TASK

Javascript II:

Conditional Statements and Looping

Visit our website

Introduction

Welcome to The Second Javascript Task!

This task will introduce you to 2 types of JavaScript control structures: *if statements* and *loops*. These are important structures that can be used to greatly enhance the functionality of the code you write. 'If' statements allow your code to make decisions and loops allow your code to repeat instructions.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor 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!

CONTROL STRUCTURES IN JAVASCRIPT

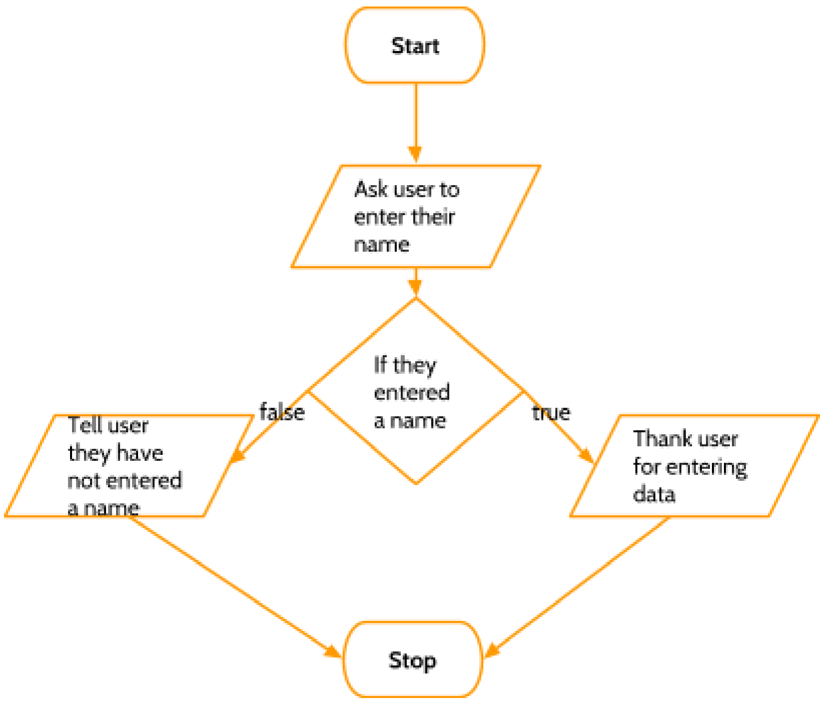
Usually, a program is executed one instruction at a time in order. The *control structures* within JavaScript allow the flow of your program to change within a unit of code or function. These statements can determine whether or not given statements are executed or they can repeat the execution of a block of code.

DECISION OR SELECTION CONTROL STRUCTURE:

Decision-making is a basic building block for all applications. Most programs do not simply go through a set of instructions, like a recipe. There is a higher logic to the code that takes into account various options and possible divergence in the execution of a program. In this task, we will learn how to write programs that can make decisions.

THE IF CONDITIONAL

Often you will want to perform different actions based on different conditions. In order to do this, you will need to use a conditional statement. The **if statement** is an example of a conditional statement. As the name suggests, it is essentially a question. It is able to compare two or more variables or scenarios and perform a certain action based on the outcome of that comparison. An example is shown in the flowchart below.


Start
Ask user to enter their name
If they entered a name
false
true
Tell user they have not entered a name
Thank user for entering data
Stop

Below is an example of an **if** statement in JavaScript. A condition, (a === b), is evaluated and, if it is true, then the block of code in the braces below will execute.

```
if (a === b)
{
        Do something
}
```

An if statement has two parts to it:

1. The condition in brackets which can either be true or false and is checked to determine whether to execute the statements in the body of the ‘if’ statement.
2. The body of code that contains statements is enclosed by curly brackets/braces ({ }) which are executed if the condition evaluates to true. If the condition evaluates to false, then the statements are ignored and the statement directly after the if statement is executed.

