HyperionDev

JavaScript Data Types and Conditional Statements

Visit our website

Introduction

In this task, you will be introduced to JavaScript. Fundamentally, it 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 and conditional statements common to most programming languages as implemented in JavaScript. You will also learn about a program's flow control. A control structure is a block of code that analyses variables and chooses a direction in which to go based on given parameters. In essence, it is a decision-making process in computing that determines how a computer responds to certain conditions.

Pseudocode

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

Take a look at the pseudocode 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 (something you will learn more about later in this lesson) called password. For now, just think of a variable as a name for a container to store some information in. Once the user's chosen password has been stored, the algorithm must request input from the user. If the input does not match the password the user originally set, the algorithm must store the incorrect passwords in a list until the correct password is entered. At that point, it must print out the value of the variable 'password' (i.e., the user's chosen password that has been stored in the password variable/container), as well as the incorrect passwords.

Pseudocode 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
          store the non-matching input for later output
when the input matches “password”
          output “password”
          and output all incorrect inputs.
```

Introduction to JavaScript

JavaScript is a versatile scripting language utilised in 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 begin learning to use JavaScript, it is important to first cover some theory and background.

Ecma International establishes standards for information and communication systems. Among its creations is ECMAScript, defining the JavaScript general-purpose programming language.

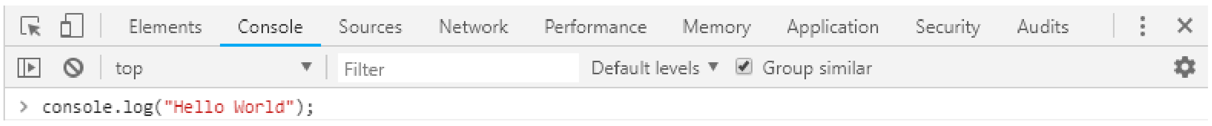
First released in 1997, ECMAScript is built on various technologies, including JavaScript and JScript. Over time, multiple versions and editions of ECMAScript have emerged. The 6th edition, known as ES6 or ES2015, represents the most substantial revision since its inception.

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 due to some code being written using older JavaScript versions. Your tasks will highlight the updates made to JavaScript since ES6.

Using the JavaScript console

All modern browsers offer built-in support for JavaScript (you are 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, but 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:

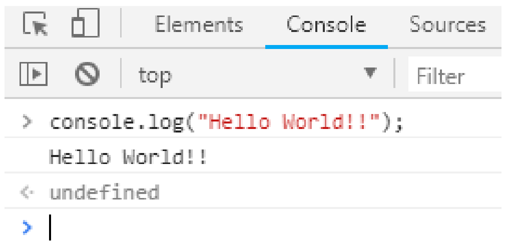
```
console.log("Whatever you would like to write to the console");
```

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:

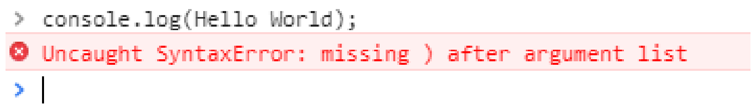
1. Open **blank.org** to access a clean, JavaScript-free page for testing your code.
2. Right-click to open 'Inspect', and then go to the Console tab.
3. Type the code `console.log("Hello World!!");` into the console.
4. Press enter.

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 have not 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 spelt 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 in the top right corner of the editor window in VS Code. Or right-click inside the code file and select Run Code to execute the JavaScript code within the file.

Once the Open in Browser extension is installed, you can use the extension to more easily explore viewing HTML files in your browser. In VS Code, within the Explorer, right-click the HTML file and choose either 'Open in Default Browser' or 'Open in Other Browsers' from the dropdown menu. For the best experience, use Google Chrome, a fast and reliable browser for running your code. However, it is perfectly fine to use another browser if you do not have Google Chrome installed.

Note: Linking JavaScript and HTML files will be covered later, so do not worry if you are not familiar with this concept yet.

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 (**"**). To avoid this, 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 '**C**onsole' is not the same as '**c**onsole'. 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 that 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 a set of instructions that are interpreted to tell the computer what to do to accomplish a certain task. Programs typically receive input data, process it, and then output the results. 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, 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
```

