thirdToLastLetter = firstName[firstName.length - 3]; // thirdToLastLetter is "l"

Use bracket notation to find the second-to-last character in the lastName string.

**Hint:** Try looking at the example above if you get stuck.

// Setup

var lastName = "Lovelace";

// Only change code below this line

var secondToLastLetterOfLastName = lastName[lastName.length-2]; // Change this line

## Basic JavaScript: Word BlanksPassed

We will now use our knowledge of strings to build a "[Mad Libs](https://en.wikipedia.org/wiki/Mad_Libs)" style word game we're calling "Word Blanks". You will create an (optionally humorous) "Fill in the Blanks" style sentence.

In a "Mad Libs" game, you are provided sentences with some missing words, like nouns, verbs, adjectives and adverbs. You then fill in the missing pieces with words of your choice in a way that the completed sentence makes sense.

Consider this sentence - "It was really **\_\_\_\_**, and we **\_\_\_\_** ourselves **\_\_\_\_**". This sentence has three missing pieces- an adjective, a verb and an adverb, and we can add words of our choice to complete it. We can then assign the completed sentence to a variable as follows:

var sentence = "It was really " + "hot" + ", and we " + "laughed" + " ourselves " + "silly" + ".";

In this challenge, we provide you with a noun, a verb, an adjective and an adverb. You need to form a complete sentence using words of your choice, along with the words we provide.

You will need to use the string concatenation operator + to build a new string, using the provided variables: myNoun, myAdjective, myVerb, and myAdverb. You will then assign the formed string to the wordBlanks variable. You should not change the words assigned to the variables.

You will also need to account for spaces in your string, so that the final sentence has spaces between all the words. The result should be a complete sentence.

var myNoun = "dog";

var myAdjective = "big";

var myVerb = "ran";

var myAdverb = "quickly";

// Only change code below this line

var wordBlanks = "The "+myNoun+" was really "+myAdjective+" and I "+myVerb+" very "+myAdverb+" because it really wanted to bite me."; // Change this line

// Only change code above this line

## Basic JavaScript: Store Multiple Values in one Variable using JavaScript ArraysPassed

With JavaScript array variables, we can store several pieces of data in one place.

You start an array declaration with an opening square bracket, end it with a closing square bracket, and put a comma between each entry, like this:

var sandwich = ["peanut butter", "jelly", "bread"].

Modify the new array myArray so that it contains both a string and a number (in that order).

**Hint**  
Refer to the example code in the text editor if you get stuck.

// Only change code below this line

var myArray = ['A',1];

## Basic JavaScript: Nest one Array within Another ArrayPassed

You can also nest arrays within other arrays, like below:

[["Bulls", 23], ["White Sox", 45]]

This is also called a multi-dimensional array.

Create a nested array called myArray.

// Only change code below this line

var myArray = [[1,2],[3,4]];

## Basic JavaScript: Access Array Data with IndexesPassed

We can access the data inside arrays using indexes.

Array indexes are written in the same bracket notation that strings use, except that instead of specifying a character, they are specifying an entry in the array. Like strings, arrays use zero-based indexing, so the first element in an array has an index of 0.

**Example**

var array = [50,60,70];

array[0]; // equals 50

var data = array[1]; // equals 60

**Note**  
There shouldn't be any spaces between the array name and the square brackets, like array [0]. Although JavaScript is able to process this correctly, this may confuse other programmers reading your code.

Create a variable called myData and set it to equal the first value of myArray using bracket notation.

// Setup

var myArray = [50,60,70];

// Only change code below this line

var myData = myArray[0];

## Basic JavaScript: Modify Array Data With IndexesPassed

Unlike strings, the entries of arrays are mutable and can be changed freely.

**Example**

var ourArray = [50,40,30];

ourArray[0] = 15; // equals [15,40,30]

**Note**  
There shouldn't be any spaces between the array name and the square brackets, like array [0]. Although JavaScript is able to process this correctly, this may confuse other programmers reading your code.

Modify the data stored at index 0 of myArray to a value of 45.

// Setup

var myArray = [18,64,99];

// Only change code below this line

myArray[0] = 45;

## Basic JavaScript: Access Multi-Dimensional Arrays With IndexesPassed

One way to think of a multi-dimensional array, is as an array of arrays. When you use brackets to access your array, the first set of brackets refers to the entries in the outer-most (the first level) array, and each additional pair of brackets refers to the next level of entries inside.

**Example**

var arr = [

[1,2,3],

[4,5,6],

[7,8,9],

[[10,11,12], 13, 14]

];

arr[3]; // equals [[10,11,12], 13, 14]

arr[3][0]; // equals [10,11,12]

arr[3][0][1]; // equals 11

**Note**  
There shouldn't be any spaces between the array name and the square brackets, like array [0][0] and even this array [0] [0] is not allowed. Although JavaScript is able to process this correctly, this may confuse other programmers reading your code.

Using bracket notation select an element from myArray such that myData is equal to 8.

// Setup

var myArray = [[1,2,3], [4,5,6], [7,8,9], [[10,11,12], 13, 14]];

// Only change code below this line

var myData = myArray[2][1];

## Basic JavaScript: Manipulate Arrays With push()Passed

An easy way to append data to the end of an array is via the push() function.

.push() takes one or more parameters and "pushes" them onto the end of the array.

Examples:

var arr1 = [1,2,3];

arr1.push(4);

// arr1 is now [1,2,3,4]

var arr2 = ["Stimpson", "J", "cat"];

arr2.push(["happy", "joy"]);

// arr2 now equals ["Stimpson", "J", "cat", ["happy", "joy"]]

Push ["dog", 3] onto the end of the myArray variable.

// Setup

var myArray = [["John", 23], ["cat", 2]];

// Only change code below this line

myArray.push(["dog", 3]);

## Basic JavaScript: Manipulate Arrays With pop()Passed

Another way to change the data in an array is with the .pop() function.

.pop() is used to "pop" a value off of the end of an array. We can store this "popped off" value by assigning it to a variable. In other words, .pop() removes the last element from an array and returns that element.

