EDUFY

Functions: 20 exercises

Maximilian Bahrami · 11 November 2026

20 exercises to practise functions, each with a short lesson first: parameters and returns, writing to the page, arrays, strings and regular expressions. Every answer runs.

You've learned how to make functions, give them parameters, return answers, use default values and run them on a click. Now you'll practise with **20 exercises** in five groups. Before each exercise there's a short lesson on what you need for it, with a small example. Then comes the task, and the answer, hidden until you open it.

Each exercise builds on the one before, the way real code grows: you write something small, then improve it step by step. Developers call that **refactoring**: changing code to do more, or to be clearer, without starting over.

How to work through this lesson

> Read the short lesson, try the exercise yourself, and only then open the answer. If you're stuck for more than ten minutes, open the answer, read it, close it, and write it again from memory. Every example runs: press **Run**, then **Edit** to try your own ideas.

The exercises about lists all use this array of names:

```
const names = ["Alex", "Sara", "Omar", "Lina", "Jonas"];
console.log(names);
```

Part 1: parameters and returns

Exercise 1: make and call a function

A function is code with a name. Writing it down with function doesn't run it; **calling** it with its name and brackets () does. You can call it as many times as you like:

```
function cheer() {
console.log("Well done!");
}

cheer();
cheer();
```

Exercise 1: say hello

> Create a function that prints "Hi! My name is Alex" with console.log. Then call it.

Answer

```
function sayHi() {
console.log("Hi! My name is Alex");
}

sayHi();
```

function sayHi() { ... } makes the function; sayHi(); runs it. Without the second line nothing would be printed.

Exercise 2: a parameter and return

A **parameter** is a name between the brackets that gets a value each time the function is called. **return** gives an answer back to the code that called the function, instead of printing it:

```
function double(n) {
return n * 2;
}

const answer = double(4);
console.log(answer); // 8
console.log(double(10)); // 20
```

Join text with +: "Hi " + "Sara" is "Hi Sara".

Exercise 2: take a name, return the greeting

Change exercise 1 so the function takes a name as a parameter and **returns** "Hi! My name is " followed by that name.

Print the result outside the function. There should be **no** console.log inside it.

Answer

```
function sayHi(name) {
return "Hi! My name is " + name;
}

console.log(sayHi("Sara"));
console.log(sayHi("Omar"));
```

sayHi("Sara") puts "Sara" into name, joins it to the greeting and **returns** the whole text. console.log outside prints what came back. The function only makes the text; whoever calls it decides what to do with it.

Exercise 3: a default value

A parameter can have a **default value**: what it uses when nothing is passed in. Write it with =:

```
function welcome(place = "EDUFY") {
return "Welcome to " + place;
}

console.log(welcome("Stockholm")); // Welcome to Stockholm
console.log(welcome()); // Welcome to EDUFY
```

Exercise 3: a default name

> Change exercise 2 so it uses the name "Alex" when no name is passed in.

Answer

```
function sayHi(name = "Alex") {
return "Hi! My name is " + name;
}

console.log(sayHi("Sara")); // Hi! My name is Sara
console.log(sayHi()); // Hi! My name is Alex
```

> name = "Alex" only takes effect when sayHi is called with nothing in the brackets.

Exercise 4: more than one parameter

A function can take several parameters, separated by commas. The values you pass in are matched **in order**: the first value goes into the first parameter, the second into the second:

```
function fullPrice(price, quantity) {
return price * quantity;
}

console.log(fullPrice(20, 3)); // 60
```

To put a space between two pieces of text, join a " " in the middle: first + " " + last.

Exercise 4: a last name too

> Change exercise 3 so the function also takes a last name.

Answer

```
function sayHi(first = "Alex", last = "") {
return "Hi! My name is " + first + " " + last;
}
```

```
console.log(sayHi("Sara", "Berg")); // Sara Berg
```

> "Sara" goes into first, "Berg" into last. The order of the values decides where they go.

