Need response times for mission critical applications within 30 minutes? Learn more →

We're hiring Blog Docs Get Support Contact Sales

Tutorials Questions Learning Paths For Businesses Product Docs Social Impact

CONTENTS

RELATED

CodeIgniter: Getting Started With a Simple Example

View

How To Install Express, a Node.js Framework, and Set Up Socket.io on a VPS

View

Tutorial Series: How To Code in JavaScript

24/37 For Loops, For...Of Loops ...

25/37 How To Define Functions ...

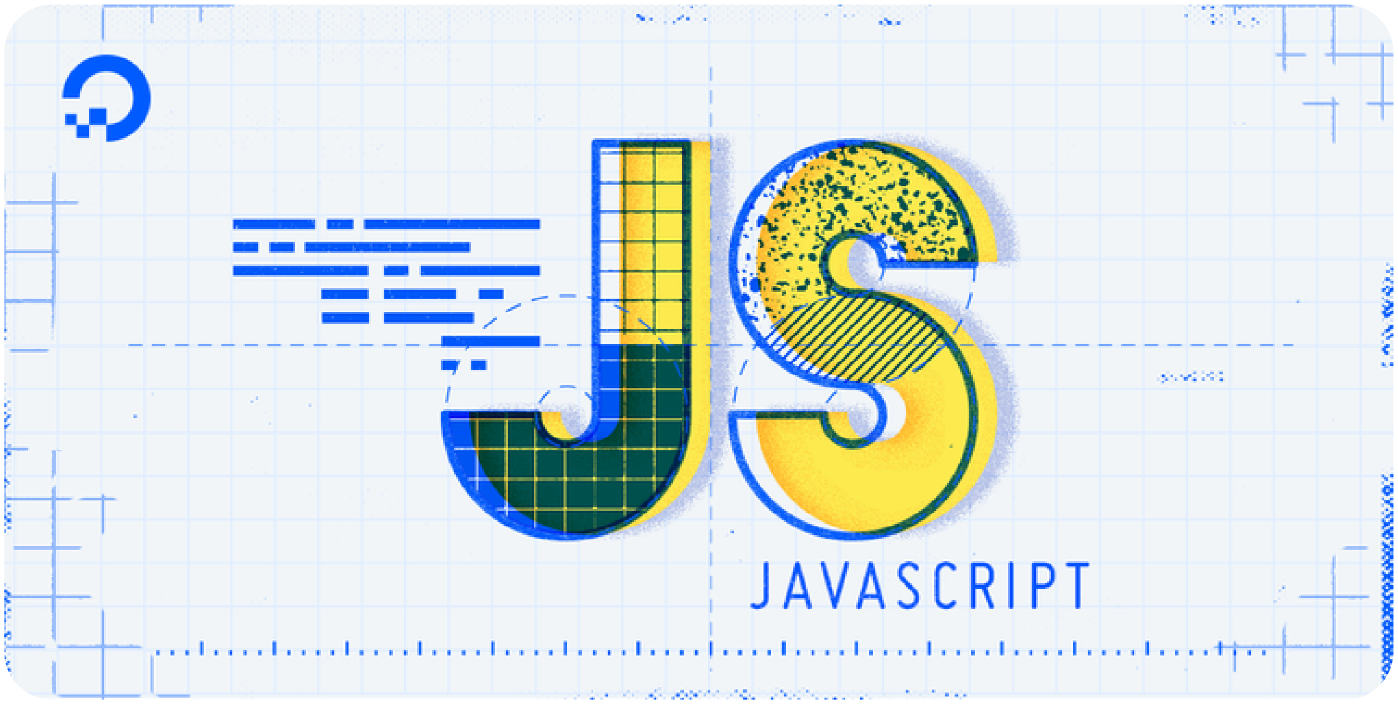
// ial //

For Loops, For...Of Loops and For...In Loops in JavaScript

Published on October 2, 2017 · Updated on August 26, 2021

JavaScript Development

By Tania Rascia



# Introduction

Loops are used in programming to automate repetitive tasks. The most basic types of loops used in JavaScript are the `while` and `do...while` statements, which you can review in "How To Construct While and Do...While Loops in JavaScript."