Any type of entry can be "popped" off of an array - numbers, strings, even nested arrays.

var threeArr = [1, 4, 6];

var oneDown = threeArr.pop();

console.log(oneDown); // Returns 6

console.log(threeArr); // Returns [1, 4]

Use the .pop() function to remove the last item from myArray, assigning the "popped off" value to removedFromMyArray.

// Setup

var myArray = [["John", 23], ["cat", 2]];

// Only change code below this line

var removedFromMyArray = myArray.pop();

## Basic JavaScript: Manipulate Arrays With shift()Passed

pop() always removes the last element of an array. What if you want to remove the first?

That's where .shift() comes in. It works just like .pop(), except it removes the first element instead of the last.

Example:

var ourArray = ["Stimpson", "J", ["cat"]];

var removedFromOurArray = ourArray.shift();

// removedFromOurArray now equals "Stimpson" and ourArray now equals ["J", ["cat"]].

Use the .shift() function to remove the first item from myArray, assigning the "shifted off" value to removedFromMyArray.

// Setup

var myArray = [["John", 23], ["dog", 3]];

// Only change code below this line

var removedFromMyArray = myArray.shift();

## Basic JavaScript: Manipulate Arrays With unshift()Passed

Not only can you shift elements off of the beginning of an array, you can also unshift elements to the beginning of an array i.e. add elements in front of the array.

.unshift() works exactly like .push(), but instead of adding the element at the end of the array, unshift() adds the element at the beginning of the array.

Example:

var ourArray = ["Stimpson", "J", "cat"];

ourArray.shift(); // ourArray now equals ["J", "cat"]

ourArray.unshift("Happy");

// ourArray now equals ["Happy", "J", "cat"]

Add ["Paul",35] to the beginning of the myArray variable using unshift().

// Setup

var myArray = [["John", 23], ["dog", 3]];

myArray.shift();

// Only change code below this line

myArray.unshift(["Paul",35]);

## Basic JavaScript: Shopping ListPassed

Create a shopping list in the variable myList. The list should be a multi-dimensional array containing several sub-arrays.

The first element in each sub-array should contain a string with the name of the item. The second element should be a number representing the quantity i.e.

["Chocolate Bar", 15]

There should be at least 5 sub-arrays in the list.

var myList = [["Coca Cola cans", 7],

            ["Packets of Doritos", 7],

            ["Snickers bars", 7],

            ["Mineral water bottles", 4],

            ["Apples", 4]

]

**Basic JavaScript: Write Reusable JavaScript with FunctionsPassed**

In JavaScript, we can divide up our code into reusable parts called *functions*.

Here's an example of a function:

function functionName() {

console.log("Hello World");

}

You can call or *invoke* this function by using its name followed by parentheses, like this: functionName(); Each time the function is called it will print out the message "Hello World" on the dev console. All of the code between the curly braces will be executed every time the function is called.

1. Create a function called reusableFunction which prints "Hi World" to the dev console.
2. Call the function.
3. function reusableFunction(){
4. console.log("Hi World");
5. }
6. reusableFunction();

**Basic JavaScript: Passing Values to Functions with ArgumentsPassed**

*Parameters* are variables that act as placeholders for the values that are to be input to a function when it is called. When a function is defined, it is typically defined along with one or more parameters. The actual values that are input (or *"passed"*) into a function when it is called are known as *arguments*.

Here is a function with two parameters, param1 and param2:

function testFun(param1, param2) {

console.log(param1, param2);

}

Then we can call testFun: testFun("Hello", "World"); We have passed two arguments, "Hello" and "World". Inside the function, param1 will equal "Hello" and param2 will equal "World". Note that you could call testFun again with different arguments and the parameters would take on the value of the new arguments.

1. Create a function called functionWithArgs that accepts two arguments and outputs their sum to the dev console.
2. Call the function with two numbers as arguments.

function functionWithArgs(param1, param2){

    console.log(param1+param2);

}

functionWithArgs(1,2);

## Basic JavaScript: Global Scope and FunctionsPassed

In JavaScript, scope refers to the visibility of variables. Variables which are defined outside of a function block have Global scope. This means, they can be seen everywhere in your JavaScript code.

Variables which are used without the var keyword are automatically created in the global scope. This can create unintended consequences elsewhere in your code or when running a function again. You should always declare your variables with var.

Using var, declare a global variable named myGlobal outside of any function. Initialize it with a value of 10.

Inside function fun1, assign 5 to oopsGlobal **without** using the var keyword.

// Declare the myGlobal variable below this line

var myGlobal = 10;

function fun1() {

  // Assign 5 to oopsGlobal Here

  oopsGlobal = 5;

}

// Only change code above this line

function fun2() {

  var output = "";

  if (typeof myGlobal != "undefined") {

    output += "myGlobal: " + myGlobal;

  }

  if (typeof oopsGlobal != "undefined") {

    output += " oopsGlobal: " + oopsGlobal;

  }

  console.log(output);

}

## Basic JavaScript: Local Scope and FunctionsPassed

Variables which are declared within a function, as well as the function parameters have local scope. That means, they are only visible within that function.

Here is a function myTest with a local variable called loc.

function myTest() {

var loc = "foo";

console.log(loc);

}

myTest(); // logs "foo"

console.log(loc); // loc is not defined

loc is not defined outside of the function.

The editor has two console.logs to help you see what is happening. Check the console as you code to see how it changes. Declare a local variable myVar inside myLocalScope and run the tests.

**Note:** The console will still have 'ReferenceError: myVar is not defined', but this will not cause the tests to fail.

function myLocalScope() {

  // Only change code below this line

   var myVar = 1;

  console.log('inside myLocalScope', myVar);

}

myLocalScope();

// Run and check the console

// myVar is not defined outside of myLocalScope

console.log('outside myLocalScope', myVar);

## Basic JavaScript: Global vs. Local Scope in FunctionsPassed

It is possible to have both local and global variables with the same name. When you do this, the local variable takes precedence over the global variable.

In this example:

var someVar = "Hat";

function myFun() {

var someVar = "Head";

return someVar;

}

The function myFun will return "Head" because the local version of the variable is present.

Add a local variable to myOutfit function to override the value of outerWear with "sweater".

