

Advanced Constructs: Advanced Function Concepts 2

Relevel
by Unacademy



Topics Covered

- Higher Order Functions
- Composability
- Arrow Functions
- Why are functions in JS 'first-class-citizen'
- IIFE
- Taking User input in JS

Higher Order Function

- Higher Order Function (HOF) are functions that takes other function as argument or return function as a result. I hope you are familiar with Arrays and their functions.
- Let me take you into deeper in this HOF concept. Arrays function such as map, filter, sort, reduce, forEach and so on are Higher Order function because these functions accept function as an argument and return desired output.

Programming Example

In this example, using the inbuilt array manipulation functions,

- Using filter function which accept function as an argument with one argument in the function and do the filtration process
- Using sort function to arrange the array in the descending order and that sort function will accept function as an argument with two argument and do the sorting.

```
JS higherOrderFunction.js > [x] filteredArr
1  const arr = [1, 2, 3, 4, 5];
2
3  // Filter the array
4  const filteredArr = arr.filter(function(item){
5    |   return item > 3
6  })
7  console.log(filteredArr); // [4, 5]
8
9  // Sort the array in descending order
10 arr.sort(function(a, b) {
11   |   return b - a;
12 })
13 console.log(arr)
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
PS G:\Github\Relevel> node .\higherOrderFunction.js
[ 4, 5 ]
[ 5, 4, 3, 2, 1 ]
```

Composability

- Function composition is a mechanism of combining multiple simple functions to build a more complicated one. The result of each function is passed to the next one.
- In mathematics, we often write something like: $f(g(x))$. So this is the result of $g(x)$ that is passed to f . In programming we can achieve the composition by writing something similar.
- Let's take a quick example. Suppose I need to make some arithmetic by doing the following operation: $2 + 3 * 5$. As you may know, the multiplication has the priority over the addition. So you start by calculating $3 * 5$ and then when add 2 to the result.

```
JS composition.js > ...  
1  const add = (a, b) => a + b;  
2  const mult = (a, b) => a * b;  
3  add(2, mult(3, 5))
```

When We use Composability

- Let me explain with real time example,
- Think of an industrial plant that produce bottles of cool drinks; first there is the operation (or function) f1f1 that puts the cool drinks inside the bottle, followed by the operation f2f2 that close the bottle with the cap.
- In the above example we need to follow certain series of action, in the similar way we need to implement certain functionality which needs to be followed one after another

Currying

- Currying is when you break down a function that takes multiple arguments into a series of function that each take only one argument.
- In the below example **curryAdd** function is returning a series of function and at the last function it is returning the value.

```
JS currying.js > ...
1 // Ordinary function for addition using 3 arguments
2 const add = (a, b, c) => {
3   return (a + b + c)
4 }
5
6 add(1, 2, 3); // 6
7
8 // Cuurying function for addition
9 const curryAdd = (sum) => {
10   return (a) => {
11     return (b) => {
12       return (c) => {
13         return sum(a, b, c);
14       }
15     }
16   }
17 }
18
19 const addition = curryAdd(add);
20 console.log(addition(1)(2)(3)); // 6
```

- Let me explain more simpler, If you want to buy a chocolate cake in shop.
- What is the process go out of the home -> take a bus -> find the shop -> check whether the chocolate cake is available or not -> if available then buy.
- In the above scenario, let us assume this as a task and we can split each and every in single function calling every function if every thing is good we will be getting the cake but any one of the task is failed we won't get cake and get proper reason and it is easy to find.
- Instead of that if we put all task in a single function and passing multiple argument that is hard to manage, that's why currying comes into picture

Arrow Function

- Traditional function expressions are `function [name]([param1[, param2[, ..., paramN]]]) {
 statements
}`
- The difference between named and unnamed functions are, If function name is omitted, it will be the variable name (implicit name). If function name is present, it will be the function name (explicit name).
- Unnamed functions are called as anonymous function.
- Arrow function is a different form of writing function compare to traditional function and it was introduced in the year 2015 ES6 (ECMAScript6) edition. They are less verbose than traditional function expression.
- Let's have a quick example and comparison of Arrow function with traditional function.
- Arrow functions are new way to write an anonymous function and are similar to Lambda function in other programming languages.
- Syntax: `(argument) => { ... Logic }`

