Learn web development

Making decisions in your code — conditionals

Overview: Building blocks

Next

In any programming language, code needs to make decisions and carry out actions accordingly depending on different inputs. For example, in a game, if the player's number of lives is 0, then **it's game over. In a weather app, if it is being looked at in the morning, show a sunrise graphic; show stars and a moon if it is nighttime. In this article we'll explore how so-called conditional statements work in JavaScript.**

Prerequisites:	Basic computer literacy, a basic understanding of HTML and CSS, JavaScript first steps.
Objective:	To understand how to use conditional structures in JavaScript.

You can have it on one condition..!

Human beings (and other animals) make decisions all the time that affect their lives, from small ("should I eat one cookie or two?") to large ("should I stay in my home country and work on my father's farm, or should I move to America and study astrophysics?")

Conditional statements allow us to represent such decision making in JavaScript, from the choice that must be made (e.g. "one cookie or two"), to the resulting outcome or those choices (perhaps the outcome of "ate one cookie" might be "still felt hungry", and the outcome of "ate two cookies" might be "felt full up, but mom scolded me for eating all the cookies".)

if ... else statements

Let's look at by far the most common type of conditional statement you'll use in JavaScript — the humble `if ... else` statement.

Basic if ... else syntax

Basic `if...else` syntax looks like so in pseudocode:

```
if (condition) {
  code to run if condition is true
} else {
  run some other code instead
}
```

Here we've got:

1. The keyword `if` followed by some parentheses.
2. A condition to test, placed inside the parentheses (typically "is this value bigger than this other value", or "does this value exist"). This condition will make use of the comparison operators we discussed in the last module, and will return `true` or `false`.
3. A set of curly braces, inside which we have some code — this can be any code we like, and will only be run if the condition returns `true`.
4. The keyword `else`.
5. Another set of curly braces, inside which we have some more code — this can be any code we

like, and will only be run if the condition is not `true`.

This code is pretty human-readable — it is saying "**if** the **condition** returns `true`, run code A, **else** run code B"

You should note that you don't have to include the `else` and the second curly brace block — the following is also perfectly legal code:

```
if (condition) {
  code to run if condition is true
}

run some other code
```

However, you need to be careful here — in this case, the second block of code is not controlled by the conditional statement, so it will **always** run, regardless of whether the condition returns `true` or `false`. This is not necessarily a bad thing, but it might not be what you want — often you will want to run one block of code *or* the other, not both.

As a final point, you may sometimes see `if...else` statements written without the curly braces, in the following shorthand style:

```
if (condition) code to run if condition is true
else run some other code instead
```

This is perfectly valid code, but using it is not recommended — it is much easier to read the code and work out what is going on if you use the curly braces to delimit the blocks of code, and use multiple lines and indentation.

A real example

To understand this syntax better, let's consider a real example. Imagine a child being asked for help with a chore by their mother or father. The parent might say "Hey sweetheart, if you help me by going and doing the shopping, I'll give you some extra allowance so you can afford that toy you wanted." In JavaScript, we could represent this like so:

```
var shoppingDone = false;

if (shoppingDone === true) {
  var childsAllowance = 10;

```

```
} else {
  var childsAllowance = 5;
}
```

This code as shown will always result in the `shoppingDone` variable returning `false`, meaning disappointment for our poor child. It'd be up to us to provide a mechanism for the parent to set the `shoppingDone` variable to `true` if the child did the shopping.

Note: You can see a more complete version of this example on GitHub (also see it running live.)

else if

The last example provided us with two choices, or outcomes — but what if we want more than two?

There is a way to chain on extra choices/outcomes to your `if...else` — using `else if`. Each extra choice requires an additional block to put in between `if() { ... }` and `else { ... }` — check out the following more involved example, which could be part of a simple weather forecast application:

```
<label for="weather">Select the weather type today: </label>
<select id="weather">
  <option value="">--Make a choice--</option>
  <option value="sunny">Sunny</option>
  <option value="rainy">Rainy</option>
  <option value="snowing">Snowing</option>
  <option value="overcast">Overcast</option>
</select>

<p></p>
```

```
var select = document.querySelector('select');
var para = document.querySelector('p');

select.addEventListener('change', setWeather);

function setWeather() {
  var choice = select.value;

  if (choice === 'sunny') {
    para.textContent = 'It is nice and sunny outside today. Wear shorts! Go t
  } else if (choice === 'rainy') {
    para.textContent = 'Rain is falling outside; take a rain coat and a broll

