# Exercise 1 – Types & Operators

## Objectives

* To understand:
  + How to create variables
  + How to use operators
  + The different datatypes JavaScript provides.

## Exercise Instructions

### Part 1 – Creating variables

In this part of the exercise, you will be exploring the most important part of any programming language. Which is how to create a variable.

1. Open (or create) a HTML file.
   1. The file must have a <script> tag, if it does not have one, create one.

***<script> </script>***

1. Within the script tag paste the following code which will create a variable called **example** with the value of **5** and then output it to the console, which allows us to check if it’s worked.

***var example = 5***

***console.log(example)***

1. Now open your HTML file that you were just working inside of in your browser (Highly recommend chrome, the exercises will be referring to things that may only be available in Chrome)
2. Once in the browser and the file is open, press F12 to open up the developer tools.
3. Once the developer tools are open go to the **console** tab, you should see the number 5 in there. This is from our code!
4. The second line of code we executed (**console.log…**) was a method that outputs the data we give it to the console, which is where we saw our number 5. We gave it the reference of “example” to the method, so it output the data that was inside of that variable which was 5.
5. Underneath the code that you wrote before, now paste the following lines into it

***var operatorExample = 5 + 5***

***console.log(operatorExample)***

1. The plus sign is something called a **binary operator**, which means there are two sides required to it, the first value and then the second value.
2. If you go back to your webpage and refresh the page after saving your new changes (or reopen the page if you closed it) you should have a 5 outputted (from before) and now a 10!
3. You can also replace the literal values of **5** with our previous variable **example** and you should get the same output, give it a try.
4. Next we will move onto the String object, another very important datatype in JavaScript.
5. A string is a word, and to represent it in JavaScript that we want the actual **word** and not a variable or keyword with that name, we wrap it in quotes.
6. Paste the following code in and run it

***var exampleString = “Hello World!”***

***console.log(exampleString)***

1. We can also use operators on strings, paste the following code and run it

***var exampleString2 = “Hello” + “ World” + “!”***

***console.log(exampleString2)***

1. However, operators like minus and divide won’t be very useful on Strings, but the addition operator is useful.
2. You can also try and use operators with *different* types, like a number and a string, see what happens! **“5” + 5** *…*
   1. Try adding a string and a number together, then output it.
   2. Try taking away a number from a string, then output it
3. Execute some of the following operations and places the results in the columns provided.

|  |  |
| --- | --- |
| **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**)**; |  |

1. There are also some special operators that are designed to save us writing lots of code. Imagine a situation where we want to increment a number by 2 over and over again, the code might look like this

***var a = 1***

***a = a + 2***

***a = a + 2***

This is a very common operation in programming, on all data types. We have something called a **compound operator** that shortens this for us.

***var a = 1***

***a += 2***

***a += 2***

For the special case of incrementing (or decrementing) something by 1, there is another special shorthand operator for this.

1. Add the following code before the next set of operations, initializing a variable **x** as 0.

***var x = 0***

1. Execute some of the following operations and places the results in the columns provided.

|  |  |
| --- | --- |
| **Arithmetic operators** | |
| **Operation** | **Result** |
| console.log(x = x + 1); |  |
| console.log(x+= 1); |  |
| console.log(x++); |  |
| console.log(++x); |  |
| console.log(x = x + 1); |  |

1. 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); |  |

1. 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); |  |

1. Lastly, try using the **parseInt()** method on the expression that adds a number and a string together, see what happens!