1 variables-intro-next-warmup.md

**Warm up 💪🏼**

1. You will use a new JavaScript operator for this: **typeof**. You can learn more about this operator [here](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof).

The first has been done for you below.

**const variableString = "I am a string";**

**console.log(typeof variableString); // should print "string"**

**const variableNumber = ???;**

**console.log(typeof variableNumber); // What goes here? // should print "number"**

**// Your code here**

**console.log(typeof variableBoolean); // should print "boolean"**

**console.log(typeof variableUndefined); // should print "undefined"**

1. Compute and store the *average* of two **number**s.
2. **const numberA = ???;**
3. **const numberB = ???;**
4. **const average = ???;**

**console.log(average); // should print the average of numberA and numberB**

2functions-intro.md

# Intro to functions

## Lecture Slides

## Comments

**//** <-- Two backslashes mark a comment. Comments are ignored by the JavaScript engine (the program which reads our code and does stuff.)

**// This line is a comment and not code.**

**// This line is also a comment.**

You can also surround multiple lines of comments with a backslash and asterisk

**/\* This is more than one line of comments**

**In fact it is two lines \*/**

## Exercises

Create a copy of the **\_lesson-template** folder called **functions-intro**.

Inside of **script.js** complete the following exercises:

### Basic Requirements

#### Printing to the console

1. Enter the following two lines into your **script.js** file. Open the **index.html** file in your browser and open the developer tools. What do you see in your developer console?
2. **5 + 6;**
3. **console.log(6 + 6);**

Why does it work this way?

Now that we are writing our JavaScript in a file rather than directly in the console, we will need to use **console.log** to \*let the computer know to print values to the console of our browser (or wherever we are running our file).

#### Functions

1. Use the **function** below to return the sum of two **number**s. Enter the following code in your script.js file:
2. **function add(numOne, numTwo) {**
3. **return numOne + numTwo;**
4. **}**
5. **// Tests**
6. **console.log(add(4, 3)); // should return 7**
7. **console.log(add(100, 42)); // => 142**

**Something Cool**: **function**s that you declare in your **script.js** are available in the console. Try entering the two tests directly in the developer console. You should continue to put your tests in the **script.js** file, but this can be useful when debugging.

1. For the above function, what happens if you give only one input? What happens if you give more than the specified number of inputs?
2. **console.log(add(100)); // What happens?**
3. **console.log(add(1, 4, 5)); // What happens?**
4. What do you think gets printed when you run the following code? Try to predict what happens and then see if you can figure out why. The extra sentence that we are printing out in the two **console.log** statements at the end will help you figure out what is happening where. Scroll to the explanation below to find the answer.
5. **function simpleFunctionA() {**
6. **return "Hello simple function A";**
7. **}**
8. **function simpleFunctionB() {**
9. **console.log("Hello simple function B");**
10. **}**
11. **console.log(simpleFunctionA(), "--> From running Simple Function A");**
12. **console.log(simpleFunctionB(), "--> From running Simple Function B");**
13. Declare a function named subtract that subtracts the second argument from the first argument. Remember to try the test cases to see if your function works.
14. **function subtract(num1, num2) {**
15. **// your code here**
16. **}**

Test cases:

**console.log(subtract(4, 3)); // => 1**

**console.log(subtract(100, 42)); // => 58**

Can you write some more test cases? Provide some different inputs and see what happens.

1. Declare a **function** named greeting that takes a name **string** as an argument and returns a hello!
2. **// Your code here**
3. **console.log(greeting("Alex")); // => "Hello, Alex!"**
4. **console.log(greeting("Beau")); // => "Hello, Beau!"**
5. Declare a **function** called average that takes two **number**s as inputs and returns the average of those **number**s. This time, write two tests for your **function** by yourself!
6. What is wrong with the following functions?
7. **function square(5) {**
8. **return 5 \* 5;**
9. **}**
10. **function square("x") {**
11. **return "x" \* "x";**
12. **}**

This one works, but what’s a better way to write it?

**function Square(monkey) {**

**return monkey \* monkey;**

**}**

1. Write a **function** called cube that returns the cube of x:
2. **function cube(x) {}**

#### Explanation

Why are we printing an **undefined** in the below code?

**function simpleFunctionA() {**

**return "Hello simple function A";**

**}**

**function simpleFunctionB() {**

**console.log("Hello simple function B");**

**}**

**console.log(simpleFunctionA(), "--> From running Simple Function A");**

**console.log(simpleFunctionB(), "--> From running Simple Function B");**

When functions don’t have an **output** (in other words, they do not have a **return** value), they return **undefined**.

With the code above: while the invocation of **simpleFunctionA** returns a string of **'Hello simple function A'**, when we run **simpleFunctionB**, that function simply prints to our console and then returns **undefined**.

