**Basic Javascript**

Understanding Uninitialized Variables

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

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.

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".

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>';

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:

To allow you to use characters you may not otherwise be able to type out, such as a carriage return.

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';

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.

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"]].

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].

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"]].

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"].

Adding a Default Option in Switch Statements

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;

}

Multiple Identical Options in Switch Statements

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.

switch(val){

    case 1:

      answer = "Low";

      break;

    case 2:

      answer = "Low";

      break;

    case 3:

      answer = "Low";

      break;

    case 4:

      answer = "Mid";

      break;

    case 5:

      answer = "Mid";

      break;

    case 6:

      answer = "Mid";

      break;

    case 7:

      answer = "High";

      break;

    case 8:

      answer = "High";

      break;

    case 9:

      answer = "High";

      break;

  }