6th November 2019

Arrow functions, the basics

There's another very simple and concise syntax for creating functions, that's often better than Function Expressions.

It's called "arrow functions", because it looks like this:

```
let func = (arg1, arg2, ...argN) => expression
```

…This creates a function `func` that accepts arguments `arg1..argN`, then evaluates the `expression` on the right side with their use and returns its result.

In other words, it's the shorter version of:

```
let func = function(arg1, arg2, ...argN) {
  return expression;
};
```

Let's see a concrete example:

```
let sum = (a, b) => a + b;

/* This arrow function is a shorter form of:

let sum = function(a, b) {
  return a + b;
};
*/

alert( sum(1, 2) ); // 3
```

As you can, see `(a, b) => a + b` means a function that accepts two arguments named `a` and `b`. Upon the execution, it evaluates the expression `a + b` and returns the result.

- If we have only one argument, then parentheses around parameters can be omitted, making that even shorter.

  For example:

  ```
  let double = n => n * 2;
  // roughly the same as: let double = function(n) { return n * 2 }

  alert( double(3) ); // 6
  ```

- If there are no arguments, parentheses will be empty (but they should be present):

  ```
  let sayHi = () => alert("Hello!");

  sayHi();
  ```

Arrow functions can be used in the same way as Function Expressions.

For instance, to dynamically create a function:

```
let age = prompt("What is your age?", 18);

let welcome = (age < 18) ?
  () => alert('Hello') :
  () => alert("Greetings!");

welcome(); // ok now
```

Arrow functions may appear unfamiliar and not very readable at first, but that quickly changes as the eyes get used to the structure.

They are very convenient for simple one-line actions, when we're just too lazy to write many words.

Multiline arrow functions

The examples above took arguments from the left of `=>` and evaluated the right-side expression with them.

Sometimes we need something a little bit more complex, like multiple expressions or statements. It is also possible, but we should enclose them in curly braces. Then use a normal `return` within them.

Like this:

```
let sum = (a, b) => {  // the curly brace opens a multiline function
  let result = a + b;
  return result; // if we use curly braces, then we need an explicit "return"
};

alert( sum(1, 2) ); // 3
```

More to come

Here we praised arrow functions for brevity. But that's not all!

Arrow functions have other interesting features.

To study them in-depth, we first need to get to know some other aspects of JavaScript, so we'll return to arrow functions later in the chapter Arrow functions revisited.

For now, we can already use arrow functions for one-line actions and callbacks.

Summary

Arrow functions are handy for one-liners. They come in two flavors:

1. Without curly braces: `(...args) => expression` – the right side is an expression: the function evaluates it and returns the result.
2. With curly braces: `(...args) => { body }` – brackets allow us to write multiple statements inside the function, but we need an explicit `return` to return something.

Tasks

Rewrite with arrow functions

Replace Function Expressions with arrow functions in the code below:

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

ask(
  "Do you agree?",
  function() { alert("You agreed."); },
  function() { alert("You canceled the execution."); }
);
```

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 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…)