Control statements in JS

- Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

- When more than one choices are available, conditional statements.

In JavaScript we have the following conditional statements:

- Use simple if to specify a block of code to be executed, if a specified condition is true
- Use if else statement to specify a block of code to be executed, if a specified condition is true otherwise else part is executed when the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

Simple if Statement

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

Note: Java script is case sensitive. if is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

Example:

```
if (num > 0) {
   result = num + " is a positive number";
}
```

if … else Statement

if block is executed when the condition is true, otherwise the statements in else block is executed.

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

Example:

```
if (num > 0) {
   result = num + " is a positive number";
}
else {
   result = num + " is a negative number";
}
```

if … else if Statement

Use the if … else if statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2
is true
}
. . .

else {
  //  block of code to be executed if the condition1 is false and condition2
is false
}
```

Example:

```
if (num > 0) {
  result = num + " is a positive number";
}
else if (num < 0) {
  result = num + " is a negative number";
}
else {
  result = num + " is Zero";
}
```

JavaScript Switch Statement

- The switch statement is used to perform different actions based on different conditions.
- Use the switch statement to select one of many code blocks to be executed.

Syntax

```
switch(expression) {
  case 1:
    // code block
    break;
  case 2:
    // code block
    break;
  default:
    // code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

- If **no default** block is found, the program continues to the statement(s) after the switch**.**

Example

The getDay() method returns the weekday as a number between 0 and 6.

(Sunday=0, Monday=1, Tuesday=2 ..)

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
     day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
}
```

break Keyword

When JavaScript reaches a break keyword, it breaks out of the switch block.

Note: If you omit the break statement, the next case will be executed even if the evaluation does not match the case.

default Keyword

- The default keyword specifies the code to run if there is no case match.
- The default case does not have to be the last case in a switch block:
- If default is not the last case in the switch block, remember to end the default case with a break.

Common Code Blocks

Sometimes you will want different switch cases to use the same code.

In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

Example:

```
switch (new Date().getDay()) {
  case 4:
  case 5:
    text = "Soon it is Weekend";
    break;
  case 6:
  case 0:
    text = "It is Weekend";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

Strict Comparison

- Switch cases use **strict** comparison (= = =).
- The values must be of the same type to match.
- A strict comparison can only be true if the operands are of the same type.

In this example there will be no match for x:

Example:

```
let x = "0";
switch (x) {
  case 0:
    text = "Off";
    break;
  case 1:
    text = "On";
    break;
  default:
    text = "No value found";
}
```

The result will be "No value found".

Looping Statements in JavaScript

- Loops can execute a block of code as long as a specified condition is true.
- Normally every loop has the following steps:

  Initialization
  Condition
  Body of the loop
  Increment/Decrement(Step size)

Different Kinds of Loops

JavaScript supports different kinds of loops:

- **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** - 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

JavaScript While Loop

- The while loop loops through a block of code as long as a specified condition is true.

Syntax

```
initialization

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

increment/decrement
}
```

Example

```
<h2>JavaScript While Loop</h2>
<p id="demo"></p>
<script>
let text = "";
let i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
} document.getElementById("demo").innerHTML = text;
</script>
```

Note:
If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser/syatem.

do … while Loop

The do while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

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

Example:

```
<body>
<h2>JavaScript Do While Loop</h2>
<p id="demo"></p>
<script>
        let text = ""
        let i = 0;

        do {
          text += "<br>The number is " + i;
          i++;
        }
        while (i < 10);

        document.getElementById("demo").innerHTML = text;
</script>
</body>
```

Note:

The example below uses a do while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Comparing For and While loops

- The loop in this example uses a while loop to collect the car names from the cars array:

Example using for loop

```
<body>
<p id="demo"></p>

<script>
const cars = ["BMW", "Volvo", "Saab", "Ford"];

    let i = 0;
    let text = "";
    while (cars[i]) {
      text += cars[i] + "<br>";
      i++;
    }

document.getElementById("demo").innerHTML = text;
</script>
</body>
```

- The loop in this example uses a for loop to collect the car names from the cars array:

Example using for loop

```
<body>
<p id="demo"></p>

<script>
    const cars = ["BMW", "Volvo", "Saab", "Ford"];
    let i = 0;
    let text = "";
    for (;cars[i];) {
```

```
    text += cars[i] + "<br>";
    i++;
  }

document.getElementById("demo").innerHTML = text;
</script>

</body>
```

The For Loop

Syntax:

```
for (initialization; condition; step size) {
  // code block to be executed
}
```

Example:

```
for (let i = 1; i <=5; i++) {
  text += "The number is " + i + "<br>"; }
```

- Loops can execute a block of code a number of times until the specified condition is true.

