Training Day17 report

1 July 2024

JavaScript Break and Continue

The break statement "jumps out" of a loop.

The continue statement "jumps over" one iteration in the loop.

The Break Statement

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch() statement.

```
for (let i = 0; i < 10; i++)
   {
   if (i === 3)
       {
          break;
       }
      console.log("The number is:", i);
  }
```

Output:

```
PS C:\Users\dell\Documents\Javascript.js> node
"c:\Users\dell\Documents\Javascript.js\break and continue.js"

The number is: 0

The number is: 1

The number is: 2
```

The Continue Statement

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

```
for (let i = 0; i < 10; i++)
  {
  if (i === 3)
      {
          continue;
      }
      console.log("The number is:", i);
}
```

Output:

```
PS C:\Users\dell\Documents\Javascript.js> node
"c:\Users\dell\Documents\Javascript.js\break and continue.js"

The number is: 0

The number is: 1

The number is: 2

The number is: 4

The number is: 5

The number is: 6

The number is: 7

The number is: 8

The number is: 9
```

Nested Loops JavaScript

A composition of loops is called a nested loop. The most common type of nested loops will be one loop inside another. The first loop is usually called the outer loop while the second loop is called the inner loop.

```
for (let i = 12; i < 13; i++) {
    for (let j = 1; j <= 10; j++) {
      console.log(`${i} x ${j} = ${i * j}`);
    }
  }
```

Output:

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

12 x 1 = 12

12 x 2 = 24

12 x 3 = 36

12 x 4 = 48

12 x 5 = 60

12 x 6 = 72

12 x 7 = 84

12 x 8 = 96

12 x 9 = 108

12 x 10 = 120
```

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

JavaScript Function Syntax

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas: **(*parameter1, parameter2, ...*)**

The code to be executed, by the function, is placed inside curly brackets: **{}**

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

```
function add()
{
    let a=10;
    let b=20;
    let c=a+b;
    console.log("Sum of two numbers is :",c);
}
add()
```

Output:

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

Sum of two numbers is : 30
```

Arrow function:

```
const ADD = () => {
    let a=20;
    let b=34;
    return console.log("sum =", a+b);
}
ADD()
```

Output:

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

sum = 54
```

Anonymous Function

It is a function that does not have any name associated with it. Normally we use the *function* keyword before the function name to define a function in JavaScript, however, in anonymous functions in JavaScript, we use only the *function* keyword without the function name.

An anonymous function is not accessible after its initial creation, it can only be accessed by a variable it is stored in as a *function as a value*. An anonymous function can also have multiple arguments, but only one expression.

Syntax:
The below-enlightened syntax illustrates the declaration of an anonymous function using the normal declaration:

```
function() {
   // Function Body
 }
```

We may also declare an anonymous function using the arrow function technique which is shown below:

```
( () => {
   // Function Body...
} )();
```

```
(() =>{
   let a=20;
   let b=34;
   return console.log("sub =", a-b);
}
)()
```

Output:

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

sub = -14
```

Parameters:

```
function add(a ,b)
{
   let c=a+b;
```

```
    return console.log("Sum of two numbers is :",c);
}
add(20 ,20)
```

output:

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

Sum of two numbers is : 40
```