TASK

JavaScript I: Variables and Operators

Visit our website

Introduction

WELCOME TO THE FIRST JAVASCRIPT TASK!

Well done! The fact that you have reached this task means that you have been able to demonstrate your competence using HTML and CSS!

You are probably very eager to make your sites more dynamic now, though. For example, you have no doubt thought, “It’s great that I can make a form, but how do I get it to work?”

You will learn to do this using JavaScript. This is a scripting language that allows you to create dynamic websites. In this task, you will learn what JavaScript is. You will also learn the basics of creating your first program using JavaScript! Once you have mastered some of the basics of creating a program with JavaScript in this task, you will learn to integrate JavaScript into your webpage in the next task.

Get in touch

Connect for support

Remember that with our courses, you’re not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

INTRODUCTION TO JAVASCRIPT

JavaScript is often used as a front-end scripting language. However, as you will learn later, it can also be used for server-side programming.

HTML, CSS and JavaScript are the three core technologies used for front-end web development. As you already know, HTML and CSS describe what content is shown on a webpage and what that content looks like. **JavaScript is used to make the webpage do things.** JavaScript is used to change web pages from static web pages (that always look the same) to dynamic web pages (pages that can dynamically change on the browser).

JAVASCRIPT BACKGROUND

Before you start learning to use JavaScript, it is important to quickly cover some theory and background related to JavaScript.

ECMAScript

ECMA is an international organisation that creates standards for information and communication systems. ECMAScript is a standard, created and maintained by ECMA, that defines the JavaScript general-purpose programming language. In other words, *ECMA* is basically *the organisation that has designed JavaScript* and *EMCAScript is basically JavaScript*. ECMAScript is based on several originating technologies, including JavaScript and JScript. ECMAScript was first released in 1997. As you can probably imagine, standards for programming languages have had to be updated massively since the ECMAScript originated to accommodate the huge changes that have taken place in technology since then. This has resulted in several versions/editions of ECMAScript. The 6th edition of the ECMAScript (ES6), which was released in 2015, is the biggest revision of ECMAScript since 1997. ES6 is sometimes also referred to as ES2015 or Harmony.

ES6, ES7, and ES8

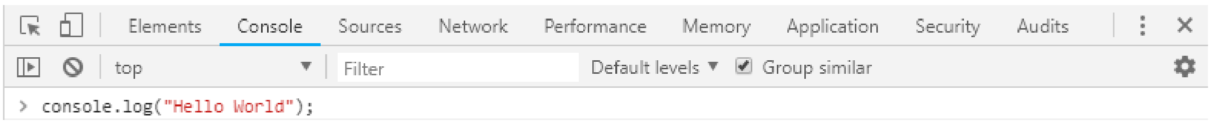
ES6 "… is the most extensive update to ECMAScript since the publication of the first edition in 1997." ES6 has introduced many changes to JavaScript (which basically is the same thing as ECMAScript) which improve the programming language greatly. Besides ES6, though, there are also already versions ES7 and ES8. This is because, since the release of ES6, the ECMA has decided to release updates to ECMAScript every year. All versions of ECMAScript after ES6 should, therefore, have fewer revisions than ES6.

When you start coding in JavaScript, you may notice that there are often different ways of writing the same code. The reason for this is often that some code will be written using the 'older' version of JavaScript while other code will be written using the changes to the language that have been introduced since ES6. The updates to JavaScript since ES6 will be highlighted in your tasks.

That's the theory out of the way! Let's get coding!

USING THE JAVASCRIPT CONSOLE

All modern browsers offer built-in support for JavaScript (you're using it without even realising it). Browsers have a built-in console that can be used for debugging web pages. The functionality of the console may differ slightly based on the browser you use. In this task, we will be using **Chrome's** DevTools Console. To access this tool, open Chrome and then press either **Ctrl+Shift+J if you are using Windows / Linux** or **Cmd+Opt+J if you are using Mac**. You should see something similar to the screen shown in the image below. The console may already display some messages. You can clear these by right-clicking on the console and then selecting "Clear console."



```
> console.log("Hello World");
```

You can input your JavaScript code directly into this console to test and debug it. To write information to the console, we use the instruction:

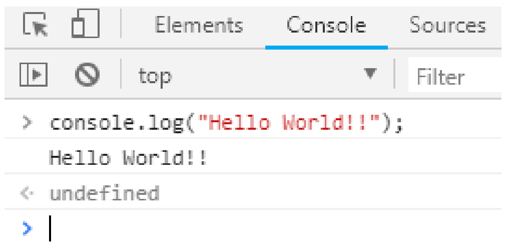
```
console.log("Whatever you would like to write to the console");
```

The above is a line of JavaScript code. It instructs your computer to log (or write) the text in the quotation marks to the console.

Try this:

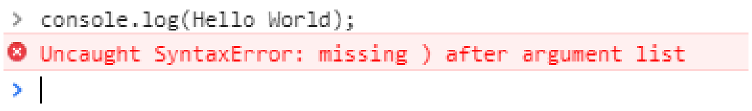
1. Open the Console.
2. Type the code `console.log("Hello World!!");` into the console.
3. Press enter.

If you typed everything correctly, "Hello World!!" should be shown in the console as shown below:



```
> console.log("Hello World!!");
  Hello World!!
<· undefined
>
```

If you haven't typed the instruction *exactly* as shown (for example if you forgot to add the quotation marks, or you used capital letters where it should have been lowercase letters, or you spelled "console" incorrectly) you may get an error. See an example of an error below:



```
> console.log(Hello World);
✖ Uncaught SyntaxError: missing ) after argument list
>
```

Like all programming languages, JavaScript has **syntax rules** that must be followed closely. Otherwise, your instructions will not be correctly interpreted and executed.

Take note:

Syntax is the "spelling and grammar rules" of a programming language and determines how you write correct, well-formed statements. These syntax rules are very strict. If you don't follow the rules exactly, your code won't be recognised and therefore, won't execute. A common syntax error you could make is forgetting to add a semicolon (**;**) at the end of each statement. The process of removing all the errors from your code is referred to as **debugging**.

Now try typing the following code into the console:

```
alert("Hello World!! I can write an instruction using JavaScript");
```

Press enter and see what happens.

Imagine that you wanted to write the code that prints the message "Welcome to my web page Tom" to the console but instead of using the name "Tom" every time, you'd like to be able to change the name based on who is currently viewing your web page. To do this, we need something called a **variable**.

VARIABLES

A script is basically a set of instructions that are interpreted by the computer to tell it what to do in order to accomplish a certain task. Programs usually process data that is somehow (there are various ways) input into the program, and then output the results of the processing. This data must be stored somewhere so that it can be used and processed according to the instructions we code. When coding, we use **variables** to store the data we need to manipulate.

A variable is a way to store information. It can be thought of as a type of "container" that holds information. We use variables in calculations to hold values that can be changed.

DECLARING VARIABLES

Before we can use variables in our code, we need to **declare** them. To declare a variable is to assign it a storage space in memory and give it a name for us to reference it with. This tells JavaScript that we want to set aside a chunk of space in the computer's memory for our program to use.

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

```
let variableName = valueYouWantToStore;
```

1. Notice from the code above that the first thing you need to do to declare a variable is to use the keyword `let`. You can also use the keywords `var` and `const` to declare variables. `var` is the old way of creating a variable. `const` is used when you want to declare a variable where the value can't change, i.e. the variable stores a constant value, whereas var is used for a variable where the value can change.
2. After that, you declare the name of the variable. You can name a variable anything you like as long as the name:
    a. Contains only letters, numbers, underscores, and dollar signs. No other characters can be used in variable names, including spaces. "My name" would thus not be an acceptable variable name.
    b. Starts with a letter.
    c. Is not a reserved word. In JavaScript, certain words are reserved. For example, you would not be able to name a variable "var", "console", or "log", because these are reserved words.