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 does not accidentally get reassigned to a different value. (**Please 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 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: Copy the code below and paste it into your browser's console 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. In the example below, 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 dialogue box (a graphical user interface element that pops up on the screen to gather user input or display information), which prompts the user to enter some information or respond to a question. Dialogue 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 displaying the dialogue box. This is because the method is designed to run in a web browser. Creating an HTML file allows the JavaScript code to be executed, displaying a dialogue box to receive user input.

Integrating JavaScript with HTML

To start using methods like `prompt()`, we first need to create HTML files and learn how to integrate our JavaScript into them. This involves embedding JavaScript directly within HTML or linking to external JavaScript files, allowing us to interact with and manipulate the web page. In this section, we will cover both methods for integrating JavaScript with HTML. First, we will learn how to add JavaScript code directly within HTML files. Then, we'll explore how to use external JavaScript files by linking them.

Adding internal JavaScript to HTML

Internal JavaScript is JavaScript code which has been directly added within a HTML file. The JavaScript code can be added to an HTML file by wrapping the JavaScript code within the `<script>` tag. It is best practice to place the `<script>` tag at the end of the

`<body>` section, just before the closing </body> tag, while still keeping it within the `<body>` section of the HTML structure. Placing the `<script>` tag at the end of the `<body>` section, helps to ensure that all the HTML elements are fully loaded before the JavaScript code is executed.

For example:

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Internal JavaScript Example</title>
  </head>
  <body>
    <h1>Internal JavaScript Example</h1>
    <script>
      let name = prompt("What is your name?");
      alert(`Hello, ${name}!`);
    </script>
  </body>
</html>
```

Explanation:

- `<script>` tag: The `<script>` tag has been used as a way to include JavaScript code to be executed within an HTML file.
- `prompt()`: The `prompt()` method is used to display a dialogue box that asks the user for input. In the provided code, `prompt("What is your name?")` shows a dialogue box asking the user to enter their name. The input is then stored in the variable `name`.
- `alert()`: The `alert()` method has been used to display a greeting message to the user in a pop-up alert box. In this case, ``alert(`Hello, ${name}!`)`` shows a greeting message that includes the user's name, which was saved in the variable `name`.

Linking external JavaScript to HTML

Linking external JavaScript involves placing your JavaScript code in a separate file with a **.js** extension and then linking it to your HTML file. Similar to internal JavaScript, the `<script>` tag is placed within the `<body>` section of your HTML file. However, instead of placing the code directly between the `<script>` tags, you link to the external JavaScript file using the `src` attribute.

To get started with linking an external JavaScript file, follow these steps:

1. To start with, create a new file named **script.js**, this will be used to hold all the JavaScript code you wish to execute.

2. Open the script.js file in your code editor (like Visual Studio Code) and add the following code:

```
let name = prompt("What is your name?");
alert(`Hello ${name}!`);
```

3. Now that we have created the JavaScript file we can create the HTML file. For this example we are going to name the HTML file **index.html**.

4. Open the created HTML file and add the following code. Take note that the `<script>` tag is being used to link the **script.js** file using the src attribute.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>External Linked JavaScript Example</title>
  </head>
  <body>
    <h1>External Linked JavaScript Example</h1>

    <!-- Linking script.js file -->
    <script src="script.js"></script>
  </body>
</html>
```

   - Please note that the **index.html** and **script.js** need to be in the same folder for this example.

5. To run the linked JavaScript code, open the index.html file within your browser.

Internal JavaScript or linked external JavaScript

Now that we have discussed the two main ways to include JavaScript code within your HTML files, let's consider when to use each of the methods. Internal JavaScript is used to add JavaScript code directly within HTML files using the <script> tags. This approach can be useful for smaller blocks of code or for simple functionality. However, as the code grows and becomes more complex, it can be difficult to manage the code when internal JavaScript is used.