Note that there is no **return** keyword–thus, it returns **undefined**.

However, our code does not end there. After **simpleFunctionB** finishes, we still need to **console.log**. Because there was no return statement, we print **undefined**. There was no value defined as the output of **simpleFunctionB**.

Do not confuse printing to the console for function outputs. **console.log** delivers a value to appear in our console, nothing more.

### Medium Requirements

1. If we run the following code, what will get printed? Why?
2. **function simpleHelloA() {**
3. **console.log("hello");**
4. **}**
5. **function simpleHelloB() {**
6. **return "hello";**
7. **}**
8. **const a = simpleHelloA();**
9. **const b = simpleHelloB();**
10. **console.log(a, b);**
11. Invoke the function below three times and assign each return value to a variable so that your test cases return what you expect. How is this working?
12. **function createGreeting(greetingPhrase, instructorName) {**
13. **return greetingPhrase + ", " + instructorName + "!";**
14. **}**
15. **console.log(morningGreeting); // should print 'Good morning, Mike!'**
16. **console.log(dayGreeting); // should print 'Hello, Beau!'**
17. **console.log(eveningGreeting); // should print 'Good evening, Alex!'**
18. Translate the geometric formulas found [in this pdf](http://www.gbcnv.edu/documents/ASC/docs/00000005.pdf) into functions.

### Advanced Requirements

For these exercises, you may need to use techniques that we haven’t covered in class.

1. Create a new file in the **functions-intro** folder called **advanced.js**. Figure out how to modify the **index.html** in **functions-intro** so that it will use both **script.js** and **advanced.js**. Add the following code in your new file. How is it showing up in the console? Look at the code for **index.html**. How are the **.js** files linking?
2. **console.log("hello advanced.js!");**
3. Below is some example code for **guessMyNumber** and **randomNumber**. Please read and test both functions in your console.
4. **function guessMyNumber(n) {**
5. **if (n > 5) {**
6. **return "Please try a number between 0 and 5.";**
7. **} else if (n === randomNumber(5)) {**
8. **return "You guessed my number!";**
9. **}**
10. **return "Nope! Wrong number!";**
11. **}**
12. **function randomNumber(n) {**
13. **return Math.floor(Math.random() \* (n + 1));**
14. **}**

The above code requires the guess to be between **0** and **5**.  
We can think of **5** as the upper bound of the guess.  
Please think over the following questions (preferably with a peer):

* + What if we wanted to change the upper bound to **6**?
  + How many changes would be required?

1. Declare a variable called **upperBound** to hold the upper bound, and then reference it instead of the number **5**. If you were asked to change the upper bound to some other number (e.g. **7**), you should only have to make one change.
2. Modify **guessMyNumber** so that if the guess is incorrect, **guessMyNumber** includes the correct guess in its output, e.g. **"Nope! The correct number was: x"** (where **x** would have been the correct number).

## Review Questions

1. What happens if you run a function with less than or more than the specified number of inputs?
2. What does the **return** keyword do? Can you use it outside of a function?

## Homework

* Complete the **Basic Requirements** for this lesson.
* Complete the **Additional Reading**.

### Additional Reading 📖

1. [JavaScript for Cats: Functions](http://jsforcats.com/#functions)

# 3 ntro to Comparisons

## Lecture Slides

## Vocabulary

* **boolean** - either **true** or **false**  
  Named after Mathematician [George Boole](https://en.wikipedia.org/wiki/George_Boole)

## Helpful Link

* [Operators - Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators)

## Exercises

Create a copy of **\_lesson-template** called **comparisons-intro**.

Inside of **script.js** complete the following exercises:

### Basic Requirements

#### Comparisons

1. Before running the code below, what do you think the expressions will resolve to? Try them in the console to see if you are correct!
2. **"true" == true;**
3. **// We never use `==`. Stick to using `===`.**
4. **"true" === true;**
5. **3 >= 3;**
6. **3 !== 4;**
7. Copy the code into your script.js file and fill in the ??? with the appropriate comparison operators or values to make the statements output the expected boolean value. Open **index.html** in Chrome and then open the Developer Console to check the results.
8. **console.log(1100 ??? 99) // should return true**
9. **console.log(1 ??? 21) // => false**
10. **console.log(62 !== ???) // => true**
11. **console.log("5" ??? 5) // => false**
12. **console.log("6" ??? "six") // => true**
13. Copy the code into your **script.js** file and change ONE of the ARITHMETIC operators in the expression below so that it evaluates to **true**. Make sure you understand the ORDER that the expressions evaluate in.
14. **console.log(2 + 3 \* 10 > 50);**
15. Add the **function** below to your **script.js** file and invoke it by replacing ??? with two DIFFERENT inputs so that the expression evaluates to **true**. Remember, **===** checks the value and type, but **==** only checks the value.
16. **function isEqual(valueOne, valueTwo) {**
17. **return valueOne == valueTwo;**
18. **}**
19. **console.log(isEqual(???,???));**
20. Add the **function** below to your **script.js** file and write a statement that returns **true** when valueOne is MORE than valueTwo.