Note that three equals signs are used in variable comparison for JavaScript. The symbol === is a boolean operator. Boolean operators are similar to the normal operators that we encountered in prior lessons, such as +, -, * and /. The difference between the mathematical operators and boolean operators is that boolean operators evaluate to a value of True or False, and not a numeric value. The table below shows some more boolean or comparison operators.

COMPARISON OPERATORS

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 type	Condition: 12 === twelve Result: False

!=	does not equal	Condition: 12 != 1 Result: True

THE IF, ELSE, IF, ELSE CONDITIONAL

You are able to add an **else** statement to the end of an **if** statement. The **if** statement and **else** statement are together referred to as an **if-else** statement. If the condition tested in the **if** statement is true, then the statements in the body of the **if** statement are executed and the statements in the body of the **else** statement are ignored. If the condition is false, the statements in the body of the **if** statement are ignored and the statements in the body of the **else** statement are executed. Consider the example below. Notice how you can use ***else if*** and ***else*** statements.

```
if (a === b)
{
        Do something
}
else if (a <= b)
{
        Do something else
}
else
{
        Do this by default
}
```

Try this:

- Open the JavaScript Console. As you learned in the previous tasks, you do this by opening Chrome and then pressing either Ctrl+Shift+J if you are using Windows / Linux or Cmd+Opt+J if you are using Mac.
- Copy and paste the code below into the console.

```
var heightInMeters = 176;

if (heightInMeters < 90){
        console.log("You are too short for this ride");
}
else {
        console.log("Enjoy the ride!");
}
```

- Execute the code and take note of the output. Make sure you understand why your code produced the output it did.

LOGICAL OPERATORS

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

```
var 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. The symbol for the AND operator is &&.

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.

```
var lotsOfMoney = false;
var receivedGift = false;
var 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 that make up that compound expression is true. The || symbols are used for the OR operation.

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.

THE SWITCH CONDITIONAL

The logic of an **if**, **else if**, **else** statement can be implemented using a **switch** statement instead. This structure allows multiple conditions to be evaluated with a much more efficient and structured layout than many ***else if*** statements. The syntax of the **switch** statement is given below:

```
switch(expression)
{
        case 1:
                Do something;
                break;
        case 2:
                Do something else;
                break;
        default:
                Else do this;
                break;
}
```

See the example of the **switch** statement in action below. The statement below uses the *getDay* function to get the current day of the week. If Date().getDay() == 0, then the variable ‘day’ is assigned the value “Sunday”, else if Date().getDay() == 1, then the variable ‘day’ is assigned the value “Monday” etc.

```
var day = new Date().getDay();
/*"new Date().getDay()" is built-in JavaScript code. To see more about how this works,
visit this site:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Dat
e/getDay */
console.log("The value of day is " + day);
switch (day) {
  case 0:
        day = "Sunday";
    break;
  case 1:
        day = "Monday";
    break;
  case 2:
        day = "Tuesday";
    break;
  case 3:
        day = "Wednesday";
    break;
  case 4:
        day = "Thursday";
    break;
  case 5:
```

```
            day = "Friday";
        break;
    case 6:
            day = "Saturday";
}
console.log("Today is " + day);
```

Ty 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.

REPETITION OR LOOPING CONTROL STRUCTURES:

We have seen a number of statements that allow us to write code that reacts differently, based on whether a condition is true or not. We now consider control structures that allow you to repeat sections of code.

THE WHILE CONDITIONAL

This structure will repeatedly execute a block of code for as long as the evaluation condition is met. For example, consider the code below. The instructions contained in the braces { }, will be repeated for as long as (or *while*) the variable 'a' is less than or equal to 10. As soon as 'a' is no longer less than or equal to 10 (i.e. as soon as 'a' equals 11), the instructions will no longer be repeated. This repetition of code is known as ***looping***.

```
var a = 1;
while (a <= 10)             //This loop prints out numbers up until 10
{
            a++;
            console.log(a);
}
```

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.

Be careful when using loops that you don't create infinite loops. An ***infinite loop*** is a loop that will never stop because the condition never becomes false. For example, if you removed the statement a++; (which you will recall from the previous task instructs the computer to add 1 to the variable a) from the example above, what would happen? Your program would just print 1 over and over again because 'a' would never become bigger than or equal to 10.