Linking an external JavaScript file helps to instead keep your JavaScript code separated from the HTML. This helps to keep the code clean and more organised as a whole. When working on larger projects, or when JavaScript is being reused across multiple HTML files, it is recommended that you use external linked JavaScript files. Storing your JavaScript code in a separate file also makes it easier for you to keep the code updated, as it will only need to be updated in one place.

Data types

Variables store data. The type of data that is stored by a variable is (intuitively) called the data type. Within JavaScript, you will find yourself working with many data types based on the kind of application you are 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 (either `""` or `''`). Note that the spaces in "23 Main Street" are part of the string and are not treated any differently from the other characters. Strings are used 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 is used to store multiple values. As shown in the table above, you put all the values you 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 may 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 numerical value that you need to convert from a string to a numeric type to be able to use the value in calculations? You can change the variable's data type using a process called 'casting'.

Casting 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; // Stores the value 0 of type number
let string0 = String(number0); // Stores the value "0" of type string
let boolean0 = Boolean(number0); // Stores the value false of type boolean

let boolean1 = true; // Stores the value true of type boolean
let string1 = String(boolean1); // Stores the value "true" of type string
let number1 = Number(boolean1); // Stores the value 1 of type number

let string2 = "2"; // Stores the value "2" of type string
let number2 = Number(string2); // Stores the value 2 of type number
let boolean2 = Boolean(string2); // Stores the value 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 and paste the following JavaScript into your code editor (e.g. VS Code). Pay close attention to 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
```

A few things to note:

- 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 learnt 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 one, indexing for most programming languages generally starts at zero. In the table below, we can see that the string "`Hello!`" has a maximum index of five, even though there are six characters (keep in mind 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]; // Stores the letter 'o'
let index = greeting.indexOf("e"); // Stores the number 1

console.log("Original string: + greeting");
console.log("The letter at index 4 is: " + letter);
console.log("The index of the first 'e' is: " + index);
```

Note that the **length** of a string is not the same as the i**ndex**. The length is the number of elements counting from one. 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); // Outputs the number 6
```

Multiline strings

Strings can be combined, or concatenated, using the addition (+) operator, as you learnt in the Take Note section 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.
```

A few things to note:

- 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 go to the next line in our code as shown above.

If statements

We are now going to learn about a vital concept when it comes to programming. We will be teaching the computer how to make decisions for itself using an *if statement*. As the name suggests, this is essentially a question, a way of comparing two or more variables or scenarios and performing a specified action based on the outcome of that comparison.

If statements contain a **condition**. In the example below, the condition is the part of the *if statement* in round brackets. Conditions are statements that can only be evaluated as true or false. If the condition is true, then the indented statements are executed. If the condition is false, then the indented statements are skipped. As such, *if statements*, and the *else* and *else if* constructs you are going to learn about soon, are all what we call ‘conditional statements’.

In JavaScript, *if statements* have the following general syntax:

```
if (condition) {
  // Code to execute if the condition is true
}
```

Here’s an example of a JavaScript *if statement*:

```
let num = 10;

// Checks if the number is less than 12
if (num < 12) {
  console.log("The variable num is lower than 12");
}
```

This *if statement* checks whether the variable number is less than 12. If it is, then the *if statement* will output the sentence letting us know. If *num* was greater than 12, then the *if statement* would not output that sentence.

Notice the following important syntax rules for an *if statement*:

- In JavaScript, the opening curly bracket is followed by code that will only run if the statement's condition is true.
- The closing curly bracket shows the end of the statement that will execute if the statement’s condition is true.

Comparison operators

You may have also noticed the less than (<) symbol above. As a programmer, it is important to remember the basic logical commands. We use comparison operators to compare values or variables in programming. These operators work well with *if statements* and *loops* to control what goes on in our programs.

