[Lesson 4]

Roi Yehoshua 2018

[DAN.IT EDUCATION]

What we learnt last time?

- JavaScript general operators
- Logic operators
- If-else construction
- Switch statement

[Our targets for today]

- What is loop?
- Types of loops in Javascript
- Breaking the loop
- Jumping to the next loop iteration

[DAN.IT] EDUCATION

Loops

- We often have a need to perform similar actions many times in a row
- For example, when we need to output goods from a list one after another. Or just
- run the same code for each number from 1 to 10.
- *Loops* are a way to repeat the same part of code multiple times
- There are three loop types in JavaScript:
  - While loops
  - Do-while loops
  - For loops

While Loop

→ The while loop has the following syntax:

```
while (condition) {
    // the loop body
}
```

→ While the condition is true, the code from the loop body is executed

→ For instance, the loop below outputs i while i < 5:

```
let i = 0;
while (i < 5) { // shows 0, 1, ..., 4
    alert(i);
    i++;
}
```

→ A single execution of the loop body is called *an iteration*

→ The loop in the example above makes 5 iterations

[DAN.IT] EDUCATION

While Loop

→ Any expression or a variable can be a loop condition, not just a comparison. They are evaluated and converted to a boolean by while.

→ For instance, the shorter way to write while (i != 0) could be while (i)

```
let i = 3;
while (i) { // when i becomes 0, the condition becomes falsy, and the loop
stops
    alert(i);
    i--;
}
```

⟶ If the loop body has a single statement, we can omit the brackets {…}:

```
let i = 3;
while (i) alert(i--);
```

The “do…while” loop

→ The condition check can be moved *below* the loop body using the do..while syntax:

```
do {
    // loop body
}
while (condition);
```

→ The loop will first execute the body, then check the condition and, while it’s truthy, execute it again and again

→ For example:

```
let i = 0;  do {
    alert(i);  i++;
} while (i < 3)
```

→ This form of syntax is rarely used except when you want the body of the loop to execute **at least once** regardless of the condition being truthy

For Loop

→ The for loop has the following syntax:

```
for (begin; condition; step) {
    // ... loop body ...
}
```

- → **begin** is executed once before entering the loop
- → **condition** is checked before every loop iteration, if fails the loop stops
- → **step** is executed after the body on each iteration, but before the condition check

→ For instance, the following loop runs alert(i) for i from 0 up to (but not including) 3:

```
for (let i = 0; i < 3; i++) {
    alert(i); // 0, 1, 2
}
```

```
// run begin  let i = 0
// if condition → run body and run step  if
(i < 3) { alert(i); i++ }
// if condition → run body and run step
if (i <  3)  { alert(i); i++ }
// if condition → run body and run step  if
(i < 3) { alert(i); i++ }
// ...finish, because now i == 3
```

For Loop - Inline Variable Declaration

→ Here the “counter” variable i is declared right in the loop

→ That’s called an “inline” variable declaration

→ Such variables are visible only inside the loop

```
for (let i = 0; i < 3; i++) {
    alert(i); // 0, 1, 2
}
alert(i); // error, no such variable
```

→ Instead of defining a variable, we can use an existing one:

```
let i = 0;

for (i = 0; i < 3; i++) { // use an existing variable
    alert(i); // 0, 1, 2
}

alert(i); // 3, visible, because declared outside of the loop
```

For Loop - Skipping Parts

→ Any part of **for** can be skipped

→ For example, we can omit begin if we don't need to do anything at the loop start

```
let i = 0; // we have i already declared and assigned

for (; i < 3; i++) { // no need for "begin"  alert(i);
    // 0, 1, 2
}
```

→ We can also remove the step part:

```
let i = 0;

for (; i < 3;) {
    alert(i++);
}
```

→ The loop became identical to while (i < 3)

[DAN.IT] EDUCATION

For Loop - Skipping Parts

→ We can actually remove everything, thus creating an infinite loop:

```
for (; ;) {
    // repeats without limits
}
```

→ Note that the two for semicolons ; must be present, otherwise it would be a syntax error

Breaking the Loop

- → Normally the loop exits when the condition becomes falsy
- → But we can force the exit at any moment using the break directive
- → For example, the loop below asks the user for a series of numbers, but “breaks” when no number is entered:

```
let sum = 0;
while (true) {
    let num = Number(prompt("Enter a number", ''));
    if (!num) break; // (*)
    sum += value;
}
alert('Sum: ' + sum);
```

  - → The break directive is activated at the line (*) if the user enters an empty line or cancels the input
  - → It stops the loop immediately, passing the control to the first line after the loop. Namely, alert.
- → The combination “infinite loop + break as needed” is great for situations when the condition must be checked not in the beginning/end of the loop, but in the middle

[DAN.IT] EDUCATION

Continue to the Next Iteration

→ The continue directive doesn't stop the whole loop. Instead it stops the current iteration and forces the loop to start a new one (if the condition allows).

→ We can use it if we're done on the current iteration and would like to move on to the next

→ The loop below uses continue to output only odd values:

```
for (let i = 0; i < 10; i++) {

    // if true, skip the remaining part of the body
    if (i % 2 == 0) continue;

    alert(i); // 1, then 3, 5, 7, 9
}
```

→ The directive continue helps to decrease nesting level

Control questions

1. In which cases do we use loops?
2. What is the difference between “while” and “do while” loops?
3. How does “for” loop work?
4. How to skip code block execution to the next iteration?
5. How to break cycle in the code block?

Materials

Core materials:
https://learn.javascript.ru/while-for

Video materials:
https://youtu.be/ITr-SzUIDpQ