JavaScript
TUTORIAL

JavaScript Functions are First-Class Citizens

Summary: in this tutorial, you'll learn that JavaScript functions are first-class citizens. This means that you can store functions in variables, pass them to other functions as arguments, and return them from other functions as values.

Storing functions in variables

Functions are first-class citizens in JavaScript. In other words, you can treat functions like values of other types.

The following defines the `add()` function and assigns the function name to the variable `sum` :

```
function add(a, b) {
    return a + b;
}

let sum = add;
```

In the assignment statement, we don't include the opening and closing parentheses `()` at the end of the `add` function name. Additionally, we don't execute but reference the function only.

By doing this, we can have two ways to execute the function. For example, we can call it normally as follows:

```
let result = add(10, 20);
```

Alternatively, we can call the `add()` function via the `sum` variable like this:

```
let result = sum(10,20);
```

Passing a function to another function

Since functions are values, you can pass them as arguments into other functions.

For example, the following declares the `average()` function that takes three arguments. The third one is a function:

```
function average(a, b, fn) {
    return fn(a, b) / 2;
}
```

Now, you can pass the `sum` function to the `average()` function as follows:

```
let result = average(10, 20, sum);
```

Put it all together:

```
function add(a, b) {
    return a + b;
}

let sum = add;

function average(a, b, fn) {
    return fn(a, b) / 2;
}

let result = average(10, 20, sum);
```

```
console.log(result);
```

Output:

```
15
```

Returning functions from functions

Since functions are values, you can return a function from another function.

The following `compareBy()` function returns a function that compares two objects by a property:

```
function compareBy(propertyName) {
  return function (a, b) {
    let x = a[propertyName],
      y = b[propertyName];

    if (x > y) {
      return 1;
    } else if (x < y) {
      return -1;
    } else {
      return 0;
    }
  };
}
```

> Note that `a[propertyName]` returns the value of the `propertyName` of the `a` object. It's equivalent to `a.propertyName`. However, if the `propertyName` contains a space like `'Discount Price'`, you need to use the square bracket notation to access it.

Suppose that you have an array of product objects where each product object has two properties: `name` and `price`.

```
let products = [
    {name: 'iPhone', price: 900},
    {name: 'Samsung Galaxy', price: 850},
    {name: 'Sony Xperia', price: 700}
];
```

You can sort an array by calling the `sort()` method. The `sort()` method accepts a function that compares two elements of the array as an argument.

For example, you can sort the product objects based on the name by passing a function returned from the `compareBy()` function as follows:

```
console.log('Products sorted by name:');
products.sort(compareBy('name'));

console.table(products);
```

Output:

```
Products sorted by name:
┌─────────┬──────────────────┬───────┐
│ (index) │       name       │ price │
├─────────┼──────────────────┼───────┤
│    0    │ 'Samsung Galaxy' │  850  │
│    1    │  'Sony Xperia'   │  700  │
│    2    │     'iPhone'     │  900  │
└─────────┴──────────────────┴───────┘
```

Similarly, you can sort the product objects by price:

```
// sort products by prices
console.log('Products sorted by price:');
products.sort(compareBy('price'));
console.table(products);
```

Output:

```
Products sorted by price:
┌─────────┬─────────────────┬───────┐
│ (index) │      name       │ price │
├─────────┼─────────────────┼───────┤
│    0    │  'Sony Xperia'  │  700  │
│    1    │ 'Samsung Galaxy'│  850  │
│    2    │    'iPhone'     │  900  │
└─────────┴─────────────────┴───────┘
```

Put it all together.

```
function compareBy(propertyName) {
  return function (a, b) {
    let x = a[propertyName],
      y = b[propertyName];

    if (x > y) {
      return 1;
    } else if (x < y) {
      return -1;
    } else {
      return 0;
    }
  };
}
let products = [
  { name: 'iPhone', price: 900 },
  { name: 'Samsung Galaxy', price: 850 },
  { name: 'Sony Xperia', price: 700 },
];

// sort products by name
console.log('Products sorted by name:');
products.sort(compareBy('name'));

console.table(products);

// sort products by price
console.log('Products sorted by price:');
products.sort(compareBy('price'));
console.table(products);
```

More JavaScript Functions are First-Class Citizens example

The following example defines two functions that convert a length in centimeters to inches and vice versa:

```
function cmToIn(length) {
    return length / 2.54;
}

function inToCm(length) {
    return length * 2.54;
}
```

The following `convert()` function has two parameters. The first parameter is a function and the second one is the length that will be converted based on the first argument:

```
function convert(fn, length) {
    return fn(length);
}
```

To convert `cm` to `in` , you can call the `convert()` function and pass the `cmToIn` function into the `convert()` function as the first argument:

```
let inches = convert(cmToIn, 10);
console.log(inches);
```

Output:

```
3.937007874015748
```

Similarly, to convert a length from inches to centimeters, you can pass the `inToCm` function into the `convert()` function, like this:

```
let cm = convert(inToCm, 10);
console.log(cm);
```

Output:

```
25.4
```

Put it all together.

```
function cmToIn(length) {
  return length / 2.54;
}

function inToCm(length) {
  return length * 2.54;
}

function convert(fn, length) {
  return fn(length);
}

let inches = convert(cmToIn, 10);
console.log(inches);

let cm = convert(inToCm, 10);
console.log(cm);
```

Output:

```
3.937007874015748
25.4
```

Summary

- Functions are first-class citizens in JavaScript.
- You can pass functions to other functions as arguments, return them from other functions as values, and store them in variables.

Quiz