// Setup

var outerWear = "T-Shirt";

function myOutfit() {

  // Only change code below this line

  var outerWear = "sweater";

  // Only change code above this line

  return outerWear;

}

myOutfit();

## Basic JavaScript: Return a Value from a Function with ReturnPassed

We can pass values into a function with arguments. You can use a return statement to send a value back out of a function.

**Example**

function plusThree(num) {

return num + 3;

}

var answer = plusThree(5); // 8

plusThree takes an argument for num and returns a value equal to num + 3.

Create a function timesFive that accepts one argument, multiplies it by 5, and returns the new value. See the last line in the editor for an example of how you can test your timesFive function.

function timesFive(num){

    return num\*5;

}

timesFive(2);

## Basic JavaScript: Understanding Undefined Value returned from a FunctionPassed

A function can include the return statement but it does not have to. In the case that the function doesn't have a return statement, when you call it, the function processes the inner code but the returned value is undefined.

**Example**

var sum = 0;

function addSum(num) {

sum = sum + num;

}

addSum(3); // sum will be modified but returned value is undefined

addSum is a function without a return statement. The function will change the global sum variable but the returned value of the function is undefined.

Create a function addFive without any arguments. This function adds 5 to the sum variable, but its returned value is undefined.

// Setup

var sum = 0;

function addThree() {

  sum = sum + 3;

}

// Only change code below this line

function addFive() {

  sum = sum + 5;

}

// Only change code above this line

addThree();

addFive();

## Basic JavaScript: Assignment with a Returned ValuePassed

If you'll recall from our discussion of [Storing Values with the Assignment Operator](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/storing-values-with-the-assignment-operator), everything to the right of the equal sign is resolved before the value is assigned. This means we can take the return value of a function and assign it to a variable.

Assume we have pre-defined a function sum which adds two numbers together, then:

ourSum = sum(5, 12);

will call sum function, which returns a value of 17 and assigns it to ourSum variable.

Call the processArg function with an argument of 7 and assign its return value to the variable processed.

// Setup

var processed = 0;

function processArg(num) {

  return (num + 3) / 5;

}

// Only change code below this line

processed = processArg(7);

## Basic JavaScript: Stand in LinePassed

In Computer Science a queue is an abstract Data Structure where items are kept in order. New items can be added at the back of the queue and old items are taken off from the front of the queue.

Write a function nextInLine which takes an array (arr) and a number (item) as arguments.

Add the number to the end of the array, then remove the first element of the array.

The nextInLine function should then return the element that was removed.

function nextInLine(arr, item) {

  // Only change code below this line

  arr.push(item);

  item = arr.shift();

  return item;

  // Only change code above this line

}

// Setup

var testArr = [1,2,3,4,5];

// Display code

console.log("Before: " + JSON.stringify(testArr));

console.log(nextInLine(testArr, 6));

console.log("After: " + JSON.stringify(testArr));

## Basic JavaScript: Understanding Boolean ValuesPassed

Another data type is the Boolean. Booleans may only be one of two values: true or false. They are basically little on-off switches, where true is "on" and false is "off." These two states are mutually exclusive.

**Note**  
Boolean values are never written with quotes. The strings "true" and "false" are not Boolean and have no special meaning in JavaScript.

Modify the welcomeToBooleans function so that it returns true instead of false when the run button is clicked.

function welcomeToBooleans() {

  // Only change code below this line

  return true; // Change this line

  // Only change code above this line

}

## Basic JavaScript: Use Conditional Logic with If StatementsPassed

If statements are used to make decisions in code. The keyword if tells JavaScript to execute the code in the curly braces under certain conditions, defined in the parentheses. These conditions are known as Boolean conditions and they may only be true or false.

When the condition evaluates to true, the program executes the statement inside the curly braces. When the Boolean condition evaluates to false, the statement inside the curly braces will not execute.

**Pseudocode**

if (*condition is true*) {  
  *statement is executed*  
}

**Example**

function test (myCondition) {

if (myCondition) {

return "It was true";

}

return "It was false";

}

test(true); // returns "It was true"

test(false); // returns "It was false"

When test is called with a value of true, the if statement evaluates myCondition to see if it is true or not. Since it is true, the function returns "It was true". When we call test with a value of false, myCondition is not true and the statement in the curly braces is not executed and the function returns "It was false".

Create an if statement inside the function to return "Yes, that was true" if the parameter wasThatTrue is true and return "No, that was false" otherwise.

function trueOrFalse(wasThatTrue) {

  // Only change code below this line

  if(wasThatTrue === true){

    return "Yes, that was true";

  }

  return "No, that was false";

  // Only change code above this line

}

## Basic JavaScript: Comparison with the Equality OperatorPassed

There are many comparison operators in JavaScript. All of these operators return a boolean true or false value.

The most basic operator is the equality operator ==. The equality operator compares two values and returns true if they're equivalent or false if they are not. Note that equality is different from assignment (=), which assigns the value on the right of the operator to a variable on the left.

function equalityTest(myVal) {

if (myVal == 10) {

return "Equal";

}

return "Not Equal";

}

If myVal is equal to 10, the equality operator returns true, so the code in the curly braces will execute, and the function will return "Equal". Otherwise, the function will return "Not Equal". In order for JavaScript to compare two different data types (for example, numbers and strings), it must convert one type to another. This is known as "Type Coercion". Once it does, however, it can compare terms as follows:

1 == 1 // true

1 == 2 // false

1 == '1' // true

"3" == 3 // true

Add the equality operator to the indicated line so that the function will return "Equal" when val is equivalent to 12.

// Setup

function testEqual(val) {

  if (val == 12) { // Change this line

    return "Equal";

  }

  return "Not Equal";

}

testEqual(10);

## Basic JavaScript: Comparison with the Strict Equality OperatorPassed

Strict equality (===) is the counterpart to the equality operator (==). However, unlike the equality operator, which attempts to convert both values being compared to a common type, the strict equality operator does not perform a type conversion.

If the values being compared have different types, they are considered unequal, and the strict equality operator will return false.

