December 7, 2022

Conditional branching: 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 in parentheses and, if the result is `true`, executes a block of code.

For example:

```
let year = prompt('In which year was ECMAScript-2015 specification published?',

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:

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

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:

```
if (0) { // 0 is falsy
  ...
}
```

…and inside this condition – it always will:

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

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

```
let cond = (year == 2015); // equality evaluates to true or false

if (cond) {
  ...
}
```

The “else” clause

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

For example:

```
let year = prompt('In which year was the ECMAScript-2015 specification publishe

if (year == 2015) {
  alert( 'You guessed it right!' );
} else {
  alert( 'How can you be so wrong?' ); // any value except 2015
}
```

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:

```
let year = prompt('In which year was the ECMAScript-2015 specification publishe

if (year < 2015) {
  alert( 'Too early...' );
} else if (year > 2015) {
  alert( 'Too late' );
} else {
  alert( 'Exactly!' );
}
```

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:

```
let accessAllowed;
let age = prompt('How old are you?', '');

if (age > 18) {
  accessAllowed = true;
} else {
  accessAllowed = false;
}

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:

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

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

For example:

```
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:

```
// the comparison operator "age > 18" executes first anyway
// (no need to wrap it into parentheses)
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`:
>
> ```
> // the same
> let accessAllowed = age > 18;
> ```

Multiple '?'

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

For instance:

```
let age = prompt('age?', 18);

let message = (age < 3) ? 'Hi, baby!' :
  (age < 18) ? 'Hello!' :
  (age < 100) ? 'Greetings!' :
  'What an unusual age!';

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`:

```
if (age < 3) {
  message = 'Hi, baby!';
} else if (age < 18) {
  message = 'Hello!';
} else if (age < 100) {
  message = 'Greetings!';
} else {
  message = 'What an unusual age!';
}
```

Non-traditional use of '?'

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

```
let company = prompt('Which company created JavaScript?', '');

(company == 'Netscape') ?
   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.

It's not recommended to use 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:

```
let company = prompt('Which company created JavaScript?', '');

if (company == 'Netscape') {
  alert('Right!');
} else {
  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?

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

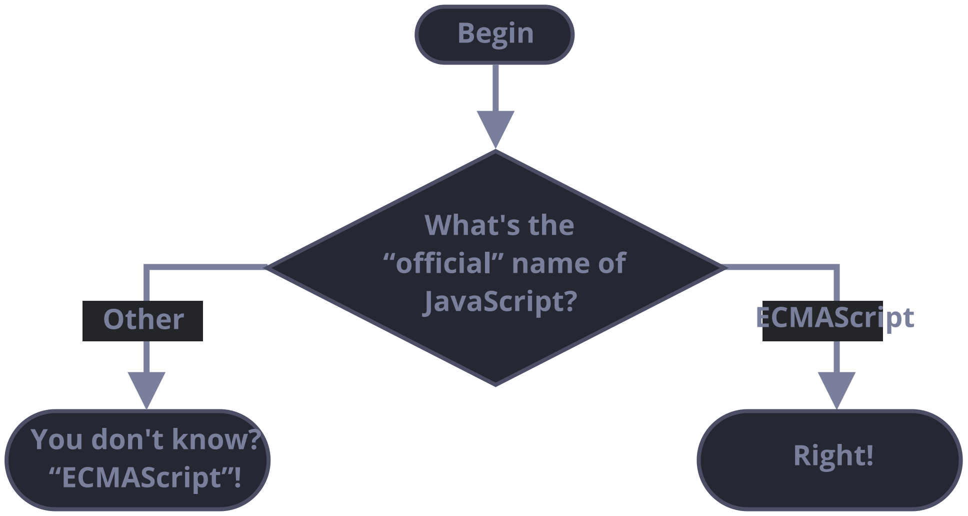
solution

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: "You don't know? ECMAScript!"


Begin
What's the "official" name of JavaScript?
Other
ECMAScript
You don't know? "ECMAScript"!
Right!

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 conditional operator `'?'` :

```
let result;

if (a + b < 4) {
  result = 'Below';
} else {
  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.

```
let message;

if (login == 'Employee') {
  message = 'Hello';
} else if (login == 'Director') {
  message = 'Greetings';
} else if (login == '') {
  message = 'No login';
} else {
  message = '';
}
```

solution

Previous lesson Next lesson

Share Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert few words of code, use the `<code>` tag, for several lines – wrap them in `<pre>` tag, for more than 10 lines – use a sandbox (plnkr, jsbin, codepen…)