1. Comment your JavaScript Code

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

There are two ways to write comments in JavaScript:

Using // will tell JavaScript to ignore the remainder of the text on the current line:

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

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.

2. Declare JavaScript Variables

In computer science, *data* is anything that is meaningful to the computer. JavaScript provides seven different *data types* which areundefined, null, boolean, string, symbol, 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 seven 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.

Instructions
Use the var keyword to create a variable called myName.

3. **Storing Values with the Assignment Operator**

   In JavaScript, you can store a value in a variable with the *assignment* operator.
   myVariable = 5;
   Assigns the Number value 5 to myVariable.
   Assignment always goes from right to left. Everything to the right of the = operator is resolved before the value is assigned to the variable to the left of the operator.

```
myVar = 5;
myNum = myVar;
```

   Assigns 5 to myVar and then resolves myVar to 5 again and assigns it to myNum.

   Instructions
   Assign the value 7 to variable a.
   Assign the contents of a to variable b.

4. **Initializing Variables with the Assignment Operator**

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

   `var myVar = 0;`

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

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

5. **Understanding Uninitialized Variables**

   When JavaScript variables are declared, they have an initial value of undefined. If you do a mathematical operation on an undefinedvariable 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".

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

6. **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 thisVariableNameIsTooLong;
```

7. **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 addition operation when placed between two numbers.

   Example

   ```
   myVar = 5 + 10; // assigned 15
   ```

8. **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; // assigned 6
   ```

9. **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; // assigned 169
   ```

10. **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; // assigned 8
    ```

11. **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.

12. **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.

13. **Create Decimal Numbers with JavaScript**

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

    Note
    Not all real numbers can accurately be represented in *floating point*. This can lead to rounding errors. **Details Here**.

14. **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.

15. **Divide one Decimal by Another with JavaScript**

    Now let's divide one decimal by another.

    Instructions

    var quotient = 0.0 / 2.0;
    Change the `0.0` so that `quotient` will equal to `2.2`.

16. **Finding a Remainder in JavaScript**

    The *remainder* operator `%` gives the remainder of the division of two numbers.
    Example

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

    Usage
    In mathematics, a number can be checked 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.

17. **Convert Celsius to Fahrenheit**

    To test your learning, you will create a solution "from scratch". Place your code between the indicated lines and it will be tested against multiple test cases.

    The algorithm to convert from Celsius to Fahrenheit is the temperature in Celsius times 9/5, plus 32.

    You are given a variable celsius representing a temperature in Celsius. Use the variable fahrenheit already defined and apply the algorithm to assign it the corresponding temperature in Fahrenheit.

    Note
    Don't worry too much about the function and return statements as they will be covered in future challenges. For now, only use operators that you have already learned.

```
function convertToF(celsius) {

  var fahrenheit;

  // Only change code below this line




  // Only change code above this line

  return fahrenheit;

}

// Change the inputs below to test your code

convertToF(30);
```

18. **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.

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

19. **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".
```

Instructions
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".
```

20. **Quoting Strings with Single Quotes**

String values in JavaScript may be written with single or double quotes, so long as you start and end with the same type of quote. Unlike some languages, single and double quotes are functionally identical in JavaScript.

```
"This string has \"double quotes\" in it"
```

The value in using one or the other has to do with the need to *escape* quotes of the same type. Unless they are escaped, you cannot have more than one pair of whichever quote type begins a string.

If you have a string with many double quotes, this can be difficult to read and write. Instead, use single quotes:

```
'This string has "double quotes" in it. And "probably" lots of them.'
```

Instructions
Change the provided string from double to single quotes and remove the escaping.

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

21. **Escape Sequences in Strings**

Quotes are not the only characters that can be *escaped* inside a string. Here is a table of common escape sequences:

Code	Output
\'	single quote
\"	double quote
\\	backslash
\n	newline
\r	carriage return
\t	tab
\b	backspace
\f	form feed

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

Instructions
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.

“FirstLine newline backslash SecondLine backslash carriage-return ThirdLine”

22. 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.

Instructions

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

23. 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.

Instructions

Build myStr over several lines by concatenating these two strings:
"This is the first sentence. " and "This is the second sentence." using the += operator.

24. **Constructing Strings with Variables**

Sometimes you will need to build a string, **Mad Libs** style. By using the concatenation operator (+), you can insert one or more variables into a string you're building.

Instructions

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

25. **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.

Instructions

Set someAdjective and append it to myStr using the += operator.

```
// Example

var anAdjective = "awesome!";

var ourStr = "CodeQuotient is ";
```

```
ourStr += anAdjective;

// Only change code below this line

var someAdjective;

var myStr = "Learning to code is ";
```

26. **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.

`"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.

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

```
// Example

var firstNameLength = 0;

var firstName = "Ada";

firstNameLength = firstName.length;

// Setup

var lastNameLength = 0;

var lastName = "Lovelace";

// Only change code below this line.

lastNameLength = lastName;
```

27. 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]`.

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

```
// Example

var firstLetterOfFirstName = "";

var firstName = "Ada";

firstLetterOfFirstName = firstName[0];

// Setup

var firstLetterOfLastName = "";

var lastName = "Lovelace";

// Only change code below this line

firstLetterOfLastName = lastName;
```

28. 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 myStrcannot 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";
```

29. 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.