THE DO WHILE CONDITIONAL

This structure has the same functionality as the while loop, with the exception of being guaranteed to iterate at least once. This is useful if you want your program to do something before variable evaluation actually begins.

```
var a = -10;
do {
   console.log("I've run at least once!");
   a++;
} while (a <= 1);
console.log("The result of a is " + a);
```

THE FOR LOOP

This loop has very similar functionality to a while loop. You will notice that the for loop in the example below actually does all the same things as the first while loop we looked at previously, just in a different format. Compare the for loop below with the first while loop we looked at previously and see the three steps they both have in common:

1. Declare a counter/control variable. The code below does this when it says "i = 1;". This creates a variable called i that contains the value 1. Something similar was done with the while loop with the code "var a = 1".
2. Increase the counter/control variable in the loop. In the for loop below, this is done with the instruction "i++" which increases i by one with each pass of the loop. Similarly, the while loop did this with the code a++.
3. Specify a condition to control when the loop will end. The condition of the for loop below is "i <= 10". This loop will carry on executing as long as i is less than or equal to 10. This loop will, therefore, execute 10 times. The condition of the while loop was "a <= 10".

```
for (i = 1; i <= 10; i++)
{
        console.log(i);
}
```

Compare the syntax of each statement. Remember to make sure you use the correct syntax rules for each statement as you code using loops.

WHICH LOOP TO CHOOSE?

How do we know which looping structure - the **for loop** or the **while loop** - is more appropriate for a given problem? The answer lies with the kind of problem we are facing. After all, both these loops have the four components that were introduced to you. Namely:

- Initialisation of control variable
- Termination condition
- Update control variable
- Body to be repeated

A **while** loop is generally used when we don't know how many times to run through the loop. Usually, the logic of the solution will decide when we break out of the loop, and not a count that we have worked out before the loop has begun. For those situations, we usually use the **for** loop.

For example, if we want to create a table with ***ten rows*** comparing various rand amounts with their equivalent dollar amount, then we would use a **for** loop because we know that it must run ten times. However, if we want to determine how many perfect squares there are below a certain number entered by a user, then we would want to use a **while** loop because we cannot tell how many times we would have to run through the loop before we found the solution.

Instructions

Open *all* the examples for this task in Sublime Text 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 Sublime Text (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 your mentor for help.

Compulsory Task 1

Follow these steps:

- Create a .js file called 'controlStructures.js'. You don't have to create an HTML page for this task. You can write all your output to the console.
- Declare three variables called "num1", "num2" and "num3". Assign each variable a number value.
- Now write a function that compares "num1" and "num2" and displays the larger value.
- Write a function that determines whether "num1" is odd or even and displays the result. (Tip: remember the modulus operator %)
- Next, write a JavaScript conditional statement to sort the three numbers from largest to smallest.
- Write a JavaScript *for loop* that will display the numbers 0 - 20.
- Now write a JavaScript *while loop* that will display the numbers 0 - 20.
- Next, create a JavaScript function that will display all even numbers between 1 and 20.
- Create a JavaScript function that will produce the following output:

```
*
**
***
****
*****
```

Use nested loops (i.e. a loop within a loop). *Tip: Refer to the example of nested loops in example.js if you need help to see how a nested loop works.*

- Write a JavaScript function to compute the greatest common divisor (GCD) of two positive integers.

Once you have completed the task in line with the instructions above, click the button below to request your mentor to review your work and provide feedback. If you have any questions while attempting the task, leave your mentor a note on the comments.txt file in your Student Dropbox folder.

Completed the task(s)?

Ask your mentor review your work!

Review work

Things to look out for:

1. Make sure that you have installed and setup all programs correctly. Follow the instructions in the FAQs document that accompanies this task to configure Sublime Text for debugging and executing JavaScript.

2. If you are not using Windows, please ask your mentor for alternative instructions if needed.

Rate us

Share your thoughts

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

Think 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.