TECHSWITCH

JavaScript Pre-Reading

(Mac)

Table of Contents

1. Introduction to JavaScript

This pre-reading is designed to give you an introduction to JavaScript programming. You should be able to follow it even if you've done little or no programming before, but if you are completely unfamiliar with JavaScript, then we suggest trying out the free 'Learn JavaScript' course on Codecademy first. The goal is to equip you with enough JavaScript knowledge for the TechSwitch online tests and interviews, and to enable you to hit the ground running in the bootcamp.

If you do have some prior programming experience, read on - there are some bits you can skip over, but don't move too fast as there could be something you're not so familiar with that's worth revising! Make sure you complete all the exercises that we suggest, and if you find anything tricky then treat that as a prompt to revisit the reading material we suggest in more detail.

Learning goals

The objective of this pre-reading is to be able to understand and use the following JavaScript programming concepts:

- Variable and assignment
- Conditional logic – the `if` statement
- Looping logic – the `for` and `while` statements
- Arrays and objects
- Functions

Anything beyond this starting point is good! But the above represents the minimum you need to know in order to be able to effectively tackle the entrance interviews.

Before you start

In order to complete this course you will need a copy of **Node.js** - this will allow you to run the code you write on your own machine and see it working. Download and install the version marked "Current" from https://nodejs.org/en/.

You will then need a development environment. We recommend **VS Code**, which you can download from https://code.visualstudio.com/.

Reading material

The recommended route to learning the necessary JavaScript is to work through the relevant sections of Mozilla's JavaScript Guide, but don't just sit down and read the guide cover-to-cover. You'll probably get bored! Instead, work through the topics in this pre-reading and we will:

- Suggest appropriate sections to read
- Provide some exercises that you should use to practice what you learn

You will almost certainly find it helpful, between reading a section of the guide and doing the exercises, to copy some of the code samples from the guide and run them. Change the code slightly, and run them again, then see if the result is what you expected. Doing this every time you meet a new concept will help you to pick it up more quickly and understand it in more depth.

If you do already have prior experience then you're welcome to skim over any of the reading material that you're already confident in. But make sure you're proud of the code you're writing, and if you're not completely confident then re-read the material even if it's largely familiar.

2. Your First JavaScript Program

This topic sees you write your first JavaScript program.

Reading material

Read through the first section "Introduction" of The JavaScript Guide to gain some background knowledge about JavaScript in general. Stop before the section called "Getting started with JavaScript", as during this course you will write and run pieces of code differently to the method described there. Don't worry about meeting any of the prerequisites mentioned in that guide - you will be able to follow this pre-reading even without any prior knowledge.

Exercise

As we progress through this module, we're going to build a (very simple!) Calculator application. It will be a console (command line) application - entirely text based.

Here's how to get started:

- Firstly, if you are unfamiliar with using the terminal, please read **Appendix: Working with the Terminal Application** before continuing.
- Create a directory that you will keep your Calculator application in. Call it something sensible, like "Calculator".
- Open up a command prompt and navigate to your calculator directory (e.g. `cd /Users/(you)/Work/Training/Calculator`)
- Run `npm init` to create a new Node.js project. Use the default answer to every question it asks you (just press "Enter") - don't worry if you don't understand what all the questions mean!
    - If it doesn't do anything after the final question "Is this ok?", double-check there's a file in the folder called `package.json` . If it's there, everything went smoothly and you can exit the `npm init` process by typing in the console `Ctrl + C`
- In **VS Code**, open your new directory via `File > Open...`
- Create a file where you will write your code via `File > New file`. Call the file "index.js" - this is the conventional name to give to the entry point of a JavaScript program.

- Write the following code in "index.js"

```
console.log('Welcome to the calculator!');
```

- Run the program: in the command prompt from earlier, still in the directory for your application, run `node index.js`
- Check that the output is what you expect it to be. Have a go at changing the message that is displayed, and re-run the program to see if it worked.

3. Data Types and Variables

This topic introduces data types, variables and assignment. We'll make some small extensions to your Calculator project.

