Exercise 12a – Types and Operators

Objectives

To understand how types and operators work in JavaScript.

Overview

There are two parts to this exercise. The first explores some primitive types and some of the string conversion functions. The second explores operators. The exercise is divided into two sections. Once you complete part one, do not go onto part two until the instructor has delivered the rest of the chapter.

This exercise is scheduled to run for **30 mins**.

Part 1 – Exploring Types

In this part of the exercise, you will be exploring some of the common JavaScript types and learning how to manipulate strings.

1. Open the **EX02** starter folder and select **Part 1 – Types**.

We now need to create a page.

2. Right click on the **Starter** folder and choose **New File** and name it as **types.html**.
3. Create a skeleton HTML document (*use the Emmet abbreviation! and press tab*).
4. Within the **<body>** of the document, add a **<script>** element. It should have a closing tag.

First, we will explore the number type.

5. Add the code below to create a number. The console.log line will allow us to check the value has been set:

```
let numTest = 45.324568;
console.log(numTest);
```

6. Save the file (note the • that appears next to the file name in the tab to show that the file is not saved yet and changes to × when the file is saved).
7. Open VSCode's in-built console by selecting **View → Integrated Terminal** (*or use the shortcut key or icon on the bottom bar*).
8. Navigate the console prompt to the **Exercises** folder (*using the* **cd** *command and the names of the folders to change in to, your path will be C:/Courseware/QAJAVSC/Exercises on a QA-build Windows computer*).
9. At the prompt, type the command **live-server**.

This should open Chrome and show a directory listing for the Exercises folder.

10. Navigate to the file you have just created and click on it, it should open a blank webpage.

Live-server is a small HTTP server that has been pre-installed on your computer. It runs, by default, on port 8080 of localhost (IP address 127.0.0.1). If the browser is closed, you can reopen the page by typing either **localhost:8080** OR **127.0.0.1:8080** in the browser's address bar.

11. Press F12 to see the developer tools and choose the Console tab. You should see the value of the numTest variable displayed.

You have created a Number data type object in the program stack, this is a 64bit number. Next, we will explore the Number type a little more with some of its methods.

We will create a new variable called twoDecimalPoints and use a method to truncate the number.

12. Under the last code you wrote in types.html, add the following code:

```
let twoDecimalPoints = numTest.toFixed(2);
console.log(twoDecimalPoints);
```

13. Save the file and your browser should automatically refresh to display the value of **twoDecimalPoints** as **45.32**.
14. Pretty straightforward, right? Try using the **toExponential** function to eight figures using the code above as a guide. If you need, help call your instructor.

We will now move onto string objects and stretch ourselves a little more. One of the most useful parts of the string type is the ability to search and slice the string value into smaller sections. Do remember that, when they are constructed, strings are immutable types. We will be looking to walk before we run here, but some of the things we are about to do could be inefficient in larger JavaScript programs.

15. Under your last line of code in **types.html**, create a **stringTest** variable, as shown in the code segment below.
    (Please be sure to add the text exactly as it appears below; otherwise, the notes will not match up to what you will see in the console).

```
let stringTest = "I am the very model of a modern major general";
let indexOfM = stringTest.indexOf("m");
console.log(indexOfM);
```

16. Save the code and observer the browser console.

You will see a value of **3**, examine the string and you will see that the **m** is the *fourth* character, so there are *three* characters before the first **m**.

17. Change the **m** within the **indexOf** method call to a capital **M**, save and observe the output in the browser again.

This time, the **console.log** will return a **-1** value. The **-1** value is telling us that there is *no match within the string* at all proving that string searches are case sensitive.

What if we convert the string to upper case?

18. Before the indexOfM line, add the following code:

```
stringTest = stringTest.toUpperCase();
```

19. Save and observe the output in the browser again.

The output will, once again, give a value of **3**. Behind the scenes, the string is an indexed collection of characters and the search function is making its way through the letters character by character until it makes a match.

With that concept in mind, we will use the principals to learn how to slice a string.

20. Add the following code under the last line, then capture **start** and **end** in a **console.log**, save and observe the output.

```
let start = stringTest.indexOf("MODEL");
let end = stringTest.lastIndexOf('MAJOR');
```

This time, we have matched based upon words, but you could search for file paths or extensions; for instance, if we were reading from a form.

The two integer values held can be used to create a substring from the longer one using string's substring method.

21. Add the following lines of code to the end of your code, save and observe the output.

```
let subStr = stringTest.substring(start, end);
console.log(subStr);
```

The console should now return a value of **"MODEL OF A MODERN"**.

Let's finish up this exercise by writing this content to the browser window using the **document.write** method (we will look at the document object in more depth later on).

22. Add the following lines of code to the end of your code, save and observe the output.

```
document.write("<p>" + subStr + "</p>");
```

We have used an operator here: the + sign which we have used to concatenate the string together and mix our string value with some hard-coded HTML to create new content to the page. With that done, let's learn some more about operators.

Do not start the next part until instructed to!

Part 2 – Operators

In this part of the exercise, you are going to explore some of the operators that JavaScript provides. We have a series of simple expression that are pre coded but commented out. You need to jot down what you believe the outcome of every expression will output.

1. Examine the following operations and write the result in the Result column before you look at any code output!

   Arithmetic operators

   Operation **Result**

   console.log(5 + 5);

   console.log(5 * 10);

   console.log(10 % 3);

   console.log(5 + 10 / 2 * 5 - 10);

   console.log((6 + 10) / 2 * 5 - 10);

2. Open **Ex02 - Types and Operators → Part 2 Operators → operators.html** uncomment the Artihmetic Operators section.
3. Navigate to the correct file using browser navigation to see the output.

Refer to instructions 8 and 9 of Part 1, if you have closed the browser or the browser does not display the expected output.

4. Repeat for assignment operators, assuming x is initialised as 0 and the statements are processed sequentially.

   Assignment operators

   Operation **Result**

   console.log(x = x + 1);

   console.log(x+= 1);

   console.log(x++);

   console.log(++x);

5. Now, we will move onto relational operators. Every expression will evaluate as either true or false.

   Relational operators

   Operation **Result**

```
console.log(5 > 3);

console.log(3 != 3);

console.log(3 <= 2 && 5 >2);

console.log(!5>3);
```

6. Finally, we will explore what happens with mismatched types.

Mismatched types

Operation	**Result**
console.log(5 + "5");	
console.log(5 + true);	
console.log(5 * "5");	
console.log(1 == true);	
console.log(1 === true);	

If you have time…

1. Examine the types outputted by Part 1 of the exercise. What do you notice?
2. Experiment with **parseInt()** on the expression that adds a number and a string together.