Training Day14 report

26 June 2024

JavaScript Loops:

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Different Kinds of Loops

JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- for/of - loops through the values of an iterable object
- while - loops through a block of code while a specified condition is true
- do/while - also loops through a block of code while a specified condition is true

For Loop:

The for statement creates a loop with 3 optional expressions:

```
Syntax: for( expression1; expression2; expression3){

// code block to be executed

}
```

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Example:

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

From the example above, you can read:

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

Expression 1

Normally you will use expression 1 to initialize the variable used in the loop (let i = 0).

This is not always the case. JavaScript doesn't care. Expression 1 is optional.

You can initiate many values in expression 1 (separated by comma):

Example

```
for (let i = 0, len = cars.length, text = ""; i < len; i++) {
  text += cars[i] + "<br>";
}
```

And you can omit expression 1 (like when your values are set before the loop starts):

Example

```
let i = 2;
let len = cars.length;
let text = "";
for (; i < len; i++) {
  text += cars[i] + "<br>";
}
```

Expression 2

Often expression 2 is used to evaluate the condition of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 2 is also optional.

If expression 2 returns true, the loop will start over again. If it returns false, the loop will end.

If you omit expression 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Expression 3

Often expression 3 increments the value of the initial variable.

This is not always the case. JavaScript doesn't care. Expression 3 is optional.

Expression 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Expression 3 can also be omitted (like when you increment your values inside the loop):

Example

```
let i = 0;
let len = cars.length;
let text = "";
for (; i < len; ) {
  text += cars[i] + "<br>";
  i++;
}
```

While loop:

The **while loop** executes a block of code as long as a specified condition is true. In JavaScript, this loop evaluates the condition before each iteration and continues running as long as the condition remains true. The loop terminates when the condition becomes false, enabling dynamic and repeated operations based on changing conditions.

Syntax:
```
while (condition) {
   //Code block to be executed
}
```

Do-While loop:

A **Do-While loop** is another type of loop in JavaScript that is similar to the while loop, but with one key difference: the do-while loop guarantees that the block of code inside the loop will be executed at least once, regardless of whether the condition is initially true or false

Syntax:

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

For-in loop:

The **for-in loop in JavaScript** iterates over the enumerable properties of an object, executing a specified block of code for each property. It's commonly used for iterating through object properties or elements in an array.

Syntax:
```
for (let i in obj1) {
   // Prints all the keys in
   // obj1 on the console
   console.log(i);
}
```

for in Loop Important Points:

- Use the for-in loop to iterate over non-array objects. Even though we can use a for-in loop for an array, it is generally not recommended. Instead, use a for loop for looping over an array.
- The properties iterated with the for-in loop also include the properties of the objects higher in the prototype chain.
- The order in which properties are iterated may not match the properties that are defined in the object.

JavaScript for...of loop is used to iterate over iterable objects such as arrays, strings, maps, sets, etc. It provides a simpler syntax compared to traditional for loops.

Syntax:

```
for ( variable of iterableObjectName) {
  // code block to be executed
}
```

Parameters:

- **Variable**: Represents the current value of each iteration from the iterable.
- **Iterable**: Any object that can be iterated over (e.g., arrays, strings, maps).

Do-while loop:

A do…while loop in JavaScript is a control structure where the code executes repeatedly based on a given boolean condition. It's similar to a repeating if statement. One key difference is that a do…while loop guarantees that the code block will execute at least once, regardless of whether the condition is met initially or not.

There are mainly two types of loops:

- **Entry Controlled loops**: In this type of loop, the test condition is tested before entering the loop body. **For Loop** and **While Loops** are entry-controlled loops.
- **Exit Controlled Loops**: In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the **do-while loop** is exit controlled loop.

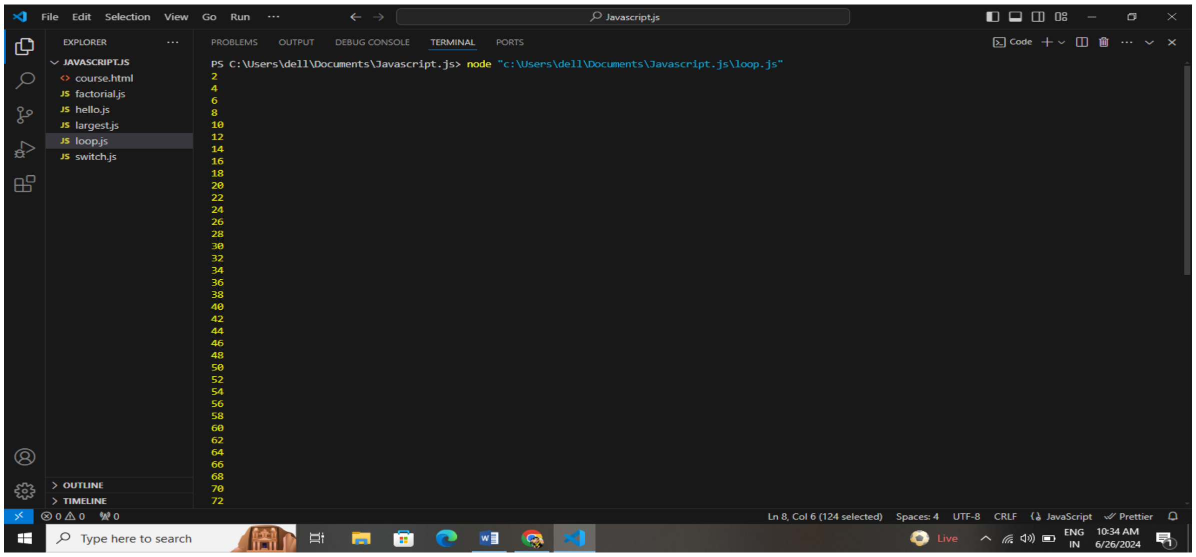
Syntax: do {

```
            //Statements
}
while(conditions)
```

Example 1:

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

Output:



```
PS C:\Users\dell\Documents\Javascript.js> node "c:\Users\dell\Documents\Javascript.js\loop.js"
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
72
```

Example 2:

```
let n = 5;

function factorial(n) {
    let ans = 1;

    if(n === 0)
        return 1;
    for (let i = 2; i <= n; i++)
        ans = ans * i;
    return ans;
}

console.log(factorial(n));
```

Output:

```
PS C:\Users\dell\Documents\Javascript.js> node
"c:\Users\dell\Documents\Javascript.js\factorial.js"

120
```

Example 3:

```
let count = 1;
while (count <= 5) {
  console.log(count);
  count++;
}
```

Output:

```
PS C:\Users\dell\Documents\Javascript.js> node
"c:\Users\dell\Documents\Javascript.js\factorial.js"

1

2

3

4

5
```

Example 4:

```
const array = [1, 2, 3, 4, 5];

for (const element of array) {
  console.log(element);
}
```

Output:

```
PS C:\Users\dell\Documents\Javascript.js> node
"c:\Users\dell\Documents\Javascript.js\tempCodeRunnerFile.js"

1

2

3

4

5
```

Example 5:

```
const str = "Hello";

for (const char of str) {
  console.log(char);
}
```

Output:

```
PS C:\Users\dell\Documents\Javascript.js> node
"c:\Users\dell\Documents\Javascript.js\factorial.js"

H

e

l

l

o
```

Example 6:

```
let test = 1;
do {
    console.log(test);
    test++;
} while(test<=5)
```

Output:

```
PS C:\Users\dell\Documents\Javascript.js> node
"c:\Users\dell\Documents\Javascript.js\factorial.js"

1
2
3
4
5
```