Reading material

Read through the sections "Grammar and types" of the JavaScript Guide. Then read the "Assignment", "Comparisons" and "Arithmetic operators" section of "Expressions and Operators". After reading these sections, you should have learned:

- How to declare a variable
- The difference between `var`, `let` and `const`
- How to assign a value to a variable
- The difference between numbers and strings in JavaScript

Exercise

Add the following functionality to your Calculator project:

- Prompt the user to enter one number
- Prompt the user to enter a second number
- Multiply the two numbers together and print out the result

You should be able to do most of this using information contained in the guide. However, you'll need a way to prompt the user for some input. **Node.js** doesn't provide a way to do this easily, but we can install a library called `readline-sync` that provides this functionality to our project.

- Open a terminal window and navigate to your project directory.
- Run `npm install --save readline-sync`
- At the top of index.js, add the following line of code:

  ```
  const readline =  require('readline-sync')
  ```

- This downloads the library to your project directory, and tells Node.js that you want to load this library in your application so that you can use its functionality. It also writes some metadata to files called `package.json` and

`package-lock.json` so that people who work on your application with you will get the same version of the library as you.

- Now you can get input from the user in the following fashion:

```
console.log('Please enter some input:');
const response = readline.prompt();
```

- Note that `readline.prompt()` returns the response as a string, so you'll need to make sure to convert the responses to numbers before multiplying them together!

Sample solution

We've put together a sample solution, that evolves as you move through the topics in this pre-reading. Here's how to use it:

- If you get stuck, it's there to help. But don't look until you've tried your best to solve the challenge yourself - you'll learn much more if you build your own code. If you do need to look at the solution, try to learn from it and then apply that knowledge in your own way, rather than just copying what we've provided.

- Once you've finished the exercise, then take a look at the sample solution. How does it compare to yours? Remember that it is only an example - it's not necessarily any more "correct" than yours. But it might approach some particular problem in a different way. Think about what's "better" and "worse" than yours. And indeed, what's just "different"?

Here's our solution to this particular exercise on GitHub: Link

4. Branching

This topic introduces branching and conditional logic, via the `if` and switch statements. We also understand a bit more about the syntax of JavaScript.

Reading material

Read the subsections called “if...else and switch” in the section "Control flow and error handling" of the JavaScript Guide. You can stop before getting to the section on exception handling - we'll cover that later on in the course.

These two statements will allow you to write code that behaves differently depending on certain conditions. You may also want to review the "Assignment" and "Comparison" sections of "Expressions and operators". After reading these sections, you should know:

- How to write code which uses the if and else statements
- How to write code using the switch statement
- The difference between `x = y` and `x == y`

Exercise

Let's enhance your Calculator a little further. We want to support more operations than just multiplication. We'll do this by prompting the user to enter an operator before they enter the numbers. So a typical program run might look like this:

```
Welcome to the calculator!
==========================
Please enter the operator: +
Please enter the first number: 10
Please enter the second number: 4
The answer is: 14
```

To keep things simple, we’ll just use four operators:

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division

You can do this exercise with either `if...else` statements or a `switch` statement. Why not try both, and see what the difference in implementation looks like?

Sample solution

One possible solution is here: Link

Remember not to look until you've pushed your own code, unless you're really stuck! And bear in mind that there are many possible solutions - this one isn't necessarily any better than yours.

5. For Loops and Arrays

This topic introduces `for` loops and arrays. We'll obviously add some more functionality to the calculator app too!

Reading material

Read the "Indexed collections" section of the JavaScript Guide. You can stop before reading about "Typed Arrays".

Then read the "Loops and iteration" section. You can skip over the `while` and the `do...while` loops for now, but we'll come back to them in the next lesson.

After reading these materials, you should know:

- How to create an array
- How to set the values in an array
- How to access elements from arrays
- How to write code that will run repeatedly inside a Exercise for loop.

