**alamWhile and do...while**

for is not the only type of loop available in JavaScript. There are actually many others and, while you don't need to understand all of these now, it is worth having a look at the structure of a couple of others so that you can recognize the same features at work in a slightly different way.

First, let's have a look at the while loop. This loop's syntax looks like so:

|  |
| --- |
| initializer  while (condition) {  // code to run  final-expression  } |

This works in a very similar way to the for loop, except that the initializer variable is set before the loop, and the final-expression is included inside the loop after the code to run, rather than these two items being included inside the parentheses. The condition is included inside the parentheses, which are preceded by the while keyword rather than for.

The same three items are still present, and they are still defined in the same order as they are in the for loop. This is because you must have an initializer defined before you can check whether or not the condition is true. The final-expression is then run after the code inside the loop has run (an iteration has been completed), which will only happen if the condition is still true.

Let's have a look again at our cats list example, but rewritten to use a while loop:

|  |
| --- |
| const cats = ['Pete', 'Biggles', 'Jasmine'];  let myFavoriteCats = 'My cats are called ';  let i = 0;  while (i < cats.length) {  if (i === cats.length - 1) {  myFavoriteCats += `and ${cats[i]}.`;  } else {  myFavoriteCats += `${cats[i]}, `;  }  i++;  }  console.log(myFavoriteCats); // "My cats are called Pete, Biggles, and Jasmine." |

|  |
| --- |
| **Note:** This still works just the same as expected — have a look at it running live on GitHub (also view the full source code). |

The do...while loop is very similar, but provides a variation on the while structure:

|  |
| --- |
| initializer  do {  // code to run  final-expression  } while (condition) |

In this case, the initializer again comes first, before the loop starts. The keyword directly precedes the curly braces containing the code to run and the final expression.

The main difference between a do...while loop and a while loop is that the code inside a do...while loop is always executed at least once. That's because the condition comes after the code inside the loop. So we always run that code, then check to see if we need to run it again. In while and for loops, the check comes first, so the code might never be executed.

Let's rewrite our cat listing example again to use a do...while loop:

|  |
| --- |
| const cats = ['Pete', 'Biggles', 'Jasmine'];  let myFavoriteCats = 'My cats are called ';  let i = 0;  do {  if (i === cats.length - 1) {  myFavoriteCats += `and ${cats[i]}.`;  } else {  myFavoriteCats += `${cats[i]}, `;  }  i++;  } while (i < cats.length);  console.log(myFavoriteCats); // "My cats are called Pete, Biggles, and Jasmine." |

|  |
| --- |
| Note: Again, this works just the same as expected — have a look at it running live on GitHub (also view the full source code). |

|  |
| --- |
| **Warning:** With while and do...while — as with all loops — you must make sure that the initializer is incremented or, depending on the case, decremented, so the condition eventually becomes false. If not, the loop will go on forever, and either the browser will force it to stop, or it will crash. This is called an infinite loop. |

1. **Match the words with their definition**

|  |  |  |  |
| --- | --- | --- | --- |
| **1.** Loop |  |  | **a.** the process of repeating a mathematical or computing process or set of instructions again and again, each time applying it to the result of the previous stage. |
| **2.** Features |  |  | **b.** cause a discrete reduction in (a numerical quantity). |
| **3.** Iteration |  |  | **c.** a set of instructions that is repeated again and again until a particular condition is satisfied. |
| **4.** Curly braces |  |  | **d.** cause a discrete increase in (a numerical quantity). |
| **5.** Incremented |  |  | **e.** either one of the marks **{** or **}** that are used as a pair around words or items that are to be considered together. |
| **6.** Decremented |  |  | **f.** something important, interesting or typical of a place or thing**.** |

[**https://www.oxfordlearnersdictionaries.com/definition/american\_english/iteration**](https://www.oxfordlearnersdictionaries.com/definition/american_english/iteration)

1. **True or False** 
   1. The only loop in JavaScript is *for*

* True
* False
  1. You need to have an initializer variable defined before you can check if a condition is true.
* True
* False

1. **Check the right answer (Only one answer is correct)**
   1. What is a true difference between the *for* loop and the *while* loop

* The *for* loop doesn’t need to end with a bracket, the while loop does
* The initializer variable is set before the loop
* The loop *for* end up being broken
* It’s strictly the same
  1. Why is the code inside a **do …** **while** loop executed?
* Because the condition comes after the code inside the loop
* Because the brackets are separated
* Because of some mystical force that runs inside the code
* Because it doesn’t need variation
  1. What is the main difference between a **while** loop and a **do while** loop

|  |
| --- |
| The main difference between a do...while loop and a while loop is that the code inside a do...while loop is always executed at least once. That's because the condition comes after the code inside the loop. So we always run that code, then check to see if we need to run it again. In while and for loops, the check comes first, so the code might never be executed. |

* 1. What can happen if the initializer is not incremented or decremented?

|  |
| --- |
| There will be an infinite loop. |

1. **Translate**

The same three items are still present, and they are still defined in the same order as they are in the *for* loop. This is because you must have an initializer defined before you can check whether or not the condition is true. The final-expression is then run after the code inside the loop has run (an iteration has been completed), which will only happen if the condition is still true.

Les trois mêmes éléments sont toujours présents et ils sont toujours définis dans le même ordre que dans la boucle for. En effet, vous devez définir un initialiseur avant de pouvoir vérifier si la condition est vraie ou non. L'expression finale est ensuite exécutée après l'exécution du code à l'intérieur de la boucle (une itération est terminée), ce qui ne se produira que si la condition est toujours vraie.