If you follow these rules, you can call your variables whatever you want. It is, however, good practice to give your variables *meaningful names*. This is important for two reasons: it helps other developers working on the code follow the logic, and it helps you remember what you were doing when you return to working on the code after a period of time

Below is an example of bad naming conventions vs good naming conventions.

myName = "Tom"	# Good variable name
variableOne = "Tom"	# Bad variable name
string_name = "Tom"	# Good variable name
h4x0r = "Tom"	# Bad variable name

myName and *string_name* are examples of descriptive variables as they reveal what content they store.
variableOne and *h4x0r* are terrible names because they are not descriptive.

To assign a value to a variable, you need to use the assignment operator. This is the equal-to sign (=) we usually use in maths. It takes the value on the right-hand side of the = and stores it in the variable on the left-hand side. For example, consider the line of JavaScript code below:

```
let myName = "Tom";
```

That statement would cause your computer to create an area in memory (i.e. a variable) called "myName" and put the value "Tom" into that area in memory.

It is important that you always put the value you want to assign (or put into) your variable on the right-hand side of the assignment operator (**=**).

Try this:

Add the following code to your console:

```
let myName = "Tom";
console.log("Hi there " + myName);
```

You have now successfully used a variable! In the example above, "myName" is referred to as a variable because the value stored in the position in memory named "myName" will *vary*/change throughout our program. You will learn how to get values to store in variables from users and other sources in later tasks.

Variables store data and the type of data that is stored by a variable is intuitively called the **data type**.

Take note:

ES6 Update: No more `var`

With previous versions of ECMAScript, you could only declare variables with the keyword `var`. With ES2015 (or ES6), variables are declared with either the keywords `let` or `const`. Variables declared with the keywords, `let` or `const` are scoped within the code block they are declared in. Variables declared using the keyword `var`, on the other hand, were declared either locally or globally in relation to a function (which sometimes lead to errors). `const` variables must be assigned a value when they are declared and the value they are assigned cannot change. `let` variables can be re-assigned a value within the block where they have been declared but they cannot be redeclared.
In summary, with ES2015:

- Don't use `var` any more.
- Use `const` whenever appropriate to initialise variables
- Use `let` whenever the value of that variable needs to be changed in a block of code.

SPOT CHECK 1

Let's see what you can remember from this section.

1. How would we write the JavaScript to output "Hello World" to the console?
2. What is a variable?

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.
- The **String** data type holds 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 as seen above. Using just quotation marks with no other characters inside them creates an 'empty' string, a string variable with no value yet (`""`). Numeric characters can be part of, or the entirety of, strings, e.g. "3 Main Road", "27".
- The **Boolean** data types can store only two values: either true or false.
- An **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 (`[ ]`). You will learn more about arrays in a later task.
- 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. You will learn a lot more about objects later.

TYPE IDENTIFICATION

In some programming languages, you have to tell the computer what data type you want a variable to store when you declare the variable. **This is not the case 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";
let myNumber = 12;
```

Note that in the above example, the variable names have no effect on the type. For example, if we swapped them around (see the code block below), we'd still have two variables where the first is a string and the second is a number, they'd just have VERY misleading names!

```
let myNumber = "12";
let myString = 12;
```

The swapped-around example above thus creates a variable called myNumber containing a string with the characters 1 and 2 in it, and a variable called myString containing a number with the value of 12. This is important to grasp - there is nothing magical about the variable names that sets the type! The type is solely determined by JavaScript's automatic assessment of the value provided for the variable.

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. This is because a string can hold numerals, whereas a number variable cannot hold text.

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 object
```

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 with. 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, giving 64. In order to add them together *as numbers*, we need to cast them - forcibly set their type - 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 used 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, we have cast the strings to numbers so that we can add them together to make 10.

SPOT CHECK 2

Let's see what you can remember from this section.

1. Give an example of a string, a number, and a boolean.
2. What are the functions to cast something to a string, a number, or a boolean?
3. How can we find out what data type a variable is?