Exercise

So far our calculator only performs operations on two numbers. Let's enhance it so it can do calculations on any number of numbers! For example:

```
3 + 4 + 5 = 12
1 * 2 * 3 = 6
12 / 2 / 2 = 3
```

Let's keep things simple by using the same operator each time. So a typical output might look like this (for the first sum above):

```
Welcome to the calculator!
==========================
Please enter the operator: +
How many numbers do you want to +? 3
Please enter number 1: 3
Please enter number 2: 4
Please enter number 3: 5
The answer is: 12
```

You may find you need two `for`-loops - one to read the numbers and put them into an array, and a second one to go through and add them all up.

See what you can come up with, and push the result. Once you're done, take a look at the example solution below - it's not necessarily any more "correct" than yours, but it might be different; what do you think the most important differences are?

Sample solution

Here's one possible solution: Link

Remember not to look until you've finished your own code, unless you're really stuck!

6. Functions and While Loops

This topic introduces the syntax for creating and calling functions, the basic building blocks of reusable (and understandable!) code. We'll also look at the `while` loop, and add some of our understanding to the calculator tool we've been building.

Reading material

Read the "Functions" section of the JavaScript Guide. You don't need to read the section about "Arrow functions", but you may find it interesting to do so if you are already familiar with the ordinary function syntax in JavaScript.

Also read about `while` and `do...while` loops in the "Loops and iteration" section.

These materials will teach you about:

- How to define your own functions
- How to call functions you have defined
- How to return values from functions
- How to use function parameters and arguments
- How to write code that will run repeatedly inside a Exercise `while` or `do...while` loop.

Exercise

One of our goals as programmers should be to write "clean code" - that is, code that is simple and understandable. Take a look at this piece of code:

```
printWelcomeMessage();
performOneCalculation();
```

It's hopefully fairly obvious what this code is trying to do, even if you don't know the details of how those functions work. **Refactor your code so it looks the same as this example** - that will involve splitting your existing code into two new functions, and then having your program just call them both in turn.

> *"Refactoring" is the process of improving your code without changing its behaviour. Those improvements might be to make it more readable, to make it easier to change and extend in future, or something else entirely.*

Now take it a step further. I'm guessing you have at least a couple of pieces of code of the form:

```
console.log('Please enter a number:');
const response = readline.prompt();
const number = +response;
```

Create a function that encapsulates this pattern, and use it to replace all the code that's similar to the above. The same function should be usable every time you want to print out a message, and interpret the response as a number.

Now **see how many further improvements you can make** to the readability of your code by splitting it off into smaller, well-named functions.

Having done all that, it should be relatively easy to add in a couple of new features, using while loops:

- **Make your calculator keep running** - once it's calculated an answer, it should just loop round and start again. Presumably you don't want to keep printing the welcome message every time though (So that you don't get stuck running your program forever, you should know that you can force it to stop by pressing `Ctrl + C` in the console while it is running).
- **Force the user to enter valid numbers** - when prompting for a number, it's annoying if your program stops working correctly if the user types in a string instead. Have it just ask again in this case.

For the second bullet you might find the `isNaN()` function useful. You can read about it here, and use it like this:

```
const maybeNumber = +"42";
if (isNaN(maybeNumber)) {
  // It didn't work - we have NaN.
} else {
  // It worked - we have a number.
}
```

Sample solution

Don't worry! This exercise is a step up from the previous ones. You should do your best to produce something that works on your own, but if you get stuck here's one possible solution: Link

There are many ways to tackle this problem though - if you do resort to looking at this model answer before you've finished your own solution, see if you can improve upon it and come up with something even better!

And as usual, if you do manage to complete the exercise by yourself, take a look at the sample solution afterwards and compare and contrast. As we build more interesting functionality, the range of possible ways you could implement the code will grow, and it becomes increasingly worthwhile thinking about the pros and cons of different alternatives. Which approach produces the shortest code? How about the most self-explanatory code? Which is "better"?

7. Strings and Objects

This topic looks in more detail at strings in JavaScript, and a new data type which is extremely important in JavaScript - objects.