Remember that you need to use **console.log** to print the output of the **function** to your console.

**function isGreaterThan(valueOne, valueTwo) {}**

Write some test cases! Invoke **isGreaterThan** with two inputs so that the output is **false**.

1. In Japan, you can drink alcohol if you are aged 20 or older. In your **script.js** file declare a **function** called isOfAge that takes a **number** as the input and returns a **boolean** that describes whether or not that person is old enough to drink.

### Medium Requirements

These exercises may require some research in order to complete.

1. Amend your isOfAge **function** to print a **string** to the console that describes whether or not that person is old enough to drink. It should still return a **boolean**.
2. Write a function called **isEven** that, given a number **n** as a parameter, returns **true** if that number is even, and **false** otherwise.
3. Write a function **validCredentials** that accepts two parameters: **username** and **password**, and returns **true** if both are long enough, and **false** otherwise. You can decide what constitutes “long enough”.

### Advanced Requirements

For these exercises, you may need to use techniques that we haven’t covered in class.

1. Adjust your **isEven** function so that it doesn’t use the **%** operator.
2. Write a function called **getGreaterValue** that returns the greater of two arguments.
3. **console.log(getGreaterValue(10, 15)); // should return 15**
4. **console.log(getGreaterValue(15, 10)); // should return 15**
5. Write a function called **getGreatestValue** that returns the greatest of any number of arguments.
6. **console.log(getGreatestValue(10, 15)); // should return 15**
7. **console.log(getGreatestValue(15, 10)); // should return 15**
8. **console.log(getGreatestValue(1, 2, 3, 4, 5)); // should return 5**
9. **console.log(getGreatestValue(1, 10, 3, 4, 5)); // should return 10**

### Nightmare Requirements 😈

These might be super hard.

1. Write a function called **flatten** that takes an array of arrays and returns a flattened version.
2. **console.log(flatten([[1, 2, 3], [4, 5, 6]])); // [1, 2, 3, 4, 5, 6]**
3. **console.log(flatten([1, 2, 3, [4, 5, 6]])); // [1, 2, 3, 4, 5, 6]**
4. **console.log(flatten([[1], [2], [3], [4, 5, 6]])); // [1, 2, 3, 4, 5, 6]**
5. Write a function called **flattenDeep** that takes an array of arrays (which may be nested) and returns a flattened version.
6. **console.log(flattenDeep([1, 2, 3, [4, 5, 6]])); // [1, 2, 3, 4, 5, 6]**
7. **console.log(flattenDeep([[1, 2, 3], [4, 5, 6]])); // [1, 2, 3, 4, 5, 6]**
8. **console.log(flattenDeep([[1], [2], [3], [4, 5, 6]])); // [1, 2, 3, 4, 5, 6]**
9. **console.log(flattenDeep([[1, [2, [3, [4, [5, [6, [7, [8, [9]]]]]]]]]])); // [1, 2, 3, 4, 5, 6, 7, 8, 9]**

## Homework

* Finish the Basic Requirements for this lesson.
* Read the **Additional Reading** materials.

### Additional Reading 📖

* [Eloquent JavaScript Chapter 2](http://eloquentjavascript.net/02_program_structure.html) - Read through “Conditional Execution”, but you do not need to go on to loops…yet. 🐶
* [JavaScript Operators, Conditionals, & Functions](https://www.sitepoint.com/javascript-operators-conditionals-functions/)

### Exploration: Truthy vs. Falsy Values

You have learned about strings, numbers and now **booleans**. There are only two types of boolean values: **true** and **false**. But JavaScript is weird and there can be some weirdness that happens when you use different operators or comparisons with different data types.

Try to see if you can guess what happens with the following:

**true + true;**

**false + false;**

**true + false;**

**false + true;**

**true + 5;**

**false + 5;**

**true \* "5";**

**false \* "5";**

**"true" + false;**

**"true" + true;**

**false / true;**

**true / false;**

**true / "1";**

**true / "a";**

**!1;**

**!0;**

**!5;**

Weird, right??

In JavaScript, values are naturally “truthy” or “falsy” and in some circumstances, **true** and **false** values get coerced into numbers **1** and **0**, respectively.

No need to try to memorize all the different possibilities. You should always avoid or be careful of using mathematical operators with non-number data types.

You can read more about this weirdness [here](https://www.sitepoint.com/javascript-truthy-falsy/) and [here](https://codeburst.io/javascript-truthy-values-dont-always-equal-true-8afaf071a4a6).