TASK

JavaScript Programming, Variables, and Data Types

Visit our website

Introduction

WELCOME TO THE JAVASCRIPT PROGRAMMING, VARIABLES, AND DATA TYPES TASK!

In this task, you will be introduced to JavaScript. This is a scripting language that allows you to create dynamic websites. You will learn what JavaScript is, the basics of creating your first program using variables, and the fundamental data types common to most programming languages as implemented in JavaScript.

PSEUDO CODE

Pseudo code is a detailed yet informal description of what a computer program or algorithm must do. It is intended for human reading rather than machine reading. Pseudo code does not need to obey any specific syntax rules, and so it can be understood by any programmer, irrespective of their programming language proficiencies. Pseudo code will help you to understand better how to implement an algorithm. It will also help you to communicate your ideas to colleagues with different programming language proficiencies to yours.

Take a look at the pseudo code example below that deals with password validation:

Problem: Write an algorithm that asks a user to input a password, and then stores the password in a variable called *password*. Subsequently, the algorithm requests input from the user. If the input does not match the password, it stores the incorrect passwords in a list until the correct password is entered, and then prints out the variable "*password*" and the incorrect passwords:

Pseudo code solution:

```
request input from the user
store input into variable called "password"
request second input from the user
if the second input is equal to "password"
          output the "password" and the incorrect inputs (which should
be none at this point)
if the second input is not equal to "password"
          request input until input matches password
when input input matches "password"
          output "password"
          and output all incorrect inputs.
```

INTRODUCTION TO JAVASCRIPT

JavaScript serves as a versatile scripting language utilised in both front-end web development and server-side programming. When combined with HTML and CSS, JavaScript plays a pivotal role in transforming static web pages into dynamic ones, enabling interactions and real-time changes. Before you start learning to use JavaScript, it is important to first cover some theory and background.

ECMA, an international organisation, establishes standards for information and communication systems. Among its creations is ECMAScript, defining the JavaScript general-purpose programming language.

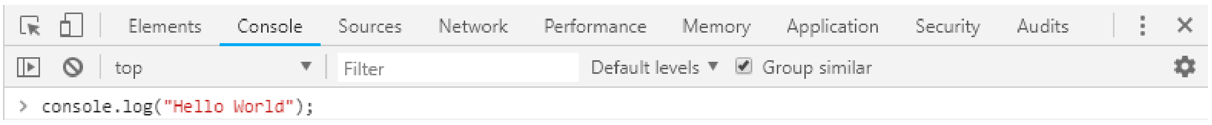
First released in 1997, ECMAScript is built on various technologies, including JavaScript and JScript. Over time, multiple versions and editions of ECMAScript have emerged. The 6th edition, known as ES6 or ES2015, represents the most substantial revision since its inception.

Subsequent releases, such as ES7 and ES8, continue to enhance JavaScript's capabilities. As you embark on your JavaScript coding journey, you may encounter different ways of writing code as a result of some code being written using older JavaScript versions. Your Compulsory Tasks will highlight the updates made to JavaScript since ES6.

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 **right-click → Inspect**, or press either **Ctrl+Shift+J if you are using Windows / Linux** or **Cmd+Opt+J if you are using macOS**. 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 the console tab to test and debug it.

To write information to the console, we use the instruction:

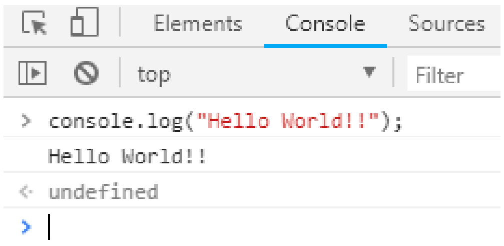
```
console.log("Whatever you would like to write to the console");
```

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

Try this:

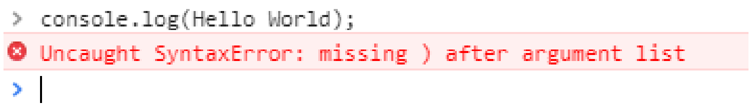
1. Open **https://blank.org/** to access a clean, JavaScript-free page for testing your code.
2. Right-click to open Inspect, and then go to the Console tab.
3. Type the code `console.log("Hello World!!");` into the console.
4. Press enter.