**Examples**

3 === 3 // true

3 === '3' // false

In the second example, 3 is a Number type and '3' is a String type.

Use the strict equality operator in the if statement so the function will return "Equal" when val is strictly equal to 7

// Setup

function testStrict(val) {

  if (val === 7) { // Change this line

    return "Equal";

  }

  return "Not Equal";

}

testStrict(10);

## Basic JavaScript: Practice comparing different valuesPassed

In the last two challenges, we learned about the equality operator (==) and the strict equality operator (===). Let's do a quick review and practice using these operators some more.

If the values being compared are not of the same type, the equality operator will perform a type conversion, and then evaluate the values. However, the strict equality operator will compare both the data type and value as-is, without converting one type to the other.

**Examples**

3 == '3' // returns true because JavaScript performs type conversion from string to number

3 === '3' // returns false because the types are different and type conversion is not performed

**Note**  
In JavaScript, you can determine the type of a variable or a value with the typeof operator, as follows:

typeof 3 // returns 'number'

typeof '3' // returns 'string'

The compareEquality function in the editor compares two values using the equality operator. Modify the function so that it returns "Equal" only when the values are strictly equal.

// Setup

function compareEquality(a, b) {

  if (a === b) { // Change this line

    return "Equal";

  }

  return "Not Equal";

}

compareEquality(10, "10");

## Basic JavaScript: Comparison with the Inequality OperatorPassed

The inequality operator (!=) is the opposite of the equality operator. It means "Not Equal" and returns false where equality would return true and vice versa. Like the equality operator, the inequality operator will convert data types of values while comparing.

**Examples**

1 != 2 // true

1 != "1" // false

1 != '1' // false

1 != true // false

0 != false // false

Add the inequality operator != in the if statement so that the function will return "Not Equal" when val is not equivalent to 99

// Setup

function testNotEqual(val) {

  if (val != 99) { // Change this line

    return "Not Equal";

  }

  return "Equal";

}

testNotEqual(10);

## Basic JavaScript: Comparison with the Strict Inequality OperatorPassed

The strict inequality operator (!==) is the logical opposite of the strict equality operator. It means "Strictly Not Equal" and returns false where strict equality would return true and vice versa. Strict inequality will not convert data types.

**Examples**

3 !== 3 // false

3 !== '3' // true

4 !== 3 // true

Add the strict inequality operator to the if statement so the function will return "Not Equal" when val is not strictly equal to 17

// Setup

function testStrictNotEqual(val) {

  if (val !== 17) { // Change this line

    return "Not Equal";

  }

  return "Equal";

}

testStrictNotEqual(10);

## Basic JavaScript: Comparison with the Greater Than OperatorPassed

The greater than operator (>) compares the values of two numbers. If the number to the left is greater than the number to the right, it returns true. Otherwise, it returns false.

Like the equality operator, greater than operator will convert data types of values while comparing.

**Examples**

5 > 3 // true

7 > '3' // true

2 > 3 // false

'1' > 9 // false

Add the greater than operator to the indicated lines so that the return statements make sense.

function testGreaterThan(val) {

  if (val > 100) {  // Change this line

    return "Over 100";

  }

  if (val > 10) {  // Change this line

    return "Over 10";

  }

  return "10 or Under";

}

testGreaterThan(10);

## Basic JavaScript: Comparison with the Greater Than Or Equal To OperatorPassed

The greater than or equal to operator (>=) compares the values of two numbers. If the number to the left is greater than or equal to the number to the right, it returns true. Otherwise, it returns false.

Like the equality operator, greater than or equal to operator will convert data types while comparing.

**Examples**

6 >= 6 // true

7 >= '3' // true

2 >= 3 // false

'7' >= 9 // false

Add the greater than or equal to operator to the indicated lines so that the return statements make sense.

function testGreaterOrEqual(val) {

  if (val >= 20) {  // Change this line

    return "20 or Over";

  }

  if (val >= 10) {  // Change this line

    return "10 or Over";

  }

  return "Less than 10";

}

testGreaterOrEqual(10);

## Basic JavaScript: Comparison with the Less Than OperatorPassed

The less than operator (<) compares the values of two numbers. If the number to the left is less than the number to the right, it returns true. Otherwise, it returns false. Like the equality operator, less than operator converts data types while comparing.

**Examples**

2 < 5 // true

'3' < 7 // true

5 < 5 // false

3 < 2 // false

'8' < 4 // false

Add the less than operator to the indicated lines so that the return statements make sense.

function testLessThan(val) {

  if (val < 25) {  // Change this line

    return "Under 25";

  }

  if (val < 55) {  // Change this line

    return "Under 55";

  }

  return "55 or Over";

}

testLessThan(10);

## Basic JavaScript: Comparison with the Less Than Or Equal To OperatorPassed

The less than or equal to operator (<=) compares the values of two numbers. If the number to the left is less than or equal to the number to the right, it returns true. If the number on the left is greater than the number on the right, it returns false. Like the equality operator, less than or equal to converts data types.

**Examples**

4 <= 5 // true

'7' <= 7 // true

5 <= 5 // true

3 <= 2 // false

'8' <= 4 // false

Add the less than or equal to operator to the indicated lines so that the return statements make sense.

function testLessOrEqual(val) {

  if (val <= 12) {  // Change this line

    return "Smaller Than or Equal to 12";

  }

  if (val <= 24) {  // Change this line

    return "Smaller Than or Equal to 24";

  }

  return "More Than 24";

}

testLessOrEqual(10);

## Basic JavaScript: Comparisons with the Logical And OperatorPassed

Sometimes you will need to test more than one thing at a time. The logical and operator (&&) returns true if and only if the operands to the left and right of it are true.

The same effect could be achieved by nesting an if statement inside another if:

if (num > 5) {

if (num < 10) {

return "Yes";

}

}

return "No";

will only return "Yes" if num is greater than 5 and less than 10. The same logic can be written as:

if (num > 5 && num < 10) {

return "Yes";

}

return "No";

Replace the two if statements with one statement, using the && operator, which will return "Yes" if val is less than or equal to 50 and greater than or equal to 25. Otherwise, will return "No".

function testLogicalAnd(val) {

  // Only change code below this line

  if (val <= 50 && val >= 25){

    return "Yes";

  }

  // Only change code above this line

  return "No";

}

testLogicalAnd(10);

## Basic JavaScript: Comparisons with the Logical Or OperatorPassed

The logical or operator (||) returns true if either of the operands is true. Otherwise, it returns false.

