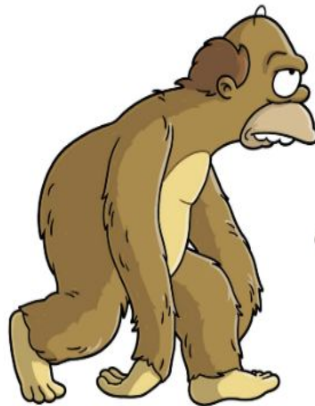
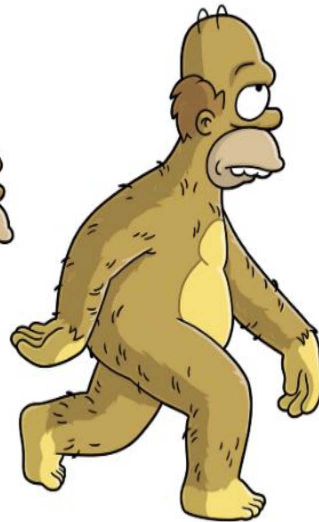
Manjula Dube
Senior Developer @BookMyShow
Mentor for javascript
Leading Google Developer Group - WTM
@manjula_dube

Functional Programming

MACHINE

ASSEMBLY


PROCEDURAL


OBJECT ORIENTED

FUNCTIONAL

Learning to Drive

JavaScript is a multi-paradigm language

Imperative

Functional

Object oriented

What is functional programming?

Paradigm

Coding style

Mindset

- Loops
    - **while**
    - **do...while**
    - **for**
    - **for...of**
    - **for...in**
- Variable declarations with **var** or **let**
- Void functions
- Object mutation (for example: **o.x = 5;**)
- Array mutator methods
    - **copyWithin**
    - **fill**
    - **pop**
    - **push**
    - **reverse**
    - **shift**
    - **sort**
    - **splice**
    - **unshift**

Why functional in javascript?

Looks cool on the resume

Established community

Object-oriented JS gets tricky
prototypes?? this ??

CURSE OF FUNCTIONAL PROGRAMMING

Once you understand it,
you lose the ability to explain it to others

OK Lets do it :)

HOW???

Do everything with functions

Input -> Output

```
> var name = " Manjula";
  var greeting = "Hi I am";
  console.log(greeting + name);
  Hi I am Manjula
```

Non Functional

```
> function greet(name){
      return 'Hi I am '+ name;
  }

  greet("Manjula")
<· "Hi I am Manjula"
```

Functional

Avoid side effects

Use “pure” fucntions

Not pure

```
var name = "Manjula";
function greet(){
    console.log("Hi I am "+ name)
}
```

Pure

```
> function greet(name){
      return ("Hi I am "+ name)
  }
```

Use higher order functions

Function can be Input or Output

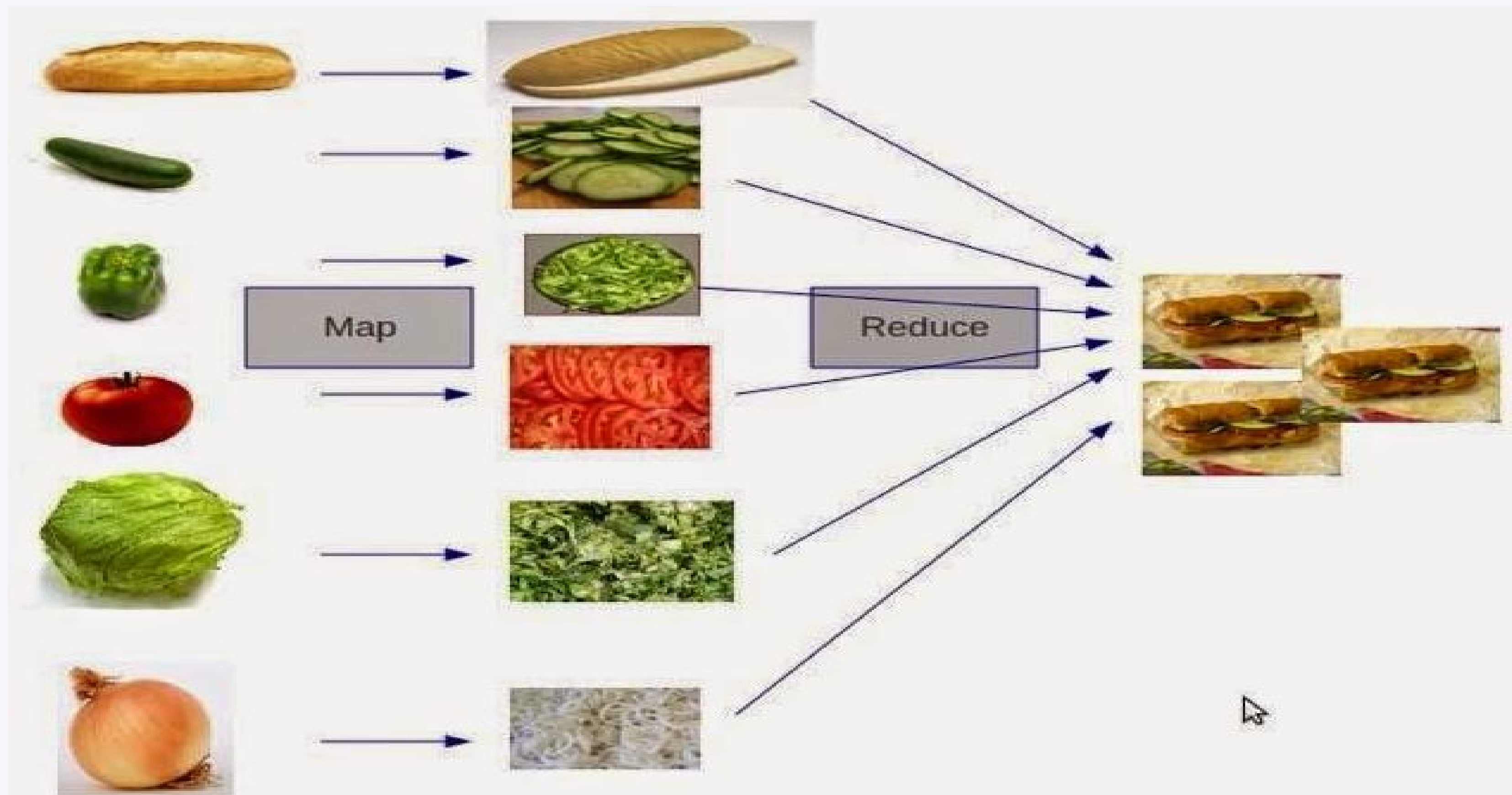
```
function makeAdjectifier(adjective)
{
    return function (string)
     {
        return adjective + “ ” + string;
     };
}

var coolifier = makeAdjectifier(“superb”);
coolifier(“meetup”);
```