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 you should have used 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.

Code Hack

Extensions to run your code:

Install **Code Runner** and **Open In Browser** from the Extension tab in VS Code. Code Runner allows you to run code snippets, whereas Open In Browser allows you to open HTML files in your default browser, both of which can be done directly within VS Code once you have installed the extensions.

After installing Code Runner, open a JavaScript file and click on the "play" button on the top right corner of the editor window in VS Code (or press Ctrl + Alt + N, or right-click inside the code file and click on "Run Code") to execute the code. Once Open In Browser is installed, open a folder that correctly links a JavaScript file to a related HTML file (both files must be in the same folder), navigate to the Explorer on VS Code (or press Ctrl + Shift + E), right-click the HTML file name in the Explorer, and from the dropdown menu select the option "Open in Default Browser" or "Open in Other Browsers". Note: it is best to open the HTML file using Google Chrome as it is a fast web browser in which to run your code.

SYNTAX RULES

All programming languages have *syntax* rules. Syntax is the "spelling and grammar rules" of a programming language and determines how you write correct, well-formed statements.

A common syntax error is forgetting to add a closing quotation mark (**"**). Remember that all opening quotation marks (**"**) require a closing one! Another common syntax error that you could make is forgetting to add a closing bracket '**)**'. Remember that all opening brackets '**(**' require a matching closing one!

Any program you write must be exactly correct. All code is case-sensitive. This means that *'Console'* is not the same as *'console'*. If you enter an invalid JavaScript command, misspell a command, or misplace a punctuation mark, you will get a syntax error when trying to run your program.

Errors appear in the JavaScript console when you try to run a program which fails. Be sure to read all errors carefully to discover what the problem is. Error reports will even show you what line of your program had an error. The process of resolving errors in code is known as *debugging*.

VARIABLES

A script is basically a set of instructions that are interpreted to tell the computer what to do in order to accomplish a certain task. Programs usually process data that is somehow input into the program, and output the results of the processing. This data must be stored somewhere so that it can be used and processed by the instructions we code. When coding, we use **variables** to store the data we need to manipulate. A variable can be thought of as a type of "container" that holds information. We use variables in calculations to hold values that can be changed/varied (thus "variable").

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 that we can use to reference it. 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 = value_you_want_to_store;
```

1. The first thing you need to do to declare a variable is to use the keyword `let` or `const`. If the value of a variable changes during the execution of a program you generally use `let`. However, if the value remains constant you should use `const` which ensures the variable doesn't accidentally get reassigned to a different value. (Note: the keyword `var` to declare variables is the old way of creating a variable and is no longer an industry standard. Avoid using it.)

2. In the next part of the declaration you can set the name of the variable to anything you like, as long as the name:

    a. contains only letters, numbers, and underscores. All other characters are prohibited from use in variable names, including spaces.

    b. begins with a letter. In JavaScript, the common naming convention used is camelCase where each word except for the first word starts with an uppercase letter.

    c. is not a reserved word. In JavaScript, certain words are reserved. For example, you would not be able to name a variable `console` or `log` because these are reserved words.

    d. is meaningful and concise. For example, "myName" and "userInput" are descriptive variable names as they explain their purpose (to store the name of a person and data entered by the user, respectively). In comparison, “h4x0r”, “x”, or “y” are not descriptive as they do not reveal what content is stored in these variables.

3. To assign a value to a variable, you need to use the assignment operator. This is the equals-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.
4. Finally, we finish the variable-declaration line with a semicolon (**;**).

Try this:

Add the code below to declare and use a variable. This code will instruct your computer to create an area in memory (i.e. a variable) called “`myName`” and store the value “Tom” in that area in memory. Remember, the value stored in the variable “`myName`” has been declared using the `let` keyword and therefore could be changed or reassigned throughout the program. The variable “`myName`” is just an area in memory - you can think of it like a box - intended to hold whatever data you give to it.

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

myName = "John"; // Variable is reused and value is changed
console.log("Hi there " + myName); // Output now returns "Hi there John"
```

