TASK

Beginner Control Structures — WHILE LOOPS

Visit our website

Copyright © 2021 HyperionDev. All rights reserved.

Introduction

WELCOME TO THE BEGINNER CONTROL STRUCTURES — WHILE LOOPS TASK!

In this task, you will be exposed to *loop structures* to understand how they can be utilised in reducing lengthy code, preventing coding errors, and paving the way for code reusability. This task begins with the **while loop**, which is the simplest loop in the group. You will then look at **for loops** and how loops can be nested within one another to solve more complex problems.

Get in touch

Connect for support

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

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

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

Copyright © 2021 HyperionDev. All rights reserved.

GET INTO THE LOOP OF THINGS

Loops are handy tools that enable programmers to do repetitive tasks with minimal effort. To count from 1 to 10, we could write the following program in which each line outputs a consecutive number to the console:

```
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
console.log(6);
console.log(7);
console.log(8);
console.log(9);
console.log(10);
```

The task will be completed correctly. The numbers 1 to 10 will be printed, but there are a few problems with this solution:

- **Efficiency:** repeatedly coding the same statements takes a lot of time.
- **Flexibility:** what if we wanted to change the start number or end number, or the amount by which the numbers incremented each time? We would have to go through and change them individually, adding extra lines of code where they're needed.
- **Scalability:** 10 repetitions are trivial, but what if we wanted 100 or even 100000 repetitions? The number of lines of code needed would be overwhelming and very tedious for a large number of iterations.
- **Maintenance:** where there is a large amount of code, the programmer is more likely to make a mistake. When updates are necessary, these take a lot of time and effort.
- **Feature:** the number of tasks is fixed and doesn't change at each execution.

Using loops, we can solve all these problems. Once you get your head around them, they will be invaluable in solving many problems in programming!

Consider the following code.

```
let i = 1;                 //create a counter variable
while (i <= 10) {           // determine how many times the loop with execute
    console.log(i);        //print the value of i to the console
    i++;                   // shorthand for i = i + 1
```

HyperionDev Copyright © 2021 HyperionDev. All rights reserved.

```
}
```

If we run the program, the same result is produced, but looking at the code, we immediately see the advantages of loops. Instead of executing 10 different lines of code, line 4 executes 10 times. 10 lines of code have been reduced to just 4. This is done through the update statement in line 4, which adds 1 to variable **`i`** each iteration until **`i == 10`**. We may change the number 10 to any number we like. Try it yourself, replace the 10 with another number - you can easily control how many times the loop executes, and thus how high a count you want output to the console. Similarly, if you changed the value by which **`i`** increments every time, you could change the step by which the count increases - setting **`i`** to increment by 2 to count in 2s, etc.

Now, note the first number that is output, and relate that back to the code. How could you rearrange the code, *without changing the initial value of* **`i`**, to output the numbers 0 to 9?

WHAT IS A WHILE LOOP?

All the examples you've been looking at have shown **while** loops. A while loop is the most general form of a loop statement. The while statement repeats its action until the controlling condition becomes false (in the example above, the controlling condition was **`i <= 10`**). In other words, the statements indented in the loop repeatedly execute "while" the condition is true (hence the name). The while statement begins with the keyword **`while`** followed by a boolean expression. The expression is tested before beginning each iteration or repetition. If the test is true, then the program passes control to the indented statements in the loop body; if the test is false, control passes to the first statement after the loop body.

Syntax:

```
while (boolean expression) {
   statement(s);
}
```

The following code shows a while statement which sums successive even integers (2 + 4 + 6 + 8 + ...) until the total is greater than 250. An update statement increments **`i`** by 2 so that it becomes the next even integer. This is an event-controlled loop (as opposed to counter-controlled loops, like the *for loop*)

HyperionDev Copyright © 2021 HyperionDev. All rights reserved.

because iterations continue until some non-counter-related condition (event) stops the process.

```
let sum1 = 0
let i = 2                 // initial even integer for the sum
while (sum1 <= 250) {
    console.log(sum1);
    sum1 += i;
    i += 2;               // update statement, shorthand for i = i + 2
}
```

INFINITE LOOPS

A while loop does, however, run the risk of being "non-terminating" - in other words, running forever - if the controlling condition never becomes false. A loop that never ends is called an **infinite loop**. Creating an infinite loop will mean that your program will run indefinitely while never in fact moving to any code below the while loop - not a desirable outcome! Make sure that your loop condition does eventually become false and that your loop is exited.

