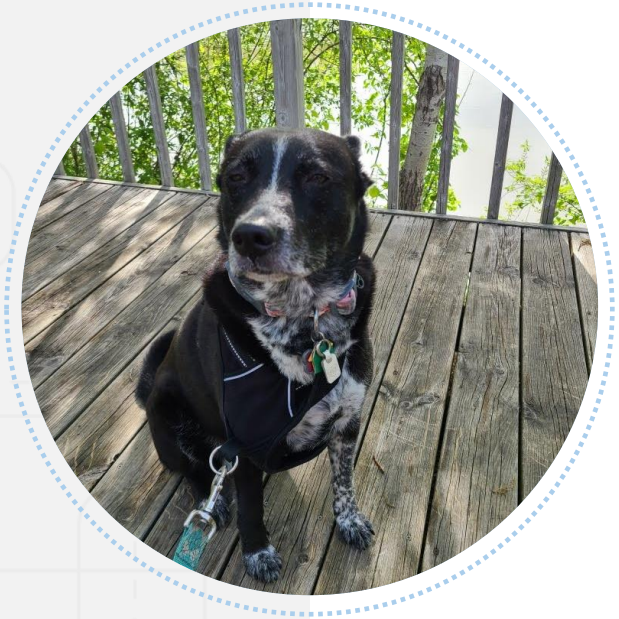
Reach Higher with Higher-Order Functions

Josh Doro



About Me

- Technical Architect with nvisia
- Coding professionally for 10 years
- Live in Milwaukee with my wife and dog
- I love Star Wars

nvisia
connect. build. enable.

About nvisia

- Software consulting company
- Chicago, Milwaukee, Madison
- A group of like-minded technologists that love to help build and maintain software at every point in the life-cycle.
- We love being involved in the community.

nvisia
connect. build. enable.

Why Do This?

What are We Going to Talk About?

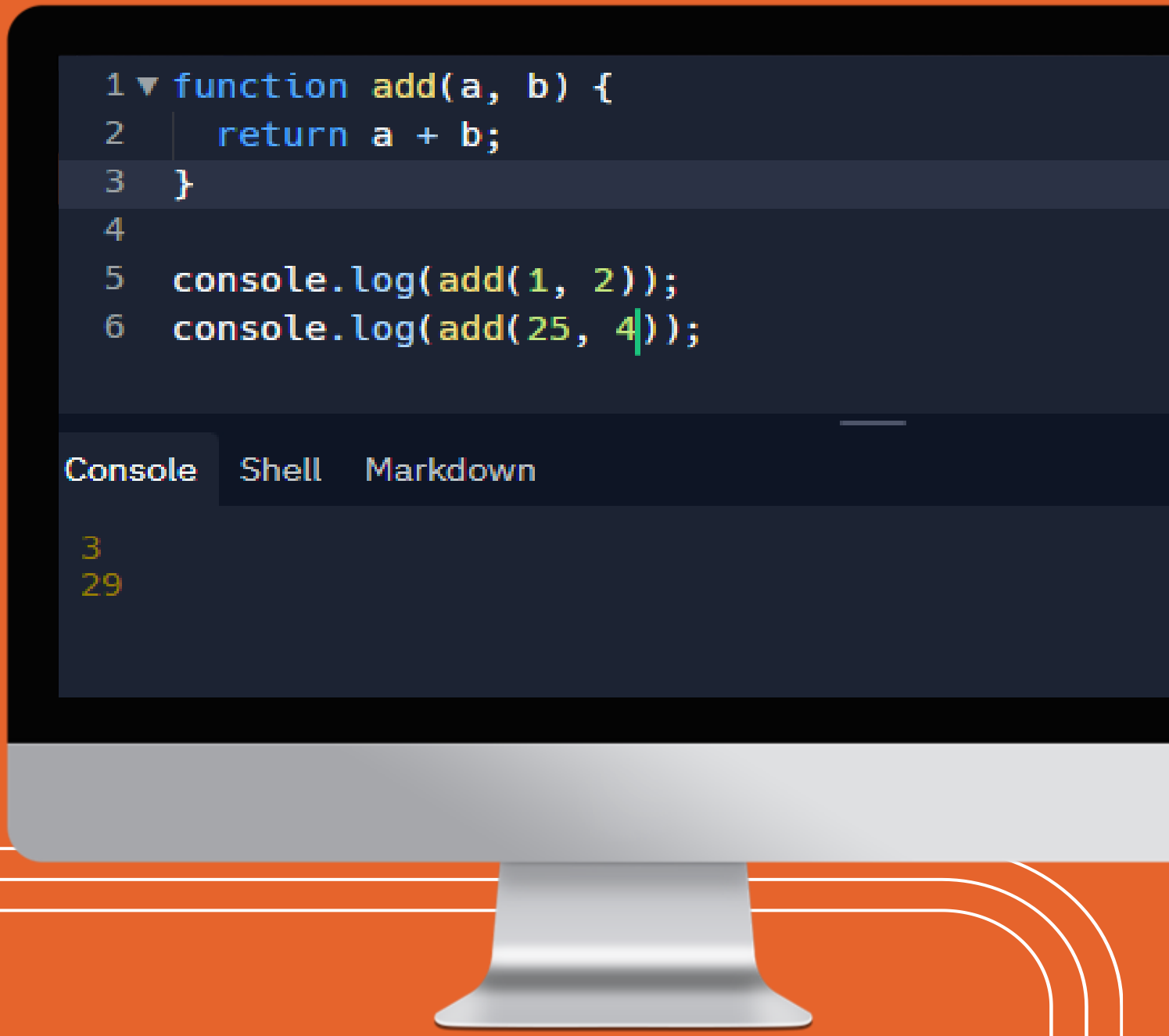
What is a higher-order function?