Reading material

Read the sections "Text formatting" and "Working with objects" of the JavaScript Guide. After reading these sections, you should know about:

- Template literals and string interpolation - e.g. `Template with ${expression}`
- How to create objects using object initializers or constructors
- How to access and set properties of an object

Exercises

Part 1

Review how your calculator deals with outputting strings at the moment. **Can you use string interpolation to improve your code?** Perhaps try adding some more informative text now that it's easier to print out more complex messages.

Part 2

We'd like to enhance the calculator by adding a calculation mode for working with strings. Specifically, we'll add support for counting the number of times each vowel appears in a given string. Working with strings doesn't really fit into the current interface, so we'll modify our main program loop to look something like this:

```
const ARITHMETIC_MODE = '1';
const VOWEL_COUNTING_MODE = '2';

printWelcomeMessage();
while (true) {
  const calculationMode = getCalculationMode();
  if (calculationMode === ARITHMETIC_MODE) {
   performOneArithmeticCalculation();
  } else if (calculationMode === VOWEL_COUNTING_MODE) {
   performOneVowelCountingCalculation();
  }
}
```

And the output might look something like this:

```
Welcome to the calculator!
==========================
Which calculator mode do you want?
 1) Arithmetic
 2) Vowel counting
> 2

Please enter a string:
> ThE QuIcK BrOwN FoX JuMpS OvEr ThE LaZy DoG

The vowel counts are:
 A: 1
 E: 3
 I: 1
 O: 4
 U: 2

 Which calculator mode do you want?
 1) Arithmetic
 2) Vowel counting
>
```

Implement some functionality along these lines - pay attention to how the example treats uppercase and lowercase vowels the same. Since we've just learned about objects, you should use an object to hold the answer when doing the vowel counting calculation.

Sample solution

Here's one possible solution: Link

Try to build your own solution before peeking!

Appendix: Working with the Terminal application

This appendix will introduce you to the concept of a command prompt/terminal if you are currently unfamiliar with them, or have not used one before.

What is a command prompt/terminal?

Developers often use a program called "command prompt" (on Windows), "terminal" (on Mac/Unix), or something similar in order to accomplish file management tasks.

Most of the file management tasks you have had to perform on a computer so far have been achievable using "Windows Explorer" (on Windows) or "Finder" (on Mac), where you have been able to open files/folders by double-clicking on them, or moving files by dragging them, or deleting them by selecting them then pressing the `Delete` key.

For the most part, a terminal prompt is simply a window in which you can type file management commands, and the commands will be executed. You can type commands to run applications, create files, delete files, edit files, and more.

How do I use a terminal window?

Let's try opening up a terminal window. It is a program already installed on your system, so you can run it by pressing `Cmd + Space` to bring up Spotlight, then typing in the words "Terminal". You should see an entry saying "Terminal" as the first search result - press `Enter`, and the program will open. It will look like a white window with some black text in it (the colour scheme may be reversed depending on your system theme. In all likelihood, the text will read something like:

```
Last login: Thu May  9 13:56:11 on ttys000
yourComputerName:~ yourUserName$
```

If you see the above, you have successfully opened a command prompt terminal window - if you type a command and press `Enter`, your command will be interpreted and executed.

The reason why we are now learning to use a command prompt is that as developers, we need to perform new tasks that are currently not achievable by clicking around in Windows Explorer or Finder. Specifically, by using a command prompt, we can pass "arguments" (or you could call them "parameters") into programs as additional information when opening them.

For example, if you wanted to run "TextEdit" on Mac, you could click on its icon in the Dock, in Finder, or in Launchpad, and TextEdit would open. However, in a terminal window, you can type the word `open -a textedit` and press `Enter`, and the TextEdit window will open. If you did this, your terminal window would now look like this:

```
Last login: Thu May  9 13:56:11 on ttys000
yourComputerName:~ yourUserName$ open -a textedit
yourComputerName:~ yourUserName$
```

Although it seems at first that the command prompt way of doing this is harder, there are a few advantages. For example, if you wanted to open a specific text file in TextEdit, you can do so easily in the Terminal by passing the name of the file as an argument to TextEdit. By typing the command notepad `open -a textedit myFile.txt`, TextEdit will open, having already opened the file `myFile.txt`.