We can also ask the user to give us the value for a variable. We do this using the `prompt()` method. The `prompt()` method displays a dialog box (a graphical user interface element that pops up on the screen to gather user input or display information) that prompts the user to enter some information or respond to a question. Dialog boxes are a common method for interacting with users in web applications.

Note: To use the `prompt()` method, you need to create an HTML file that contains the JavaScript code responsible for showing the dialog box.

The code below shows how the JavaScript code is linked to the HTML file using the `<script>` tag. When the browser loads the HTML file, the JavaScript code runs. In this case, it uses the `prompt()` method to display a dialog box with a message asking the user to enter their name, and then logs “Hi there, “ and their name to the console.

```
<body>
    <script>
        // JavaScript code asking the user to enter their name
        let myName = prompt("What is your name?");
        console.log("Hi there, " + myName); // Output: "Hi there, [myName]"
    </script>
</body>
```

DATA TYPES

Variables store data. The type of the data that is stored by a variable is (intuitively) called the data type! Within JavaScript, you'll find yourself working with many data types based on the kind of application you're dealing with. There are five 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 string values that must always be put within quotation marks (`""`). Note that the spaces in "23 Main Street" are part of the string and not treated any differently to the other characters. We use strings to store and manipulate text. In "23 Main Street" the "23" is treated as a text and not a number because it has been input within quotation marks.
- **Booleans** are data types that 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 into 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 created a person object, for example, you could store a collection of related information that describes a person such as their name, surname, date of birth, address, etc.

TYPE IDENTIFICATION

JavaScript is smart, in the sense that it's able to detect each variable's type automatically based on the value that you assign to the variable. If a value is enclosed within quotation marks, JavaScript knows that the value is a string. If a value is not enclosed within quotation marks, JavaScript will determine the data type as either a number, boolean, or object. Sometimes, you may also want to check some data to inspect its data-type property. This can be done by making use of the built-in `typeof` function.

Try this: Enter the following code inside a JavaScript file. Be sure to understand how the code results in the output displayed in the console.

```
let age = 25;
console.log(typeof age); // Output: number

let name = "Tom";
console.log(typeof name); // Output: string

let isLoggedIn = true;
console.log(typeof isLoggedIn); // Output: boolean

// Numeric and String type is automatically converted to a String
let unknownType = 10 + "Chocolates";
console.log(typeof unknownType); // Output: string
```

CASTING TO DIFFERENT TYPES

Sometimes you need to change a variable from one data type to another. For example, when getting user input, JavaScript automatically assumes that the input is always a string. What if the user inputs a a numerical value that you need to convert from a string to a numeric type so as to be able to use the value in calculations?. You can change the variable's data type using a process called "casting". This requires using the `Number()`, `String()`, or `Boolean()` constructor, depending on what we want the final data type to be. Try testing the code below with and without the constructors to observe the difference in results.

```
// Convert from a String to a Number
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
```

Once a variable has been converted to a number, the full range of 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 number0 = 0; // Output: 0 of type number
let string0 = String(number0); // Output: "0" of type string
let boolean0 = Boolean(number0); // Output: false of type boolean

let boolean1 = true; // Output: true of type boolean
let string1 = String(boolean1); // Output: "true" of type string
let number1 = Number(boolean1); // Output: 1 of type number

let string2 = "2"; //Output: "2" of type string
let number2 = Number(string2); // Output: 2 of type number
let boolean2 = Boolean(string2); //Output: true of type boolean
```

MATHEMATICAL CALCULATIONS

Doing calculations with numbers in JavaScript is similar to doing calculations in regular mathematics; the only difference 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)	--

Try this: Copy-paste the following JavaScript in VS Code. Press take careful note of the output.