Operator	Description	Example
>	greater than	Condition: 12 > 1 Result: True
<	less than	Condition: 12 < 1 Result: False
>=	greater than or equal to	Condition: 12 >= 1 Result: True

`<=`	less than or equal to	Condition: 12 <= 12 Result: True
`==`	equals	Condition: 12 == 1 Result: False
`===`	Equal value and equal data type	Condition: 12 === “12” Result: False
`!=`	does not equal	Condition: 12 != 1 Result: True

Take note that the symbol we use in conditional statements to check if the values are the same is ‘==’, and not ‘=’. This is because ‘==’ literally means ‘equals’ (e.g., `i == 4` means *i* is equal to 4). On the other hand, ‘=’ is used to assign a value to a variable (e.g., `i = “blue”` means we assign the value of ‘*blue*’ to *i*). This is a subtle but important difference.

Similarly, we use ‘===’ in conditional statements to check if the value and type are the same, for example, `5 === 5` would return `true` because both operands are numbers that have the same value of 5. In comparison, `5 === "5"` would return `false` because the first operand is a number and the second operand is a string, so they have different data types.

Comparing boolean values

When making a comparison to a boolean variable, it is unnecessary to use == or === to check if the value stored within the boolean variable is true. This is because the if statement in JavaScript automatically evaluates the boolean variable directly to check if the value is **truthy** or **falsy**.

For example:

```
isOldEnough = true;

if (isOldEnough === true) {
  console.log("You are old enough to drive!");
}

if (isOldEnough) {
  console.log("You are old enough to drive!");
}
```

Explanation:

In the above example, the isOldEnough variable stores the value of true. As a result there are two approaches which you could take to check if the value stored within the variable is indeed true.

1. For the first approach, if (`isOldEnough === true`), the conditional statement checks if the value is equal to true. While this works, it is redundant because the isOldEnough variable is a boolean.
2. On the other hand, if (`isOldEnough`), directly evaluates the boolean value stored within the variable. This takes advantage of JavaScript's ability to automatically check if isOldEnough is truthy or falsy.

Logical operators

Sometimes you need to use logical operators: **AND** (&&), **OR** (||), or **NOT** (!). The AND and OR operators can be used to combine comparison expressions. The resulting boolean expression will be evaluated as either true or false. For example, consider the code below:

```
let num = 12;

// Checks if the number is greater than or equal 10 and less than or equal 15
if (num >= 10 && num <= 15) {
  console.log(num + " is a value between 10 and 15");
}
```

The boolean expression (`num >= 10 && num <= 15`) is true if both `num >= 10` and `num <= 15` are true. This is a conjunction, meaning both conditions must be true for the statement to be true. If either condition is false, the statement is false.

For the OR operator, consider this example: You can buy a nice car if you have enough money, if someone gives you the money, or if you can get a loan.

```
let lotsOfMoney = false;
let receivedGift = false;
let loanApproved = true;

// Check if any condition allows purchasing a car
if (lotsOfMoney || receivedGift || loanApproved) {
  console.log("Can purchase a car");
}
```

A disjunction operation requires at least one condition to be true for the whole statement to be true. For example, the boolean expression `(lotsOfMoney || receivedGift || loanApproved)` is true if at least one condition is true.

The NOT operator checks if a condition is false. For example, to see if a grocery store is closed when it's open from 8:00 to 19:00, you would use the NOT operator to indicate times outside this range.

```
let time = 12;

if (time > 8 && time < 19) {
  console.log("The grocery store is Open")
}
```

We can rewrite this above code with the NOT operator to show when the store is closed.

```
let time = 7;

if (!(time > 8 && time < 19)) {
  console.log("The grocery store is Closed")
}
```

Try this:

- Open your code editor (e.g. Visual Studio Code).
- Copy and paste the code above into a new JavaScript file.
- Execute the code and take note of the output. Make sure you understand why your code produced the output it did.
- Modify the code and see how your changes affect the output.

Take note

Keep in mind the difference between operands and operators. The operands represent the data stored in the variables which are being manipulated by the operators. For example, when adding two numbers, the numbers are the operands whereas '+' is the operator. Operators (+, -, *, etc.) tell us how the operation is being performed, whereas the operands enable us to apply the operation to something. You may refer to this **MDN resource** that provides further insight into JavaScript expressions, operators, and operands.