MATHEMATICAL CALCULATIONS WITH JAVASCRIPT

Doing calculations with numbers in JavaScript is similar to the way you'd do normal maths. The only difference between calculations in real mathematics and in 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. Often called “mod”, e.g. you would say five mod two equals on.)	%
Increment (add one to) a variable (e.g. 2++ = 3)	++
Decrement (subtract one from) a variable (e.g. 2-- = 1)	--

THE MODULUS OPERATOR

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

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

Given the division problem 152 / 10, the answer can be given as two parts. The quotient is 15 and the remainder is 2. We informally say the answer is 15 remainder 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 = " + Number(num1+num2));
console.log("num1 / num2 = " + num1/num2);
console.log("num2 % num1 = " + num2%num1);
console.log("num1++ = " + num1++);
console.log("num2-- = " + num2--);
```

Note that in the above example,
`console.log("num1 + num2 = " + Number(num1+num2));`

Could also just be written as:
`console.log("num1 + num2 = " + num1+num2);`

The variables are already numbers and so do not need to be cast as numbers, but doing so does not mess up the calculation.

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.

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 link or join 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 = ”`, giving an output of `num1 + num2 = 46.`

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!”. Each character in the string has a position in the string which is called an index. We can look at the characters in a string based on the index. Unlike normal counting, which starts at 1, indexing in programming usually 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. (It’s useful to remember that spaces and punctuation also count as characters!). Indexing is useful because we can extract 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]; //sets the value of the variable 'letter' to 'o'
let index = greeting.indexOf("e"); //sets the value of the variable 'index' to 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, so you count as normal, starting from 1. The string "Hello!" has a length of 6 and a maximum index value of 5. Thinking about this example we can see a rule that always holds true about the length of any string and the index of its last element - 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.
```

Note that the last character of the variable greeting in the example above is a space. If you concatenate two strings and the first does not contain a space as its last element, there will be no space between the elements in the concatenated string:

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

If we wanted to print two variables we're concatenating onto two different lines, we could use an escape character. For a new line, we use `\n`. Therefore, 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.

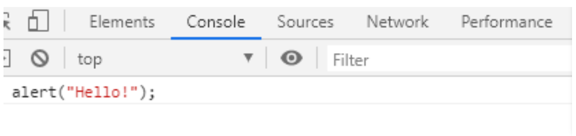
SPOT CHECK 3

Let's see what you can remember from this section.

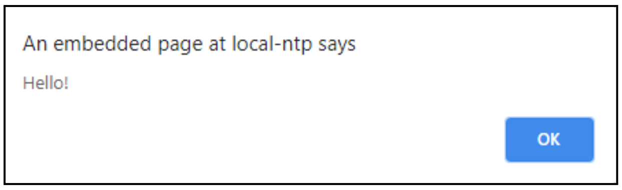
1. What does the modulus (%) operator do?
2. If you have the string "baby shark", what index is the space between "y" and "s"?
3. What is the format of template literals?

DISPLAYING IN JAVASCRIPT

There are a few ways of displaying output in JavaScript, but for now we are going to focus on `console.log` and `alert`. If you've tried out the example code above, you would have noticed that **`alert`** displays output in an **`alert`** box, while **`console.log`** displays output in the browser console. Have a look at the examples below:

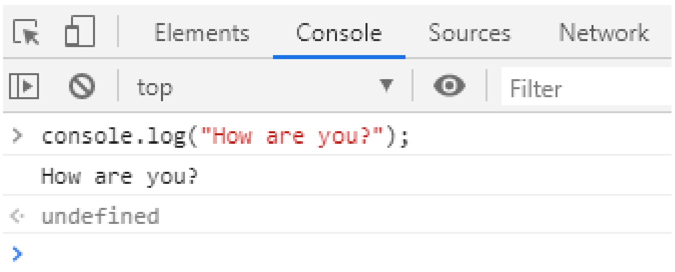


```
alert("Hello!");
```



An embedded page at local-ntp says

Hello!



```
> console.log("How are you?");
  How are you?
< undefined
>
```

Play around with these and use them based on what each task requires: does the person viewing the web page need to see the output or not?

CREATING .JS FILES

All the code we have written so far has been written directly into Google's DevTools Console. As soon as you close your browser though, all the code you wrote in the console will be lost. To write JavaScript code that you can save locally and reuse for your websites, you need to create JavaScript files. To do this, simply create a new file (using VSCode), enter the JavaScript instructions in this file one instruction per line, and then save this file with the **.js** extension. You will learn how to get your JavaScript files to work with your HTML files very soon.