Why Use Them?

Examples

What is a function?

- A block of code that encapsulates logic
- Invocable multiple times with different inputs
- Breaks down our problems



nvisia
connect. build. enable.

First-Order Function

```
1  const num1 = 5;
2  const num2 = 2;
3
4  function add(a, b) {
5    return a + b;
6  }
7
8  console.log(add(num1, num2));
9
10 function subtract(a, b) {
11   return a - b;
12 }
13
14 console.log(subtract(num1, num2));
15
16 function multiply(a, b) {
17   return a * b;
18 }
19
20 console.log(multiply(num1, num2));
21
22 function add5(a) {
23   return a + 5;
24 }
25
26 console.log(add5(num1));
```

Console | Shell | Markdown

```
7
3
10
10
```

nvisia
connect. build. enable.

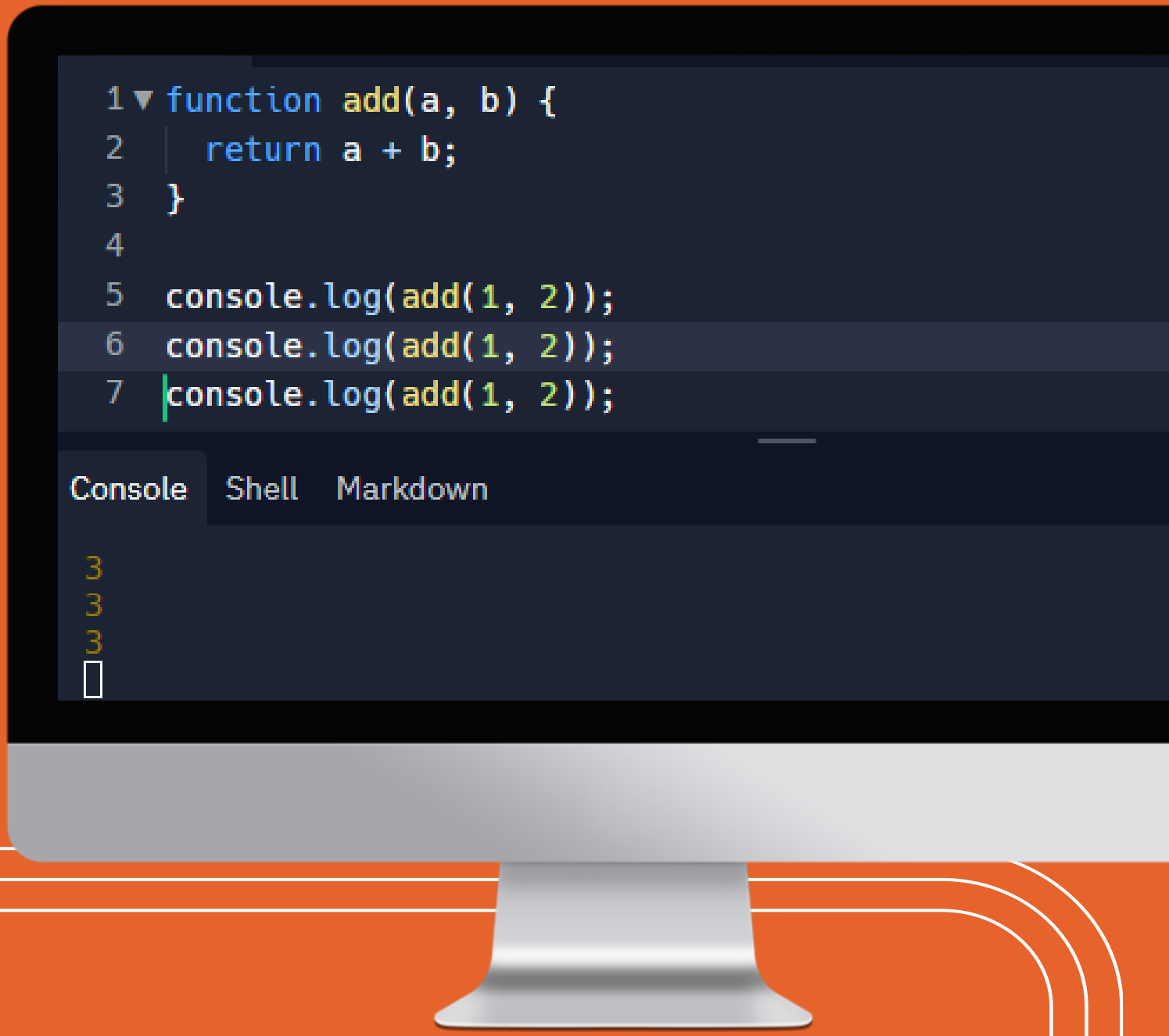
Drive thru: Will that be all?

Me:

Pure Function

Idempotent

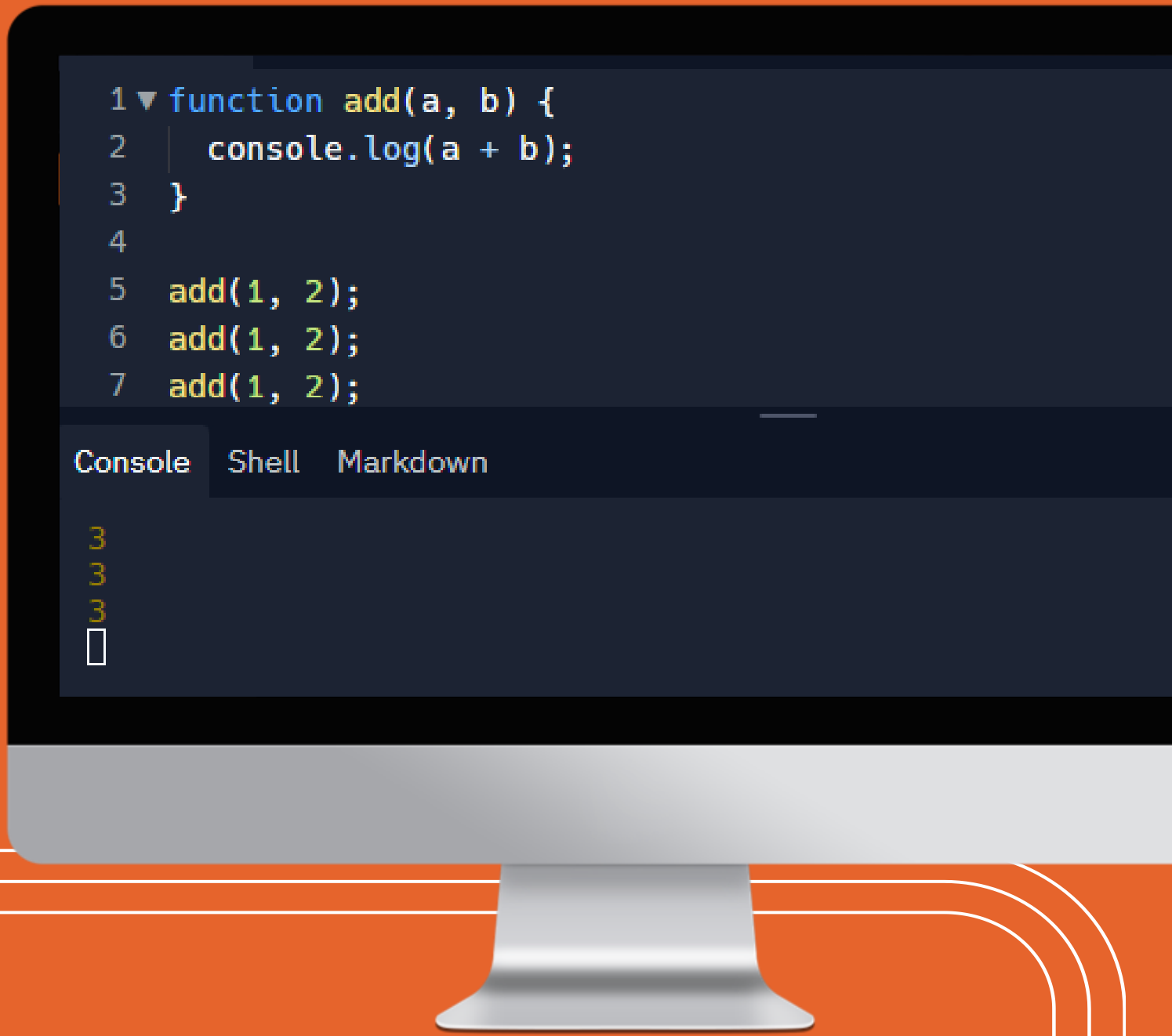
- Given a set of arguments the function will always produce the same result.



