TASK

Variables and Data Types

Visit our website

Introduction

WELCOME TO THE VARIABLES AND DATA TYPES TASK!

In this task, you will learn about variables and the fundamental data types not just in JavaScript, but in programming in general. These include numbers, strings, booleans, arrays and objects. This task will go into detail on the first three data types. You will learn how to use these data types to create functional programs.

RECAP ON VARIABLES

A variable is a way to store information. It can be thought of as a type of "container" that holds information. To declare a variable is to assign it a storage space in memory and give it a name for us to reference it with.

In JavaScript we use the following format to create a variable and assign a value to it:

```
let variableName = value_you_want_to_store;
```

ESSENTIAL DATA TYPES

Within JavaScript, you'll find yourself working with many data types based on the kind of application you're dealing with. There are four main data types with which you should familiarise yourself as they form the basis of any JavaScript program:

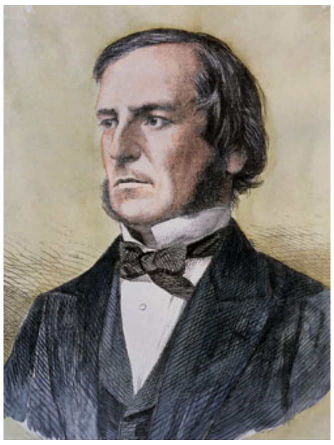
Data Type	**Declaration**
Numeric	`let someNumber = 25;`
String	`let someName = "Joe";`
Boolean	`let someBool = true;`
Array	`let someArray = [15, 17, 19];`
Object	`let someObject = {firstName: "James", lastName: "Bond"}`

- **Numeric** data types describe any numbers that you store.
- **Strings** refer to a combination of characters. “Joe” and “23 Main Street” are examples of strings. We use strings to store and manipulate text. String values must always be put within quotation marks (`""`).
- **Booleans** are data types that can store only two values: either **true** or **false**.
- **Array** is a data type that we use to store multiple values. As shown in the table above, we put all the values we want to store in an array variable in a comma-separated list that is enclosed by square brackets (`[ ]`).
- An **object** is a data type that stores a collection of related data. If you wanted to create a person object, for example, you would store a collection of related information that describes a person such as their name, surname, date of birth, address etc.

A note from the

HyperionDev Team

You have just learned what a boolean data type is, but why is it called a Boolean? It is named after George Boole (1815 - 1864), an English mathematician who helped establish modern symbolic logic and devised Boolean algebra. Boole introduced Boolean algebra in his first book, The Mathematical Analysis of Logic (1847), and developed his ideas in the book for which he is better known, An Investigation of the Laws of Thought (1854).



Boole was born in Lincoln, Lincolnshire, England. His father, a struggling shoemaker, encouraged him to take an interest in mathematics. Boole went to an elementary school and trade school for a short time, but he mostly educated himself. By the age of fifteen, Boole began teaching. At age nineteen, he set up a school in Lincoln. Boole was appointed as a professor of mathematics at Queen’s College, Cork in 1849, despite not holding any university degree.

- George Boole

TYPE IDENTIFICATION

In some programming languages, you must tell the computer what data type you want a variable to store when you declare the variable. This is not true with JavaScript. JavaScript is smart in the sense that it's able to detect variables' types automatically based on the value that you assign to your variable. If a value is in quotation marks, JavaScript knows that the value is a string. It would likewise automatically know that it should store the value 12 as a number if it is not inside quotation marks.

```
let myString = "12"; // String
let myNumber = 12; // Number
```

Note that the variable names in the above example have been chosen to include the words "String" and "Number" in accordance with the guiding principle of creating meaningful variable names that help the programmer remember something about the variable. Consider the following:

```
let myNumber = "12"; // String
let myString = 12; // Number
```

In this example, the *first variable is still a string, and the second is still a number*, despite the change to their names. This is because the data are still assigned the same way - as a string (with the "12" in inverted commas) to the first variable, and as a number (just 12) to the second variable. It is the data on the right-hand side of the equation that determines the variable type, not the variable name. This is a common source of error amongst students learning to code, and so, if you're confused here, take some time to reread this section and make sure you really understand it.

JAVASCRIPT INTELLIGENCE

So what happens if you were to code this line?

```
let unknownType = 53 + "Bond";
```

In many programming languages, this would cause a *type conflict error* because mathematically you can't add a number and a string, but JavaScript simply solves this issue by converting the entire variable contents into a string. The logic is that JavaScript first sees a number and assumes a number variable type but, once it detects a string, the variable is reclassified as a string.