Because `while` and `do...while` statements are conditionally based, they execute when a given statement returns as evaluating to `true`. Similar in that they are also conditionally based, `for` statements also include extra features such as a **loop counter**, allowing you to set the number of iterations of the loop beforehand.

In this tutorial, we will learn about the `for` statement, including the `for...of` and `for...in` statements, which are essential elements of the JavaScript programming language.

[illegible] Loop

The `for` statement is a type of loop that will use up to three optional expressions to implement the repeated execution of a code block.

Let's take a look at an example of what that means.

Copy
```
for (initialization; condition; final expression) {
        // code to be executed
}
```

In the syntax above there are three expressions inside the `for` statement: the **initialization**, the **condition**, and the **final expression**, also known as incrementation.

Let's use a basic example to demonstrate what each of these statements does.

forExample.js

Copy
```
// Initialize a for statement with 5 iterations
for (let i = 0; i < 4; i++) {
        // Print each iteration to the console
        console.log(i);
}
```

When we run the code above, we'll receive the following output:

```
Output
0
1
2
3
```

In the above example, we initialized the `for` loop with `let i = 0`, which begins the loop at `0`. We set the condition to be `i < 4`, meaning that as long as `i` evaluates as less than `4`, the loop will continue to run. Our final expression of `i++` increments the count for each iteration through the loop. The `console.log(i)` prints out the numbers, starting with `0` and stopping as soon as `i` is evaluated as `4`.

Without using a loop, we could have achieved that same output by using the following code.

noLoop.js

Copy
```
// Se[illegible]itial variable to 0
let i = 0;
```

```
// Manually increment variable by 1 four times
console.log(i++);
console.log(i++);
console.log(i++);
console.log(i++);
```

Without the loop in place, the code block is repetitive and consists of more lines. If we needed to increment through more numbers we would have needed to write even more lines of code.

Let’s go over each expression in the loop to understand them fully.

# Initialization

Our first expression is the **initialization**. This is what it looks like.

```
let i = 0;
```

Copy

We’re declaring a variable called `i` with the `let` keyword (the keyword `var` may also be used) and giving it a value of `0`. While the variable can be named anything, `i` is most frequently used. The variable `i` stands for **i**teration, is consistent, and keeps the code compact.

# Condition

Just as we saw in the `while` and `do...while` loops, `for` loops usually contain a **condition**. Here is our condition statement.

```
i < 4;
```

Copy

We already established that our iteration variable, `i`, represents `0` to start. Now we are saying that the condition is `true` as long as `i` is less than `4` in this example.

# Final Expression

The **final expression** is a statement that is executed at the end of each loop. It is most often used to increment or decrement a value, but it can be used for any purpose.

```
i++
```

Copy

In our example, we are incrementing the variable by one, with `i++`. This is the same as running `i = i + 1`.

Unlike the initialization and condition expressions, the final expression does not end with a semicolon.

# Putting it Together

Now that we've reviewed our three expressions contained in the `for` loop, we can take a look at the complete loop again.

```
// Initialize a for statement with 5 iterations
for (let i = 0; i < 4; i++) {
	console.log(i);
}
```
Copy

First, we are declaring `i` and setting it to `0`. Then, we are setting a condition for the loop to run until `i` is less than `4`. Finally, we're incrementing `i` by one 1 each iteration. Our code block prints the value of `i` to the console, so our result is `0`, `1`, `2`, and `3` as output.

# Optional Expressions

All three expressions in the `for` loop are optional. For example, we can write the same `for` statement without the initialization expression by initializing the variable outside of the loop.

```
// Declare variable outside the loop
let i = 0;

// Initialize the loop
for (; i < 4; i++) {
	console.log(i);
}
```
Copy

```
Output
0
1
2
3
```

In this [illegible], the first `;` is necessary to denote whether the statement refers to initialization, condition, or final expression, even when it's omitted.