nvisia
connect. build. enable.

Side-Effects

- References entities outside of the function's scope.
- Pure functions don't have any
- Unavoidable



nvisia
connect. build. enable.

First Class

- Like every other variable


```
1 ▼ function add(a, b) {
2       return a + b;
3   }
4
5   const num = 5;
6   const greeting = 'hello';
7
8   console.log(add(num, 2));
9
10  const plus = add; // no parens
11  console.log(plus(num, 2))
12
13  const concat = plus;
14  console.log(concat(greeting, ' world!'))
15
16  console.log(add(num, 2));
```

Console | Shell | Markdown

```
7
7
hello world!
7
```

nvisia
connect. build. enable.

What are Higher Order Functions?

Any function that takes a function as an argument or returns a function as its result.

```
1 ▼ function vader() {
2       console.log('I am your Father');
3   }
4
5 ▼ function luke() {
6       console.log('NOOOOOOOO!!!');
7   }
8   vader();
9   luke();
```

Console Shell Markdown

```
I am your Father
NOOOOOOOO!!!
```

```
1 ▼ function luke() {
2       console.log('NOOOOOOOO!!!');
3   }
4
5 ▼ function vader(responseFunc) {
6       console.log('I am your Father');
7       responseFunc();
8   }
9   vader(luke);
```

Console Shell Markdown

```
I am your Father
NOOOOOOOO!!!
```

nvisia
connect. build. enable.

```
1 function vaderSpeaks() {
2   console.log('Vader: I am your Father');
3 }
4
5 function lukeSpeaks() {
6   console.log('Luke: NOOOOOOOO!!!');
7 }
8 vaderSpeaks();
9 lukeSpeaks();
```

Console Shell Markdown

```
Vader: I am your Father
Luke: NOOOOOOOO!!!
```

```
1  function actor(name) {
2    return function speak(dialogue) {
3      console.log(`${name}: ${dialogue}`)
4    }
5  }
6
7  const vaderSpeaks = actor('Vader')
8  const lukeSpeaks = actor('Luke');
9
10 vaderSpeaks('Obi-Wan never told you what happened to your
   father.');
11 lukeSpeaks('He told me enough! It was you who killed him.');
12 vaderSpeaks('No. I am your Father');
13 lukeSpeaks('NOOOOOOOO!!!');
14
15 actor('Obi-Wan')('Luke, use the force!');
```

Console Shell Markdown

```
Vader: Obi-Wan never told you what happened to your father.
Luke: He told me enough! It was you who killed him.
Vader: No. I am your Father
Luke: NOOOOOOOO!!!
Obi-Wan: Luke, use the force!
```