FINDING THE TYPE

Sometimes, you may want to check some data to inspect its data type property. This is done by making use of the `typeof` built-in function.

```
typeof "Bond";       // this returns a string
typeof [4,6,2];     // this returns an array
```

Try this:

1. Enter the following code into your console and then press enter.

```
let myName = "Tom";
let num = 33.33;
let pass = true;

console.log(myName);
console.log(num);
console.log(pass);

let myDataType = typeof num;
console.log(myDataType);

console.log(typeof pass);
```

2. Take careful note of the output. Be sure to understand how the code you entered resulted in the output displayed in the console.

CASTING TO DIFFERENT TYPES

When coding, it may be necessary to change a variable from one data type to another. For example, when getting user input, JavaScript automatically assumes that whatever the user gives the program is a string. This could be a problem if we're trying to get numbers from the user to do calculations. Have a look at the code below:

```
let num1 = prompt("Enter the first number: ");         // User enters 6
let num2 = prompt("Enter the second number: ");        // User enters 4
console.log(num1 + num2);                              // Output: 64
```

Why is the output 64 and not 10? The program assumes that the input given is a string and so puts the two numbers together as if they were words or characters. In order to add them together as numbers, we need to cast them as such.

To cast to different data types we use the following functions:

```
let str3 = "3";               //Output: "3" of type string
let num3 = Number(str3);      // Output: 3 of type number
let bool3 = Boolean(str3);    //Output: true of type boolean

let num0 = 0;                 // Output: 0 of type number
let str0 = String(num0);      // Output: "0" of type string
let bool0 = Boolean(num0);    // Output: false of type boolean

let bool1 = true;             // Output: true of type boolean
let str1 = String(bool1);     // Output: "true" of type string
let num1 = Number(bool1);     // Output: 1 of type number
```

Once a variable has been converted to a number, any mathematical operations can be performed with it. Take note of what happens when we cast a variable to a boolean. If we cast a number to a boolean, any number except zero (0) will return `true`. If we cast a string to a boolean, any string except an empty string (“”) will return `true`.

Let’s look back to our previous example. If we want to be able to add numbers given to us by the user, we need to cast them as numbers. See below:

```
let num1 = Number(prompt("Enter the first number: "));  // User enters 6
let num2 = Number(prompt("Enter the second number: ")); // User enters 4
console.log(num1 + num2);                               // Output: 10
```

Now that we have cast the strings to numbers, we can add them together to make 10.

MATHEMATICAL CALCULATIONS WITH JAVASCRIPT

Doing calculations with numbers in JavaScript is similar to the way they would be performed in regular mathematics. The only difference between calculations in mathematics and programming is the symbols you use, as shown below:

Arithmetic Operations	Symbol used in JavaScript
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus (Divides left-hand operand by right-hand operand and returns remainder, e.g. 5%2 = 1)	%
Add one to a variable (e.g. 2++ = 3)	++
Subtract one from a variable (e.g. 2-- = 1)	--

THE MODULUS OPERATOR

It is important that we discuss one special operator, the modulus operator. It is crucial for a programmer to know how to use this operator because it is used to solve many computational problems. Here is an example of it in use:

```
let remainder = 152 % 10;
```

Given the division problem 152 / 10, the answer is given in two parts. The quotient is 15 and the remainder is 2. We informally say the answer is 15 remainder is 2. The modulus operator is a means of getting the remainder of a division problem directly. So, the result of the expression above would be 2.

Try this:

1. Type the following JavaScript into the console.

```
let num1 = 12;
let num2 = 34;

console.log("num1 = " + num1);
console.log("num2 = " + num2);
console.log("num1 + num2 = " + (num1+num2));
console.log("num1 / num2 = " + num1/num2);
console.log("num2 % num1 = " + num2%num1);
console.log("++num1 = " + ++num1);
console.log("--num2 = " + --num2);
```

2. Press enter and take careful note of the output. Be sure to understand how the code you entered resulted in the output displayed in the console.

NOTE

When we are using ++ and - - it is essential to place them in the correct position. If you are placing the operator afterwards, it will update after the line is completed, if it is placed before, then it will affect the current line.

```
let num1 = 12;
let num2 = 34;

console.log("num1++ = " + num1++); //12
console.log("num2-- = " + num2--); // 34
console.log("New value of num1 =" + num1); //13
console.log("New value of num2 =" + num2); //33

let num3 = 12;
let num4 = 34;
console.log("++num3 = " + ++num3); //13
console.log("--num4 = " + --num4); //33
```

Take note:

In JavaScript, the + sign can be used to add two numbers OR to concatenate a value to a string. To concatenate things means to join those things together. Consider this line of code in the example above: `console.log("num1 + num2 = " + num1+num2);`. Here the variables `num1` and `num2` are added together and then the result is concatenated with the string **"`num1 + num2 =` "**.