Below, we can also remove the condition from the loop. We will use an `if` statement combined with `break` to tell the loop to stop running once `i` is greater than `3`, which is the reverse of the `true` condition.

```
// Declare variable outside the loop
let i = 0;

// Omit initialization and condition
for (; ; i++) {
        if (i > 3) {
                break;
        }
        console.log(i);
}
```

Copy

```
Output
0
1
2
3
```

Warning: The `break` statement *must* be included if the condition is omitted, otherwise the loop will run forever as an infinite loop and potentially crash the browser.

Lastly, the final expression can be removed by putting it at the end of the loop instead. Both semicolons must still be included, or the loop will not function.

```
// Declare variable outside the loop
let i = 0;

// Omit all statements
for (; ;) {
        if (i > 3) {
                break;
        }
        console.log(i);
        i++;
}
```

Copy

```
Out
0
1
```

```
2
3
```

As we can see from the above examples, including all three statements generally produces the most concise and readable code. However, it's useful to know that statements can be omitted in case you encounter it in the future.

# Modifying an Array

We can use `for` loops to modify an array.

In the next example, we'll create an empty array and populate it with the loop counter variable.

modifyArray.js

Copy

```
// Initialize empty array
let arrayExample = [];

// Initialize loop to run 3 times
for (let i = 0; i < 3; i++) {
        // Update array with variable value
        arrayExample.push(i);
        console.log(arrayExample);
}
```

Running the JavaScript code above will result in the following output.

```
Output
[ 0 ]
[ 0, 1 ]
[ 0, 1, 2 ]
```

We set a loop that runs until `i < 3` is no longer `true`, and we're telling the console to print the `arrayExample` array to the console at the end of each iteration. With this method, we can see how the array updates with the new values.

# Length of an Array

Sometimes, we might want a loop to run a number of times without being certain of what the number of iterations will be. Instead of declaring a static number, as we did in previous examples, we can make use of the `length` property of an array to have the loop run as many times as there are items in the array.

loopThroughArray.js

```
// Declare array with 3 items
let fish = [ "flounder", "salmon", "pike" ];

// Initalize for loop to run for the total length of an array
for (let i = 0; i < fish.length; i++) {
	// Print each item to the console
	console.log(fish[i]);
}
```

Copy

We'll receive the following output.

```
Output
flounder
salmon
pike
```

In this example, we increment through each index of the array with `fish[i]` (e.g. the loop will increment through `fish[0]`, `fish[1]`, etc.). This causes the index to dynamically update with each iteration.

More detail on the `for` statement is available on the Mozilla Developer Network.

# For...In Loop

The `for...in` statement iterates over the properties of an object. To demonstrate, we will make a simple `shark` object with a few *name:value* pairs.

shark.js

```
const shark = {
	species: "great white",
	color: "white",
	numberOfTeeth: Infinity
}
```

Copy

Using the `for...in` loop, we can easily access each of the property names.

```
//   int property names of object
for        ibute in shark) {
	 onsole.log(attribute);
}
```

Copy

```
Output
species
color
numberOfTeeth
```

We can also access the values of each property by using the property name as the index value of the object.

Copy

```
// Print property values of object
for (attribute in shark) {
        console.log(shark[attribute]);
}
```

```
Output
great white
white
Infinity
```

Putting them together, we can access all the names and values of an object.

Copy

```
// Print names and values of object properties
for (attribute in shark) {
        console.log(`${attribute}`.toUpperCase() + `: ${shark[attribute]}`);
}
```

```
Output
SPECIES: great white
COLOR: white
NUMBEROFTEETH: Infinity
```

We used the `toUpperCase()` method to modify the property name, and following it by the property value. `for...in` is an extremely useful way to iterate through object properties.

Review `for...in` on the Mozilla Developer Network for more detailed information.