Closure

```
1 ▼ function init() {
2     const message = 'Hello There!';
3 ▼   function displayMessage() {
4         console.log(message);
5     }
6     displayMessage();
7 }
8
9 init();
10
11 console.log(message);
```

Console Shell Markdown

```
Hello There!
/home/runner/HOF-Presentation/index.js:11
console.log(message);
            ^

ReferenceError: message is not defined
```

```
1 ▼ function init() {
2     const message = 'Hello There!';
3 ▼   function displayMessage() {
4         console.log(message);
5     }
6     return displayMessage;
7 }
8
9 const initialize = init();
10 initialize();
```

Console Shell Markdown

```
Hello There!
```

Why Use Higher-Order Functions?

Composition

```
function isEven(arg) {
  return (arg % 2) === 0;
}

function greaterThan(val) {
  return function(arg) {
    return arg > val;
  }
}

function and(predA, predB) {
  return function(arg) {
    return predA(arg) && predB(arg);
  }
}
```

nvisia
connect. build. enable.

Composition

```
17  const greaterThan2 = greaterThan(2)
18
19  const isEvenAndGreaterThan2 = and(isEven, greaterThan2);
20
21  console.log(isEvenAndGreaterThan2(2));
22  console.log(isEvenAndGreaterThan2(3));
23  console.log(isEvenAndGreaterThan2(4));
24  console.log(isEvenAndGreaterThan2(5));
```

Console Shell Markdown

```
false
false
true
false
```

nvisia
connect. build. enable.

Composition

```
26 ▼ function or(predA, predB) {
27 ▼   return function(arg) {
28       return predA(arg) || predB(arg);
29     }
30   }
31
32   const isEvenAndGreaterThan2OrAnyOver10 = or(isEvenAndGreaterThan2,
     greaterThan(10));
33
34   console.log(isEvenAndGreaterThan2OrAnyOver10(2));
35   console.log(isEvenAndGreaterThan2OrAnyOver10(3));
36   console.log(isEvenAndGreaterThan2OrAnyOver10(4));
37   console.log(isEvenAndGreaterThan2OrAnyOver10(5));
38   console.log(isEvenAndGreaterThan2OrAnyOver10(11));
```

Console Shell Markdown

```
false
false
true
false
true
```

Testability

- We can write tests and feel confident!

```
test("test-isEven-odd-arg", async function() {
  expect(index.isEven(3)).toBe(false);
});
test("test-isEven-even-arg", async function() {
  expect(index.isEven(2)).toBe(true);
});
```

```
test("test-and-both-true", async function() {
  function passThrough(val) { return val}
  expect(index.and(passThrough, passThrough)(true))
    .toBe(true);
});
test("test-and-both-false", async function() {
  function passThrough(val) { return val}
  expect(index.and(passThrough, passThrough)(false))
    .toBe(false);
});
test("test-and-first-false", async function() {
  function passThrough(val) { return val}
  function returnFalse() {return false}
  expect(index.and(returnFalse, passThrough)(true))
    .toBe(false);
});
test("test-and-second-false", async function() {
  function passThrough(val) { return val}
  function returnFalse() {return false}
  expect(index.and(passThrough, returnFalse)
(true)).toBe(false);
});
```

Declarative

- This will make your code more declarative. Your code will describe what needs to be done instead of how to do it.

```
or(and(isEven, greaterThan(2)), greaterThan(10));
```

You may already be using them!

- React
- Callbacks - not the best use-case but sometimes necessary
- Promise chains
- array loop methods
- Any lambda in any language

Examples

Array Loop Methods

- .map() – transform each entry
- .filter() – remove entries that don’t meet this condition
- .reduce() – combine all entries to one value

Transform Each Entry

```
1  const arr = [1, 2, 3, 4];
2  const newArr = [];
3  for(let i = 0; i < arr.length; i++) {
4      newArr.push(arr[i] * 2);
5  }
6
7  console.log(newArr);
```

Console Shell Markdown

```
[ 2, 4, 6, 8 ]
```

```
1  const arr = [1, 2, 3, 4];
2
3  function double(x) {
4      return x * 2;
5  }
6
7  const newArr = arr.map(double)
8
9  console.log(newArr);
```

Console Shell Markdown

```
[ 2, 4, 6, 8 ]
```

nvisia
connect. build. enable.