Output: “superb meetup”

Don’t iterate

use map, reduce, filter


Map
Reduce

Avoid mutability

Use immutable data

Mutation (bad)

```
> var rooms = ["H1", "H2", "H3"];
  rooms[2] = "H4";
  rooms

<· ▶ (3) ["H1", "H2", "H4"]
```

No mutation (good!!)

```
> var rooms = ["H1", "H2", "H3"];
  var newRooms = rooms.map(function (rm)
   {
      if (rm == "H3")
      {
          return "H4"; }
      else
      {
          return rm;
      }
  });
  rooms
<· ▶ (3) ["H1", "H2", "H3"]
> newRooms
<· ▶ (3) ["H1", "H2", "H4"]
```

Persistent data structures for efficient immutability

Mori, Immutable.js

Ready to try it out?

Mori (http://swannodette.github.io/mori)

Immutable (https://facebook.github.io/immutable-js/)

Underscore (http://underscorejs.org)

Lodash (https://lodash.com)

Ramda (http://ramdajs.com)

EXAMPLE

```javascript
let heightRequirement = 46;

function canRide(height) {
 return height >= heightRequirement;
}

function multiply(a, b) {
 console.log('Arguments: ', a, b);
 return a * b;
}
```

```
let heightRequirement = 46;
function canRide(height) {
 return height >= heightRequirement;



setInterval(() => heightRequirement = Math.floor(Math.random() * 201),
500);

const mySonsHeight = 47;
// Every half second, check if my son can ride.
// Sometimes it will be true and sometimes it will be false.
setInterval(() => console.log(canRide(mySonsHeight)), 500);
```

```
const heightRequirement = 46;



function canRide(height) {
 return height >= heightRequirement;
}
```

```javascript
const constants = {
 heightRequirement: 46,
 // ... other constants go here
};


function canRide(height) {
 return height >= constants.heightRequirement;
}
```

The object should be immutable and the variable cannot be reassigned. This is what we want to achieve!!!!!

```javascript
const o4 = Object.freeze({ foo: 'never going to change me'
});
// Cannot mutate the object
// o4.foo = 'talk to the hand' // Error!



// Cannot reassign the variable
// o4 = { message: "ain't gonna happen, sorry" }; // Error
```

```javascript
const a = Object.freeze([4, 5, 6]);
// Instead of: a.push(7, 8, 9);
const b = a.concat(7, 8, 9);
// Instead of: a.pop();
const c = a.slice(0, -1);
// Instead of: a.shift();
const e = a.slice(1);
// Instead of: a.unshift(1, 2, 3);
const d = [1, 2, 3].concat(a);
```

Function composition

Recursion

n! = n * (n-1) * (n-2) * ... * 1.

That is, **n!** is the product of all the integers from **n** down to **1**. We can write a loop that computes that for us easily enough.

```
function iterativeFactorial(n) {
 let product = 1;
 for (let i = 1; i <= n; i++) {
   product *= i;
 }
 return product;
}
```

Notice that both **product** and **i** are repeatedly being reassigned inside the loop. This is a standard procedural approach to solving the problem. How would we solve it using a functional approach?

```
function factorial(n, product = 1) {
 if (n === 0) {
   return product;
 }
 return factorial(n - 1, product * n)
}
```

Currying

https://we-are.bookmyshow.com/currying-in-javascript-c5fe318c1829

Want to learn more?

“An introduction to functional programming”

by Mary Rose Cook

https://codewords.recurse.com/issues/one/an-introduction-to-functional-programming

Thank you

@manjula_dube