This will become more important when you are working with tools like **Node.js** which need to be operated on using the command line. If you have a script called `myScript.js` you will need to run it in the command line by typing `node myScript.js`.

The Working Directory

The command prompt actually tells you what directory you are currently operating in. Specifically, `yourComputerName:~` means that this terminal window is currently operating inside the `~` directory. That seems odd, but `~` is actually a short way of saying "the current user's home directory" in Unix (and Mac OS is a type of Unix). This shortcut exists because your home directory is often used. If you want to find out exactly where this directory is supposed to be, you can type in `pwd` and the pwd program will print out the full path to your current working directory:

```
Last login: Thu May  9 13:56:11 on ttys000
yourComputerName:~ yourUserName$ open -a textedit
yourComputerName:~ yourUserName$ pwd
/Users/yourUserName
yourComputerName:~ yourUserName$
```

You should know that `/` at the beginning of the path means the "root" of your hard drive, i.e. the top-most level directory. And now we can see that your user's home directory lives in a folder called `Users`, which itself is located at the top level of your drive. If you store your project files somewhere else, e.g. your `Dropbox` folder, which might be located in your user directory, then you can use the `cd` command to change the working directory to it. So, it would look like this:

```
Last login: Thu May  9 13:56:11 on ttys000
yourComputerName:~ yourUserName$ open -a textedit
yourComputerName:~ yourUserName$ pwd
/Users/yourUserName
yourComputerName:~ yourUserName$ cd Dropbox
yourComputerName:Dropbox yourUserName$
```

Now you can see the prompt has changed to reflect the directory you are currently in (`yourComputerName:Dropbox yourUserName$`). If you type `pwd` now, you should see the full path to your new current directory:

```
Last login: Thu May  9 13:56:11 on ttys000
yourComputerName:~ yourUserName$ open -a textedit
yourComputerName:~ yourUserName$ pwd
/Users/yourUserName
yourComputerName:~ yourUserName$ cd Dropbox
yourComputerName:Dropbox yourUserName$ pwd
/Users/yourUserName/Dropbox
yourComputerName:Dropbox yourUserName$
```

The `cd` command can be used in a few different ways:

- `cd DirectoryName` will open `DirectoryName` inside your current directory.
- `cd C:\Users\YourName\SomeFolder\SomeOtherFolder` will open that path, regardless of where you are at the moment (contrast this with the above command, which is sensitive to your current location). The difference is including the `/` at the start of the path (which indicates the root of your hard drive, remember?)

- `cd ..` is a special command that will go up one folder (i.e. it will take you from `/Users/YourName` to `/Users/`)

By the way, `cd` stands for "Change Directory".

Other useful commands

If you want to experiment more with command prompt, here are a few more commands that might come in useful:

- `ls` lists all the files in your current directory
- `mkdir DirectoryName` creates a new directory inside your current one
- `rm fileName.txt` deletes a file - be careful with this
- `cp file1.txt file2.txt` duplicates an existing `file1.txt`, calling the new file `file2.txt`
- Here's a fun one (unmute, turn up your volume): say "`I'm a wonderful computer`" will use your Mac's TTS (text to speech) engine to read out whatever you want. You can even choose which of its inbuilt voices you'd like it to use: `say -v Samantha "Hi there"` (you can view the full list of voices like this: `say -v ?`)
- There are also `mv` (move) and `cat` (concatenates files and writes them to the console) commands, and much more - have a search on the internet if you are curious, but remember that the command prompt is a very powerful tool so **do not run any command that you do not fully understand** - the risks of doing so include accidentally deleting files you did not intend to, or being tricked by someone on the internet into running something malicious.

You're Ready!

Now that you have completed this pre-reading you can go on to take the online coding and aptitude tests by following the link in the email you were sent.

IMPORTANT: Remember to select JavaScript as the language you would like to take the test in (the default language is C). You will need to do this for each test you take.

Please get in touch if you have any problems with the above via info@techswitch.co.uk

Good luck!

TechSwitch Team