Only Bring These Over

```
1  const arr = [1, 2, 3, 4];
2  const newArr = [];
3  for(let i = 0; i < arr.length; i++) {
4    if(arr[i] % 2) newArr.push(arr[i]);
5  }
6
7  console.log(newArr);
```

Console Shell Markdown

```
[ 1, 3 ]
```

```
1  const arr = [1, 2, 3, 4];
2  function isOdd(x) {
3    return x % 2;
4  }
5  const newArr = arr.filter(isOdd);
6
7  console.log(newArr);
8  console.log(arr);
```

Console Shell Markdown

```
[ 1, 3 ]
[ 1, 2, 3, 4 ]
```

nvisia
connect. build. enable.

Combine Everything Into One

```
1  const arr = [1, 2, 3, 4];
2  let sum = 0;
3  for(let i = 0; i < arr.length; i++) {
4      sum += arr[i];
5  }
6
7  console.log(sum);
```

Console Shell Markdown

```
10
```

```
1  const arr = [1, 2, 3, 4];
2  function add(a, b) {
3      return a + b;
4  }
5  const sum = arr.reduce(add, 0);
6
7  console.log(sum);
```

Console Shell Markdown

```
10
[ 1, 2, 3, 4 ]
```

```
const sum = arr.reduce(function(currentSum, currentValue){
  return currentSum + currentValue
}, 0);
```

nvisia
connect. build. enable.

Disclaimer

- These are my own simplified implementations of these functions
- They could probably use some polish
- Many already exist in libraries

I don’t need those other arguments

```
40  const arr = [11, 5, 9, 17, 25, 3];
41
42  arr.forEach(console.log);
43
```

Console Shell Markdown

```
11 0 [ 11, 5, 9, 17, 25, 3 ]
5 1 [ 11, 5, 9, 17, 25, 3 ]
9 2 [ 11, 5, 9, 17, 25, 3 ]
17 3 [ 11, 5, 9, 17, 25, 3 ]
25 4 [ 11, 5, 9, 17, 25, 3 ]
3 5 [ 11, 5, 9, 17, 25, 3 ]
```

```
40  const arr = [11, 5, 9, 17, 25, 3];
41
42  function unary(func) {
43    return function(firstArg) {
44      return func(firstArg);
45    }
46  }
47
48  arr.forEach(unary(console.log));
```

Console Shell Markdown

```
11
5
9
17
25
3
```

nvisia
connect. build. enable.

Not() what I wanted

```
50  const arr = [11, 5, 9, 17, 25, 3];
51
52  arr.filter(function(x){
53    return !greaterThan(10)(x)
54  }).forEach(unary(console.log));
```

Console Shell Markdown

```
5
9
3
```

```
50  const arr = [11, 5, 9, 17, 25, 3];
51
52  function not(pred) {
53    return function(...args) {
54      return !pred(...args);
55    }
56  }
57
58  arr.filter(not(greaterThan(10))).forEach(unary(console.log));
```

Console Shell Markdown

```
5
9
3
```

nvisia
connect. build. enable.

Constant Arguments

```
64 function add(a, b) {
65   return a + b;
66 }
67
68 function subtract(a, b) {
69   return a - b;
70 }
71
72 function multiply(a, b) {
73   return a * b;
74 }
75
76 console.log(add(5, 3));
77 console.log(subtract(5, 3));
78 console.log(multiply(5, 3));
79
```

Console Shell Markdown

```
8
2
15
```

```
80 function withArgs(...args) {
81   return function (func) {
82     return func(...args);
83   }
84 }
85
86 const funcs = [add, subtract, multiply];
87
88 funcs.map(withArgs(5, 3)).forEach(unary(console.log));
89
```

Console Shell Markdown

```
8
2
15
```

nvisia
connect. build. enable.