Consider the following code:

```
let a = 0;
while (a < 10) {
    a++;                  // shorthand for a = a + 1
    console.log(a);
}
```

In any loop, the following three steps are usually used:

1. **Declare a counter/control variable.** The code above does this when it says `a = 0;` This creates a variable called **a** that contains the value zero.
2. **Increase the counter/control variable in the loop.** In the loop above, this is done with the instruction `a++` which increases **a** by one (*increments* **a**) with each pass of the loop.
3. **Specify a condition to control when the loop ends.** The condition of the while loop above is `a < 10`. This loop will carry on executing as long as `a` is less than ten. The loop will, therefore, execute 10 times. Would there be a difference in the output if the control variable, **a**, was incremented after the console.log statement? If so, what would the difference be?

THE DO WHILE CONDITIONAL

Copyright © 2021 HyperionDev. All rights reserved.

This structure has the same functionality as the while loop, except that it is guaranteed to iterate at least once. This is useful if you want your program to do something before the variable evaluation begins!

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

Considering the code above, how many times will the statement "I've run at least once" be output? What will the value of that output be? What will the last statement output as the value of **a**?

The Break Statement

A loop could also contain a **break statement**. Within a loop body, a break statement causes an immediate exit from the loop to the first statement after the loop body. The break allows for an exit at any intermediate statement in the loop.

```
break;
```

Using a break statement to exit a loop has limited but important applications. For example, a program may use a loop to input data from a file. In such a case, the number of iterations depends on the amount of data in the file. The task of reading from the file is part of the loop body, which becomes the place where the program discovers that data is exhausted. When the end-of-file condition becomes true, a break statement exits the loop.

Instructions

Open **example.js** in Visual Studio Code 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 expected as you learn the keywords and syntax rules of this

Copyright © 2021 HyperionDev. All rights reserved.

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 Visual Studio Code (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 an expert code reviewer for help on Discord, as well as discuss the problem with peers.

Compulsory Task 1

Follow these steps:

- *Note: For this task, you will need to create an HTML file to get input from a user. If you need a refresher on how to do this, go back to the **example.js** and **index.html** files in your Task 2 folder for examples.*
- Create a JavaScript file called **palindrome.js**.
- Write a program that asks the user to enter a word.
- Using a **while loop**, the program must determine whether or not the word is a palindrome, i.e. whether it reads the same forwards and backwards. An example of a palindrome is the word "racecar".
- Output the given word and whether or not it is a palindrome, e.g. `"racecar is a palindrome"`

Compulsory Task 2

Follow these steps:

- *Note: For this task you will need to create an HTML file to get input from a user. If you need a refresher on how to do this, go back to the **example.js** and **index.html** files in your Task 2 folder for examples.*
- Create a new file called **while.js**.
- Write a program that always asks the user to enter a number.
- When the user enters -1, the program should stop requesting the user to enter a number.

HyperionDev
Copyright © 2021 HyperionDev. All rights reserved.

- The program must then calculate the average of the numbers entered, excluding the -1.
- Make use of the **while loop** repetition structure to implement the program.
- Compile, save, and run your file.

Compulsory Task 3

Follow these steps:

- *Note: For this task, you will need to create an HTML file to get input from a user. If you need a refresher on how to do this, go back to the* ***example.js*** *and* ***index.html*** *files in your Task 2 folder for examples.*
- Create a JavaScript file in this folder called **quiz.js**.
- Create a **do-while loop** in which you ask the user a multiple choice question. If they choose the incorrect answer, they are asked if they want to try again: if they answer that they do, they are asked the question again. The program ends when the user either guesses correctly, or they do not wish to guess again.
- Here is an example of the possible output:

```
What colour is the sky?
a. Purple
b. Pink
c. Blue
d. Yellow
Enter a, b, c, or d: [user enters b]
That’s incorrect! Would you like to try again? y/n: [user enters y]
[Question is asked again. User enters c]
That’s correct! [Program exits]
```

HyperionDev

Copyright © 2021 HyperionDev. All rights reserved.

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.

HyperionDev Copyright © 2021 HyperionDev. All rights reserved.