Else statements

The *else statement* represents an alternative path for the flow of logic if the condition of the *if statement* turns out to be *false*.

Imagine: you ask your friend to buy chocolate and they can't find any because you didn't mention alternatives, they would need to return each time for more instructions. In programming, an 'else' statement provides a default action when none of the 'if' conditions are met, avoiding the need for multiple checks.

In JavaScript, the general if-else syntax is:

```
if (condition) {
  // Code to execute if the condition is true
} else {
  // Code to execute if the condition is false
}
```

If the condition turns out to be *false*, the statements in the indented block following the *if statement* are skipped, and the statements in the indented block following the *else statement* are executed.

Take a look at the following example:

```
let num = 10;

if (num <= 12) {
  console.log("The variable num is lower than 12");
} else {
  console.log("The variable num is greater than or equal to 12");
}
```

Now instead of nothing happening when the condition of the *if statement* is *false* (e.g., `num` is greater than or equal to 12), the else statement will be executed.

Another example of using an *else statement* with an *if statement* can be found below. The value that the variable `hour` holds determines which string is assigned to the `greeting`.

```
let hour = 10;
let greeting;

if (hour < 18) {
  greeting = "Good morning";
} else {
  greeting = "Good evening";
}
```

```
console.log(greeting);
```

We are faced with decisions like this daily. For instance, if it is cold outside you would likely wear a jacket. However, if it is not cold you might not find a jacket necessary. This is a type of branching logic. If one condition is true, do one thing, and if the condition is false, do something else. This type of branching decision-making can be implemented in JavaScript using *if-else statements*.

Else if statements

The *else if statement* allows you to test multiple conditions in a single conditional structure. If the initial if condition is *false*, each *else if* is checked in sequence. If none are *true*, the final *else statement* is executed.

The syntax for *if*, *else if*, and *else statements* in JavaScript is as follows:

```
if (condition) {
  // Code to execute if the condition is true
} else if (condition2) {
  // Code to execute if the condition2 is true
} else {
  // Code to execute if all conditions are false
}
```

Look at the following example:

```
let num = 10;

if (num > 12) {
  console.log("The variable num is greater than 12");
} else if (num > 10) {
  console.log("The variable num is greater than 10");
} else if (num < 10) {
  console.log("The variable num is less than 10");
} else {
  console.log("The variable num is 10");
}
```

Note how you can combine *if*, *else if* and *else* into one big statement.

Some important points to note on the syntax of *if*, *else if*, and *else statements*:

- Make sure that each indented statement starts with an opening curly bracket and ends with a closing curly bracket for *if*, *else if*, and *else statements*.

- Ensure that your indentation is correct (i.e., statements that are part of a certain control structure's 'code block' need the same indentation).
- To have an *else if* you must have an if above it.
- To have an *else* you must have an *if* or *else if* above it.
- You cannot have an *else* without an *if* – think about it!
- You can have many *else if statements* under an *if*, but only one *else* right at the bottom. It is like the fail-safe statement that executes if the other *if* or *else if statements* fail!

Nested if statements

We can also nest an *if statement* inside another *if statement*:

```
// Outer if statement
if (condition1) {
  // Code block for outer if statement
  if (condition2) {
    // Code block for inner if statement
  } else {
    // Code block for inner else statement
  }
} else {
  // Code block for outer else statement
}
```

Look at the following example:

```
let num = 12;

if (num > 10) {
  // Nested if statement
  if (num % 2 == 0) {
    console.log("num is larger than 10 and an even number.");
  } else {
    console.log("num is larger than 10 and an odd number.");
  }
} else {
  console.log("num is smaller than or equal to 10.");
}
```

Instructions

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

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give the task your best attempt and submit it when you are ready.

After you click "Request review" on your student dashboard, you will receive a 100% pass grade if you've submitted the task.