```

```
    } else if (choice === 'snowing') {
      para.textContent = 'The snow is coming down — it is freezing! Best to sta
    } else if (choice === 'overcast') {
      para.textContent = 'It isn\'t raining, but the sky is grey and gloomy; it
    } else {
      para.textContent = '';
    }
  }
```

Select the weather type today: --Make a choice--

1. Here we've got an HTML `<select>` element allowing us to make different weather choices, and a simple paragraph.
2. In the JavaScript, we are storing a reference to both the `<select>` and `<p>` elements, and adding an event listener to the `<select>` element so that when its value is changed, the `setWeather()` function is run.
3. When this function is run, we first set a variable called `choice` to the current value selected in the `<select>` element. We then use a conditional statement to show different text inside the paragraph depending on what the value of `choice` is. Notice how all the conditions are tested in `else if() {...}` blocks, except for the first one, which is tested in an `if() {...} block`.
4. The very last choice, inside the `else {...}` block, is basically a "last resort" option — the code inside it will be run if none of the conditions are `true`. In this case, it serves to empty the text out of the paragraph if nothing is selected, for example if a user decides to re-select the "--Make a choice--" placeholder option shown at the beginning.

Note: You can also find this example on GitHub (see it running live on there also.)

A note on comparison operators

Comparison operators are used to test the conditions inside our conditional statements. We first looked at comparison operators back in our Basic math in JavaScript — numbers and operators article. Our choices are:

- === and !== — test if one value is identical to, or not identical to, another.
- < and > — test if one value is less than or greater than another.
- <= and >= — test if one value is less than or equal to, or greater than or equal to, another.

Note: Review the material at the previous link if you want to refresh your memories on these.

We wanted to make a special mention of testing boolean (`true`/`false`) values, and a common pattern you'll come across again and again. Any value that is not `false`, `undefined`, `null`, `0`, `NaN`, or an empty string (`''`) actually returns `true` when tested as a conditional statement, therefore you can simply use a variable name on its own to test whether it is `true`, or even that it exists (i.e. it is not undefined.) So for example:

```
var cheese = 'Cheddar';

if (cheese) {
  console.log('Yay! Cheese available for making cheese on toast.');
} else {
  console.log('No cheese on toast for you today.');
}
```

And, returning to our previous example about the child doing a chore for their parent, you could write it like this:

```
var shoppingDone = false;

if (shoppingDone) { // don't need to explicitly specify '=== true'
  var childsAllowance = 10;
} else {
  var childsAllowance = 5;
}
```

Nesting if ... else

It is perfectly OK to put one `if...else` statement inside another one — to nest them. For example, we could update our weather forecast application to show a further set of choices depending on what the temperature is:

```
if (choice === 'sunny') {
  if (temperature < 86) {
    para.textContent = 'It is ' + temperature + ' degrees outside — nice and
  } else if (temperature >= 86) {
    para.textContent = 'It is ' + temperature + ' degrees outside — REALLY HO
  }
}
```

Even though the code all works together, each `if...else` statement works completely independently of the other one.

Logical operators: AND, OR and NOT

If you want to test multiple conditions without writing nested `if...else` statements, logical operators can help you. When used in conditions, the first two do the following:

- && — AND; allows you to chain together two or more expressions so that all of them have to individually evaluate to `true` for the whole expression to return `true`.
- || — OR; allows you to chain together two or more expressions so that one or more of them have to individually evaluate to `true` for the whole expression to return `true`.

To give you an AND example, the previous example snippet can be rewritten to this:

```
if (choice === 'sunny' && temperature < 86) {
  para.textContent = 'It is ' + temperature + ' degrees outside — nice and su
} else if (choice === 'sunny' && temperature >= 86) {
  para.textContent = 'It is ' + temperature + ' degrees outside — REALLY HOT!
}
```

So for example, the first code block will only be run if `choice === 'sunny'` *and* `temperature < 86` return `true`.

Let's look at a quick OR example:

```
if (iceCreamVanOutside || houseStatus === 'on fire') {
  console.log('You should leave the house quickly.');
} else {
  console.log('Probably should just stay in then.');
}
```

The last type of logical operator, NOT, expressed by the `!` operator, can be used to negate an expression. Let's combine it with OR in the above example:

```
if (!(iceCreamVanOutside || houseStatus === 'on fire')) {
  console.log('Probably should just stay in then.');
} else {
  console.log('You should leave the house quickly.');
}
```

In this snippet, if the OR statement returns `true`, the NOT operator will negate it so that the overall expression returns `false`.

