

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

**Tutorial**

# Day 2: Loops



by [AvmnuSng](#)

Problem

Submissions

Leaderboard

Discussions

Editorial

## Loops

### JavaScript Loops

*Loops* are a quick and easy way to repeatedly perform a series of instructions, and they are typically run a finite number of times. JavaScript has the following types of loops:

- *for*

- *do-while*
- *for-in*
- *for-of*

## *for*

---

The *for* statement creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by one or more statements that will be executed in the loop.

### Basic Syntax

```
for (initialization; condition; finalExpression) {  
    statement(s);  
}
```

### Components

- ***initialization***: An expression or variable declaration that is typically used to initialize a counter variable.
- ***condition***: This is the *termination condition*, which is an expression that's evaluated before each pass through the loop. If this expression evaluates to *true*, then ***statement*** is executed. If the expression evaluates to *false*, execution jumps to the first line of code after the end of the loop. If this statement is omitted, then ***condition***

### Table Of Contents

---

[\*for\*](#)

[\*while\*](#)

[\*do-while\*](#)

[\*for-in\*](#)

[\*for-of\*](#)

OK

- ***finalExpression***: An expression to be evaluated at the end of each loop iteration. This occurs before the next evaluation of ***condition***.
- ***statement***: The statement (or statements) that is executed each time ***condition*** evaluates to *true*.

It's important to note that:

- The ***initialization***, ***condition***, and ***finalExpression*** in the head of the *for* loop are *optional*, but are generally always used.
- The head of a for loop typically looks like `for (var i = 0; i < maxValue; i++)`, where ***maxValue*** is the maximum value you wish to iterate until.

## Table Of Contents

---

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

### - EXAMPLE

Print all the integers in the range from 1 to some number given as input.

```

1 process.stdin.on('data', function (data) {
2     main(+data);
3 });
4 /**** Ignore above this line. ****/
5
6 function main(input) {
7     for (var i = 1; i <= input; i++) {
8         process.stdout.write(i + " ");
9     }
10 }
```

Input

10

Run

Output

## Initialize

In this example, we omit the *initialization* expression and instead initialize the variable used in *condition* and *finalExpression* before our loop:

```
1 process.stdin.on('data', function (data) {
2     main(+data);
3 });
4 /**** Ignore above this line. ****/
5
6 function main(input) {
7     var i = 1;
8
9     for (; i <= input; i++) {
10         process.stdout.write(i + " ");
11     }
12 }
```

Input

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

Output

## Condition

In this example, we omit the *condition* expression and instead add an *if* statement inside the loop that terminates the loop once a the condition `i > input` is satisfied:

```
1 process.stdin.on('data', function (data) {
2     main(+data);
3 });
4 /**** Ignore above this line. ****/
5
6 function main(input) {
7
8     for (var i = 1;; i++) {
9         if (i > input) {
10             break;
11         }
12
13         process.stdout.write(i + " ");
14     }
15 }
```

Input

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

## Output

## Infinite Loop

If we omit all three blocks, our loop will run infinitely or until such a time as we call `break`; from inside the loop. In this example, we do just that:

```
1 process.stdin.on('data', function (data) {
2     main(+data);
3 });
4 /**** Ignore above this line. ****/
5
6 function main(input) {
7     var i = 1;
8
9     for (;;) {
10         if (i > input) {
11             break;
12         }
13
14         process.stdout.write(i + " ");
15         i++;
16     }
17 }
```

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

10

Output

## *while*

The *while* statement creates a loop that executes its internal statement(s) as long as the specified ***condition*** evaluates to *true*. The condition is evaluated before executing the statement.

### Basic Syntax

```
while (condition) {  
    statement(s);  
}
```

- ***condition***: This is the *termination condition*, which is an expression that's evaluated before each pass through the loop. If this expression evaluates to *true*, then ***statement*** is executed; if it evaluates to *false*, execution jumps to the first line of code after the end of the loop.

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

- EXAMPLE

Print all the integers from **1** to **10**.

```
1 process.stdin.on('data', function (data) {
2     main(+data);
3 });
4 /**** Ignore above this line. ****/
5
6 function main(input) {
7     var i = 1;
8
9     while (i <= input) {
10         process.stdout.write(i + " ");
11
12         i++;
13     }
14 }
```

Input

10

Run

Output

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)



The *do-while* statement creates a loop that executes its internal statement(s) until the specified **condition** evaluates to false. The condition is evaluated after executing the internal statement(s), so the contents of the loop always execute *at least* once.

## Basic Syntax

```
do {  
    statement(s);  
} while (condition);
```

- **condition**: This is the *termination condition*, and it's evaluated *after* each pass through the loop (meaning the loop will always run at least once). Once the statement(s) inside the loop is executed, **condition** is evaluated. If this expression evaluates to *true*, then **statement** is executed again; if it evaluates to *false*, execution jumps to the first line of code after the end of the loop.
- **statement**: The statement (or statements) that is executed each time **condition** evaluates to *true*.

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

-	EXAMPLE
Print all the integers in the range from 1 to some number given as input.	
1	process.stdin.on('data', function (data) {
2	main(+data);

```
6 function main(input) {
7     var i = 1;
8
9     do {
10         process.stdout.write(i + " ");
11
12         i++;
13     } while (i <= input);
14 }
```

Input

10

Run

Output

## Table Of Contents

---

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

## *for-in*

---

This loop iterates (in an arbitrary order) over the *name* of each enumerable property in an object, allowing statements to be executed for each distinct property.