Exercise 5: a default made from another parameter

A default value can be worked out from the parameters **before** it. When the function runs, the earlier ones already have their values:

```
function box(width, height = width) {
return width + " × " + height;
}

console.log(box(4, 2)); // 4 × 2
console.log(box(4)); // 4 × 4: height copies width
```

Exercise 5: a last name made from the first

> Change exercise 4 so that, if no last name is passed in, the last name is the first name + "sson".

> With no names at all it should print "Hi! My name is Alex Alexsson".

Answer

```
function sayHi(first = "Alex", last = first + "sson") {
return "Hi! My name is " + first + " " + last;
}

console.log(sayHi("Sara", "Berg")); // Sara Berg
console.log(sayHi("Omar")); // Omar Omarsson
console.log(sayHi()); // Alex Alexsson
```

> With no names, first becomes "Alex", and then last becomes first + "sson": "Alexsson". The order of the parameters matters: last can use first, but not the other way round.

Part 2: writing to the page

Exercise 6: add a paragraph

The **DOM** is the page as JavaScript sees it: every element is something your code can find and change. Adding a paragraph takes three steps:

1. **Find** the element to add to: document.getElementById("id").
2. **Make** a new element: document.createElement("p"), and give it text with textContent.
3. **Add** it: box.appendChild(p) puts it at the end of the box.

```
<!DOCTYPE html>
<html lang="en">
<body style="font-family: sans-serif">
<div id="news"><b>News</b></div>
<script>
const box = document.getElementById("news");
const p = document.createElement("p");
p.textContent = "The shop opens at nine.";
box.appendChild(p);
</script>
</body>
</html>
```

Exercise 6: here I am

> Create a **function** that adds a paragraph saying "Here I am!" to the . Then call it.

Answer

```
<!DOCTYPE html>
<html lang="en">
<body style="font-family: sans-serif">
<div id="exercise6"><b>Exercise 6</b></div>
<script>
function addHere() {
const box = document.getElementById("exercise6");
const p = document.createElement("p");
p.textContent = "Here I am!";
box.appendChild(p);
}

addHere();
</script>
</body>
</html>
```

> The three steps (find, make, add) are inside a function, so you can add the paragraph again just by calling addHere().

Exercise 7: add any text

Right now the function always adds the same words. Give it a **parameter** for the text, and the same function can add anything: one function, many uses.

Exercise 7: add any text

> Change exercise 6 so the function takes a string and adds **that** text to the .

Answer

```
<!DOCTYPE html>
<html lang="en">
<body style="font-family: sans-serif">
<div id="exercise7"><b>Exercise 7</b></div>
<script>
function addText(text) {
const box = document.getElementById("exercise7");
const p = document.createElement("p");
p.textContent = text;
box.appendChild(p);
}

addText("Here I am!");
addText("And here too.");
</script>
</body>
</html>
```

> p.textContent = text uses whatever was passed in. Each call adds one more paragraph.

Exercise 8: functions working together

The answer one function **returns** can go straight into another function. addText(sayHi("Sara")) first runs sayHi("Sara"), then hands the text it returned to addText. Work from the inside out:

```
function double(n) { return n * 2; }
function addOne(n) { return n + 1; }

console.log(addOne(double(5))); // double(5) is 10, then 11
```

Exercise 8: show the greetings on the page

> Change exercise 7 so it adds a paragraph with the result of exercise 5. Run it three times:

> 1. with both names,
> 2. with only a first name,
> 3. with no names.

Answer

```
<!DOCTYPE html>
<html lang="en">
<body style="font-family: sans-serif">
<div id="exercise8"><b>Exercise 8</b></div>
<script>
```

```
function sayHi(first = "Alex", last = first + "sson") {
return "Hi! My name is " + first + " " + last;
}

function addText(text) {
const box = document.getElementById("exercise8");
const p = document.createElement("p");
p.textContent = text;
box.appendChild(p);
}

addText(sayHi("Sara", "Berg"));
addText(sayHi("Omar"));
addText(sayHi());
</script>
</body>
</html>
```

> Two small functions, each with one job: sayHi makes the text, addText puts text on the page. Small functions that each do one thing are easier to understand, test and reuse.

