

[Home](#) → [The JavaScript language](#) → [JavaScript Fundamentals](#)

06 May 2019

Conditional operators: if, '?'

Sometimes, we need to perform different actions based on different conditions.

To do that, we can use the `if` statement and the conditional operator `?`, that's also called a “question mark” operator.

The “if” statement

The `if` statement evaluates a condition and, if the condition's result is `true`, executes a block of code.

For example:

```
1 let year = prompt('In which year was ECMAScript-2015 specification published?');
2
3 if (year == 2015) alert('You are right!');
```

In the example above, the condition is a simple equality check (`year == 2015`), but it can be much more complex.

If we want to execute more than one statement, we have to wrap our code block inside curly braces:

```
1 if (year == 2015) {
2   alert('That's correct!');
3   alert('You're so smart!');
4 }
```

We recommend wrapping your code block with curly braces `{}` every time you use an `if` statement, even if there is only one statement to execute. Doing so improves readability.

Boolean conversion

The `if (...)` statement evaluates the expression in its parentheses and converts the result to a boolean.

Let's recall the conversion rules from the chapter [Type Conversions](#):

- A number `0`, an empty string `""`, `null`, `undefined`, and `NaN` all become `false`. Because of that they are called “falsy” values.
- Other values become `true`, so they are called “truthy”.

So, the code under this condition would never execute:

```
1 if (0) { // 0 is falsy
2   ...
3 }
```

...and inside this condition – it always will:

```
1 if (1) { // 1 is truthy
2   ...
3 }
```

We can also pass a pre-evaluated boolean value to `if`, like this:

```
1 let cond = (year == 2015); // equality evaluates to true or false
2
3 if (cond) {
4   ...
5 }
```

The “else” clause

The `if` statement may contain an optional “else” block. It executes when the condition is false.

For example:

```
1 let year = prompt('In which year was the ECMAScript-2015 specification published?');
2
3 if (year == 2015) {
4   alert( 'You guessed it right!' );
5 } else {
6   alert( 'How can you be so wrong?' ); // any value except 2015
7 }
```

Several conditions: “else if”

Sometimes, we’d like to test several variants of a condition. The `else if` clause lets us do that.

For example:

```
1 let year = prompt('In which year was the ECMAScript-2015 specification published?');
2
3 if (year < 2015) {
4   alert( 'Too early...' );
5 } else if (year > 2015) {
6   alert( 'Too late' );
7 } else {
8   alert( 'Exactly!' );
9 }
```

In the code above, JavaScript first checks `year < 2015` . If that is falsy, it goes to the next condition `year > 2015` . If that is also falsy, it shows the last `alert` .

There can be more `else if` blocks. The final `else` is optional.

Conditional operator ‘?’

Sometimes, we need to assign a variable depending on a condition.

For instance:

```
1 let accessAllowed;
2 let age = prompt('How old are you?', '');
3
4 if (age > 18) {
5   accessAllowed = true;
6 } else {
7   accessAllowed = false;
8 }
9
10 alert(accessAllowed);
```

The so-called “conditional” or “question mark” operator lets us do that in a shorter and simpler way.

The operator is represented by a question mark `?` . Sometimes it’s called “ternary”, because the operator has three operands. It is actually the one and only operator in JavaScript which has that many.

The syntax is:

```
1 let result = condition ? value1 : value2;
```

The `condition` is evaluated: if it’s truthy then `value1` is returned, otherwise – `value2` .

For example:

```
1 let accessAllowed = (age > 18) ? true : false;
```

Technically, we can omit the parentheses around `age > 18` . The question mark operator has a low precedence, so it executes after the comparison `>` .

This example will do the same thing as the previous one:

```
1 // the comparison operator "age > 18" executes first anyway
2 // (no need to wrap it into parentheses)
3 let accessAllowed = age > 18 ? true : false;
```

But parentheses make the code more readable, so we recommend using them.

Please note:

In the example above, you can avoid using the question mark operator because the comparison itself returns `true/false`:

```
1 // the same
2 let accessAllowed = age > 18;
```

Multiple ‘?’

A sequence of question mark operators `?` can return a value that depends on more than one condition.

For instance:

```
1 let age = prompt('age?', 18);
2
3 let message = (age < 3) ? 'Hi, baby!' :
4   (age < 18) ? 'Hello!' :
5   (age < 100) ? 'Greetings!' :
6   'What an unusual age!';
7
8 alert( message );
```

It may be difficult at first to grasp what's going on. But after a closer look, we can see that it's just an ordinary sequence of tests:

1. The first question mark checks whether `age < 3`.
2. If true – it returns `'Hi, baby!'`. Otherwise, it continues to the expression after the colon `“:”`, checking `age < 18`.
3. If that's true – it returns `'Hello!'`. Otherwise, it continues to the expression after the next colon `“:”`, checking `age < 100`.
4. If that's true – it returns `'Greetings!'`. Otherwise, it continues to the expression after the last colon `“:”`, returning `'What an unusual age!'`.

