JavaScript Part 4: Functions and Arrow Functions

A function is a named block of code that we write once and use many times. Without functions, the same rule gets copied into several places, and the copies slowly drift apart. Four steps make functions dependable: define, return, scope and arrow. Each step answers one simple question, and every example can be tried in the browser console. In this part, Nader, an operations analyst at an online retailer, uses them to give the whole shop one shipping fee rule.

1. Define

A function declaration starts with the keyword function, followed by a name, a pair of parentheses and a block of code in curly braces. Inside the parentheses we list parameters, which are names for the values that the function needs. When we call the function by writing its name with parentheses, the values we pass are called arguments, and they fill the parameters. The question is: what does this function need to receive, and what is its single job? A good function name is a verb phrase such as calculateShippingFee, because it tells the reader what happens. A function that does one job is easy to test and easy to reuse. A parameter can also have a default value, which is used when the caller leaves that argument out.

Example: A coffee machine needs two things from us, the type of coffee and the size of the cup, and then it does one job. It does not matter who presses the button, because the steps are always the same. A new colleague gets exactly the same cup on the first day. A function works in the same way: its parameters are what we ask for and its body is the recipe, so every caller gets a predictable result.

At work: Nader finds the shipping rule copied into the cart page, the checkout page and the confirmation email, and the copies disagree. He asks the Define question and writes one function named calculateShippingFee that needs the weight and the delivery zone. The zone has a default value for domestic orders, so the common call is short. All three places now call the same function, so the copies can never disagree again.

2. Return

A function usually exists to produce a result. The return statement sends a value back to the code that called the function and ends the function at once. A function without a return statement gives back undefined, even if it printed something on the way. Printing with console.log shows a value to a developer, but it does not hand the value to the rest of the program. The question is: what should this function give back? Returning a value lets the caller store it in a variable, compare it or pass it to another function. Functions that return values are easier to combine than functions that only print.

Example: A cashier who announces the price aloud has told the customer something, but a cashier who prints the receipt has produced something that the customer can carry away. Only the receipt can be filed, checked or passed to accounting. A function that only prints shouts the price, and a function that returns a value hands over the receipt, so the returned value is the receipt.

At work: The first version of Nader's shipping function printed the fee in the console, so the checkout page received undefined and showed nothing. He asks the Return question and writes a return statement instead of printing. The checkout page stores the fee in a variable, adds it to the total and passes it to the email function. The fee now appears in all three places, because the function hands it back instead of printing it.

This document is a property of Asaseeyat Academy - © Copy Right Protected.

3. Scope

Scope decides where a variable can be seen. A variable declared with let or const inside a function exists only inside that function, and the code outside cannot read it. A function can read variables from the outer area around it, but not the other way round. The question is: where does this variable need to be visible? Keeping variables as local as possible stops two parts of a program from changing the same value by accident. If we try to read a local variable from outside, JavaScript reports that it is not defined. Values that a function needs should arrive through its parameters, and results should leave through the return value.

Example: A project team keeps its working notes in a private folder, and only the final report goes to the shared drive. Colleagues see the report but not the scribbles, so nobody overwrites a draft by accident. The team can open the shared drive, but the drive cannot open the private folder. A function keeps its local variables private in the same way, and this is what scope means.

At work: One of Nader's functions calculated a surcharge in a variable named extra, and another function tried to read extra from outside. JavaScript reported that extra is not defined. He asks the Scope question and sees that the other function should not depend on a private detail. He passes the surcharge in as a parameter instead, so both functions now work independently.

4. Arrow

Arrow functions are a shorter way to write a function. We list the parameters, then an arrow made of an equals sign and a greater than sign, then the body. When the body is a single expression, the braces and the return keyword can be left out, and the value is returned automatically. For example, `const addTax = (price) => price * rate` is a complete function that takes a price and returns the price times the rate. The question is: is this small helper short enough to read in one line? Arrow functions are especially useful when we pass a function to another function, which we will see with arrays in the next part. For longer logic, a normal function with a clear name is still easier to read.

Example: A manager who explains a task in a long meeting could have sent a one-line message instead. When the instruction is short and obvious, such as add the tax to this price, the short form is better. When it needs care, such as the full delivery rule, the long form is better. An arrow function is the one-line message, so we use it for short helpers.

At work: Nader needs a tiny helper that adds the tax rate to a price. He asks the Arrow question and writes it as an arrow function on a single line. The longer shipping logic stays as a normal function with a clear name, so every colleague can find it and read it. The checkout code now reads almost like a sentence, which is the benefit of a short arrow where it fits.

Conclusion

Nader's shop now has one shipping rule that every page uses. Define asks what the function needs to receive and what its single job is, and return asks what it should give back. Scope asks where a variable must be visible, and arrow asks whether a small helper is short enough to read in one line. In the next part, arrays and objects let us hold lists and records that functions can work on.

Sources

- MDN Web Docs: Functions (JavaScript Guide) – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions
- MDN Web Docs: Arrow function expressions – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
- MDN Web Docs: Default parameters – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters
- MDN Web Docs: Grammar and types (variable scope) – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types

This document is a property of Asaseeyat Academy - © Copy Right Protected.