Instructions

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

```
// Example

var firstName = "Ada";

var secondLetterOfFirstName = firstName[1];

// Setup

var lastName = "Lovelace";

// Only change code below this line.

var thirdLetterOfLastName = lastName;
```

30. 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 = "Charles"`, you can get the value of the last letter of the string by using`firstName[firstName.length - 1]`.

Instructions

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

```
// Example

var firstName = "Ada";

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

// Setup

var lastName = "Lovelace";

// Only change code below this line.

var lastLetterOfLastName = lastName;
```

31. Use Bracket Notation to Find the NthtoLast 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 = "Charles"` string by using`firstName[firstName.length - 3]`

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

```
// Example

var firstName = "Ada";

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

// Setup

var lastName = "Lovelace";

// Only change code below this line

var secondToLastLetterOfLastName = lastName;
```

32. Word Blanks

We will now use our knowledge of strings to build a "**Mad Libs**" style word game we're calling "Word Blanks". You will create an (optionally humorous) "Fill in the Blanks" style sentence.

You will need to use string operators to build a new string, `result`, using the provided variables: `myNoun`, `myAdjective`, `myVerb`, and`myAdverb`.

You will also need to use additional strings, which will not change, and must be in between all of the provided words. The output should be a complete sentence.

We have provided a framework for testing your results with different words. The tests will run your function with several different inputs to make sure all of the provided words appear in the output, as well as your extra strings.

```
function wordBlanks(myNoun, myAdjective, myVerb, myAdverb) {

  var result = "";

  // Your code below this line
```

```
  // Your code above this line

  return result;

}

// Change the words here to test your function

wordBlanks("dog", "big", "ran", "quickly");
```

33. 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"]`.

Instructions
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.

34. Nest one Array within Another Array

You can also nest arrays within other arrays, like this: `[["Bulls", 23], ["White Sox", 45]]`. This is also called a *Multi-dimensional Array.*

Instructions
Create a nested array called `myArray`.

35. 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 is element `0`.

Example

```
var array = [1,2,3];
array[0]; // equals 1
var data = array[1];  // equals 2
```

Instructions

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

36. Modify Array Data With Indexes

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

Example

```
var ourArray = [3,2,1];
ourArray[0] = 1; // equals [1,2,1]
```

Instructions

Modify the data stored at index 0 of myArray to a value of 3.

37. Access MultiDimensional 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 bracket 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
```

Instructions

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

38. 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.

```
var arr = [1,2,3];
arr.push(4);
// arr is now [1,2,3,4]
```

Instructions

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

39. 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.

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

For example, for the code var oneDown = [1, 4, 6].pop();
the variable oneDown now holds the value 6 and the array becomes [1, 4].

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

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

var removedFromMyArray;
```

40. 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.

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

```
// Example

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

removedFromOurArray = ourArray.shift();

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

// Setup

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

// Only change code below this line.

var removedFromMyArray;
```

41. 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.

Instructions

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

```
// Example

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

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

ourArray.unshift("Happy");

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

// Setup

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

myArray.shift();



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

42. 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.

43. Global Scope and Functions

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.

Instructions
Declare a global variable myGlobal outside of any function. Initialize it to have a value of 10

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

```
// Declare your variable here

function fun1() {

  // Assign 5 to oopsGlobal Here

}

// 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);

}
```

44. Local Scope and Functions

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(); // "foo"
console.log(loc); // "undefined"
```

loc is not defined outside of the function.

Instructions
Declare a local variable myVar inside myLocalScope. Run the tests and then follow the instructions commented out in the editor.

```
function myLocalScope() {

  'use strict';

  console.log(myVar);

}

myLocalScope();

// Run and check the console

// myVar is not defined outside of myLocalScope

console.log(myVar);

// Now remove the console log line to pass the test
```

45. Global vs Local Scope in Functions

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.

Instructions

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

```
// Setup

var outerWear = "T-Shirt";

function myOutfit() {

  // Only change code below this line

  // Only change code above this line

  return outerWear;

}

myOutfit();
```

46. Golf Code

In the game of **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"

Strokes	Return
Par	"Par"
par + 1	"Bogey"
par + 2	"Double Bogey"
>= par + 3	"Go Home!"

par and strokes will always be numeric and positive.

```
function golfScore(par, strokes) {
  // Only change code below this line
  return "Change Me";
  // Only change code above this line
}
// Change these values to test
golfScore(5, 4);
```

47. Stand in Line

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

```
function nextInLine(arr, item) {
  // Your code here
```

```
  return item;  // Change this line
}

// Test Setup
var testArr = [1,2,3,4,5];

// Display Code
console.log("Before: " + JSON.stringify(testArr));
console.log(nextInLine(testArr, 6)); // Modify this line to test
console.log("After: " + JSON.stringify(testArr));
```

48. Counting Cards

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**.

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 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"

```
var count = 0;

function cc(card) {

  // Only change code below this line

  return "Change Me";

  // Only change code above this line

}

// Add/remove calls to test your function.

// Note: Only the last will display

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