The "switch" statement

A `switch` statement can replace multiple `if` checks.

It gives a more descriptive way to compare a value with multiple variants.

The syntax

The `switch` has one or more `case` blocks and an optional default.

It looks like this:

```
switch(x) {
  case 'value1':  // if (x === 'value1')
    ...
    [break]

  case 'value2':  // if (x === 'value2')
    ...
    [break]

  default:
    ...
    [break]
}
```

The value of `x` is checked for a strict equality to the value from the first `case` (that is, `value1`) then to the second (`value2`) and so on.

If the equality is found, `switch` starts to execute the code starting from the corresponding `case`, until the nearest `break` (or until the end of `switch`).

If no case is matched then the `default` code is executed (if it exists).

An example

An example of `switch` (the executed code is highlighted):

```
let a = 2 + 2;

switch (a) {
  case 3:
    alert( 'Too small' );
    break;
  case 4:
    alert( 'Exactly!' );
    break;
  case 5:
    alert( 'Too big' );
    break;
```

```
  default:
    alert( "I don't know such values" );
}
```

Here the `switch` starts to compare `a` from the first `case` variant that is `3`. The match fails.

Then `4`. That's a match, so the execution starts from `case 4` until the nearest `break`.

If there is no `break` then the execution continues with the next `case` without any checks.

An example without `break`:

```
let a = 2 + 2;

switch (a) {
  case 3:
    alert( 'Too small' );
  case 4:
    alert( 'Exactly!' );
  case 5:
    alert( 'Too big' );
  default:
    alert( "I don't know such values" );
}
```

In the example above we'll see sequential execution of three `alert`s:

```
alert( 'Exactly!' );
alert( 'Too big' );
alert( "I don't know such values" );
```

Any expression can be a `switch/case` argument

Both `switch` and `case` allow arbitrary expressions.

For example:

```
let a = "1";
let b = 0;

switch (+a) {
  case b + 1:
    alert("this runs, because +a is 1, exactly equals b+1");
    break;

  default:
    alert("this doesn't run");
}
```

Here `+a` gives `1`, that's compared with `b + 1` in `case`, and the corresponding code is executed.

Grouping of “case”

Several variants of `case` which share the same code can be grouped.

For example, if we want the same code to run for `case 3` and `case 5`:

```
let a = 3;

switch (a) {
  case 4:
    alert('Right!');
    break;

  case 3: // (*) grouped two cases
  case 5:
    alert('Wrong!');
    alert("Why don't you take a math class?");
    break;

  default:
    alert('The result is strange. Really.');
}
```

Now both `3` and `5` show the same message.

The ability to “group” cases is a side-effect of how `switch/case` works without `break`. Here the execution of `case 3` starts from the line `(*)` and goes through `case 5`, because there’s no `break`.

Type matters

Let’s emphasize that the equality check is always strict. The values must be of the same type to match.

For example, let’s consider the code:

```
let arg = prompt("Enter a value?");
switch (arg) {
  case '0':
  case '1':
    alert( 'One or zero' );
    break;

  case '2':
    alert( 'Two' );
    break;

  case 3:
    alert( 'Never executes!' );
    break;
  default:
    alert( 'An unknown value' );
}
```

1. For `0`, `1`, the first `alert` runs.
2. For `2` the second `alert` runs.
3. But for `3`, the result of the `prompt` is a string `"3"`, which is not strictly equal `===` to the number `3`. So we've got a dead code in `case 3`! The `default` variant will execute.