You can combine as many logical statements together as you want, in whatever structure. The following example executes the code inside only if both OR statements return true, meaning that the overall AND statement will return true:

```
if ((x === 5 || y > 3 || z <= 10) && (loggedIn || userName === 'Steve')) {
  // run the code
}
```

A common mistake when using the logical OR operator in conditional statements is to try to state the variable whose value you are checking once, and then give a list of values it could be to return true, separated by || (OR) operators. For example:

```
if (x === 5 || 7 || 10 || 20) {
  // run my code
}
```

In this case the condition inside `if(...)` will always evaluate to true since 7 (or any other non-zero value) always evaluates to true. This condition is actually saying "if x equals 5, or 7 is true — which it always is". This is logically not what we want! To make this work you've got to specify a complete test either side of each OR operator:

```
if (x === 5 || x === 7 || x === 10 ||x === 20) {
  // run my code
}
```

switch statements

`if...else` statements do the job of enabling conditional code well, but they are not without their downsides. They are mainly good for cases where you've got a couple of choices, and each one requires a reasonable amount of code to be run, and/or the conditions are complex (e.g. multiple logical operators). For cases where you just want to set a variable to a certain choice of value or print out a particular statement depending on a condition, the syntax can be a bit cumbersome, especially if you've got a large number of choices.

`switch` statements are your friend here — they take a single expression/value as an input, and then look through a number of choices until they find one that matches that value, executing the corresponding code that goes along with it. Here's some more pseudocode, to give you an idea:

```
switch (expression) {
  case choice1:
    run this code
    break;

  case choice2:
    run this code instead
    break;

  // include as many cases as you like

  default:
    actually, just run this code
}
```

Here we've got:

1. The keyword `switch`, followed by a set of parentheses.
2. An expression or value inside the parentheses.
3. The keyword `case`, followed by a choice that the expression/value could be, followed by a colon.
4. Some code to run if the choice matches the expression.
5. A `break` statement, followed by a semi-colon. If the previous choice matches the expression/value, the browser stops executing the code block here, and moves on to any code that appears below the switch statement.
6. As many other cases (bullets 3–5) as you like.
7. The keyword `default`, followed by exactly the same code pattern as one of the cases (bullets 3–5), except that `default` does not have a choice after it, and you don't need to `break` statement as there is nothing to run after this in the block anyway. This is the default option that runs if none of the choices match.

Note: You don't have to include the `default` section — you can safely omit it if there is no chance that the expression could end up equaling an unknown value. If there is a chance of this however, you need to include it to handle unknown cases.

A switch example

Let's have a look at a real example — we'll rewrite our weather forecast application to use a switch statement instead:

```
<label for="weather">Select the weather type today: </label>

```

```
<select id="weather">
  <option value="">--Make a choice--</option>
  <option value="sunny">Sunny</option>
  <option value="rainy">Rainy</option>
  <option value="snowing">Snowing</option>
  <option value="overcast">Overcast</option>
</select>

<p></p>
```

```
var select = document.querySelector('select');
var para = document.querySelector('p');

select.addEventListener('change', setWeather);


function setWeather() {
  var choice = select.value;

  switch (choice) {
    case 'sunny':
      para.textContent = 'It is nice and sunny outside today. Wear shorts! Go
      break;
    case 'rainy':
      para.textContent = 'Rain is falling outside; take a rain coat and a bro
      break;
    case 'snowing':
      para.textContent = 'The snow is coming down — it is freezing! Best to s
      break;
    case 'overcast':
      para.textContent = 'It isn\'t raining, but the sky is grey and gloomy;
      break;
    default:
      para.textContent = '';
  }
}
```

Select the weather type today: --Make a choice--

Note: You can also find this example on GitHub (see it running live on there also.)

Ternary operator

There is one final bit of syntax we want to introduce you to, before we get you to play with some examples. The ternary or conditional operator is a small bit of syntax that tests a condition and returns one value/expression if it is `true`, and another if it is `false` — this can be useful in some situations, and can take up a lot less code than an `if...else` block if you simply have two choices that are chosen between via a `true`/`false` condition. The pseudocode looks like this:

```
( condition ) ? run this code : run this code instead
```

So let's look at a simple example:

```
var greeting = ( isBirthday ) ? 'Happy birthday Mrs. Smith — we hope you have
```

Here we have a variable called `isBirthday` — if this is `true`, we give our guest a happy birthday message; if not, we given her the standard daily greeting.