THE STRING DATA TYPE

As you have learned above, a string is a combination of characters between quotation marks, e.g. "This is a string!". We can access the characters in a string based on the index. Unlike normal counting, which starts at 1, indexing starts at zero. Have a look at the table below:

Character	H	e	l	l	o	!
Index	0	1	2	3	4	5

Here we can see that the string "Hello!" has a maximum index of 5, even though there are 6 characters (spaces and punctuation also count as characters). This is useful because we access a specific element in a string by using its index, or find the index of an element by using the character.

```
let greeting = "Hello!";
let letter = greeting[4];     // returns the letter 'o'
let index = greeting.indexOf("e");  // returns the number 1
```

We can also find the length of a string in the following way. Note that the *length* is not the same as the *index*. The length is the number of elements counting from 1. Therefore, the index of the last element in a string will be equal to its *length - 1*.

```
let greeting = "Hello!";
console.log(greeting.length);       // returns the number 6
```

Strings can be combined, or concatenated, using the plus (**+**) sign. For example:

```
let greeting = "Hello ";
let name = "Tom.";
let sentence = greeting + name;
console.log(sentence);  // Hello Tom.
```

If we wanted to print these on different lines, we could use an escape character, specifically the 'newline' character, `\n`. We would change the value of *greeting* to **"Hello\n"**:

```
let greeting = "Hello \n";
let name = "Tom.";
let sentence = greeting + name;
console.log(sentence);
// Hello
// Tom.
```

A more succinct way of doing both concatenation and multiline strings is by using template literals. This means using backticks (`) and the format of `${expression}`. We adapt the code above using template literals below:

```
let greeting = "Hello";
let name = "Tom";
console.log(`${greeting} ${name}.`);      // Hello Tom.
```

Note how we no longer need the *sentence* variable. We simply put the variable name of the value we want within the curly brackets. Any characters between the backticks are also printed, which means that we don't need to put a space after "Hello" or a full stop after "Tom". This also means that we don't need escape characters; if we want a new line, we simply create a new line. See below:

```
let greeting = "Hello";
let name = "Tom";
console.log(`${greeting}
${name}.`);
// Hello
// Tom.
```

We can also get information from strings and manipulate them with certain methods and properties.

Instructions

Open **example.js from Task 2** in Visual Studio Code and read through the comments before attempting these tasks.

Getting to grips with JavaScript takes practice. You will make mistakes in this task. This is completely to be expected as you learn the keywords and syntax rules of this programming language. It is vital that you learn to debug your code. To help with this remember that you can:

- Use either the JavaScript console or Visual Studio Code (or another editor of your choice) to execute and debug JavaScript in the next few tasks.
- Remember that if you really get stuck, you can contact a mentor for help.

Compulsory Task 1

Follow these steps:

- Create a JavaScript file called **myVariables.js**.
- Create the following variables with the correct data types (as indicated by their names) as their values. You can choose the values themselves: ***myFirstNumber***, ***mySecondNumber***, ***myFirstString***, ***mySecondString*** and ***myBoolean***.
- Output the result of multiplying the two numerical variables.
- Concatenate the two strings and output the result.
- Using template literals, output a multiline string which might read as follows:

```
The boolean is: true
The first number is: 6
The second number is: 5
6 x 5 = 30
The first string is: This is my first string.
The second string is: This is my second string!
These two together make: This is my first string. This is my
second string!
```

Compulsory Task 2

Follow these steps:

- Note: For this task, you will need to create an HTML file to get input from a user. Please refer back to previous tasks for a refresher if needed.
- Create a JavaScript file called **fortuneTeller.js**.
- You are going to create a fortune teller based on information that you receive from the user. You need to ask for the following information:

1. The user's mother's first name
2. The name of the street they grew up on
3. Their favourite colour as a child
4. Their current age
5. A number between 1 and 10

- With this information, you can work out the following:
    a. (5) is the number of years in which they will meet their best friend.
    b. Their best friend's name will be (1) + (2).
    c. (4) divided by (5) (rounded off) is the number of years in which they will get married. *(Hint: look up `Math.round()`)*
    d. The remainder of (4) divided by (5) is how many children they will have.
    e. (4) minus (5) is in how many years they will dye their hair (3).
- Output the result of the above in a multiline string. For example, the output may be:

```
In 7 years you are going to meet your best friend named Mary Washington.
You will get married in 4 years and have 6 children.
In 20 years you are going to dye your hair blue.
```

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.