Note: Often loops are used when working with arrays:

Instead of writing:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
```

```
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

We can write:

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

Variations in for loop:

Example:

1. **More than initialization**

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

2. **Initialization before for loop**

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

3. Increment/Decrement(Step size) inside for loop

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

4. Condition is omitted in for loop

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

if (i >= len )
    break;
i++;
}
```

Note : If condition is omitted in for loop, we must provide a break inside the loop. Otherwise the loop will never end. This will crash your browser.

Loop Scope

➔ **Using var in a loop:**

```
var i = 5;
for (var i = 0; i < 10; i++) {
  // some code     // Original value is changed
}

// Here i is 10
```

➔ **Using let in a loop:**

```
let i = 5;
for (let i = 0; i < 10; i++) {
  // some code  --- Block level value will not affect outside value
}
// Here i is 5
```

JavaScript For In Loop

- The JavaScript for in statement loops through the properties of an Object:

Syntax

```
for (key in object) {
  // code block to be executed
}
```

Example:

```
const person = {fname:"Indra", lname:"Gandhi", age:60};

let text = "";
for (let x in person) {
  text += person[x];
}
```

Explanation

- The **for in** loop iterates over a **person** object
- Each iteration returns a **key** (x)
- The key is used to access the **value** of the key, that is **person[x]**

For In Over Arrays

- The JavaScript for in statement can also loop over the properties of an Array:

Syntax

```
for (variable in array) {
  block of code
}
```

Example:

```
const numbers = [153, 370, 371, 407 ];

let txt = "";
for (let x in numbers) {
  txt += numbers[x];
}
document.write("Armstrong Numbers : " + txt);
```

Note :

- Do not use for in over an Array if the index order is important.
- The **index order is implementation-dependent**, and array values may not be accessed in the order you expect.
- **It is better to use a for loop, a for of loop, or Array.forEach() when the order is important.**

Array.forEach()

- The forEach() method calls a function (a callback function) once for each array element.

Example1:

```
<h2>JavaScript Array.forEach()</h2>
<p>Calls a function once for each array element.</p>
<p id="demo"></p>

<script>
const numbers = [153, 370, 371, 407];
let txt = "";
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = txt;

function myFunction(value, index, array) {
  txt += value + "<br>";
}
</script>
```

Note:
In the above example the function takes 3 arguments:

- The item value
- The item index
- The array itself

Example2:

The example above uses only the value parameter. It can be rewritten to:

```
<h2>JavaScript Array.forEach()</h2>
<p>Calls a function once for each array element.</p>
<p id="demo"></p>
```

```
<script>
const numbers = [153, 370, 371, 407];
let txt = "";
numbers.forEach(myFunction);
document.getElementById("demo").innerHTML = txt;

function myFunction(value) {
  txt += value + "<br>";
} </script>
```

JavaScript For Of Loop

- The JavaScript for of statement loops through the values of an iterable object added to JS in 2015 (ES6)
- It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, etc
- It is not supported in internet explorer

Syntax:

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

- **variable** - For every iteration the value of the next property is assigned to the variable. *Variable* can be declared with const, let, or var.
- **iterable** - An object that has iterable properties.

- **Looping over an Array**

```
const cars = ["BMW", "Volvo", "Audi"];

let text = "";
for (let x of cars) {
  text += x;
}
```

- **Looping over a String**

```
<h2>JavaScript For Of Loop</h2>

<p>The for of statement loops through the values of an iterable object.</p>

<p id="demo"></p>

<script>

let language = "JavaScript";

let text = "";

for (let x of language) {

  text += x + "<br>";

}

document.getElementById("demo").innerHTML = text;

</script>
```

JavaScript Break and Continue

- The `break` statement "jumps out" of a loop.
- The `continue` statement "jumps over" one iteration in the loop by skipping set of statemetns.

break statement

The `break` statement can also be used to jump out of a loop:

Example:

```
<h2>JavaScript Loops</h2>
<p>A loop with a <b>break</b> statement.</p>
<p id="demo"></p>

<script>
let text = "";
for (let i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>
```

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.

This example skips the value of 3:

```
<body>
<h2>JavaScript Loops</h2>

<p>A loop with a <b>continue</b> statement.</p>
<p>A loop which will skip the step where i = 3.</p>
<p id="demo"></p>

<script>
let text = "";
for (let i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
```

JavaScript Labels

- To label JavaScript statements you precede the statements with a label name and a colon:

label:
statements

Note:

The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.

Syntax:

break *labelname*;

continue *labelname*;

- The continue statement (with or without a label reference) can only be used to **skip one loop iteration**.

- The break statement, without a label reference, can only be used to **jump out of a loop or a switch**.

- With a label reference, the break statement can be used to **jump out of any code block**:

```
<body>

<h2>JavaScript break</h2>

<p id="demo"></p>

<script>
const cars = ["BMW", "Volvo", "Audi", "Ford"];
let text = "";

list: {
  text += cars[0] + "<br>";
  text += cars[1] + "<br>";
  break list;
  text += cars[2] + "<br>";
  text += cars[3] + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
```