### Basic Syntax

```
}
```

- **variable**: A variable that refers to a different property *name* during each iteration of the loop. You can declare this with `var` or `let`.
- **object**: The object whose enumerable properties are being iterated through.

## Table Of Contents

[for](#)[while](#)[do-while](#)[for-in](#)[for-of](#)

### - EXAMPLE

In the code below, we create an object (referenced by the **actress** variable) and iterate over its enumerable properties:

```
1 var actress = {
2   firstName: "Julia",
3   lastName: "Roberts",
4   dateOfBirth: "October 28, 1967",
5   nationality: "American",
6   firstMovie: "Satisfaction"
7 };
8
9 for (var property in actress) {
10   console.log("actress." + property + " = " + actress[property]);
11 }
```

Output

Run

```
actress.firstName = Julia
actress.lastName = Roberts
actress.dateOfBirth = October 28, 1967
actress.nationality = American
actress.firstMovie = Satisfaction
```

In this code, we create a *Monster* object named ***monster***, then print the object followed by its individual properties.

### Input Format

The first line contains a string, ***name***, denoting the type of monster.

The second line contains a string, ***home***, denoting the location where the monster lives.

The third line contains a string, ***description***, describing the monster.

```
1 'use strict';
2 process.stdin.on('data', function (data) {
3     main(String(data).trim().split(new RegExp("[\n]+")));
4 });
5 /**** Ignore above this line. ****/
6
7 class Monster {
8     constructor(name, home, description) {
9         this.name = name;
10        this.home = home;
11        this.description = description;
12    }
```

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

```
15 function main(input) {
16     var monster = new Monster(input[0], input[1], input[2]);
17
18     // Print array
19     console.log(monster);
20
21     // Print each of its elements on a new line
22     for (let property in monster) {
23         console.log(property + ": " + monster[property]);
24     }
25 }
```

Input

Minotaur  
Labyrinth

Run

Output

The code above produces the following output for the given input:

```
Monster {
  name: 'Minotaur',
  home: 'Labyrinth',
  description: 'Bull head, man body.' }
name: Minotaur
home: Labyrinth
```

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

---

## for-of

---

This loop iterates over iterable objects such as an *Array*, *Map*, *Set*, *String*, *TypedArray*, *arguments object*, etc. It essentially iterates over the *value* of each distinct property in the structure, such as each letter in a word or each element in an array.

### Basic Syntax

```
for (let variable of iterable) {  
    statement(s);  
}
```

- **variable**: A variable that refers to a different property *value* during each iteration of the loop. You can declare this with `var` or `let`.
- **object**: The object whose enumerable properties are being iterated through.

-

#### EXAMPLE

The code below splits the input into an array and prints it. It then iterates over each element of the array and prints it on a new line.

#### Input Format

Space and/or newline-separated words.

---

### Table Of Contents

---

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

```

3   main(String(data).trim());
4   });
5   /**** Ignore above this line. ****/
6
7   function main(input) {
8       // Split the words read as input into an array of words
9       var array = input.split(new RegExp("[ \n]+"));
10
11      // Print array
12      console.log(array);
13
14      // Print each of its elements on a new line
15      for (let value of array) {
16          console.log(value);
17      }
18  }

```

Input

hi bye  
hello goodbye

Run

Output

The code above produces the following output:

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)

```
hello  
goodbye
```

In this code, we iterate over the set of *Key-Value* pairs in a *Map*, first printing each *Key-Value* pair and then printing each individual *Key* and its paired *Value*.

```
1 'use strict';  
2  
3 let actress = new Map([  
4   ["firstName", "Julia"],  
5   ["lastName", "Roberts"],  
6   ["dateOfBirth", "October 28, 1967"],  
7   ["nationality", "American"],  
8   ["firstMovie", "Satisfaction"]  
9 ]);  
10  
11 // Print each Key-Value pair in the map  
12 for (let info of actress) {  
13   console.log(info);  
14 }  
15  
16 // Print each Key and Value as "Key: Value"  
17 console.log();  
18 for (let info of actress) {  
19   console.log(info[0] + ": " + info[1]);  
20 }
```

Output

## Table Of Contents

[for](#)

[while](#)

[do-while](#)

[for-in](#)

[for-of](#)



The code above produces the following output:

```
[ 'firstName', 'Julia' ]  
[ 'lastName', 'Roberts' ]  
[ 'dateOfBirth', 'October 28, 1967' ]  
[ 'nationality', 'American' ]  
[ 'firstMovie', 'Satisfaction' ]
```

```
firstName: Julia  
lastName: Roberts  
dateOfBirth: October 28, 1967  
nationality: American  
firstMovie: Satisfaction
```

## Table Of Contents

---

*[for](#)*

*[while](#)*

*[do-while](#)*

*[for-in](#)*

*[for-of](#)*

---

[Contest Calendar](#) | [Interview Prep](#) | [Blog](#) | [Scoring](#) | [Environment](#) | [FAQ](#) | [About Us](#) | [Support](#) | [Careers](#) | [Terms Of Service](#) | [Privacy Policy](#) | [Request a Feature](#)