1. Comentarios:

// This is an in-line comment.

/\* This is a

multi-line comment \*/

2. Var myName

3. var a;

a = 7;

**4.** **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.

*Solution:*

// Setup

var a;

a = 7;

var b;

// Only change code below this line

var b = a;

**5. Initializing Variables with the Assignment Operator**Passed

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.

*Solution:*

var a = 9;

**6. Understanding Uninitialized Variables**Passed

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 = 5;

var b = 10;

var c = "I am a";

// Only change code above this line

a = a + 1;

b = b + 5;

c = c + " String!";

8. **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;

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;

**9. 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:**

myVar = 5 + 10;

myVar now has the value 15.

Change the 0 so that sum will equal 20.

var sum = 10 + 10;

**10. Subtract One Number from Another with JavaScript**

We can also subtract one number from another.

JavaScript uses the - symbol for subtraction.

**Example**

myVar = 12 - 6;

myVar would have the value 6.

Change the 0 so the difference is 12.

var difference = 45 - 33;

**11. Multiply Two Numbers with JavaScript**

We can also multiply one number by another.

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

**Example**

myVar = 13 \* 13;

myVar would have the value 169.

Change the 0 so that product will equal 80.

var product = 8 \* 10;

**12. Divide One Number by Another with JavaScript**

We can also divide one number by another.

JavaScript uses the / symbol for division.

**Example**

myVar = 16 / 2;

myVar now has the value 8.

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

var quotient = 66 / 33;

**13. 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.

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

var myVar = 87;

// Only change code below this line

myVar ++;

**14. 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.

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

var myVar = 11;

// Only change code below this line

myVar --;

**15. Create Decimal Numbers with JavaScript**Passed

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_arithmetic#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 = 4.2

**16. Multiply Two Decimals with JavaScript**

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;

**16. Divide One Decimal by Another with JavaScript**Passed

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

**17. Finding a Remainder in JavaScript**Passed

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;

**18. 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.

var myVar = 1;

myVar += 5;

console.log(myVar);

6 would be displayed in the console.

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;

**19. 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;

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;

**20.** **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;

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;

**21. 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;

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;

**22. Declare String Variables**

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;

var myLastName;

myFirstName = "Jaime";

myLastName = "Dargallo";

**23. Escaping Literal Quotes in Strings**

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

**24. 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!"';

Here badStr will throw 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>';

**25. Escape Sequences in Strings**

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.

**Note:** The indentation for SecondLine is achieved with the tab escape character, not spaces.

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

**26. Concatenating Strings with Plus Operator**

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.";

The string I come first. I come second. would be displayed in the console.

Build myStr from the strings This is the start. and This is the end. using the + operator. Be sure to include a space between the two strings.

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

**27. Concatenating Strings with the Plus Equals Operator**

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 now has a value of the string 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 example and be sure to include a space between the two strings. 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.";

**28. Constructing Strings with Variables**

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 would have a value of the string 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 = "Jaime";

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

**29. Appending Variables to Strings**

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 would have the value 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 = "fantastic!";

var myStr = "Learning to code is ";

myStr += someAdjective;

**30. Find the Length of a String**

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

console.log("Alan Peter".length);

The value 10 would be displayed in the console.

For example, if we created a variable var firstName = "Ada", we could find out how long the string Ada 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;

**31. Use Bracket Notation to Find the First Character in a String**

*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 would have a value of the string 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

**32. Understand String Immutability**

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";

// Setup

var myStr = "Jello World";

// Only change code below this line

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

// Only change code above this line

**33. Use Bracket Notation to Find the Nth Character in a String**

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 would have a value of the string 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

**34. Use Bracket Notation to Find the Last Character in a String**

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

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

Example:

var firstName = "Ada";

var lastLetter = firstName[firstName.length - 1];

lastLetter would have a value of the string a.

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

**35. Use Bracket Notation to Find the Nth-to-Last Character in a String**

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 = "Augusta" string by using firstName[firstName.length - 3]

Example:

var firstName = "Augusta";

var thirdToLastLetter = firstName[firstName.length - 3];

thirdToLastLetter would have a value of the string s.

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

**36. Word Blanks**

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 = "Luckie was a " + myNoun + " very " + myAdjective + " and " +  myVerb + " very " + myAdverb + "."; // Change this line

// Only change code above this line

**37. Store Multiple Values in one Variable using JavaScript Arrays**

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).

// Only change code below this line

var myArray = ["Pigs", 3, "and a Wolf"];

**38. Nest one Array within Another Array**

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 = [["Oliver", 10],["Benji", 1]];

**39. Access Array Data with Indexes**

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];

var data = array[1];

array[0] is now 50, and data has the value 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.

var myArray = [50,60,70];

myArray[2];

var myData = myArray[0];

**40.Modify Array Data With Indexes**

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

**Example**

var ourArray = [50,40,30];

ourArray[0] = 15;

ourArray now has the value [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;

**41. Access Multi-Dimensional Arrays With Indexes**

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];

arr[3][0];

arr[3][0][1];

arr[3] is [[10, 11, 12], 13, 14], arr[3][0] is [10, 11, 12], and arr[3][0][1] is 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.

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

var myData = myArray[2][1];

**42. Manipulate Arrays With push()**

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);

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

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

arr1 now has the value [1, 2, 3, 4] and arr2 has the value ["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]);

**43. Manipulate Arrays With pop()**

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);

console.log(threeArr);

The first console.log will display the value 6, and the second will display the value [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();

**44. Manipulate Arrays With shift()**

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 would have a value of the string Stimpson, and ourArray would have ["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();

**45. Manipulate Arrays With unshift()**

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.unshift("Happy");

After the shift, ourArray would have the value ["J", "cat"]. After the unshift, ourArray would have the value ["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]);

**46. Shopping List**

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.