Difference Between Arrow and Regular function

	Regular Function	Arrow Function
Constructor	<pre>function Car(color) { this.color = color; } const redCar = new Car('red'); redCar instanceof Car; // => true</pre> <p>We can create an instance for Car</p>	<pre>const Car = (color) => { this.color = color; }; const redCar = new Car('red'); // TypeError: Car is not a constructor</pre> <p>We cannot create an instance</p>
Argument Object	<pre>let user = { show(){ console.log(arguments); // 1, 2, 3 } }; user.show(1, 2, 3);</pre> <p>Argument object are available</p>	<pre>let user = { show_ar : () => { console.log(...arguments); // error } }; user.show_ar(1, 2, 3);</pre> <p>Argument object are not available</p>
this Keyword	<pre>let user = { name: "Revel", regularfn(){ console.log("hello " + this.name); // 'this' binding here } }; user.regularfn();</pre> <p>this binding here</p>	<pre>let user = { name: "Revel", arrowfn:()=> { console.log("hello " + this.name); // no 'this' binding here } }; user.arrowfn();</pre> <p>no this binding here</p>
Implicit return	<pre>normalfn () { 12; return; }</pre> <p>normalfn(); // undefined</p>	<pre>const arrowfn = () => 44 arrowfn(); // 44</pre>

Programming Example

In this program, function **addTwoNumberTraditional** is the function expression we have seen in the class which is the traditional way of declaring the function.

The function **addTwoNumberArrow** is called as Arrow function because in the expression we are using `=>` and this function is assigning to a variable using `const` keyword, so hoisting will consider this as a variable.

```
JS arrowFunction.js > ...
1  // Traditional function
2  function addTwoNumberTraditional (a, b) {
3    |   return (a + b);
4    | }
5
6  // Arrow function
7  const addTwoNumberArrow = (a, b) => {
8    |   return (a + b);
9    | }
10
11 console.log(addTwoNumberTraditional(1, 2));
12
13 console.log(addTwoNumberArrow(1, 2))
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
PS G:\Github\Relevel> node .\arrowFunction.js
3
3
```

Function – A first class citizen

Function in javascript are first class citizen which means you can store function in a variable, pass function as an argument, return function as a result.

```
JS firstClassCitizen.js > ...
1 // Storing function in a variable
2 const add = (a, b) => {
3   |   return (a + b);
4 }
5
6 const addition = add;
7 add(1, 2);
8 addition(1, 2);
9
10 // passing function as an argument
11 const pass = (func) => {
12   |   return func(1, 2);
13 }
14
15 pass(add);
16
17 // Return function as a result
18 const funcReturn = (a, b) => {
19   |   return () => {
20     |     console.log(a + b + 5);
21   }
22 }
23
24 funcReturn(1, 3)();
```

IIFE (Immediately Invoked Function Expression)

- IIFE is a function that runs as soon as it is declared. Example will help you to understand what is IIFE.
- This is similar to declaring the function and invoking the function but only difference here is it will invoke as soon as it is declared.

```
JS IIFE.js
1  ✓ (() => {
2    |   console.log('IIFE, I am Invoked')
3    | } )();
4
5  // Output : IIFE, I am Invoked
```

Advantages

Secure Variables Scope

As you know var keyword scope is global so to secure the reference we can use IIF

```
(function () {  
  var greeting = 'Good morning! How are you today?';  
  console.log(greeting); // Good morning! How are you today?  
})();console.log(greeting); // error: Uncaught ReferenceError: greeting is not defined
```

As you can see in the example above, what happens in the IIFE scope, stays in the IIFE scope. You can't use the variable defined inside IIFE from the outside.

Avoid Naming Conflict

Using many JavaScript libraries can cause conflicts because some of them might export an object with the same name. Let's say you're using jQuery. We all know it export \$ as its main object. So, if there's any library in your dependencies using \$ as its exported object as well, a conflict will occur.

Fortunately, you can use IIFEs to solve this problem by applying the aliasing technique:

```
(function ($) {  
  // You're safe to use jQuery here  
})(jQuery);
```

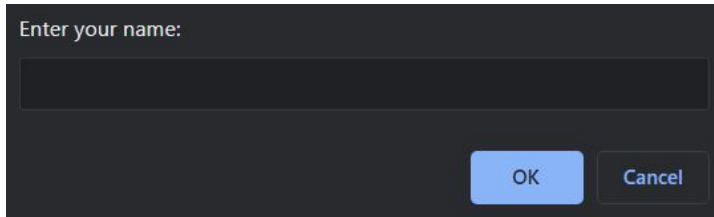
By wrapping your code inside an IIFE that takes jQuery as an argument, we will make sure that the \$ symbol now refers to jQuery, not other libraries.

Taking user input in JS

- We can take user input in various way using javascript, but as a beginner we should be aware of the prompt which will get user input and do the logic as per the next process.
- JavaScript has a few window object methods that you can use to interact with your users. The prompt() method lets you open a client-side window and take input from a user.

```
const name = window.prompt("Enter your name: ");  
alert("Your name is " + name);
```

- Window object will prompt for user input and asking the user to enter name



Enter your name:

OK Cancel

- As per the 2nd line code the will show the name which user had given as an alert.



