JavaScript – theory and tasks

Functions

A function is a block of code designed to perform a task.

Functions are like recipes. They take data or variables, perform a set of tasks on them, and then return the result. The beauty of functions is that they allow us to write a chunk of code once, then we can reuse it over and over without writing the same code over and over.

Take a look at this code:

var calculatorOn = false;

function pressPowerButton() {

if (calculatorOn) {

console.log('Calculator turning off.');

calculatorOn = false;

} else {

console.log('Calculator turning on.');

calculatorOn = true;

}

}

pressPowerButton();

// Output: Calculator turning on.

pressPowerButton();

// Output: Calculator turning off.

This code turns the calculator on if it is currently off, and turns it off if the calculator is currently on.

See if you can figure out how this code works. In the next exercise, we'll walk through it line by line.

How does this code work?

var calculatorOn = false;

function pressPowerButton() {

if (calculatorOn) {

console.log('Calculator turning off.');

calculatorOn = false;

} else {

console.log('Calculator turning on.');

calculatorOn = true;

}

}

pressPowerButton();

// Output: Calculator turning on.

pressPowerButton();

// Output: Calculator turning off.

1. On line 1, we have a variable named calculatorOn set to false. Our program starts with the calculator in the off position.
2. On line 3, there's a function named pressPowerButton. Functions follow this syntax:

* They begin with the JavaScript keyword function.
* After function comes the name of the function. pressPowerButton is the name of the function. Notice there are no spaces in the name and each new word is capitalized. This is a convention in the JavaScript community called camelCase.
* After the function's name, comes parentheses (). We'll learn about these in the next exercise.
* Finally, the function has a block of code it executes between the curly braces {}.

1. Inside the function is an if/else statement.
2. On the last few lines, we make the function run by writing pressPowerButton(). This term for this is calling the function. We call it with pressPowerButton(), and it runs all the code in the block of the function.
3. We executed the code in the block of the function twice without having to write it twice. Functions can make code reusable!

***Instructions***

*Write your code in the pizza.js file.*

1. Imagine you work at a pizza restaurant and you want to write a JavaScript program to take orders so you don't have to write them out by hand. You can write a function to perform this task!

Start by writing a function named takeOrder. Inside of its block, use console.log to print 'Order: pizza'.

1. Under the function, let's take an order. Call the takeOrder function on the last line.

The calculator program should be able to perform a math operation on a number. We should be able to give a calculator a number, have it perform a task on it like multiplication, then print a result.

Currently, we have no way to give a function a number. To do this, we can use parameters. Parameters are variables that we can set when we call the function. For example:

function multiplyByThirteen(inputNumber) {

console.log(inputNumber \* 13);

}

multiplyByThirteen(9);

// Output: 117

1. We added inputNumber within the parentheses of the multiplyByThirteen function. inputNumber is a parameter.
2. Inside the multiplyByThirteen function, we use console.log to print the inputNumber by 13.
3. When we call the multiplyByThirteen function on the last lines, we set the inputNumber parameter. For instance, multiplyByThirteen(9) calls the multiplyByThirteen function, and sets inputNumber as 9. Then, inside the block of the multiplyByThirteen function, 9 is multiplied by 13, resulting in 117 printing to the console.
4. Note on terminology: inputNumber is a parameter, but when we call multiplyByThirteen(9), the 9 is called an argument. Therefore, arguments are provided when you call a function, and parameters receive arguments as their value. So, inputNumber is a parameter and its value is the argument 9, since we wrote multiplyByThirteen(9).

Parameters let us write logic inside functions that can be modified based on when we call the function, which will help make our functions more flexible.

***Instructions***

*Write your code in the pizza.js file.*

1. Let's include a parameter in the takeOrder function to make the orders we log to the console more descriptive.

Inside the parentheses of the takeOrder function, add a parameter named topping.

1. Now, let's include the topping parameter inside the takeOrder function. Modify the console.log to interpolate the topping parameter to print a string like this:

*Order: pizza topped with bacon*

1. At the end of the program, modify the takeOrder function call to include an argument for topping.

If we can set one parameter, can we set two?

We can set as many parameters as we'd like by adding them when we declare the function, separated by commas, like this:

function getRemainder(numberOne, numberTwo) {

console.log(numberOne % numberTwo);

}

getRemainder(365, 27);

// Output: 14

1. The getRemainder function has two parameters: numberOne and numberTwo.
2. When we call the getRemainder function on the last line, we include two numbers as the parameters, also separated by commas. This is referred to as passing in parameters to a function.

In this case, we are telling the function to assign numberOne the value of 365 and numberTwo the value of 27. We are passing in 365 and 27 to the getRemainder function.