Part 3: working with arrays

Exercise 9: a loop through an array

An **array** is a list in square brackets. To do something with **every** item, use a loop. for...of gives you the items one at a time, in order:

```
const fruits = ["apple", "pear", "plum"];

for (const fruit of fruits) {
console.log("I like " + fruit);
}
```

Read it as: for each fruit of the fruits, run the code in { }. The name fruit is yours to choose; it holds a different item each time round.

Exercise 9: print every name

> Go through the names array and print each name with console.log.

Answer

```
const names = ["Alex", "Sara", "Omar", "Lina", "Jonas"];

for (const name of names) {
console.log(name);
}
```

> The loop runs five times, once per name. Add a sixth name to the array and it runs six times, with no other change.

Exercise 10: call a function inside a loop

Anything can go inside a loop, including a call to your own function. The function then runs once for each item:

```
function double(n) { return n * 2; }

for (const n of [1, 2, 3]) {
console.log(double(n)); // 2, 4, 6
}
```

Exercise 10: greet every name

> Go through the names array and print the result of exercise 2 for each name.

Answer

```
const names = ["Alex", "Sara", "Omar", "Lina", "Jonas"];

function sayHi(name) {
return "Hi! My name is " + name;
}

for (const name of names) {
console.log(sayHi(name));
}
```

Exercise 11: from a loop to the page

Put the three page steps (find, make, add) together with a loop. Find the box **once**, before the loop; then make and add one paragraph each time round.

Exercise 11: greet every name on the page

> Change exercise 10 so each greeting is added as a paragraph to the .

Answer

```
<!DOCTYPE html>
<html lang="en">
<body style="font-family: sans-serif">
<div id="exercise11"><b>Exercise 11</b></div>
<script>
const names = ["Alex", "Sara", "Omar", "Lina", "Jonas"];
const box = document.getElementById("exercise11");

function sayHi(name) {
return "Hi! My name is " + name;
```

```
}

for (const name of names) {
const p = document.createElement("p");
p.textContent = sayHi(name);
box.appendChild(p);
}
</script>
</body>
</html>
```

> The box is found once, outside the loop: there's no need to look for it five times.

Exercise 12: a helper function with two parameters

When the same few lines appear in many places, move them into a **helper function**. Give it parameters for everything that changes: here, where the paragraph goes and what it says. Then the helper works for any box on the page.

Exercise 12 (extra): one function to add paragraphs

> Change exercise 11 so the loop calls a second function that adds the paragraph. That function takes **two** arguments: the div to add to, and the text for the paragraph.

Answer

```
<!DOCTYPE html>
<html lang="en">
<body style="font-family: sans-serif">
<div id="exercise12"><b>Exercise 12</b></div>
<script>
const names = ["Alex", "Sara", "Omar", "Lina", "Jonas"];

function sayHi(name) {
return "Hi! My name is " + name;
}

function addParagraph(box, text) {
const p = document.createElement("p");
p.textContent = text;
box.appendChild(p);
}

const box = document.getElementById("exercise12");
for (const name of names) {
addParagraph(box, sayHi(name));
}
</script>
</body>
</html>
```

> addParagraph(box, text) doesn't care which box or which text: you'll use exactly this helper in exercise 16.

Part 4: working with strings

A **string** is text. Strings come with their own tools, used with a dot after the text. These four are all you need here:

Write	Gives you	Example
text.toUpperCase()	the text in capital letters	"Sara" → "SARA"
text.length	how many characters it has	"Sara" → 4
text.includes("s")	true if it contains "s"	"Sara".includes("s") → false
text.indexOf("a")	where "a" first is (counting from 0), or -1	"Sara".indexOf("a") → 1

Exercise 13: capital letters

toUpperCase() gives back a **new** string in capitals; the original stays as it was. Notice the brackets: it's a function that belongs to the string.

```
const city = "Paris";
console.log(city.toUpperCase()); // PARIS
console.log(city); // Paris: unchanged
```

Exercise 13: names in capitals

> Write a function that prints every name in the names array in capital letters.