The logical or operator is composed of two pipe symbols: (||). This can typically be found between your Backspace and Enter keys.

The pattern below should look familiar from prior waypoints:

if (num > 10) {

return "No";

}

if (num < 5) {

return "No";

}

return "Yes";

will return "Yes" only if num is between 5 and 10 (5 and 10 included). The same logic can be written as:

if (num > 10 || num < 5) {

return "No";

}

return "Yes";

Combine the two if statements into one statement which returns "Outside" if val is not between 10 and 20, inclusive. Otherwise, return "Inside".

function testLogicalOr(val) {

  // Only change code below this line

  if (val<10 || val >20){

    return "Outside";

  }

  // Only change code above this line

  return "Inside";

}

testLogicalOr(15);

## Basic JavaScript: Introducing Else StatementsPassed

When a condition for an if statement is true, the block of code following it is executed. What about when that condition is false? Normally nothing would happen. With an else statement, an alternate block of code can be executed.

if (num > 10) {

return "Bigger than 10";

} else {

return "10 or Less";

}

Combine the if statements into a single if/else statement.

function testElse(val) {

  var result = "";

  // Only change code below this line

  if (val > 5) {

    result = "Bigger than 5";

  }

  else {

    result = "5 or Smaller";

  }

  // Only change code above this line

  return result;

}

testElse(4);

## Basic JavaScript: Introducing Else If StatementsPassed

If you have multiple conditions that need to be addressed, you can chain if statements together with else if statements.

if (num > 15) {

return "Bigger than 15";

} else if (num < 5) {

return "Smaller than 5";

} else {

return "Between 5 and 15";

}

Convert the logic to use else if statements.

function testElseIf(val) {

  if (val > 10) {

    return "Greater than 10";

  }

  else if (val < 5) {

    return "Smaller than 5";

  }

  else {

    return "Between 5 and 10";

  }

}

testElseIf(7);

## Basic JavaScript: Logical Order in If Else StatementsPassed

Order is important in if, else if statements.

The function is executed from top to bottom so you will want to be careful of what statement comes first.

Take these two functions as an example.

Here's the first:

function foo(x) {

if (x < 1) {

return "Less than one";

} else if (x < 2) {

return "Less than two";

} else {

return "Greater than or equal to two";

}

}

And the second just switches the order of the statements:

function bar(x) {

if (x < 2) {

return "Less than two";

} else if (x < 1) {

return "Less than one";

} else {

return "Greater than or equal to two";

}

}

While these two functions look nearly identical if we pass a number to both we get different outputs.

foo(0) // "Less than one"

bar(0) // "Less than two"

Change the order of logic in the function so that it will return the correct statements in all cases.

function orderMyLogic(val) {

  if (val < 5) {

    return "Less than 5";

  } else if (val < 10) {

    return "Less than 10";

  } else {

    return "Greater than or equal to 10";

  }

}

orderMyLogic(7);

## Basic JavaScript: Chaining If Else StatementsPassed

if/else statements can be chained together for complex logic. Here is pseudocode of multiple chained if / else if statements:

if (condition1) {

statement1

} else if (condition2) {

statement2

} else if (condition3) {

statement3

. . .

} else {

statementN

}

Write chained if/else if statements to fulfill the following conditions:

num < 5 - return "Tiny"  
num < 10 - return "Small"  
num < 15 - return "Medium"  
num < 20 - return "Large"  
num >= 20 - return "Huge"

function testSize(num) {

  // Only change code below this line

  if(num < 5){

    return "Tiny";

  }

  else if (num < 10){

    return "Small";

  }

  else if (num < 15){

    return "Medium";

  }

  else if (num < 20){

    return "Large";

  }

  else {

    return "Huge";

  }

  return "Change Me";

  // Only change code above this line

}

testSize(7);

## Basic JavaScript: Golf CodePassed

In the game of [golf](https://en.wikipedia.org/wiki/Golf) each hole has a par meaning the average number of strokes a golfer is expected to make in order to sink the ball in a hole to complete the play. Depending on how far above or below par your strokes are, there is a different nickname.

Your function will be passed par and strokes arguments. Return the correct string according to this table which lists the strokes in order of priority; top (highest) to bottom (lowest):

| **Strokes** | **Return** |
| --- | --- |
| 1 | "Hole-in-one!" |
| <= par - 2 | "Eagle" |
| par - 1 | "Birdie" |
| par | "Par" |
| par + 1 | "Bogey" |
| par + 2 | "Double Bogey" |
| >= par + 3 | "Go Home!" |

par and strokes will always be numeric and positive. We have added an array of all the names for your convenience.

var names = ["Hole-in-one!", "Eagle", "Birdie", "Par", "Bogey", "Double Bogey", "Go Home!"];

function golfScore(par, strokes) {

  // Only change code below this line

  if(strokes === 1){

    return names[0];

  }

  else if (strokes <= par - 2){

    return names[1];

  }

  else if (strokes === par - 1){

    return names[2];

  }

  else if (strokes === par){

    return names[3];

  }

  else if (strokes === par + 1){

    return names[4];

  }

  else if (strokes === par + 2){

    return names[5];

  }

  else if (strokes >= par + 3){

    return "Go Home!"

  }

  return "Change Me";

  // Only change code above this line

}

golfScore(5, 4);

## Basic JavaScript: Selecting from Many Options with Switch StatementsPassed

If you have many options to choose from, use a switch statement. A switch statement tests a value and can have many case statements which define various possible values. Statements are executed from the first matched case value until a break is encountered.

Here is an example of a switch statement:

switch(lowercaseLetter) {

case "a":

console.log("A");

break;

case "b":

console.log("B");

break;

}

case values are tested with strict equality (===). The break tells JavaScript to stop executing statements. If the break is omitted, the next statement will be executed.

