

# Loops

In the real world, that is rarely how things work and, in many cases, we specifically want our code to do things more than once, usually as many times as is necessary and sometimes even endlessly. For this, we have loops. Loops are a vital part of all programming languages and will play a vital role in most JavaScript code. At their core, loops are simple. We create some sort of loop condition and as long as this condition holds or true, the loop will keep running.

## for loop

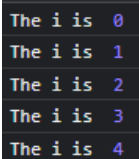
**for** loop runs the statement as long as the condition is true.

*Syntax*

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

**Example**) use a for loop to display an increasing counter from 0 to 4

```
for (let i = 0; i < 5; i++) {  
    console.log( "The current i is ", i);  
}
```



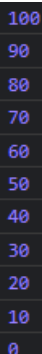
```
The i is 0  
The i is 1  
The i is 2  
The i is 3  
The i is 4
```

**Example**) write a JS code that uses for loop to print even numbers between 0 and 100 exclusive

```
for(let num = 0; num<100; num++){  
    if (num%2 ===0){console.log(num)}  
}
```

**Example**) write a JS code that uses for loop to print numbers from 100 to 0, inclusive, with a decrement of 10

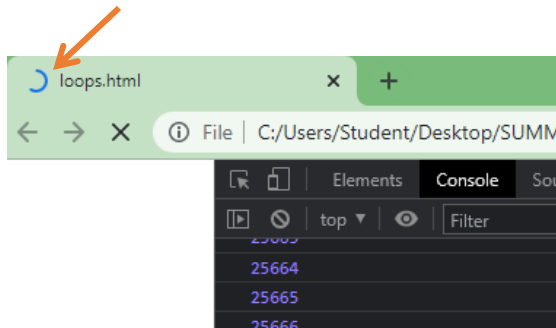
```
for(let num = 100; num>=0; num -=10){  
    console.log(num)  
}
```



```
100  
90  
80  
70  
60  
50  
40  
30  
20  
10  
0
```

## Infinite loops

Infinite loops are loops that do not stop, they run forever! Infinite loops are not very recommended since it can use all our computer memory and overheat our microprocessor.



## for loop in an array

**for** loop is very useful to loop to each values in an array. When we work with for loop in an array, we have to keep in mind that zero is the most common initial value because it is the initial index value in an array.

```
let cars = ['Mazda', 'Tesla', 'Dodge','BMW', 'Porshe'];
for(let i=0; i<cars.length; i++){
  console.log('Car ', i+1, ' = ', cars[i]);
}
```

```
Car 1 = Mazda
Car 2 = Tesla
Car 3 = Dodge
Car 4 = BMW
Car 5 = Porshe
```

There is also a specific statement in a for loop that works with array, which is the **for... of** statement. The **of** statement in a for loop will loop to each item in the list

```
let cars = ['Mazda', 'Tesla', 'Dodge','BMW', 'Porshe'];
for(let eachItem of cars){
  console.log(eachItem);
}
```

```
Mazda
Tesla
Dodge
BMW
Porshe
```

### for loop in a string

We can also use **for** loop to go to each of the character in an string:

```
P
r
o
f
.
W
u

let myString = "Prof. Wu";
for(let eachLetter of myString){
  console.log(eachLetter);
}
```

**Example)** Nest statements: Display a message *I want some* \_\_\_\_\_ with each food within an array. All even foods will display upper cases.

```
let foods = ['salad', 'pizza', 'tofu', 'cheese burger'];
for(let i=0; i<foods.length;i++){
  if (i%2 === 0){
    x=foods[i].toUpperCase();
    console.log('I want some ' + x);
  }
  else{
    console.log('I want some ' + foods[i]);
  }
}
```

```
I want some SALAD
I want some pizza
I want some TOFU
I want some cheese burger
```

### Nesting for loops

Nesting for loops, basically means that for each iteration of the outer for loop, it runs one complete inner for loop.