1. When the getRemainder runs, the function knows what numberOne and numberTwo equal since we passed in two parameters when we called the function. Therefore it evaluates 365 % 27, which produces the result 14.

By adding multiple parameters, we can build functions that are more flexible. Now the function has two variables that we can define when we call the function.

***Instructions***

*Write your code in the pizza.js file.*

1. Let's add another parameter to the takeOrder function to make the order even more descriptive.

Add a parameter named crustType so that we can add this to the console output in the upcoming steps.

1. Inside the takeOrder function, interpolate the crustType parameter to construct a sentence like this:

*Order: thin crust pizza topped with bacon*

1. Below the takeOrder function, call the function 3 times and pass in different arguments each time for topping and crustType.

Using console.log as the result of a function isn't the best use of a function. The purpose of a function is to take some input, perform some task on that input, then return a result.

To return a result, we can use the return keyword. Take a look at our function from the last exercise, now re-written slightly:

function getRemainder(numberOne, numberTwo) {

return numberOne % numberTwo;

}

console.log(getRemainder(365, 27));

// Output: 14

1. Instead of using console.log inside the getRemainder function, we used the return keyword. return will take the result of the math operation and give it back to whatever calls it.
2. On the last line, we called the getRemainder function inside of a console.log statement, which outputted the result of 14.
3. This code achieved the same output as before, however now our code is better. Why? If we wanted to use the getRemainder function in another place in our program, we could without printing the result to the console. Using return is generally a best practice when writing functions, as it makes your code more maintainable and flexible.

***Instructions***

*Write your code in the pizza.js file.*

1. Now that we have the pizza orders, you want to add them up to find the cost of the pizzas for the check. Let's imagine that each pizza is $7.50, no matter the topping and crust type.

We will need to do three things to write this in JavaScript:

* Create a variable to hold the number of pizzas ordered.
* Whenever a pizza is ordered, add one to the number of pizzas ordered.
* Take the total number of pizzas and multiply them by 7, since each pizza is $7.50.

Begin by creating a variable named orderCount set equal to 0 at the top of your code.

1. Inside the takeOrder function, set orderCount equal to orderCount plus 1, so that each time the takeOrder function runs, 1 is added to the orderCount.
2. Now it's time to calculate the subtotal of the pizzas. This is the perfect job for a function.

Declare a function named getSubTotal that has one parameter named itemCount.

1. Inside the getSubTotal function's block, use return to output the itemCount multiplied by 7.5.
2. On the last line of your program, after the takeOrder function calls, call the getSubTotal function inside a console.log statement.

getSubTotal has a parameter that represents the amount of items ordered. Pass in the orderCount as an argument when making the function call.

In the last exercise, we pointed out that using return makes programs more maintainable and flexible, but how exactly?

When functions return their value, we can use them together and inside one another. If our calculator needed to have a Celsius to Fahrenheit operation, we could write it with two functions like so:

function multiplyByNineFifths(celsius) {

return celsius \* (9/5);

}

function getFahrenheit(celsius) {

return multiplyByNineFifths(celsius) + 32;

}

console.log('The temperature is ' + getFahrenheit(15) + '°F');

// Output: The temperature is 59°F

Take a look at the getFahrenheit function. Inside of its block, we called multiplyByNineFifths and passed it the degrees in celsius. The multiplyByNineFifths function multiplied the celsius by (9/5). Then it returned its value so the getFahrenheit function could continue on to add 32 to it.

Finally, on the last line, we interpolated the function call within a console.log statement. This works because getFahrenheit returns it's value.

We can use functions to section off small bits of logic or tasks, then use them when we need to. Writing functions can help take large and difficult problems and break them into small and manageable problems.

***Instructions***

1. It's your job to calculate two more numbers for each order:

* A sales tax of 6% needs to be calculated for every full order. This should be based on the subtotal.
* The total, which is the subtotal plus tax, should also be computed.

Let's start with calculating the tax. Under the getSubTotal function, declare a function named getTax. It should take no parameters.

1. Inside the getTax function's block, multiply the subtotal times 6% (0.06). Make sure to return the result of this operation.
2. Now that you calculated the tax, declare another function named getTotal beneath the getTax function. The getTotal function should have no parameters.

Inside the getTotal function's block, add the subtotal to the tax, then return the result.

1. On the last line of the program, call the getTotal function inside of a console.log statement to view the result.

This unit introduced you to functions.

* Functions are written to perform a task.
* Functions take data or variables, perform a set of tasks on them, and then return the result.
* We can define parameters when calling the function.
* When calling a function, we can pass in arguments, which will set the function's parameters.
* We can use return to return the result of a function which allows us to call functions anywhere, even inside other functions.

Great work so far. Next up: Scope! Scope informs us where variables are accessible from within our programs.