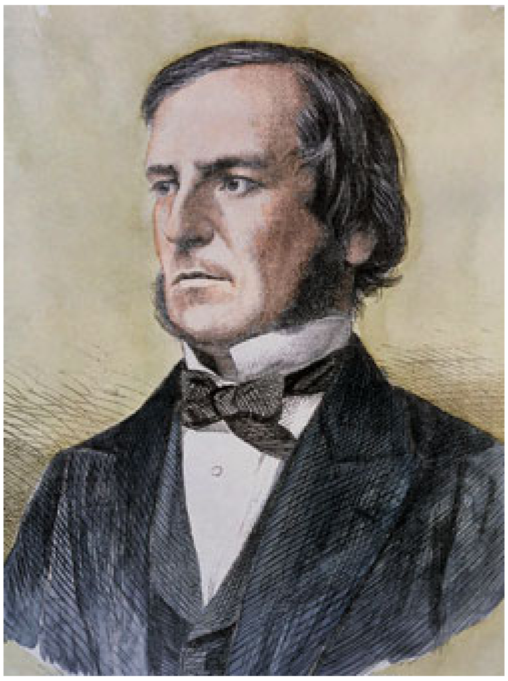
Save all your compulsory tasks as JavaScript files!

A note from the

HyperionDev Team

By now you should be familiar with the Boolean data type, and the vital role Boolean algebra plays in computer programming. Why is it called a Boolean datatype? It is named after George Boole (1815 - 1864) an English mathematician who helped establish modern symbolic logic and devised Boolean algebra. Boole is best known as the author of The Laws of Thought (1854), which contains Boolean algebra.

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

Instructions

Open **example.js** and **index.html** in VSCode 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 VSCode (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 code reviewer for help.

Compulsory Task 1

Follow these steps:

- Create a JavaScript file called **myVariables.js**.
- Create a variable called “language” and assign it the value “JavaScript”. Then create a variable called “score” and assign it the value “10”. Write both variables to the console.
- Create two variables called “length” and “width”. Assign each variable a value. Use the variables to calculate the area of a rectangle with the length and width specified (remember area = length x width). Write the following to the console: “The area of the rectangle is ...” where ... is the area you calculated.
- Create two variables called “num1” and “num2.” Assign each variable a value. Calculate and display what the remainder is if num1 is divided by num2.
- OPTIONAL: Using the variables you created in the previous step, output the results of the division in the following format: “num1 / num2 = x remainder y.” (Research **Math.trunc** to get this to work.)
- Use Beautify to improve the readability of your code before submitting it to your mentor.

Compulsory Task 2

Follow these steps:

- Create a .js file called **dataTypes.js**.
- Define variables of the following data types, assigning them appropriate values of your choice:
    - Integer
    - String
    - Array
    - Object
- Once you've created the variables, display them to the console.
- Use the 'typeof' function to check each variable.
- Create an array with numbers and strings. Comment what type you expect this to be.
- Use the 'typeof' function on this array.
- Create a null variable and an undefined variable (you'll need to do some research to do this).
- Use Beautify to improve the readability of your code before submitting it to your mentor.

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Completed the task(s)?

Ask an expert to review your work!

Review work

Things to look out for:

1. Make sure that you have installed and set up all programs correctly. You have set up **Dropbox** correctly if you are reading this, but **Visual Studio Code** may not be installed correctly.

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.

SPOT CHECK 1 ANSWERS:

1. `console.log("Hello World");`
2. A variable is a container of information that we give a name to.

SPOT CHECK 2 ANSWERS:

1.
   a. String: Cat
   b. Number: 6.2
   c. Boolean: true

2.
    a. String: `String(VariableYouWantToChange)`
    b. Number: `Number(VariableYouWantToChange)`
    c. Boolean: `Boolean(VariableYouWantToChange)`
3. We can use the `typeof` function.

SPOT CHECK 3 ANSWERS:

1. The modulus operator determines what the remainder would be if you divided one number by another. E.g. 5 % 2 = 1 because 5 divided by 2 equals 2 remainder 1.
2. 4
3. ``` `I type what I want and insert variables like this: ${variable}` ```