Just Constant

```
121 function constant(val) {
122   return function() {
123     return val;
124   }
125 }
126
127 const ten = constant(10);
128
129 console.log(ten(1));
130 console.log(ten(6));
131 console.log(ten(888));
132 console.log(ten('hello'));
133 console.log(ten('hello', 5, 2));
134
```

Console Shell Markdown

```
10
10
10
10
10
```

Only in some cases

```
103  const arr = [11, 5, 9, 17, 25, 3];
104
105  function capAt10(arg) {
106    if(arg > 10) return 10;
107    return arg;
108  }
109
110  arr.map(capAt10).forEach(unary(console.log));
111
```

Console Shell Markdown

```
10
5
9
10
10
3
```

```
103  const arr = [11, 5, 9, 17, 25, 3];
104
105  function conditional(pred, func) {
106    return function(arg) {
107      if(pred(arg)) return func(arg);
108      return arg;
109    }
110  }
111
112  arr.map(conditional(greaterThan(10),
     ten)).forEach(unary(console.log));
113
```

Console Shell Markdown

```
10
5
9
10
10
3
```

nvisia
connect. build. enable.

Get a property From an Object

```
114 const records = [
115   { user: { name: {first: 'Luke', last: 'Skywalker' } } },
116   { user: { name: {first: 'Luke', last: 'Warm' } } },
117   { user: { name: {first: 'Luke', last: 'Atmee' } } }
118 ]
119
120 function getLastName(record) {
121   const user = record ? record.user : undefined;
122   const name = user ? user.name : undefined;
123   return name.last;
124 }
125
126 records.map(getLastName).forEach(unary(console.log));
127
```

Console Shell Markdown

```
Skywalker
Warm
Atmee
```

```
128 function getProp(keyPath) {
129   return function(obj) {
130     return keyPath.split('.')
131     .reduce(function(currObj, currKey) {
132       if(currObj) return currObj[currKey];
133     }, obj)
134   }
135 }
136
137 records.map(getProp('user.name.last')).forEach(unary(console.log))
138
```

Console Shell Markdown

```
Skywalker
Warm
Atmee
```

```
139 records.map(getProp('user.name.first'))
140   .forEach(unary(console.log));
141
```

Console Shell Markdown

```
Luke
Luke
Luke
```

nvisia
connect. build. enable.

Once

```
180 function once(func) {
181   let hasRun = false;
182   return function(...args) {
183     if (!hasRun) {
184       func(...args);
185       hasRun = true;
186     };
187   }
188 }
189
190 function initApp() {
191   console.log('initialized app');
192 }
193
194 const initialize = once(initApp);
195
196 initialize();
197 initialize();
198 initialize();
```

Console Shell Markdown

```
initialized app
```

Don’t ever let go

```
200 function expensiveApiCall(arg) {
201   console.log('making expensive call for arg: ' + arg);
202   return arg;
203 }
204
205 const cache = {};
206
207 function memoizeExpensiveApiCall(arg) {
208   if (cache[arg]) return cache[arg];
209   const result = expensiveApiCall(arg);
210   cache[arg] = result;
211   return result;
212 }
213
214 console.log(memoizeExpensiveApiCall(5))
215 console.log(memoizeExpensiveApiCall(5))
216 console.log(memoizeExpensiveApiCall(11))
217 console.log(memoizeExpensiveApiCall(11))
218 console.log(memoizeExpensiveApiCall(5))
```

Console Shell Markdown

```
making expensive call for arg: 5
5
5
making expensive call for arg: 11
11
11
5
```

```
220 function memoize(func) {
221   const cache = {};
222   return function(arg) {
223     if (cache[arg]) return cache[arg];
224     const result = func(arg);
225     cache[arg] = result;
226     return result;
227   }
228 }
229
230 const memoizeExpensiveApiCall = memoize(expensiveApiCall);
231
232 console.log(memoizeExpensiveApiCall(5))
233 console.log(memoizeExpensiveApiCall(5))
234 console.log(memoizeExpensiveApiCall(11))
235 console.log(memoizeExpensiveApiCall(11))
236 console.log(memoizeExpensiveApiCall(5))
```

Console Shell Markdown

```
making expensive call for arg: 5
5
5
making expensive call for arg: 11
11
11
5
```

nvisia
connect. build. enable.

nvisia
connect. build. enable.

Tips

1. Try to find pure abstractions
2. Use closures for private scope
3. Be mindful of mutation
4. Keep your naming consistent

Questions?

Thank You!

Libraries:

- Lodash
- Ramda

Books:

- Functional-Light Javascript by Kyle Simpson
- Mastering Javascript Functional Programming by Federico Kereki

nvisia
connect. build. enable.