When you submit the task, you will receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer. Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey for this task, which you can use as a self-assessment tool. Please take a moment to complete the survey.

Once you've done that, feel free to progress to the next task.

Auto-graded task 1

Follow these steps:

- For this task, you will need to create an HTML file to get input from a user. Please refer to the accompanying file titled **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

- Note: The numbers in parentheses below (e.g., (1), (2)) refer to each of the corresponding prompts above.
- With this information, you can work out the following using each of the numbered prompts:
    - (5) is the number of years in which they will meet their best friend.
    - Their best friend’s name will be (1) + (2) (concatenation of the mother's first name and the street name).
    - (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 to be able to add the values.
    - The remainder of (4) divided by (5) is how many children they will have.
        - Note: Refer to the modulus operator `%` discussed earlier in the lesson.
    - (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:

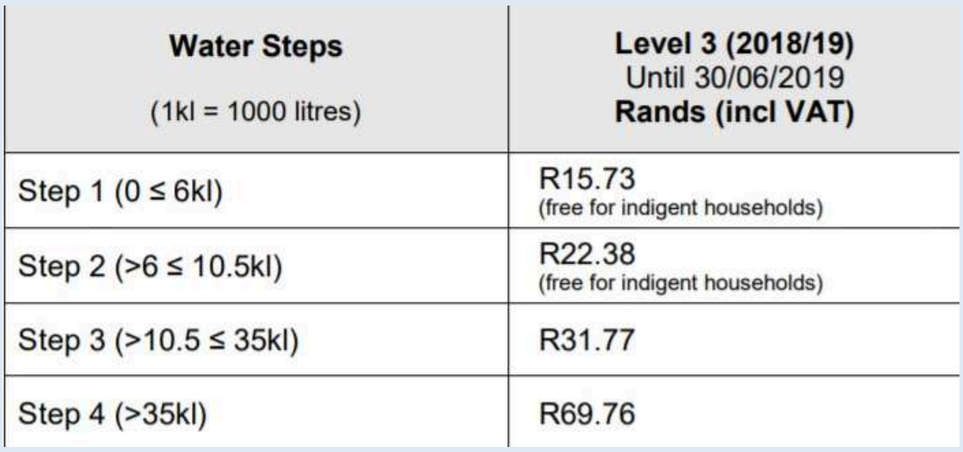
```
In 7 years you are going to meet your best friend named Mary
Washington.
You will get married in 4 years and have 6 children.
In 20 years you are going to dye your hair blue.
```

Be sure to place files for submission inside your task folder and click "Request review" on your dashboard.

Auto-graded task 2

Follow these steps:

- For this task, you will need to create an HTML file to get input from a user.
- Create a JavaScript file called **waterTariffs.js**. Your task is to create a calculator to determine a user's water bill.
- These are the Level 3 water tariffs for the City of Cape Town (**Source**):



Water Steps (1kl = 1000 litres)	**Level 3 (2018/19)** Until 30/06/2019 **Rands (incl VAT)**
Step 1 (0 ≤ 6kl)	R15.73 (free for indigent households)
Step 2 (>6 ≤ 10.5kl)	R22.38 (free for indigent households)
Step 3 (>10.5 ≤ 35kl)	R31.77
Step 4 (>35kl)	R69.76

- The table above states that the first 6,000 litres will cost R15.73 per kilolitre. Next, water consumption above 6,000 litres but less than or equal to 10,500 litres will be charged at R22.38 per kilolitre. Therefore, a household that has used 8,000 litres will pay R139.14 ((15.73 x 6) + (22.38 x 2)).
- The calculator should ask the user to input the number of litres of water they have used.
- Once the user has provided the number of litres of water that has been used, they should be prompted to ask if they are an indigent household, as steps 1 and 2 are free for indigent households.
- Finally, after performing the necessary calculations based on the user's input, the total amount in rands (R) that the user needs to pay should be displayed.

Be sure to place files for submission inside your task folder and click "Request review" on your dashboard

Share your thoughts

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

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

Share your thoughts anonymously using this **form**.