# For...Of Loop

The `for...in` statement is useful for iterating over object properties, but to iterate over iterable objects like arrays and strings, we can use the `for...of` statement. The `for...of` statement is a newer feature as of ECMAScript 6. ECMAScript (or ES) is a scripting-language specification created to standardize JavaScript.

In this example of a `for...of` loop, we will create an array and print each item in the array to the console.

sharks.js

Copy

```
// Initialize array of shark species
let sharks = [ "great white", "tiger", "hammerhead" ];

// Print out each type of shark
for (let shark of sharks) {
	console.log(shark);
}
```

We'll receive the following as output from the `for...of` statement.

```
Output
great white
tiger
hammerhead
```

It is also possible to print out the index associated with the index elements using the `entries()` method.

sharks.js

Copy

```
...
// Loop through both index and element
for (let [index, shark] of sharks.entries()) {
	console.log(index, shark);
}
```

```
Output
0 'great white'
1 'tiger'
2 'hammerhead'
```

A string can be iterated through in the same way as an array.

sharkString.js

```
// Assign string to a variable
let sharkString = "sharks";

// Iterate through each index in the string
for (let shark of sharkString) {
        console.log(shark);
}
```

Copy

Output
```
s
h
a
r
k
s
```

In this case, we looped through each character in the string, printing them in sequential order.

For a more detailed account of the differences between `for...in` and `for...of`, read about `for...of` loops on the Mozilla Developer Network.

# Conclusion

In this tutorial, we learned how to construct `for` loops in JavaScript, consisting of the `for`, `for...of` and `for...in` statements.

Loops are an integral part of programming in JavaScript, and are used for automating repetitive tasks and making code more concise and efficient.

Next in series: How To Define Functions in JavaScript →

Tutorial Series: How To Code in JavaScript

JavaScript is a high-level, object-based, dynamic scripting language popular as a tool for making webpages interactive.

Subscribe

JavaScript Development

Browse Series: 37 articles

Expand to view all

About the authors

Tania Rascia Author

Lisa Tagliaferri Editor

Still looking for an answer? Ask a question

Search for more help

Was this helpful? Yes No

Comments

1 Comments

Leave a comment...

This textbox defaults to using `Markdown` to format your answer.

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

anapimolodec • July 19, 2022

Hello, thank you for this post. Is there any difference between using For Loop and For Of Loop for arrays? I have the same code but with different loop types and they are giving different answers. And I cannot understand why, please help! thanks!

Reply

Try DigitalOcean for free

Click below to sign up and get **$200 of credit** to try our products over 60 days!

Sign up →

Popular Topics

Ubuntu

Linux Basics

JavaScript

Python

MySQL

Doc

Kubern

All tutorials →

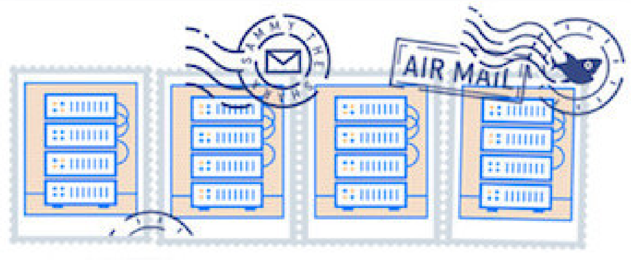
Free Managed Hosting →

Congratulations on unlocking the whale ambience easter egg! Click the whale button in the bottom left of your screen to toggle some ambient whale noises while you read.

Thank you to the Glacier Bay National Park & Preserve and Merrick079 for the sounds behind this easter egg.

Interested in whales, protecting them, and their connection to helping prevent climate change? We recommend checking out the Whale and Dolphin Conservation.

