JS | Conditionals and Loops

Learning Goals

After this lesson, you will be able to:

- Understand what statements are
- Understand why conditionals are necessary
- Understand and use the `if..else` statement
- Understand and use the `switch` statement
- Identify when it's better to use `switch` over `if..else`
- Understand why the loops and iterations are necessary
- Understand and use the `while` statement
- Understand and use the `for` statement

Statements - general overview

In JavaScript we write programs as a `sequence of statements` - so we can say that **statements are code snippets that perform some action**.

Some examples of statements are:

- **if…else**
- **switch**
- **do…while**
- **for**
- **forEach**
- **break**
- **while**

You can see the full list on the official MDN site.

Conditional Statements

During our life, we have to make decisions every day. What to wear for a job interview? Which train to take to get faster to work? What to buy for your partner for our anniversary? Some of them are more important than others, but they all have something in common: we have to decide what to do.

In programming, we have to make some decisions too. In our applications we will have to give more than one option to our user so they have to be able to choose - all this means we have to know how and when to use **conditional statements**.

`if..else`

The `if` statement **executes a statement if the specified condition is true**. If the condition is false, another statement can be executed.

```
if (condition) {
  // code to execute if the condition is true
}
```

We can add to our `if` an `else` statement, that will be executed just if the condition is not true.

```
if (condition) {
  // code to execute if the condition is true
} else {
  // code to execute if the condition is false
}
```

Sometimes, we have more than one option or better saying, more than one condition to check before we make a decision and for this we use the `else if` statement.

```
if (condition1) {
  // code to execute if condition1 is true
} else if (condition2) {
  // code to execute if condition2 is true
} else {
  // code to execute if condition1 and condition2 are false
}
```

Let's create a simple example to see how the `if ... else` statement works. Let's create a script to check if we have any kind of discount in the cinema. We will have a discount if we are younger than 16, or older than 65.

```
const age = parseInt(prompt("Welcome to Ironhack cinema. How old are you?"));

if (age <= 16) {
  console.log ("You have a teenager discount.");
} else if (age >= 65) {
  console.log ("You have a senior citizen discount :)");
} else {
  console.log ("Sorry, you don't have any discount :(");
}
```

Easy, right? Okay, let's complicate it a little bit. We can have nested `if..else` statements. The syntax is as it follows:

```
if (condition) {
  if (nestedCondition) {
    // The code will be executed if
    // condition === true && nestedCondition === true
  } else {
    // The code will be executed if
    // condition === true && nestedCondition === false
  }
} else {
  // The code will be executed if
  // condition === false
}
```

Let's play around with two numbers to see how nested `if` works:

```
const number1 = parseInt (prompt ("First number:"));
const number2 = parseInt (prompt ("Second number:"));

if (number1 === number2) {
  console.log ("The numbers are equal.");
} else {
  if (number1 > number2) {
    console.log("Number 1 is bigger than number 2.");
  } else {
    console.log("Number 1 is smaller than number 2.");
  }
}
```

Time to practice - Custom Hello, world!

Let’s write an improved version of the ‘Hello, world!’ program. Let’s ask the user in which language they want to see the message. You must have, at least, three different languages.

- If the user wants the message in Spanish, you have to log in the console **“Hola, mundo!”**.
- If the user wants the message in French, you have to log in the console **“Bonjour tout le monde”.**
- Finally, if we don’t have the indicated language, we will show **“Hello, world!”**.

Use the if…else and else if statements we have just learned

Too many conditions

Sometimes, an `if` statement can become very complicated. Let’s suppose we want to figure out the house of our favorite main character of Game of Thrones. We could do something like this:

```
const name  = prompt ("Favorite Game of Thrones main character:");
let house = "";

if (name === "Khal Drogo") {
  house = "Dothraki Horselord";
} else if (name === "Daenerys") {
  house = "Targaryen";
} else if (name === "Jon Snow" || name === "Sansa" || name === "Arya") {
  house = "Stark";
} else if (name === "Cersei" || name === "Tyrion" || name === "Ser Jaime") {
  house = "Lannister";
} else {
  house = "Other";
}

console.log(`Your favorite character is from the house ${house}.`);
```

This solution works but **not everything that works is the correct/best solution**. We have too many `else if` statements.

