Passing Arguments to Functions

Functions with Arguments

Remember that a function is a group of statements that performs a particular task. A function becomes more useful and reusable when passing parameters.

Think of a function as a machine, it takes in an input and produces an output. Parameters are the input. They allow the function to do something useful and reusable based on the input so that you don't have to re-write the same code over and over again.

For example, if you have a function on your website that greets your users by name, you don't want to write code for every single user you have. You can just create a function that takes the user's name as a parameter, and prints the greeting message using that input.

Parameter passing

A parameter is information that the function needs to work. The function parameters are defined in parentheses immediately following the name of the function. You can then use the parameter value in the body of the function.

You need to supply the parameter value when calling the function. This value is called an `argument`.

Let's see an example of adding a parameter to the `greetings` function:

```
function greetings(name){
    const message = `Hello, ${name}!`
    return message
}
console.log(greetings('Dr. Falken'))
console.log(greetings('Joshua'))
```

The function contains the parameter name. It is best practice to pass no more than three or four parameters.

You must respect the order when calling a function with multiple arguments. Otherwise, you'll have unexpected results.

```
function greetings(name, age){
    const message = `Hello, ${name}! Happy ${age}th birthday!`
    return message
}
//right way
console.log(greetings('Dr. Falken', 42))
//wrong way
console.log(greetings(42, 'Dr. Falken')) //Hello 42! Happy Dr. Falkenth
birthday!
```

Naming functions and parameters Well

Function naming is just as important as variable naming. You should choose names that clearly express the purpose of the function clearly and follow a naming convention.

When creating functions, you will often run into situations where you need to write code to check multiple conditions. In JavaScript, this is made easy using the `switch` statement.

Switch conditionals

So far in this course, we've been using `if..else` statements to assess different conditions. Now, it's time to `switch` things up!

Whenever you find yourself in a situation where you need to write multiple `if..else` statements, you should consider using `switch` instead. Here's how the `switch` statement works:

- The `switch` statement evaluates an expression.
- Then, it compares the result of the expression with a specific case.
- Then, it executes any code that's inside the case block until the keyword break is found.
- Alternatively, if no cases are matched, an optional default case is defined and executed. Here's an example of a switch statements:

```
const fruit = 'watermelon';

switch (fruit) {
  case 'Mandarins':
    console.log('Mandarins are $2.59 a pound.');
    break;
  case 'cantaloupes':
    console.log('cantaloupes are $5.29 a pound.');
    break;
  case 'watermelon':
    console.log('cantaloupes and watermelons are $3.79 a pound.');
    // expected output: "cantaloupes and watermelons are $3.79 a pound."
    break;
  default:
    // If not cases are matched, then execute the following code
    console.log(`Sorry, we are out of ${fruit}.`);
}
```

In this example, the constant `fruit` is evaluated based on 3 different cases: does it equal 'Mandarins', 'cantaloupes' or 'watermelon'? This would be comparable to the following `if..else` statements:

```
const fruit = 'watermelon';

if(fruit==='Mandarins')
    console.log('Mandarins are $2.59 a pound.');
```

```
else if(fruit==='cantaloupes')
    console.log('cantaloupes are $5.29 a pound.');
else if(fruit==='watermelon')
    console.log('cantaloupes and watermelons are $3.79 a pound.');
else
    console.log(`Sorry, we are out of ${fruit}.`);
```

Using a `switch` statement makes your code more readable when you have a lot of conditions to write.

Task instructions

In this exercise, you have to create a simple calculator in JavaScript that can perform four operations: Addition, subtraction, multiplication, and division. You can use the `switch` statement to test what the operator should be, then perform the appropriate operation based on that.

- Create a function called calculator. This function take the following arguments:
  - firstnumber: This is the first number in the operation.
  - operator: This is the operator. Could be: "+", "-", "*", or "/".
  - secondnumber: This is the second number in the operation.
- For simplicity, you can assume that all arguments are positive numbers. Here's an example call to the function:

```
//we call the function with 3 parameters: 2 numbers and a string in that
order
calculator(2, '+', 2) // results 4
calculator(3, '-', 2) // results 1
calculator(4, '*', 2) // results 8
calculator(20, '/', 2) // results 10
//Be aware of the case when the divided by zero, it should be `Infinity`
calculator(8, '/', 0) // results Infinity
```

Expected output:

```
4
1
8
10
Infinity
```

Add your code to the file main0.js under the activity1 folder.

Tip: Try one operation at a time, check the results and if that works continue with the rest of the operations. Practice the skill of debugging in the console.

Task

Create a function that works like a calculator.

Passing Objects as Arguments

Function calls with arguments as objects

Check if keys exist in an object

Here is an example of how to check if a key exists in a given object.

```
const bankInfo = {
    name: 'ING',
    address: '79 Strasse',
    city: 'Amsterdam',
    country: 'Europe'
}
console.log(bankInfo['name']) //the key name is ING

// use `in` within a conditional
if('address' in bankInfo){
    console.log('result: ', bankInfo['address'])
}
//will print result:  79 Strasse
//remember to use quotes around the key
```

Iterate over the keys in an object

To loop over the object keys, we can use the `for...in` loop.

Let's say we have an object called `pet`. It has four `key/value pairs`:

```
let pet = {
    name: 'kitty',
    age: 2,
    sound: 'meow',
    owner: 'Atha'
}

for(let key in pet){
    console.log(pet[key]) // 'kitty'
}
//will print
// 'kitty'
//2
//meow
//Atha
```

Task instructions

Compare objects by passing objects into a function. Add you code to the file `main.js` under the activity1 folder.

- Define the function `compareCities`, that accepts two objects as arguments.
- compareCities should return `true` if both objects have exactly the same key/value pairs. Otherwise, `compareCities` should return false. Check the task when done.

Task

- Compare Objects by passing objects into function.