JavaScript

Higher-Order Functions

Au Mau Duong

Technical Trainers

Table of Contents

Higher-Order Functions

- A function that accepts and/or returns another function is called a higher-order function
- It’s higher-order because instead of strings, numbers, or booleans, it goes higher to operate on functions. Pretty meta
- With functions in JavaScript, you can
  - Store them as variables
  - Use them in arrays
  - Pass them as arguments
  - Assign them as object properties (methods)
  - Return them from other functions

Functions Operate on Data (1)

- Strings Are Data

```
sayHi = (name) => `Hi, ${name}!`;
result = sayHi('User');
console.log(result); // 'Hi, User!'
```

- Numbers Are Data

```
double = (x) => x * 2;
result = double(4);
console.log(result); // 8
```

Functions Operate on Data (2)

- Booleans Are Data

```
getClearance = (allowed) => (allowed ? 'Access granted' :
'Access denied');


result1 = getClearance(true);

result2 = getClearance(false);


console.log(result1); // 'Access granted'

console.log(result2); // 'Access denied'
```

Functions Operate on Data (3)

- Objects Are Data

```
getFirstName = (obj) => obj.firstName;


result = getFirstName({
  firstName: 'Yazeed'
});


console.log(result); // 'Yazeed'
```

Functions Operate on Data (4)

- Arrays Are Data

```
len = (array) => array.length;
result = len([1, 2, 3]);


console.log(result); // 3
```

Functions Can Be Data Too (1)

- Functions as Arguments

```
isEven = (num) => num % 2 === 0;

result = [1, 2, 3, 4].filter(isEven);

console.log(result); // [2, 4]
```

- isEven, a function, was a parameter to another function
- It’s called by filter for each number, and uses the returned value *true* or *false* to determine if a number should be *kept* or *discarded*

Functions Can Be Data Too (2)

- Returning Functions

```
add = (x) => (y) => x + y;
```

add requires two parameters, but not all at once

```
result = add(10)(20);
console.log(result); // 30
```

You can still supply 10 and 20 immediately

```
add10 = add(10);
add20 = add10(20);
console.log(result); // 30
```

Or 10 now and 20 later

JavaScript’s premiere Array methods

- Here’s a list of users. We’re going to do some calculations with their information

```
users = [
  { name: 'Yazeed',
    age: 25 },
  { name: 'Sam',
    age: 30 },
  { name: 'Bill',
    age: 20 }
];
```

JavaScript’s premiere Array methods – Map (1)

- Without higher-order functions, we’d always need loops to mimic map's functionality

```
getName = (user) => user.name;
usernames = [];
for (let i = 0; i < users.length; i++) {
  const name = getName(users[i]);
  usernames.push(name);
}
console.log(usernames);
// ["Yazeed", "Sam", "Bill"]
```

```
users = [
  { name: 'Yazeed',
    age: 25 },
  { name: 'Sam',
    age: 30 },
  { name: 'Bill',
    age: 20 }
];
```

JavaScript’s premiere Array methods – Map (2)

- Or we could do this!

```
usernames = users.map(getName);


console.log(usernames);
// ["Yazeed", "Sam", "Bill"]
```

```
users = [
  { name: 'Yazeed',
    age: 25 },
  { name: 'Sam',
    age: 30 },
  { name: 'Bill',
    age: 20 }
];
```

JavaScript’s premiere Array methods – Filter (1)

- Without higher-order functions, we’d still need loops to recreate filter's functionality

```
startsWithB = (string) =>
string.toLowerCase().startsWith('b');
namesStartingWithB = [];
for (let i = 0; i < users.length; i++) {
  if (startsWithB(users[i].name)) {
    namesStartingWithB.push(users[i]);
  }
}
console.log(namesStartingWithB);
// [{ "name": "Bill", "age": 20 }]
```

```
users = [
  { name: 'Yazeed',
    age: 25 },
  { name: 'Sam',
    age: 30 },
  { name: 'Bill',
    age: 20 }
];
```

JavaScript’s premiere Array methods – Filter (2)

- Or we could do this!

```
namesStartingWithB = users.filter((user)
=> startsWithB(user.name));

console.log(namesStartingWithB);
// [{ "name": "Bill", "age": 20 }]
```

```
users = [
  { name: 'Yazeed',
    age: 25 },
  { name: 'Sam',
    age: 30 },
  { name: 'Bill',
    age: 20 }
];
```

JavaScript’s premiere Array methods – Reduce (1)

- Without higher-order functions, we’d still need loops to recreate reduce's functionality

```
total = 0;
for (let i = 0; i < users.length; i++) {
  total += users[i].age;
}
console.log(total); // 75
```

```
users = [
  { name: 'Yazeed',
    age: 25 },
  { name: 'Sam',
    age: 30 },
  { name: 'Bill',
    age: 20 }
];
```

JavaScript’s premiere Array methods – Reduce (2)

- Or we could do this!

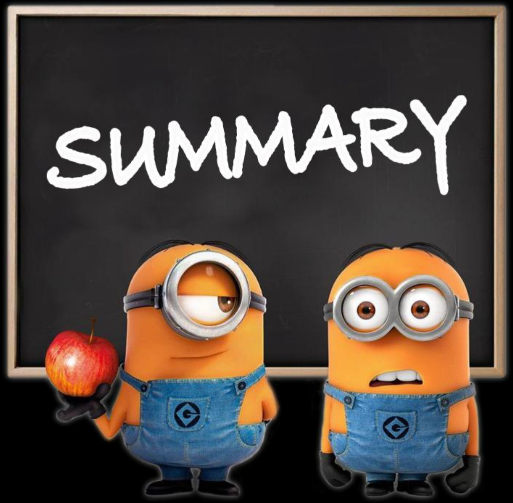
```
totalAge = users.reduce((total,
user) => user.age + total, 0);


console.log(totalAge);
// 75
```

```
users = [
  { name: 'Yazeed',
    age: 25 },
  { name: 'Sam',
    age: 30 },
  { name: 'Bill',
    age: 20 }
];
```

Summary

- Functions Operate on Data
- Functions Can Be Data
- JavaScript’s premiere Array methods



JavaScript – Higher-Order Functions

Questions?