CS472 WAP

JavaScript Functional Programming

JavaScript Functional Programming

Maharishi International University Fairfield, Iowa © 2020

Main Point Preview

- Functional programming methods map, filter, reduce make code more understandable and error free by automating details of general-purpose looping mechanisms, indicating their intent by their name, and not changing the state of the original array.

- *Science of Consciousness: Growth of consciousness makes behavior simpler and more error free because intentions that arise from deep levels are spontaneously in accord with and supported by all the laws of nature.*

First-class functions

*

```
//Functions can be assigned to variables
const myfunc = function(a, b) {
 return a * b;
};
//Functions can be passed as parameters
function apply(a, b, applyFunc) {
 const y = applyFunc(a, b);
 return y;
}
const x = apply(2, 3, myfunc); // 6

//Functions can be return values
function getAlert(str) {
 return function() { alert(str); } }

const whatsUpAlert= getAlert("What's up!");
whatsUpAlert(); // “What’s up!”

const mottoAlert= getAlert("Knowledge Is Structured in Consciousness");
mottoAlert(); // “Knowledge Is Structured in Consciousness
```

Arrow functions (ES6)

- Arrow functions(ES6)
  - Syntactic sugar++ for anonymous functions (ala Java lambdas)

- Arrow functions can be a shorthand for an anonymous function in callbacks

```
(param1, param2, …, paramN) => { statements } // return required for { statements }

(param1, param2, …, paramN) => expression // equivalent to: => { return expression; }


(singleParam) => { statements }
singleParam => { statements } // Parentheses optional when only one parameter


// A function with no parameters requires parentheses:
() => { statements }
```

Arrow Functions Example

```
const multiply = function(num1, num2) {
 return num1 * num2;
}
console.log("output: " + multiply(5,5));

const multiply = (num1, num2) => num1 * num2;
console.log("output: " + multiply(5,5));
```

Arrow Functions Example – map

```
//creates new array with results of calling provided function on every element in
  calling array.
const atoms = [
 "Hydrogen",
 "Helium",
 "Lithium",
 "Beryllium"
];
const atomsLength = atoms.map(function(s) { return s.length });
console.log("a2: " + atomsLength);

const atomsLengthV2 = atoms.map(s => s.length);
console.log("a3: " + atomsLengthV2);
```

Arrow Functions Example – filter

//filter returns Array containing all elements that pass the test. If no elements pass returns empty array.

```
const atoms = [
 "Hydrogen",
 "Helium",
 "Lithium",
 "Beryllium"
];
const atomsLong = atoms.filter(function(s) {return s.length > 7 });
const atomsLong2 = atoms.filter( s => s.length > 7 );
```

`//find` returns value of first element that satisfies test. Otherwise undefined is returned.

```
const longAtom = atoms.find( s => s.length > 7 );
const longAtom2 = atoms.findIndex( s => s.length > 7 );
```


Reduce with initial value

- initialValue: optional value for first argument to first call of callback.
  - If no initial value, the first element in array will be used.
- in array *of objects* you **must** supply an initial value
  - otherwise reduce will not know what to use as value from the first object

```
var initialValue = 0;

var sum = [{x: 1}, {x:2}, {x:3}].reduce(
  function (accumulator, currentValue) {
  return accumulator + currentValue.x;},initialValue);

console.log(sum); // logs 6
```

Arrow Functions Example - reduce

- executes a **reducer** function (that you provide) on each member of the array resulting in a single output value.
  - will execute the callback function starting at index 1, using the first value as the initial value

- returned value is assigned to the accumulator, whose value is remembered across each iteration throughout the array and ultimately becomes the final, single resulting value.

```
const array1 = [1, 2, 3, 4];

const reducer = (accumulator, currentValue) => accumulator +
 currentValue;

console.log(array1.reduce(reducer));// expected output: 10
```

Array methods: map, filter, reduce

```
//functional programming:  map, filter, reduce can replace many loops

const a = [1,3,5,3,3];

//return new array with elements translated/mapped to another set of values
const b = a.map(function(elem, i, array) {
  return elem + 3;})// [4,6,8,6,6]

//return new array containing select elements based on a condition
const c = a.filter(function(elem, i, array){
  return elem !== 3;});//[1,5]

//find first element or index of first element satisfying condition
const d = a.find(function(elem) {return elem > 1;}); //3
const e = a.findIndex(function(elem) {return elem > 1;}); //1

//accumulate a single value based on elements across the array
const f = a.reduce(function(accumulator, elem){
  return accumulator + elem;}, 0); //0 is initial value, 15 is sum
```

map/filter/find/reduce are “pure” functions

- Important principle of “functional” programming
- Pure functions have no side effects
  - Do not change state information
  - Do not modify the input arguments
- Take arguments and return a new value
- Valuable benefits for automated program verification, parallel programming, reuse, and readable code
  - no ‘side effects’
  - low coupling

forEach for side effects on array

- ‘forEach’ is another array convenience method that executes a provided function once for each Array element.
  - forEach returns undefined rather than a new array
  - Intended use is for side effects, e.g., writing to output, etc.

```
const arr = ['a', 'b', 'c'];

arr.forEach(function(element) {
element = element + "ha ha ha";
    console.log(element);
});
```

‘for of’ vs ‘for in’ –ES6

- Both for..of and for..in statements iterate over arrays;
- for..in returns keys and works on objects as well as arrays
- for..of returns values of arrays but does not work with object properties

```
let letters = ['x', 'y', 'z'];

for (let i in letters) {
  console.log(i); } // "0", "1", "2",

for (let i of letters) {
  console.log(i); } // “x", “y", “z"
```

‘for in’ over object literal/Arrays –ES6

```
//returns property keys (index) of object
const things = {
 'a': 97,
 'b': 98,
 'c': 99
};

for (const key in things) {
 console.log(key + ', ' + things[key]);
}
```

```
a, 97
b, 98
c, 99
```

Summary ‘for’ loops

- ‘for’ is the basic for loop in JavaScript for looping
  - Almost exactly like Java for loop
  - If not sure what need, use this
- ‘forEach’ is another array convenience method that executes a provided function once for each Array element.
  - forEach returns undefined rather than a new array
  - Intended use is for side effects, e.g., writing to output, etc.
- ‘for of’ is a new convenience (ES6) method for looping through values of ‘iterable’ collections
  - e.g., Array, Map, Set, String
- ‘for in’ is useful for iterating through the properties of objects
  - can also be used to go through the indices of an array
- Best practice to use convenience methods when possible
  - Avoids bugs associated with indices at end points
  - map, filter, find, reduce, forEach, for of, for in

Main Point

- Functional programming methods map, filter, reduce make code more understandable and error free by automating details of general-purpose looping mechanisms, indicating their intent by their name, and not changing the state of the original array.

- *Science of Consciousness: Growth of consciousness makes behavior simpler and more error free because intentions that arise from deep levels are spontaneously in accord with and supported by all the laws of nature.*


MAHARISHI
HIGHER EDUCATION
FOR HIGHER CONSCIOUSNESS
INTERNATIONAL UNIVERSITY