`switch`

```
switch (expression) {
  case value1:
    // executed code when the expression === value1
    break;
  case value2:
    // executed code when the expression === value2
    break;
  case value3:
    // executed code when the expression === value3
    break;
  default:
    // executed code when none of the values match the expression
}
```

The `switch` statement helps us to do exactly the same thing we did previously in the *"Game of Thrones"* example. **It evaluates an expression,or better saying - checks the condition and executes statements associated with that case**.

Let's change the previous Game of Thrones example to see how `switch` works. We will go step by step:

STEP 1 - Khal Drogo

```
const name  = prompt ("Favorite Game of Thrones main character:");
let house = "";

switch (name) {
  case "Khal Drogo":
    house = "Dothraki Horselord";
}

console.log(`Your favorite character is from the house ${house}.`);
```

If we pass "Khal Drogo" as the answer, we will get as a result "Your favorite character is from the house Dothraki Horselord". Cool, huh? :)

STEP 2 - Daenerys

```javascript
const name  = prompt ("Favorite Game of Thrones main character:");
let house = "";

switch (name) {
  case "Khal Drogo":
    house = "Dothraki Horselord";
  case "Daenerys":
    house = "Targaryen";
}

console.log(`Your favorite character is from the house ${house}.`);
```

Now, if you introduce “Daenerys”, you will get as a result “Your favorite character is from the house Targaryen”. But… what happens if you say “Khal Drogo”? “Khal Drogo” is not from Targaryen house!

We have to add a **`break` statement** at the end of each `case` clause. What is `break` for?

`break` statement terminates the current switch statement, or better saying, the current case.

The problem we have right now is that we are running both `case` statements, and the second one sets the `house` variable to “Targaryen”. If we add the `break` statement at the end of each `case`, the problem will be solved.

```javascript
const name  = prompt ("Favorite Game of Thrones main character:");
let house = "";

switch (name) {
  case "Khal Drogo":
    house = "Dothraki Horselord";
    break;
  case "Daenerys":
    house = "Targaryen";
    break;
}

console.log(`Your favorite character is from the house ${house}.`);
```

All right, this is much better, right?

STEP 3 - Stark House

Now we have a `case` with several options. Jon Snow, Arya and Sansa are from Stark House. What happens if we set the `case` expression like the following:

```
case "Jon Snow" || "Sansa" || "Arya"
```

Does it work for all the options? No, right? It just works with the first option. In this case, it just works as we expect with “Jon Snow”.

❗ **We can’t have several options in the same `case` statement, but we can set the same code to execute for different `case`.**

Let’s take a look at how to do that:

```
const name  = prompt ("Favorite Game of Thrones main character:");
let house = "";

switch (name) {
  case "Khal Drogo":
    house = "Dothraki Horselord";
    break;
  case "Daenerys":
    house = "Targaryen";
    break;
  case "Jon Snow":
  case "Sansa":
  case "Arya":
    house = "Stark";
    break;
}

console.log(`Your favorite character is from the house ${house}.`);
```

Now we have our favorite Starks in our super amazing script 😄

📝 **Time to practice**

Use the same technique to add the Lannister house to our script. Don’t forget to add the `break` statement to avoid conflicts between houses. We don’t want a house misunderstanding that will cause a war!

STEP 4 - Other characters

We all know that Game of Thrones is an entire world. We can't consider all the characters here: it would take a lot of time, and we have more stuff to cover. So let's create the last statement to consider all the other houses. We use the `default` statement to do that:

```
const name  = prompt ("Favorite Game of Thrones main character:");
let house = "";

switch (name) {
  case "Khal Drogo":
    house = "Dothraki Horselord";
    break;
  case "Daenerys":
    house = "Targaryen";
    break;
  case "Jon Snow":
  case "Sansa":
  case "Arya":
    house = "Stark";
    break;
  default:
    house = "other";
}

console.log(`Your favorite character is from the house ${house}.`);
```

And that's it! This is how `switch` works. Wait… we have forgotten the `break` in the `default` statement, right? Do you remember what the `break` statement does?

As we have seen, the `break` statement terminates the current switch statement and transfers our script just at the end of the `switch` statement. **`default` is our last statement inside the `switch`, so it's not necessary to add the `break` statement in this case**.