Write a switch statement which tests val and sets answer for the following conditions:  
1 - "alpha"  
2 - "beta"  
3 - "gamma"  
4 - "delta"

function caseInSwitch(val) {

  var answer = "";

  // Only change code below this line

  switch(val){

    case 1:

      answer = "alpha";

      break;

    case 2:

      answer = "beta";

      break;

    case 3:

      answer = "gamma";

      break;

    case 4:

      answer = "delta";

  }

  // Only change code above this line

  return answer;

}

caseInSwitch(1);

## Basic JavaScript: Adding a Default Option in Switch StatementsPassed

In a switch statement you may not be able to specify all possible values as case statements. Instead, you can add the default statement which will be executed if no matching case statements are found. Think of it like the final else statement in an if/else chain.

A default statement should be the last case.

switch (num) {

case value1:

statement1;

break;

case value2:

statement2;

break;

...

default:

defaultStatement;

break;

}

Write a switch statement to set answer for the following conditions:  
"a" - "apple"  
"b" - "bird"  
"c" - "cat"  
default - "stuff"

function switchOfStuff(val) {

  var answer = "";

  // Only change code below this line

 switch(val) {

    case "a":

      answer = "apple";

      break;

    case "b":

      answer = "bird";

      break;

    case "c":

      answer = "cat";

      break;

    default:

      answer = "stuff";

      break;

 }

  // Only change code above this line

  return answer;

}

switchOfStuff(1);

## Basic JavaScript: Multiple Identical Options in Switch StatementsPassed

If the break statement is omitted from a switch statement's case, the following case statement(s) are executed until a break is encountered. If you have multiple inputs with the same output, you can represent them in a switch statement like this:

var result = "";

switch(val) {

case 1:

case 2:

case 3:

result = "1, 2, or 3";

break;

case 4:

result = "4 alone";

}

Cases for 1, 2, and 3 will all produce the same result.

Write a switch statement to set answer for the following ranges:  
1-3 - "Low"  
4-6 - "Mid"  
7-9 - "High"

**Note**  
You will need to have a case statement for each number in the range.

function sequentialSizes(val) {

  var answer = "";

  // Only change code below this line

  switch(val){

    case 1:

    case 2:

    case 3:

      answer = "Low";

      break;

    case 4:

    case 5:

    case 6:

      answer = "Mid";

      break;

    case 7:

    case 8:

    case 9:

      answer = "High";

      break;

  }

  // Only change code above this line

  return answer;

}

sequentialSizes(1);

## Basic JavaScript: Replacing If Else Chains with SwitchPassed

If you have many options to choose from, a switch statement can be easier to write than many chained if/else if statements. The following:

if (val === 1) {

answer = "a";

} else if (val === 2) {

answer = "b";

} else {

answer = "c";

}

can be replaced with:

switch(val) {

case 1:

answer = "a";

break;

case 2:

answer = "b";

break;

default:

answer = "c";

}

Change the chained if/else if statements into a switch statement.

function chainToSwitch(val) {

  var answer = "";

  // Only change code below this line

  switch(val){

    case "bob":

      answer = "Marley";

      break;

    case 42:

      answer = "The Answer";

      break;

    case 1:

      answer = "There is no #1";

      break;

    case 99:

      answer = "Missed me by this much!";

      break;

    case 7:

      answer = "Ate Nine";

      break;

  }

  // Only change code above this line

  return answer;

}

chainToSwitch(7);

## Basic JavaScript: Returning Boolean Values from FunctionsPassed

You may recall from [Comparison with the Equality Operator](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/comparison-with-the-equality-operator) that all comparison operators return a boolean true or false value.

Sometimes people use an if/else statement to do a comparison, like this:

function isEqual(a,b) {

if (a === b) {

return true;

} else {

return false;

}

}

But there's a better way to do this. Since === returns true or false, we can return the result of the comparison:

function isEqual(a,b) {

return a === b;

}

Fix the function isLess to remove the if/else statements.

function isLess(a, b) {

  // Only change code below this line

  return a < b;

  // Only change code above this line

}

isLess(10, 15);

## Basic JavaScript: Return Early Pattern for FunctionsPassed

When a return statement is reached, the execution of the current function stops and control returns to the calling location.

**Example**

function myFun() {

console.log("Hello");

return "World";

console.log("byebye")

}

myFun();

The above outputs "Hello" to the console, returns "World", but "byebye" is never output, because the function exits at the return statement.

Modify the function abTest so that if a or b are less than 0 the function will immediately exit with a value of undefined.

**Hint**  
Remember that [undefined is a keyword](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/understanding-uninitialized-variables), not a string.

// Setup

function abTest(a, b) {

  // Only change code below this line

  if (a < 0 || b < 0 ) {

    return;

  }

  // Only change code above this line

  return Math.round(Math.pow(Math.sqrt(a) + Math.sqrt(b), 2));

}

abTest(2,2);

## Basic JavaScript: Counting CardsPassed

In the casino game Blackjack, a player can gain an advantage over the house by keeping track of the relative number of high and low cards remaining in the deck. This is called [Card Counting](https://en.wikipedia.org/wiki/Card_counting).

Having more high cards remaining in the deck favors the player. Each card is assigned a value according to the table below. When the count is positive, the player should bet high. When the count is zero or negative, the player should bet low.

| **Count Change** | **Cards** |
| --- | --- |
| +1 | 2, 3, 4, 5, 6 |
| 0 | 7, 8, 9 |
| -1 | 10, 'J', 'Q', 'K', 'A' |

You will write a card counting function. It will receive a card parameter, which can be a number or a string, and increment or decrement the global count variable according to the card's value (see table). The function will then return a string with the current count and the string Bet if the count is positive, or Hold if the count is zero or negative. The current count and the player's decision (Bet or Hold) should be separated by a single space.