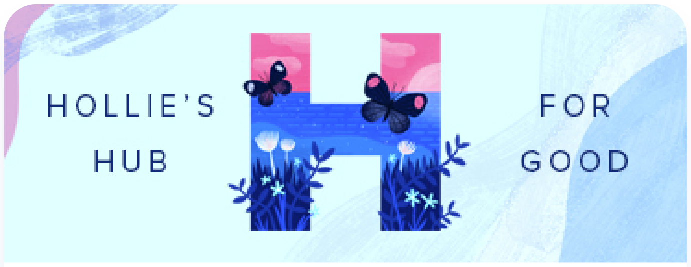
Reset easter egg to be discovered again / Permanently dismiss and hide easter egg



Get our biweekly newsletter

Sign up for Infrastructure as a Newsletter.

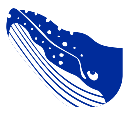
Sign up →

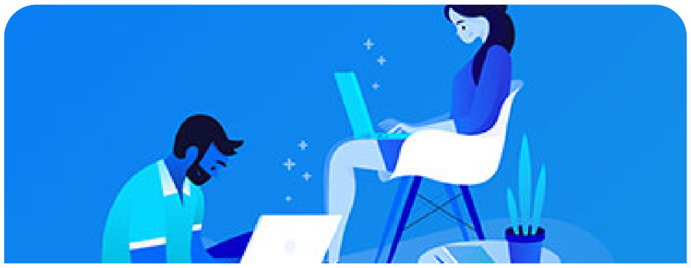


Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →





Become a contributor

You get paid; we donate to tech nonprofits.

Learn more →

Featured on Community

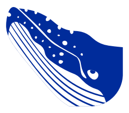
Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products

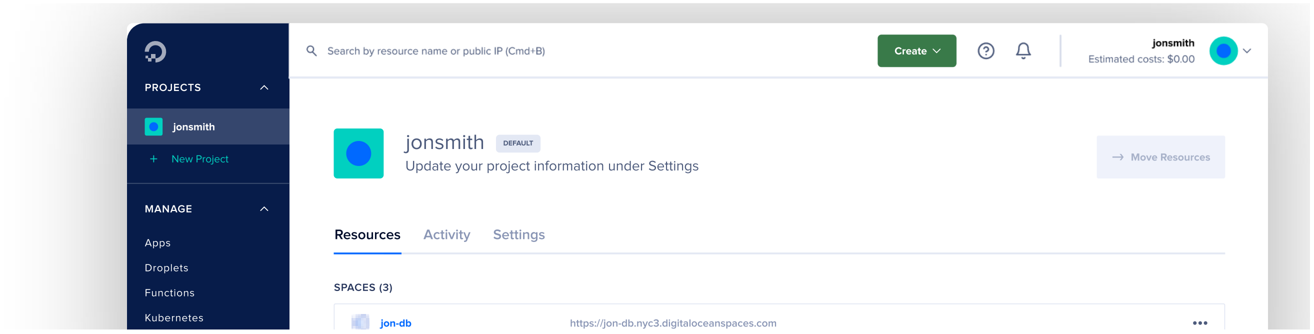
Cloudways Virtual Machines Managed Databases Managed Kubernetes Block Storage Object Storage Marketplace VPC Load Balancers

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.



Learn more →



Company

About

Leadership

Blog

Careers

Customers

Partners

Channel Partners

Referral Program

Affiliate Program

Press

Legal

Security

Investor Relations

DO Impact

Products

Products Overview

Droplets

Kubernetes

App Platform

Functions

Cloudways

Managed Databases

Spaces

Marketplace

Load Balancers

Block Storage

Tools & Integrations

API

Pricing

Documentation

Release Notes

Uptime

Community

Tutorials

Q&A

CSS-Tricks

Write for DOnations

Currents Research

Hatch Startup Program

deploy by DigitalOcean

Shop Swag

Research Program

Open Source

Code of Conduct

Newsletter Signup

Meetups

Solutions

Website Hosting

VPS Hosting

Web & Mobile Apps

Game Development

Streaming

VPN

SaaS Platforms

Cloud Hosting for Blockchain

Startup Resources

Contact

Support

Sales

Report Abuse

System Status

Share your ideas