5. 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 <u>true</u>, the loop will keep running.
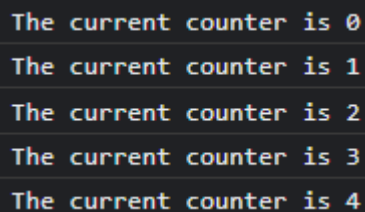
for loop

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

Syntax
```
for (initial_value; loop condition; update_initial_value) {
  // code block to be executed
}
```

Example 1) use a for loop to display an increasing counter from 0 to 4

```
for(let counter = 0; counter<=4; counter++){
  console.log(`The current counter is ${counter}`)
}
```

```
The current counter is 0
The current counter is 1
The current counter is 2
The current counter is 3
The current counter is 4
```

Example 2) write a JS code that uses for loop to print numbers that are multiple of 5 between 0 and a number less than 90 but greater than 0, exclusive. Initial number will be collected from the dialog box

```
let initialNum = parseInt(prompt("Enter a number between 0 and 90"))
for(initialNum; initialNum>=0; initialNum--){
    if(initialNum%5===0){
        console.log(`counting = ${initialNum}`)
    }
}
```

This page says

Enter a number between 0 and 90

20

OK   Cancel
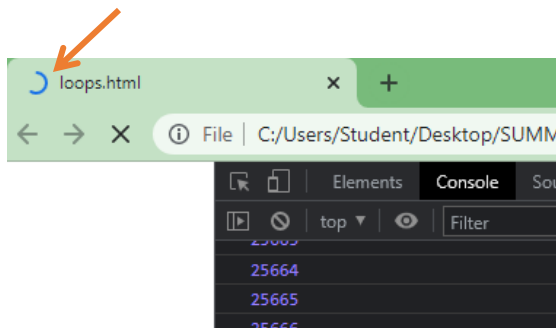
```
counting = 20
counting = 15
counting = 10
counting = 5
counting = 0
```

Example 3) 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)
    }
```

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.

Example 4) Use a for loop to display each item in array `cars[]`

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

Example 5) use a **for … of** loop to print each item in array

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

```
Mazda
Tesla
Dodge
BMW
Porshe
```

for loop in a string

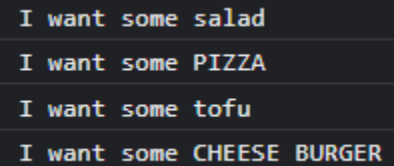We can also use **for** loop to go to each of the character in an string:

Example 6) use a **for…of** loop to print each character in a string

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

```
P
r
o
f
.
W
u
```

Example 7) Nest statements: create an array for *foods*. Use a **for** loop to display a message with each value in array `foods`. For example: *I want some _____.* All even foods will be uppercase.

```
let foods =['salad','pizza','tofu','cheese burger'];
for(let item = 0; item<foods.length ; item++){
   if((item+1)%2===0){
        let itemUpper = foods[item].toUpperCase();
        console.log(`I want some ${itemUpper}`)
      }
      else{
        console.log(`I want some ${foods[item]}`)
      }
 }
```

```
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 8) write a JS code that will print a decrement from 30 to 0, step of 10, five times

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

```
OUTER LOOP COUNT          1
Inner counter = 30
Inner counter = 20
Inner counter = 10
Inner counter = 0
OUTER LOOP COUNT          2
Inner counter = 30
Inner counter = 20
Inner counter = 10
Inner counter = 0
OUTER LOOP COUNT          3
Inner counter = 30
Inner counter = 20
Inner counter = 10
Inner counter = 0
OUTER LOOP COUNT          4
Inner counter = 30
Inner counter = 20
Inner counter = 10
Inner counter = 0
OUTER LOOP COUNT          5
Inner counter = 30
Inner counter = 20
Inner counter = 10
Inner counter = 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 (loop condition) {
  // code block to be executed
}
```
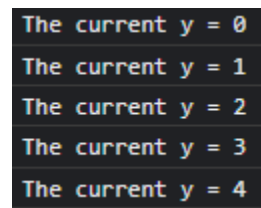
One of the ways to create **while** loop is to have the initial value of the while loop statement declares before the while loop. Once in the while loop, inside the while loop should also have the update of the initial value. Otherwise, if the initial value of the condition statement of the while loop is not updated, the while loop will run infinite times, which is not recommended!

Syntax

```
let initial_value = value

while (loop condition) {
  // code block to be executed

  update initial_value
}
```

Example 9) 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 10) 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 terminates the entire loop. You have already seen the **break** statement used in a **switch()** statement. It basically works the same way in loop.

Example 11) Create a loop that will print decrement counter from 5 to -10, inclusive, and stop when the loop counter reaches -3.

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

The **continue** statement breaks or skips one iteration (in the loop) if a specified condition occurs, and continues with the next iteration in the loop.
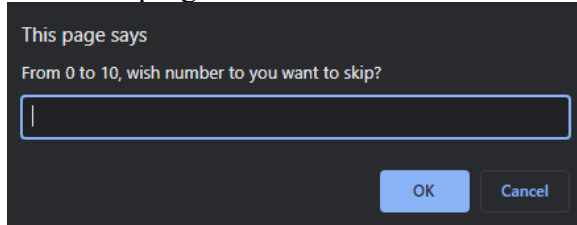
Example 12) Create a JS code that will **skip** even numbers from -10 to 10, inclusive.
```
for(let y = -10; y<=10; y++){
    if(y%2!==0){
        continue
    }
    else{
        console.log(y)
    }
}
```

Example 13) Create a JS code that will **skip** a number between 10 and 0, inclusive.

```
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)
}
```
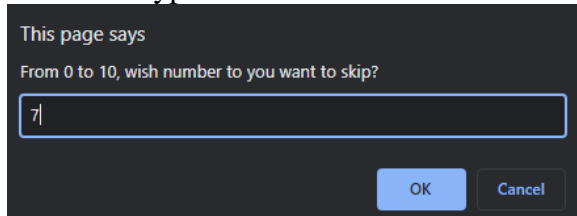
When the program runs…

This page says

From 0 to 10, wish number to you want to skip?

> [   ]

> OK   Cancel

If the user types 7 and click OK

This page says
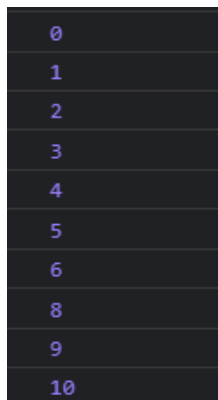
From 0 to 10, wish number to you want to skip?

> [7   ]

> OK   Cancel

The console displays…

```
0
1
2
3
4
5
6
8
9
10
```