Chapter 18: Loops

Section 18.1: Standard "for" loops

Standard usage

```
for (var i = 0; i < 100; i++) {
    console.log(i);
}
```

Expected output:

> 0
> 1
> ...
> 99

Multiple declarations

Commonly used to cache the length of an array.

```
var array = ['a', 'b', 'c'];
for (var i = 0; i < array.length; i++) {
    console.log(array[i]);
}
```

Expected output:

> 'a'
> 'b'
> 'c'

Changing the increment

```
for (var i = 0; i < 100; i += 2 /* Can also be: i = i + 2 */) {
    console.log(i);
}
```

Expected output:

> 0
> 2
> 4
> ...
> 98

Decremented loop

```
for (var i = 100; i >=0; i--) {
    console.log(i);
}
```

Expected output:

```
100
99
98
...
0
```

Section 18.2: "for ... of" loop

Version ≥ 6

```
const iterable = [0, 1, 2];
for (let i of iterable) {
    console.log(i);
}
```

Expected output:

```
0
1
2
```

The advantages from the for...of loop are:

- This is the most concise, direct syntax yet for looping through array elements
- It avoids all the pitfalls of for...in
- Unlike `forEach()`, it works with break, continue, and return

Support of for...of in other collections
Strings

for...of will treat a string as a sequence of Unicode characters:

```
const string = "abc";
for (let chr of string) {
  console.log(chr);
}
```

Expected output:

```
a b c
```

Sets

for...of works on Set objects.

Note:

- A Set object will eliminate duplicates.
- Please check this reference for `Set()` browser support.

```
const names = ['bob', 'alejandro', 'zandra', 'anna', 'bob'];

const uniqueNames = new Set(names);

for (let name of uniqueNames) {
  console.log(name);
}
```

Expected output:

> bob
> alejandro
> zandra
> anna

Maps

You can also use for...of loops to iterate over Maps. This works similarly to arrays and sets, except the iteration variable stores both a key and a value.

```
const map = new Map()
  .set('abc', 1)
  .set('def', 2)

for (const iteration of map) {
  console.log(iteration) //will log ['abc', 1] and then ['def', 2]
}
```

You can use destructuring assignment to capture the key and the value separately:

```
const map = new Map()
  .set('abc', 1)
  .set('def', 2)

for (const [key, value] of map) {
  console.log(key + ' is mapped to ' + value)
}
/*Logs:
  abc is mapped to 1
  def is mapped to 2
*/
```

Objects

for...of loops *do not* work directly on plain Objects; but, it is possible to iterate over an object's properties by switching to a for...in loop, or using `Object.keys()`:

```
const someObject = { name: 'Mike' };

for (let key of Object.keys(someObject)) {
  console.log(key + ": " + someObject[key]);
}
```

Expected output:

> name: Mike

Section 18.3: "for ... in" loop

> **Warning**
> for...in is intended for iterating over object keys, not array indexes. Using it to loop through an array is generally discouraged. It also includes properties from the prototype, so it may be necessary to check if the key is within the object using `hasOwnProperty`. If any attributes in the object are defined by the `defineProperty/defineProperties` method and set the param `enumerable: false`, those attributes will be inaccessible.

```
var object = {"a":"foo", "b":"bar", "c":"baz"};
// `a` is inaccessible
Object.defineProperty(object , 'a', {
        enumerable: false,
});
for (var key in object) {
    if (object.hasOwnProperty(key)) {
      console.log('object.' + key + ', ' + object[key]);
    }
}
```

Expected output:

> object.b, bar
> object.c, baz

Section 18.4: "while" Loops

Standard While Loop

A standard while loop will execute until the condition given is false:

```
var i = 0;
while (i < 100) {
    console.log(i);
    i++;
}
```

Expected output:

> 0
> 1
> ...
> 99

Decremented loop

```
var i = 100;
while (i > 0) {
```

```
    console.log(i);
    i--; /* equivalent to i=i-1 */
}
```

Expected output:

> 100
> 99
> 98
> ...
> 1

Do...while Loop

A do...while loop will always execute at least once, regardless of whether the condition is true or false:

```
var i = 101;
do {
    console.log(i);
} while (i < 100);
```

Expected output:

> 101

Section 18.5: "continue" a loop

Continuing a "for" Loop

When you put the `continue` keyword in a for loop, execution jumps to the update expression (`i++` in the example):

```
for (var i = 0; i < 3; i++) {
    if (i === 1) {
        continue;
    }
    console.log(i);
}
```

Expected output:

> 0
> 2

Continuing a While Loop

When you `continue` in a while loop, execution jumps to the condition (`i < 3` in the example):

```
var i = 0;
while (i < 3) {
    if (i === 1) {
        i = 2;
        continue;
```

```
    }
    console.log(i);
    i++;
}
```

Expected output:

```
0
2
```

Section 18.6: Break specific nested loops

We can name our loops and break the specific one when necessary.

```
outerloop:
for (var i = 0;i<3;i++){
    innerloop:
    for (var j = 0;j <3; j++){
        console.log(i);
        console.log(j);
        if (j == 1){
            break outerloop;
        }
    }
}
```

Output:

```
0
0
0
1
```

Section 18.7: "do ... while" loop

```
var availableName;
do {
    availableName = getRandomName();
} while (isNameUsed(name));
```

A `do while` loop is guaranteed to run at least once as it's condition is only checked at the end of an iteration. A traditional `while` loop may run zero or more times as its condition is checked at the beginning of an iteration.

Section 18.8: Break and continue labels

Break and continue statements can be followed by an optional label which works like some kind of a goto statement, resumes execution from the label referenced position

```
for(var i = 0; i < 5; i++){
  nextLoop2Iteration:
  for(var j = 0; j < 5; j++){
    if(i == j) break nextLoop2Iteration;
    console.log(i, j);
  }
```

```
}
```

> ***i=0 j=0 skips rest of j values***
> 1 0
> ***i=1 j=1 skips rest of j values***
> 2 0
> 2 1 ***i=2 j=2 skips rest of j values***
> 3 0
> 3 1
> 3 2
> ***i=3 j=3 skips rest of j values***
> 4 0
> 4 1
> 4 2
> 4 3
> ***i=4 j=4 does not log and loops are done***