**Example)** write a JS code that will print a decrement from 30 to 0, decrement of 10, five times

```
for(let i=1; i<=5 ; i++){
  console.log(`OUTER LOOP COUNT \t\t${i}`)
  for(let j=30; j>=0 ; j-= 10){
    console.log(`Inner loop count  ${j}`)
  }
}
```

OUTER LOOP COUNT	1
Inner loop count	30
Inner loop count	20
Inner loop count	10
Inner loop count	0
OUTER LOOP COUNT	2
Inner loop count	30
Inner loop count	20
Inner loop count	10
Inner loop count	0
OUTER LOOP COUNT	3
Inner loop count	30
Inner loop count	20
Inner loop count	10
Inner loop count	0
OUTER LOOP COUNT	4
Inner loop count	30
Inner loop count	20
Inner loop count	10
Inner loop count	0
OUTER LOOP COUNT	5
Inner loop count	30
Inner loop count	20
Inner loop count	10
Inner loop count	0

## while loop

The for loop assumes you know how many times you want to loop to run but sometimes you just want to run the loop until some condition changes. In that case, you can use a while loop instead.

The while loop allows us to create more advanced functions inside the core block and run the loop as long as these or other external conditions are true.

### Syntax

```
while (condition) {
  // code block to be executed
}
```

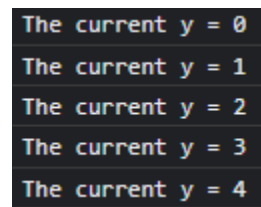
One of the ways to create while loop is to have the initial value of the while loop statement declares before the while loop. Once in the while loop, inside the while loop should also have the update of the initial value. Otherwise, if the initial value of the condition statement of the while loop is not updated, the while loop will run infinite times, which is not recommended!

### Syntax

```
let initial_value = value
while (condition) {
  // code block to be executed
  update initial_value
}
```

**Example)** use **while** loop to display number from 0 to 4

```
let y = 0; //initial value
while (y<5) {
  console.log(`The current y = ${y}`);
  y++; //update initial value
}
```



```
The current y = 0
The current y = 1
The current y = 2
The current y = 3
The current y = 4
```

**Example)** Create a JS code, using while loop, that asks the user to guess a secret number. If the user guesses the wrong number, it will continue asking the user to enter another number. The program stops when the user guessed the secret number.

```
const SECRET = 8;

let guessNum = parseInt(prompt("Guess a number between 0 and 10"));
while (guessNum !== SECRET){
  guessNum = parseInt(prompt("WRONG! Guess another number between 0 and 10"));
}
```

### **Break and continue keyword**

The **break** keyword "jumps out" or terminate a loop. You have already seen the **break** statement used in a **switch()** statement. It basically works the same way in loop.

**Example)** Create a JS code that will **stop** a decrement counter from 20 to 0, inclusive, when the counter reaches to 9.

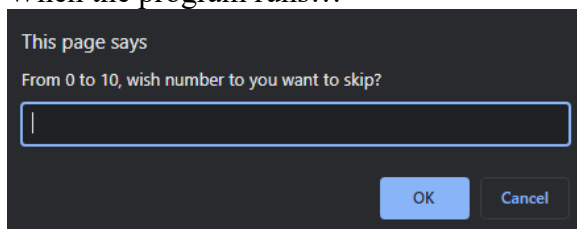
```
for(let counter = 20; counter>=0 ; counter--){  
    if(counter===9){break}  
    console.log(counter)  
}
```

The **continue** statement breaks one iteration (in the loop) if a specified condition occurs, and continues with the next iteration in the loop.

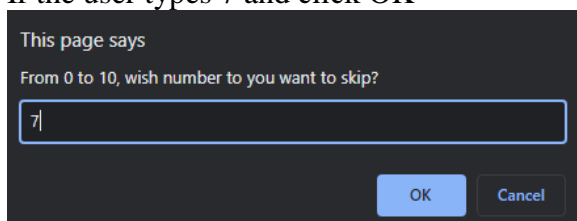
**Example)** Create a JS code that will skip a number of an increment counter from 0 to 10, inclusive. The skip number is entered by user from a dialog box.

```
let skipNum = parseInt(prompt("From 10 to 0, wish number to you want to skip?"  
"))  
for(let counter = 0; counter>=10 ; counter++){  
    if(counter===skipNum){continue}  
    console.log(counter)  
}
```

When the program runs...



If the user types 7 and click OK



The console displays...