📝 **Time to practice - Even more custom `Hello, world!`**

Improve your custom hello, world! exercise by using a `switch` statement instead of `if..else`. Now it's more practical to have more than three languages. Add a few more languages to your program. 😉

Loops & Iterations

In JavaScript, loops and iterations are used to execute repetitive tasks.

For example, if we want to print numbers from 1 to 100, we won't type them all. We will use `while` or `for` statement to do that. They both offer us a quick and easy way to do something repeatedly.

`while` statement

```
while (condition) {
  // code to be executed while the condition is true
}
```

`while` statement creates a loop that executes a specified code as long as the condition evaluates true.

Let's print in the console the numbers from 0 to 100 with a `while` statement. The code is the following:

```
let i = 0;

while (i <= 100) {
  console.log (i);
  i++;  // this is the same as i = i + 1
}
```

The script follows the following rules:

1. Check if the value of `i` is lower or equal than 100
2. If yes, print in the console the value of `i`
3. Increment the value of the `i` by 1
4. Again check if the value of `i` is lower or equal to 100 and as long as it is, executes these steps over and over again.

`do while` statement

The `do/while` statement creates a loop that executes a block of code once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

The `do/while` statement is used when you want to run a loop at least one time, no matter what.

```
do {
    // block of code to be executed
} while (condition);
```

Time to practice

Let’s do a small exercise to practice the new stuff we have learned in this lesson. The exercise will have two small iterations:

Iteration 1 - Create a script that counts from 0 to 30. Expected output:

1

2

3

…

28

29

30

Iteration 2 - Change the previous script to write “ten” when the value of `i` is 10, and “twenty” when the value if 20. Expected output:

1

…

9

ten

11

…

19

twenty

21

…

30

How can you modify the basic flow of the script?

`for` statement

The `for` statement creates a loop with three different values separated by semicolons: **initialization, condition and final expression**.

```
for (initialization; condition; finalExpression) {
  // code to execute
}
```

The **initialization** is the variable declaration. The **condition** is what has to happen to finish the iteration. The **final expression** is the increment of the variable used to iterate our code.

Let's do the same exercise than before, this time using a `for` statement. Let's print in the console the numbers from 0 to 100. The code is:

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

As you can see, you don't have to increment the variable inside the statement. If you forget to iterate the variable inside a `while`, you can create an infinite loop.

Continue and Break statements

Break Statement

The `break` statement exits a switch statement or a loop (for, for … in, while, do … while).

When the `break` statement is used with a switch statement, it breaks out of the switch block.

When the `break` statement is used in a loop, it breaks the loop and continues executing the code after the loop (if there's any code after).

```
let i = 0;

while (i < 5) {
  i++;
  console.log(`The number is: ${i}.`);
  if (i === 3) {
    break;
  }
}
```

The result of text will be

```
The number is: 1.
The number is: 2.
The number is: 3.
```

💡 Please note that `break` statement does not exit an `if`, it only exits the loops!

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.

❗ The difference between `continue` and the `break` statement, is instead of "jumping out" of a loop, the continue statement "jumps over" one iteration in the loop.

However, when the `continue` statement is executed, it behaves differently for different types of loops:

- In a while loop, the condition is tested, and if it is true, the loop is executed again

- In a for loop, the increment expression (e.g. i++) is first evaluated, and then the condition is tested to find out if another iteration should be done

```
let i = 0;

while (i < 5) {
  i++;
  if (i === 3) {
    continue;
  }
  console.log(`The number is: ${i}.`);
}
```

The result of text will be

```
The number is: 1.
The number is: 2.
The number is: 4.
The number is: 5.
```

Time to practice

Let's write a loop that will iterate from 0 to 20. For each iteration, it will check if the current number is even or odd, and print that on the screen. You will need the remainder operator to do the exercise. Expected output:

1 is odd

2 is even

…

19 is odd

20 is even

Summary

We use boolean values (true or false) in the conditional statements like `if..else` or `switch`, and depending in the condition we know which block of code will execute.

If you are evaluating one condition and you have too many options to check, the best solution is to change the `if..else` statement for a `switch` statement.

Finally, we have learned how to iterate several times with two different statements, `while` and `for`.

Extra Resources

- MDN - Statements and declarations
- MDN - Logical operators
- MDN - `if...else` statement
- MDN - `switch` statement
- MDN - `for` statement
- MDN - `while` statement