```
let num1 = 152;
let num2 = 10;

console.log("num1 = " + num1); // num1 = 152
console.log("num2 = " + num2); // num2 = 10
console.log("num1 + num2 = " + (num1 + num2)); // num1 + num2 = 162
console.log("num1/num2 = " + num1 / num2); // num1 / num2 = 15.2
console.log("num1 % num2 = " + (num1 % num2)); // num1 % num2 = 2
console.log("++num1 = " + ++num1); // ++num1 = 153
console.log("--num2 = " + --num2); // --num2 = 9
```

NOTES

- When we are using `++` and `--` it is essential to place them in the correct position. If you are placing the operator after the variable, it will update after the line is completed, whereas if it is placed before the variable, it will affect the current line.
- It is crucial for a programmer to know how to use the **modulus operator %** because it is used to solve many computational problems. Given the above division problem, 152 / 10, the answer can be expressed as 15 remainder 2. The modulus operator returns only the remainder of a division problem. So, the result of the expression `152 % 10` would be 2.

Take note:

In JavaScript, the + operator can be used to add two numbers OR to concatenate a string with another string. To concatenate means to join values together. Consider this line of code from the example above: `console.log("num1+num2="+(num1+num2));`. Here the variables `num1` and `num2` are added and the result is concatenated with the string literal enclosed in quotation marks, producing `num1+num2=162` as output.

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 their position in the string, known as their index. Unlike normal counting, which starts at 1, indexing starts at zero. In the table below, we can see that the string “Hello!” has a maximum index of 5, even though there are 6 characters (remember that spaces and punctuation also count as characters).

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

This is useful because we can access any specific element in any string by using its index, or find the index of an element by using the character:

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

Note that the `length` of a string 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 always be equal to its *length - 1*. We can find the length of a string in the following way:

```
console.log(greeting.length); // returns the number 6
```

MULTI-LINE STRINGS

Strings can be combined, or concatenated, using the addition (+) operator, as you learned in the Take Note box earlier. If we wanted to print a string on separate lines, we could use an escape character, specifically the newline character ‘`\n`’, or template literals. Let’s see what the code for this would look like.

```
let greeting = "Hello";
let name = "Tom";
let age = 25;

// String concatenation and the newline character:
```

```
console.log(
  greeting + "!\nMy name is " + name + ".\nI am " + age + " years old.\n"
);

// Template literals:
console.log(`${greeting}!
My name is ${name}.
I am ${age} years old.`);

// Output for both approaches will be the same:
// Hello!
// My name is Tom.
// I am 25 years old.
```

NOTES

- Concatenation using the + **operator** may create code that is longer to read, especially if many strings are joined together. **Template literals** can provide a more succinct way of doing both concatenation and multiline strings, making the code more readable and easier to maintain. This means using backticks (`) and the format of `${expression}` as seen in the example above.
- Note: when using template literals, we simply place the variable name within the curly brackets. Any characters between the backticks such as spaces, punctuation, and string literals are also printed. Furthermore, we don't need escape characters; if we want a new line, we simply create a new line.

Instructions

To become more comfortable with the concepts covered in this task, read and run the accompanying example files before doing the Compulsory Task.

Compulsory Task

Follow these steps:

- Note: For this task, you will need to create an HTML file to get input from a user. Please refer to the accompanying file entitled **prompt_example.html**.
- 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 prompt the user 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) + (5) is the number of years in which they will get married. *(Note: ensure to convert the string types to numbers using the* `Number()` *constructor, so as to be able to add the values.)*
    d. The remainder of (4) divided by (5) is how many children they will have. *(Refer to the modulus operator* `%` *discussed earlier in the lesson.)*
    e. (4) divided by (5) (rounded off) is how many years until they dye their hair (3). *(Hint: look up* `Math.round()` *.)*
- Using template literals, 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.