Ternary operator example

You don't just have to set variable values with the ternary operator; you can also run functions, or lines of code — anything you like. The following live example shows a simple theme chooser where the styling for the site is applied using a ternary operator.

```
<label for="theme">Select theme: </label>
<select id="theme">
  <option value="white">White</option>
  <option value="black">Black</option>
</select>

<h1>This is my website</h1>
```

```
var select = document.querySelector('select');
var html = document.querySelector('html');
document.body.style.padding = '10px';

function update(bgColor, textColor) {
  html.style.backgroundColor = bgColor;

```

```
    html.style.color = textColor;
  }

  select.onchange = function() {
    ( select.value === 'black' ) ? update('black','white') : update('white','bl
  }
```

Select theme: White

This is my website

Here we've got a `<select>` element to choose a theme (black or white), plus a simple `<h1>` to display a website title. We also have a function called `update()`, which takes two colors as parameters (inputs). The website's background color is set to the first provided color, and its text color is set to the second provided color.

Finally, we've also got an onchange event listener that serves to run a function containing a ternary operator. It starts with a test condition — `select.value === 'black'`. If this returns `true`, we run the `update()` function with parameters of black and white, meaning that we end up with background color of black and text color of white. If it returns `false`, we run the `update()` function with parameters of white and black, meaning that the site color are inverted.

Note: You can also find this example on GitHub (see it running live on there also.)

Active learning: A simple calendar

In this example you are going to help us finish a simple calendar application. In the code you've got:

- A `<select>` element to allow the user to choose between different months.
- An `onchange` event handler to detect when the value selected in the `<select>` menu is changed.
- A function called `createCalendar()` that draws the calendar and displays the correct month in

the `<h1>` element.

We need you to write a conditional statement inside the onchange handler function, just below the `// ADD CONDITIONAL HERE` comment. It should:

1. Look at the selected month (stored in the `choice` variable. This will be the `<select>` element value after the value changes, so "January" for example.)
2. Set a variable called `days` to be equal to the number of days in the selected month. To do this you'll have to look up the number of days in each month of the year. You can ignore leap years for the purposes of this example.

Hints:

- You are advised to use logical OR to group multiple months together into a single condition; many of them share the same number of days.
- Think about which number of days is the most common, and use that as a default value.

If you make a mistake, you can always reset the example with the "Reset" button. If you get really stuck, press "Show solution" to see a solution.

Select month: January

January

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28
29	30	31	

```
var select =
document.querySelect
or('select');
var list =
document.querySelect
or('ul');
var h1 =
document.querySelect
or('h1');

select.onchange =
function() {
  var choice =
select.value;

  // ADD
CONDITIONAL HERE


createCalendar(days,
 choice);
}

function
createCalendar(days,
 choice) {
  list.innerHTML =
'';
  h1.textContent =
```

Active learning: More color choices!

In this example you are going to take the ternary operator example we saw earlier and convert the ternary operator into a switch statement that will allow us to apply more choices to the simple website. Look at the `<select>` — this time you'll see that it has not two theme options, but five. You need to add a switch statement just underneath the `// ADD SWITCH STATEMENT` comment:

- It should accept the `choice` variable as its input expression.
- For each case, the choice should equal one of the possible values that can be selected, i.e. white, black, purple, yellow, or psychedelic.
- For each case, the `update()` function should be run, and be passed two color values, the first one for the background color, and the second one for the text color. Remember that color values are strings, so need to be wrapped in quotes.

If you make a mistake, you can always reset the example with the "Reset" button. If you get really stuck, press "Show solution" to see a solution.

Select theme: White

This is my website

```
var select =
document.querySelect
or('select');
var html =
document.querySelect
or('.output');

select.onchange =
function() {
  var choice =
select.value;

  // ADD SWITCH
STATEMENT
}

function
update(bgColor,
textColor) {

html.style.backgroun
dColor = bgColor;
  html.style.color
= textColor;
}
```

Reset Show solution

Conclusion

And that's all you really need to know about conditional structures in JavaScript right now! I'm sure

you'll have understood these concepts and worked through the examples with ease; if there is anything you didn't understand, feel free to read through the article again, or contact us to ask for help.

See also

- Comparison operators
- Conditional statements in detail
- if...else reference
- Conditional (ternary) operator reference

Overview: Building blocks Next

Was this article helpful?

Learn the best of web development

Sign up for our newsletter:

you@example.com

SIGN UP NOW