JavaScript Loops

Loops can execute a block of code a number of times. Loops are handy, if you want to run the same code over and over again, each time with a different value.

When working with arrays, instead of writing:

```
text += cars[0]";
text += cars[1]";
text += cars[2]";
text += cars[3]";
text += cars[4]";
text += cars[5]";
```

You can write:

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

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

The For Loop

The `for` statement creates a loop with 3 optional expressions:

```
for (expression 1; expression 2; expression 3) {
  // 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";
}
```

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.

How to use Expression 1

Expression 1 is used to initialize the variable(s) used in the loop (let i = 0). But, expression 1 is optional.You can omit expression 1 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]";
}
```

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

Example

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

How to use Expression 2

Expression 2 is used to evaluate the condition of the initial variable (i < len). But, expression 2 is also optional.

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

Note: 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.

How to use Expression 3

- Expression 3 increments the value of the initial variable (i++). But, expression 3 is also 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]";
  i++;
}
```

Loop Scope

Using `var` in a loop:

Example

```
var i = 5;

for (var i = 0; i < 10; i++) {
  // some code
}

// Here i is 10
```

Using `let` in a loop:

Example

```
let i = 5;

for (let i = 0; i < 10; i++) {
  // some code
}
```

```
// Here i is 5
```

In the first example, using `var`, the variable declared in the loop redeclares the variable outside the loop.

In the second example, using `let`, the variable declared in the loop does not redeclare the variable outside the loop.

When `let` is used to declare the i variable in a loop, the i variable will only be visible within the loop.