**Example Output**  
-3 Hold  
5 Bet

**Hint**  
Do NOT reset count to 0 when value is 7, 8, or 9.  
Do NOT return an array.  
Do NOT include quotes (single or double) in the output.

var count = 0;

function cc(card) {

  // Only change code below this line

  switch (card){

    case 2:

    case 3:

    case 4:

    case 5:

    case 6:

      count+=1;

      break;

    case 7:

    case 8:

    case 9:

      count = count;

      break;

    case 10:

    case 'J':

    case 'Q':

    case 'K':

    case 'A':

      count-=1;

      break;

  }

  if (count > 0){

    return count+" Bet";

  }

  if (count === 0 || count < 0){

    return count+ " Hold";

  }

}

cc(2); cc(3); cc(7); cc('K'); cc('A');

## Basic JavaScript: Build JavaScript ObjectsPassed

You may have heard the term object before.

Objects are similar to arrays, except that instead of using indexes to access and modify their data, you access the data in objects through what are called properties.

Objects are useful for storing data in a structured way, and can represent real world objects, like a cat.

Here's a sample cat object:

var cat = {

"name": "Whiskers",

"legs": 4,

"tails": 1,

"enemies": ["Water", "Dogs"]

};

In this example, all the properties are stored as strings, such as - "name", "legs", and "tails". However, you can also use numbers as properties. You can even omit the quotes for single-word string properties, as follows:

var anotherObject = {

make: "Ford",

5: "five",

"model": "focus"

};

However, if your object has any non-string properties, JavaScript will automatically typecast them as strings.

Make an object that represents a dog called myDog which contains the properties "name" (a string), "legs", "tails" and "friends".

You can set these object properties to whatever values you want, as long as "name" is a string, "legs" and "tails" are numbers, and "friends" is an array.

var myDog = {

// Only change code below this line

"name": "Chocho",

"legs": 4,

"tails": 1,

"friends": ["Lemurs", "Forest"]

// Only change code above this line

};

## Basic JavaScript: Accessing Object Properties with Dot NotationPassed

There are two ways to access the properties of an object: dot notation (.) and bracket notation ([]), similar to an array.

Dot notation is what you use when you know the name of the property you're trying to access ahead of time.

Here is a sample of using dot notation (.) to read an object's property:

var myObj = {

prop1: "val1",

prop2: "val2"

};

var prop1val = myObj.prop1; // val1

var prop2val = myObj.prop2; // val2

Read in the property values of testObj using dot notation. Set the variable hatValue equal to the object's property hat and set the variable shirtValue equal to the object's property shirt.

// Setup

var testObj = {

  "hat": "ballcap",

  "shirt": "jersey",

  "shoes": "cleats"

};

// Only change code below this line

var hatValue = testObj.hat;

var shirtValue = testObj.shirt;

## Basic JavaScript: Accessing Object Properties with Bracket NotationPassed

The second way to access the properties of an object is bracket notation ([]). If the property of the object you are trying to access has a space in its name, you will need to use bracket notation.

However, you can still use bracket notation on object properties without spaces.

Here is a sample of using bracket notation to read an object's property:

var myObj = {

"Space Name": "Kirk",

"More Space": "Spock",

"NoSpace": "USS Enterprise"

};

myObj["Space Name"]; // Kirk

myObj['More Space']; // Spock

myObj["NoSpace"]; // USS Enterprise

Note that property names with spaces in them must be in quotes (single or double).

Read the values of the properties "an entree" and "the drink" of testObj using bracket notation and assign them to entreeValue and drinkValue respectively.

// Setup

var testObj = {

  "an entree": "hamburger",

  "my side": "veggies",

  "the drink": "water"

};

// Only change code below this line

var entreeValue = testObj["an entree"];

var drinkValue = testObj["the drink"];

## Basic JavaScript: Accessing Object Properties with VariablesPassed

Another use of bracket notation on objects is to access a property which is stored as the value of a variable. This can be very useful for iterating through an object's properties or when accessing a lookup table.

Here is an example of using a variable to access a property:

var dogs = {

Fido: "Mutt", Hunter: "Doberman", Snoopie: "Beagle"

};

var myDog = "Hunter";

var myBreed = dogs[myDog];

console.log(myBreed); // "Doberman"

Another way you can use this concept is when the property's name is collected dynamically during the program execution, as follows:

var someObj = {

propName: "John"

};

function propPrefix(str) {

var s = "prop";

return s + str;

}

var someProp = propPrefix("Name"); // someProp now holds the value 'propName'

console.log(someObj[someProp]); // "John"

Note that we do not use quotes around the variable name when using it to access the property because we are using the value of the variable, not the name.

Set the playerNumber variable to 16. Then, use the variable to look up the player's name and assign it to player.

// Setup

var testObj = {

  12: "Namath",

  16: "Montana",

  19: "Unitas"

};

// Only change code below this line

var playerNumber = 16;

var player = testObj[playerNumber];

## Basic JavaScript: Updating Object PropertiesPassed

After you've created a JavaScript object, you can update its properties at any time just like you would update any other variable. You can use either dot or bracket notation to update.

For example, let's look at ourDog:

var ourDog = {

"name": "Camper",

"legs": 4,

"tails": 1,

"friends": ["everything!"]

};

Since he's a particularly happy dog, let's change his name to "Happy Camper". Here's how we update his object's name property: ourDog.name = "Happy Camper"; or ourDog["name"] = "Happy Camper"; Now when we evaluate ourDog.name, instead of getting "Camper", we'll get his new name, "Happy Camper".

Update the myDog object's name property. Let's change her name from "Coder" to "Happy Coder". You can use either dot or bracket notation.

// Setup

var myDog = {

  "name": "Coder",

  "legs": 4,

  "tails": 1,

  "friends": ["freeCodeCamp Campers"]

};

// Only change code below this line

myDog.name = "Happy Coder";

## Basic JavaScript: Add New Properties to a JavaScript ObjectPassed

You can add new properties to existing JavaScript objects the same way you would modify them.

Here's how we would add a "bark" property to ourDog:

ourDog.bark = "bow-wow";

or

ourDog["bark"] = "bow-wow";

Now when we evaluate ourDog.bark, we'll get his bark, "bow-wow".

Example:

var ourDog = {

"name": "Camper",

"legs": 4,

"tails": 1,

"friends": ["everything!"]

};

ourDog.bark = "bow-wow";

Add a "bark" property to myDog and set it to a dog sound, such as "woof". You may use either dot or bracket notation.