Here's how this looks using `if..else`:

```
1 if (age < 3) {
2   message = 'Hi, baby!';
3 } else if (age < 18) {
4   message = 'Hello!';
5 } else if (age < 100) {
6   message = 'Greetings!';
7 } else {
8   message = 'What an unusual age!';
9 }
```

Non-traditional use of ‘?’

Sometimes the question mark `?` is used as a replacement for `if`:

```
1 let company = prompt('Which company created JavaScript?', '');
2
3 (company == 'Netscape') ?
4   alert('Right!') : alert('Wrong.');
```

Depending on the condition `company == 'Netscape'`, either the first or the second expression after the `?` gets executed and shows an alert.

We don't assign a result to a variable here. Instead, we execute different code depending on the condition.

We don't recommend using the question mark operator in this way.

The notation is shorter than the equivalent `if` statement, which appeals to some programmers. But it is less readable.

Here is the same code using `if` for comparison:

```
1 let company = prompt('Which company created JavaScript?', '');
2
3 if (company == 'Netscape') {
4   alert('Right!');
5 } else {
6   alert('Wrong.');
```

Our eyes scan the code vertically. Code blocks which span several lines are easier to understand than a long, horizontal instruction set.

The purpose of the question mark operator `?` is to return one value or another depending on its condition. Please use it for exactly that. Use `if` when you need to execute different branches of code.

✓ Tasks

if (a string with zero) [↗](#)

importance: 5

Will `alert` be shown?

```
1 if ("0") {
2   alert( 'Hello' );
3 }
```

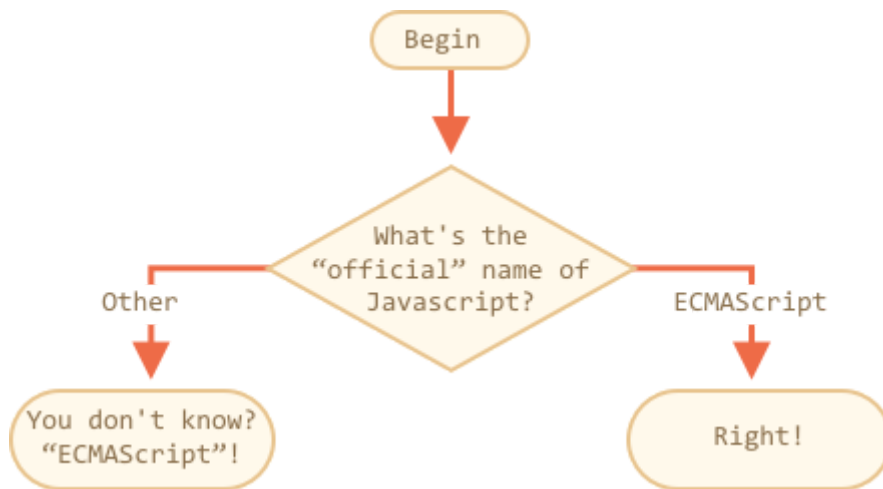
solution

The name of JavaScript [↗](#)

importance: 2

Using the `if...else` construct, write the code which asks: 'What is the "official" name of JavaScript?'

If the visitor enters "ECMAScript", then output "Right!", otherwise – output: "Didn't know? ECMAScript!"



[Demo in new window](#)

[solution](#)

Show the sign [↗](#)

importance: 2

Using `if...else`, write the code which gets a number via `prompt` and then shows in `alert`:

- `1`, if the value is greater than zero,
- `-1`, if less than zero,
- `0`, if equals zero.

In this task we assume that the input is always a number.

[Demo in new window](#)

[solution](#)

Rewrite 'if' into '?' [↗](#)

importance: 5

Rewrite this `if` using the ternary operator `'?'`:

```
1 if (a + b < 4) {  
2   result = 'Below';  
3 } else {  
4
```

```
5   result = 'Over';  
}
```

solution

Rewrite 'if..else' into '?' [↗](#)

importance: 5

Rewrite `if..else` using multiple ternary operators `'?'`.

For readability, it's recommended to split the code into multiple lines.

```
1  let message;  
2  
3  if (login == 'Employee') {  
4    message = 'Hello';  
5  } else if (login == 'Director') {  
6    message = 'Greetings';  
7  } else if (login == '') {  
8    message = 'No login';  
9  } else {  
10   message = '';  
11 }
```

solution



Previous lesson

Next lesson



Share  

 [Tutorial map](#)

Comments

- You're welcome to post additions, questions to the articles and answers to them.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox ([plnkr](#), [JSBin](#), [codepen](#)...)
- If you can't understand something in the article – please elaborate.

