
TASK

# Control Structures - If, Else-if, and Else Statements

Visit our website

# Introduction

## WELCOME TO THE CONTROL STRUCTURES - IF, ELSE, AND ELSE-IF STATEMENTS TASK!

In this task, you will 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.

## 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. The condition is the part of the *if statement* in brackets in the example below. 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) {
   indented statements;
}
```

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

```
let num = 10;

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* were 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's 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<br>Result: True |
| `<` | less than | Condition: 12 < 1<br>Result: False |
| `>=` | greater than or equal to | Condition: 12 >= 1<br>Result: True |
| `<=` | less than or equal to | Condition: 12 <= 12<br>Result: True |
| `==` | equals | Condition: 12 == 1<br>Result: False |
| `===` | Equal value and equal type | Condition: 12 === “twelve”<br>Result: False |
| `!=` | does not equal | Condition: 12 != 1<br>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.

## 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 evaluate as either true or false. For example, consider the code below:

```
let num = 12;
if (num >= 10 && num <= 15){
   console.log(num + " is a value between 10 and 15");
}
```

The boolean expression (`num >= 10 && num <= 15`) evaluates to true in the example above because both the expression `num >= 10` AND the expression `num <= 15` are true. This is an example of a *conjunction operation* where both conditions need to be true for the whole statement to be true. In other words, if one of the conditions is false, the whole statement becomes false.

To illustrate how the OR operator is used, consider a real-life example: you could buy a very nice car if you have enough money OR if someone gives you the money as a gift OR if you can get a loan.

```
let loanApproved = true;

if (lotsOfMoney || receivedGift || loanApproved){
    console.log("Can purchase a car");
}
else {
    console.log("Sorry! Can't afford a car");
}
```

This is a *disjunction operation* where at least one of the conditions needs to be true for the whole statement to be true. The boolean expression (`lotsOfMoney ||`

`receivedGift || loanApproved`) is true because at least one of the expressions in the compound expression is true.

Now let's look at the NOT operator. In the example below we are testing whether a grocery store is open. If the time is after 8:00 AND before 19:00 then the store is open.

```
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 the JavaScript Console.
- Copy and paste the code above into the console.
- Execute the code and take note of the output. Make sure you understand why your code produced the output it did.

**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** which provides further insight about JavaScript expressions, operators, and operands.

## ELSE STATEMENTS

*If statements* are one of the most important concepts in programming, but on their own, they are a bit limited. 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 are craving something sweet. You decide on chocolate and your friend offers to go to the shop to buy it. When they get to the shop, they find no chocolate in stock and leave because you didn't mention any alternatives. They would have to keep coming back for instructions every time they didn't find what you wanted unless you provided them with an alternative. Similarly, instead of us having many 'if' statements to test each scenario, we can add an 'else' statement to give us a single alternative.

In JavaScript, the general if-else syntax is:

```
if (condition) {
    indented statements;
}
else {
    indented statements;
}
```

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

We are faced with decisions like this on a daily basis. 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 last piece of the puzzle when it comes to branching logic is *else-if statements*. With this statement, we can add more conditions to the *if statement*, making it possible to test multiple parameters in the same statement.

Unlike with the *else statement*, you can have multiple *else-if* statements in an *if/else-if/else statement*. If the condition of the *if statement* is *false*, the condition of the next *else-if* statement is checked. If the first *else-if statement* condition is also *false*, the condition of the next *else-if statement* is checked, etc. If all the *else-if* conditions are *false*, the *else statement* and its indented block of statements are executed.

In JavaScript, *if/else-if/else statements* have the following syntax:

```
if (condition1) {
      indented statements;
}
else if (condition2) {
      indented statements;
}
else if (condition3) {
      indented statements;
```

```
}
else if (condition4) {
      indented statements;
}
else {
      indented statements;
}
```

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, and else-if into one big statement.

**Some important points to note on the syntax of *if/else-if/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/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 can't 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's like the fail-safe statement that executes if the other *if/else if statements* fail!

## NESTED IF STATEMENTS

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

```
if (condition1){
    indented statements;

    // Nested if statement
    if (condition2) {
        indented statements;
    }
}
else{
    indented statements;
}
```

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

Open **control_structures_example.js** in Visual Studio Code and read through the comments, to become more comfortable with the concepts covered, before attempting this Compulsory Task.

## Compulsory Task 1

Follow these steps:

- *Note: For this Compulsory 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**<br>(1kl = 1000 litres) | **Level 3 (2018/19)**<br>Until 30/06/2019<br>**Rands (incl VAT)** |
|---|---|
| Step 1 (0 ≤ 6kl) | R15.73<br>(free for indigent households) |
| Step 2 (>6 ≤ 10.5kl) | R22.38<br>(free for indigent households) |
| Step 3 (>10.5 ≤ 35kl) | R31.77 |
| Step 4 (>35kl) | R69.76 |

- The table above states that the first 6000 litres will cost R15.73 per kilolitre. Next, water consumption above 6000 litres but less than or equal to 10500 litres will be charged at R22.38 per kilolitre. Therefore, a regular household that has used 8000 litres will pay R139.14 ((15.73 x 6) + (22.38 x 2)). An indigent household would pay R0 since they receive the first 2 tiers of the tariff for free.
- The calculator should ask the user to input the number of litres of water they have used. It should also ask the user if they are an indigent or regular household. Based on the user's input the calculator should output the total amount in Rands (R) that they need to pay.

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.