// Setup

var myDog = {

  "name": "Happy Coder",

  "legs": 4,

  "tails": 1,

  "friends": ["freeCodeCamp Campers"]

};

// Only change code below this line

myDog.bark = "wowowowowowowooooof";

## Basic JavaScript: Delete Properties from a JavaScript ObjectPassed

We can also delete properties from objects like this:

delete ourDog.bark;

Example:

var ourDog = {

"name": "Camper",

"legs": 4,

"tails": 1,

"friends": ["everything!"],

"bark": "bow-wow"

};

delete ourDog.bark;

After the last line shown above, ourDog looks like:

{

"name": "Camper",

"legs": 4,

"tails": 1,

"friends": ["everything!"]

}

Delete the "tails" property from myDog. You may use either dot or bracket notation.

// Setup

var myDog = {

  "name": "Happy Coder",

  "legs": 4,

  "tails": 1,

  "friends": ["freeCodeCamp Campers"],

  "bark": "woof"

};

// Only change code below this line

delete myDog.tails;

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**TRICKY JAVASCRIPT PROBLEMS**

**Basic JavaScript: Record CollectionPassed**

You are given a JSON object representing a part of your musical album collection. Each album has several properties and a unique id number as its key. Not all albums have complete information.

Write a function which takes an album's id (like 2548), a property prop (like "artist" or "tracks"), and a value (like "Addicted to Love") to modify the data in this collection.

If prop isn't "tracks" and value isn't empty (""), update or set the value for that record album's property.

Your function must always return the entire collection object.

There are several rules for handling incomplete data:

If prop is "tracks" but the album doesn't have a "tracks" property, create an empty array before adding the new value to the album's corresponding property.

If prop is "tracks" and value isn't empty (""), push the value onto the end of the album's existing tracks array.

If value is empty (""), delete the given prop property from the album.

**Hints**  
Use bracket notation when [accessing object properties with variables](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/accessing-object-properties-with-variables).

Push is an array method you can read about on [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push).

You may refer back to [Manipulating Complex Objects](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/manipulating-complex-objects) Introducing JavaScript Object Notation (JSON) for a refresher.

// Setup

var collection = {

  2548: {

    album: "Slippery When Wet",

    artist: "Bon Jovi",

    tracks: [

      "Let It Rock",

      "You Give Love a Bad Name"

    ]

  },

  2468: {

    album: "1999",

    artist: "Prince",

    tracks: [

      "1999",

      "Little Red Corvette"

    ]

  },

  1245: {

    artist: "Robert Palmer",

    tracks: [ ]

  },

  5439: {

    album: "ABBA Gold"

  }

};

// Only change code below this line

function updateRecords(id, prop, value) {

  if (value === "") {

    delete collection[id][prop];

  }else if (prop === "tracks") {

    collection[id][prop] = collection[id][prop] || [];

    collection[id][prop].push(value);

  }else {

    collection[id][prop] = value;

  }

  return collection;

}

updateRecords(5439, "artist", "ABBA");

## Profile Lookup

We have an array of objects representing different people in our contacts lists.

A lookUpProfile function that takes name and a property (prop) as arguments has been pre-written for you.

The function should check if name is an actual contact's firstName and the given property (prop) is a property of that contact.

If both are true, then return the "value" of that property.

If name does not correspond to any contacts then return "No such contact".

If prop does not correspond to any valid properties of a contact found to match name then return "No such property".

// Setup

var contacts = [

    {

        "firstName": "Akira",

        "lastName": "Laine",

        "number": "0543236543",

        "likes": ["Pizza", "Coding", "Brownie Points"]

    },

    {

        "firstName": "Harry",

        "lastName": "Potter",

        "number": "0994372684",

        "likes": ["Hogwarts", "Magic", "Hagrid"]

    },

    {

        "firstName": "Sherlock",

        "lastName": "Holmes",

        "number": "0487345643",

        "likes": ["Intriguing Cases", "Violin"]

    },

    {

        "firstName": "Kristian",

        "lastName": "Vos",

        "number": "unknown",

        "likes": ["JavaScript", "Gaming", "Foxes"]

    }

];

function lookUpProfile(name, prop){

  for (var i = 0; i < contacts.length; i++) {

    if(contacts[i].firstName === name) {

      return contacts[i][prop] || "No such property";

    }

  }

  return "No such contact";

}

lookUpProfile("Akira", "likes");

## Use Recursion to Create a CountdownPassed

In a [previous challenge](https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/basic-javascript/replace-loops-using-recursion), you learned how to use recursion to replace a for loop. Now, let's look at a more complex function that returns an array of consecutive integers starting with 1 through the number passed to the function.

As mentioned in the previous challenge, there will be a base case. The base case tells the recursive function when it no longer needs to call itself. It is a simple case where the return value is already known. There will also be a recursive call which executes the original function with different arguments. If the function is written correctly, eventually the base case will be reached.

For example, say you want to write a recursive function that returns an array containing the numbers 1 through n. This function will need to accept an argument, n, representing the final number. Then it will need to call itself with progressively smaller values of n until it reaches 1. You could write the function as follows:

function countup(n) {

if (n < 1) {

return [];

} else {

const countArray = countup(n - 1);

countArray.push(n);

return countArray;

}

}

console.log(countup(5)); // [ 1, 2, 3, 4, 5 ]

At first, this seems counterintuitive since the value of n decreases, but the values in the final array are increasing. This happens because the push happens last, after the recursive call has returned. At the point where n is pushed into the array, countup(n - 1) has already been evaluated and returned [1, 2, ..., n - 1].

We have defined a function called countdown with one parameter (n). The function should use recursion to return an array containing the integers n through 1 based on the n parameter. If the function is called with a number less than 1, the function should return an empty array. For example, calling this function with n = 5 should return the array [5, 4, 3, 2, 1]. Your function must use recursion by calling itself and must not use loops of any kind.

// Only change code below this line

function countdown(n){

  if (n < 1) {

    return [];

  } else {

    const countArray = countdown(n - 1);

    countArray.unshift(n);

    return countArray;

  }

}

// Only change code above this line