Get Input using Node

Guide to install Node:

Step 1:

Go to the official website of [Node.js downloads](#) then according to your OS and system bit download the Node.js .msi installer.

Step 2:

Run the downloaded Node.js .msi Installer then follow the steps like accepting the license, selecting the destination, and authenticating for the install. This requires Administrator privileges, and you may need to authenticate for the same.

Step 3:

To check if Node.js has been installed, run `node -v` in your terminal - you should get something like v16.14.2

Bravo! Node.js is installed into your system and it's ready to work.

Method 1:

- To access input from user, you need to create an Interface instance that is connected to an input stream.
- You create the Interface using `readline.createInterface()` method, while passing the input and output options as an object argument.
- Create a file named `testReadlineInterface.js` or a name you prefer and write the below mentioned code in it and then try running the code.

```
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.question("What is your name? ", function (answer) {
  console.log(`Oh, so your name is ${answer}`);
  console.log("Closing the interface");
  rl.close();
});
```

<https://www.jdoodle.com/iembed/v0/pZw>

Method 2:

- Using process, we can get the input in a single time
- Create a file named testStdinUsingProcess.js or a name you prefer and write the below mentioned code in it and then try running the code.

```
process.stdin.resume();
process.stdin.setEncoding('utf8');

// declare global variables
var input_stdin = "";
var chunks = "";
var input_currentline = 0;

// standard input is stored into input_stdin
process.stdin.on('data', function (data) {
    input_stdin += data;
});

// standard input is done and stored into an array
process.stdin.on('end', function () {
    chunks = input_stdin.split("\n");
    start();
});

function start(){
    maxcount = chunks.shift();
    chunks = chunks.filter(function(e){return e});

    chunks.forEach(function(val,index) {
        if(maxcount-- >0){
            solveProblem(val);
        }
    });
}
```

<https://www.jdoodle.com/iembed/v0/pZx>

```
function solveProblem(numba) {  
  var solution = checkfactorial(numba);  
  console.log(solution);  
}  
  
function checkfactorial(number) {  
  var temp = 1;  
  var sum = 0;  
  
  while(temp <= number) {  
    temp = temp*5;  
    sum = sum + Math.floor(number/temp);  
  }  
  
  return sum;  
}
```

MCQ

1. From the below snippet, what is the function used here ?

```
((fn, val) => {  
  return fn(val);  
})(console.log, 5);
```

- A. Higher Order Function
- B. Immediately Invoked Function
- C. Both A and B
- D. None of the above

Answer: C

2. What is the output of the given code ?

```
const obj = {  
  1: 1,  
  2: 2,  
  3: 3,  
};  
console.log(Object.keys(obj) == Object.values(obj));
```

- A. true
- B. false
- C. equal
- D. Null

Answer: B

3. What is the output of the below snippet ?

```
(function(a){  
  return (function(){  
    console.log(a);  
    a = 6;  
  })()  
})(21);
```

- A. 21
- B. 6
- C. undefined
- D. NaN

Answer: A

4. Which of the following are closures in Javascript ?

- A. Variables
- B. Functions
- C. Objects
- D. All of the above

Answer: D

5. Javascript function to display an “Relevel” in the alert box ?

- A. alert(“Relevel”)
- B. message(“Relevel”)
- C. alertBox(“Relevel”)
- D. messageBox(“Relevel”)

Answer: A

Assignments

1. Bikers A and B started the journey and A having X litres of petrol and B having Y litres of petrol. K and L are the km per litre for A and B bikes.

Let us consider halt as an array a1, a2, a3, an where n is the length of the array. a represent distance from one halt to another

Write a program to find each biker will cross how many halts based on the petrol quantity and the litre per km.

2. Write a program to get an input ('How are you') and by default it should be "good" in the prompt text box.

3. Seetha is working as a professor in a college and she is having 100+ student records and she wants to convert student's name into uppercase. Write a program to help Seetha to Change the student's name into uppercase.

4. Assume that the input will be a word with odd letter count.
Print the output as given below. start with middle letter from first line. Next line two letter from the middle. Continue still you print all letters in last line. Then start with the first letter and continue for the remaining letters.

Output:

```
      C
     CO
    COM
   COME
  COMEW
 COMEWE
COMEWEL
```

Upcoming Class Teaser

- Sort array of 0 1
- Sort array of 0 1 2
- Target sum pair in an array
- Target Sum Triplet in an array
- Rain water trapping

Thank You!