Answer

```
const names = ["Alex", "Sara", "Omar", "Lina", "Jonas"];

function printUpper(list) {
for (const name of list) {
console.log(name.toUpperCase());
}
}

printUpper(names);
```

> The array is passed in as a parameter (list), so the same function works for any list of names.

Exercise 14: length

length tells you how many characters a string has, spaces included. It has **no** brackets: it's a value the string keeps, not something it does.

```
console.log("JavaScript".length); // 10
console.log("hi there".length); // 8: the space counts
```

Exercise 14: how long is each name?

> Write a function that prints the length of each name in the names array.

Answer

```
const names = ["Alex", "Sara", "Omar", "Lina", "Jonas"];

function printLengths(list) {
for (const name of list) {
console.log(name + ": " + name.length + " letters");
}
}

printLengths(names);
```

Exercise 15: includes, with if and else

includes() answers a yes/no question, so it's a boolean: perfect for an if. Capital letters count as different letters, so use toLowerCase() first when capitals shouldn't matter:

```
const word = "Stockholm";
console.log(word.includes("s")); // false: it's a capital S
console.log(word.toLowerCase().includes("s")); // true
```

Exercise 15: does it contain an s?

> Write a function that prints each name and whether it does or does not contain the letter "s".

Answer

```
const names = ["Alex", "Sara", "Omar", "Lina", "Jonas"];

function checkForS(list) {
for (const name of list) {
if (name.toLowerCase().includes("s")) {
console.log(name + " contains an s");
} else {
console.log(name + " does not contain an s");
}
}
}
```

```
checkForS(names);
```

> toLowerCase() turns "Sara" into "sara" before the check, so its capital S counts too.

Exercise 16: indexOf

indexOf("a") gives the **position** of the first "a", counting from **0**, or -1 if there isn't one. So -1 means "not found", and anything else means "found here":

```
console.log("banana".indexOf("n")); // 2: b=0, a=1, n=2
console.log("plum".indexOf("a")); // -1: no a
```

People count from 1, so add 1 when you show the position to a reader.

Exercise 16: where is the a?

> Change exercise 15: for **each** name that contains the letter "a", add a paragraph to the page saying where in the name the a appears.

Answer

```
<!DOCTYPE html>
<html lang="en">
<body style="font-family: sans-serif">
<div id="exercise16"><b>Exercise 16</b></div>
<script>
const names = ["Alex", "Sara", "Omar", "Lina", "Jonas"];

function addParagraph(box, text) {
const p = document.createElement("p");
p.textContent = text;
box.appendChild(p);
}

function findA(list, box) {
for (const name of list) {
const at = name.toLowerCase().indexOf("a");
if (at !== -1) {
addParagraph(box, name + ": letter " + (at + 1));
}
}
}

findA(names, document.getElementById("exercise16"));
</script>
</body>
</html>
```

> at !== -1 means "the name has an a". It finds the **first** a: in "Sara" that's letter 2. The helper from exercise 12 is used again as it was.

Part 5: regular expressions

A **regular expression** (regex) is a pattern that text has to match. You write it between two slashes, and pattern.test(text) answers true or false. These are the pieces you need:

Part	Means
^ … $	the start and the end: the **whole** text must match, not just a piece of it
[a-z]	one small letter; [A-Z] one capital letter; [0-9] one digit
[a-zA-Z0-9]	one letter (small or capital) or digit
+	one or more of what's just before it
{3}	exactly three of what's just before it

```
const onlyDigits = /^[0-9]+$/;
console.log(onlyDigits.test("2026")); // true
console.log(onlyDigits.test("20x6")); // false: x isn't a digit
console.log(/[0-9]+/.test("20x6")); // true: no ^ $, so a piece is enough
```

The last line shows why ^ and $ matter: without them, the pattern only needs to be found **somewhere** in the text.

Exercise 17: letters or numbers

Put letters and digits in **one** pair of square brackets to mean "any of these", then + for "one or more", between ^ and $.

Exercise 17: letters or numbers only

> Write a function that tests whether a user's input is made **only** of letters and numbers (no spaces or other signs).

Answer

```
function isLettersOrNumbers(input) {
return /^[a-zA-Z0-9]+$/.test(input);
}

console.log(isLettersOrNumbers("Alex2026")); // true
console.log(isLettersOrNumbers("Alex 2026")); // false: space
console.log(isLettersOrNumbers("hi!")); // false: !
```

> The function returns the true/false from test, so it can go straight into an if.

Exercise 18: an exact count

{5} means "exactly five of what's before it". With ^ and $ around the pattern, the text can't be longer or shorter:

```
console.log(/^[0-9]{4}$/.test("2026")); // true: 4 digits
console.log(/^[0-9]{4}$/.test("20261")); // false: 5 digits
```

Exercise 18: exactly 5 letters

> Write a function that tests whether the input is exactly 5 letters.

Answer

```
function isFiveLetters(input) {
return /^[a-zA-Z]{5}$/.test(input);
}

console.log(isFiveLetters("Jonas")); // true
console.log(isFiveLetters("Sara")); // false: 4 letters
console.log(isFiveLetters("Jon4s")); // false: a digit
```

Exercise 19: capital letters only

[A-Z] matches capital letters only, so leave out a-z. Regex patterns care about capitals, just like includes.

Exercise 19: exactly 5 capital letters

> Write a function that tests whether the input is exactly 5 capital letters.

Answer

```
function isFiveCapitals(input) {
return /^[A-Z]{5}$/.test(input);
}

console.log(isFiveCapitals("JONAS")); // true
console.log(isFiveCapitals("Jonas")); // false
```

Exercise 20: one part after another

A pattern can have several parts **in a row**. Each part matches the next piece of the text, from left to right:

```
// one capital letter, then one digit
console.log(/^[A-Z][0-9]$/.test("B7")); // true
console.log(/^[A-Z][0-9]$/.test("7B")); // false: wrong order
```

Exercise 20: 2 capitals, then 3 numbers

> Write a function that tests whether the input is 2 capital letters followed by 3 numbers, like AB123.

Answer

```
function isCode(input) {
return /^[A-Z]{2}[0-9]{3}$/.test(input);
}

console.log(isCode("AB123")); // true
console.log(isCode("ab123")); // false: small letters
console.log(isCode("AB1234")); // false: 4 numbers
```

> This is the kind of check a form uses for a booking code or a car number plate: the pattern describes exactly what's allowed.

Try it on a real page

Here's exercise 20 in a form, checking as you type. Press **Run**, then type a code:

```
<!DOCTYPE html>
<html lang="en">
<body style="padding: 16px; font-family: sans-serif">
<input id="code" placeholder="For example AB123" />
<p id="answer">Type a code</p>
<script>
function isCode(input) {
return /^[A-Z]{2}[0-9]{3}$/.test(input);
}

const box = document.getElementById("code");
const answer = document.getElementById("answer");

box.addEventListener("input", () => {
answer.textContent = isCode(box.value)
? "That's a valid code"
: "2 capital letters, then 3 numbers";
});
</script>
</body>
</html>
```

Key points

- Start small, then refactor: each exercise improved the one before instead of starting over.
- Return values from functions; let the caller print them or put them on the page.

- Find, make, add: getElementById, createElement with textContent, then appendChild.
- for (const item of list) goes through every item; strings have toUpperCase(), length, includes() and indexOf().
- A regular expression like /^[A-Z]{2}[0-9]{3}$/ checks that text has exactly the shape you want.