Getting better

Getting better

- Dirty Function
- Clear function
- Wrapping
- Composition
- Immutability
- Curry
- Recursion

Dirty Function

```
let y = 2;
let z = 3;

function f(x) {
   y = y * x;
   z = y * x;
}
```

Dirty Function

```
f(5);
console.log("y " + y);
console.log("z " + z);

f(5);
console.log("y " + y);
console.log("z " + z);
```

EVOLUTION KILLS

Clear function

```
function bar(x, y, z) {
    f(x);
    return [y, z];

    function f(x) {
        y = y * x;
        z = y * x;
    }
}
```

Clear function

```
let res1 = bar(5, 2, 3);
let res2 = bar(5, ...res1);
console.log(res1);
console.log(res2);


let res3 = bar(5, ...bar(5, 2, 3));
console.log(res3);
```

Wrapping

```
let y = 2;
let z;

function f(x) {
    y++;
    z = x * y;
}

f(10);
console.log("y " + y);
console.log("z " + z);

f(5);
console.log("y " + y);
console.log("z " + z);
```

Wrapped

```
function bar(x, y) {
    let z;

    f(x);
    return [y, z];

    function f(x) {
        y++;
        z = x * y;
    }
}

let res1 = bar(10, 2);
console.log("res1");
console.log(res1);

let res2 = bar(10, res1[0]);
console.log("res2");
console.log(res2);
```

Composition

```
let sum = (x, y) => x + y;
let mul = (x, y) => x * y;

let z = mul(2, 3); // side effect
let x = sum(z, 5);
```

Composition - manual

```
let sum = (x, y) => x + y;
let mul = (x, y) => x * y;

let x = sum(mul(2, 3), 5);

console.log(x);
```

Composition - manual

```
let sum = (x, y) => x + y;
let mul = (x, y) => x * y;

let mulAndSum = (x, y, z) => sum(mul(x, y), z);

let x = mulAndSum(2, 3, 5);
console.log(x);
```

Composition - Utility

```
let sum = (x, y) => x + y;
let mul = (x, y) => x * y;

function compose(f1, f2) {
    return function fc() {
        var args = [].slice.call(arguments);
        return f2(f1(args.shift(), args.shift()), args.shift());
    };
}

let mulAndSum = compose(mul,sum);

let x = mulAndSum(2, 3, 5);
console.log(x);
```

Composition - Utility

```
let sum = (x, y) => x + y;
let mul = (x, y) => x * y;

function compose(f1, f2) {
    return function fc() {
        var args = [].slice.call(arguments);
        return f2(f1(args.shift(), args.shift()), args.shift());
    };
}

let x = compose(mul,sum)(2, 3, 5);

console.log(x);
```

Immutability

```
let x = 1;
x++;

const y = 1;
y++;
```

Immutability

```
const z = [1, 2];
z = 10;
z = [3, 4];
z[0] = 3;
z.length = 0;

const w = Object.freeze([1, 2]);
w = 10;
w = [3, 4];
w[0] = 3;
w.length = 0;
```

Immutability

```
const z = [1, 2];
const w = Object.freeze(z);

z[0] = 3;
w[0] = 3;

console.log(z);
console.log(w);
```

Immutability

```
const z = [1, 2];
z[0] = 3;

const w = Object.freeze(z);
z[0] = 3;
w[0] = 3;

console.log(z);
console.log(w);
```

Immutability

```
const z = [1, 2];

let temp = z;

const w = Object.freeze(temp);
z[0] = 3;
temp[0] = 3;
w[0] = 3;

console.log(z);
console.log(w);
```

Immutability

```
const z = [1, 2];
z = Object.freeze(z);
```

Immutability

```
let doubleMe = ar => {
    for (let i = 0; i < ar.length; i++) {
        ar[i] *= 2;
    }
};

let myAr = [0, 1, 2];
doubleMe(myAr);
console.log(myAr);
```

Immutability

```
let doubleMeImmutableStyle = ar => {
    let result = [];
    for (let i = 0; i < ar.length; i++) {
        result.push(ar[i] * 2);
    }
    return result;
};

let myAr = [0, 1, 2];
console.log(doubleMeImmutableStyle(myAr));
console.log(myAr);
```

Curry

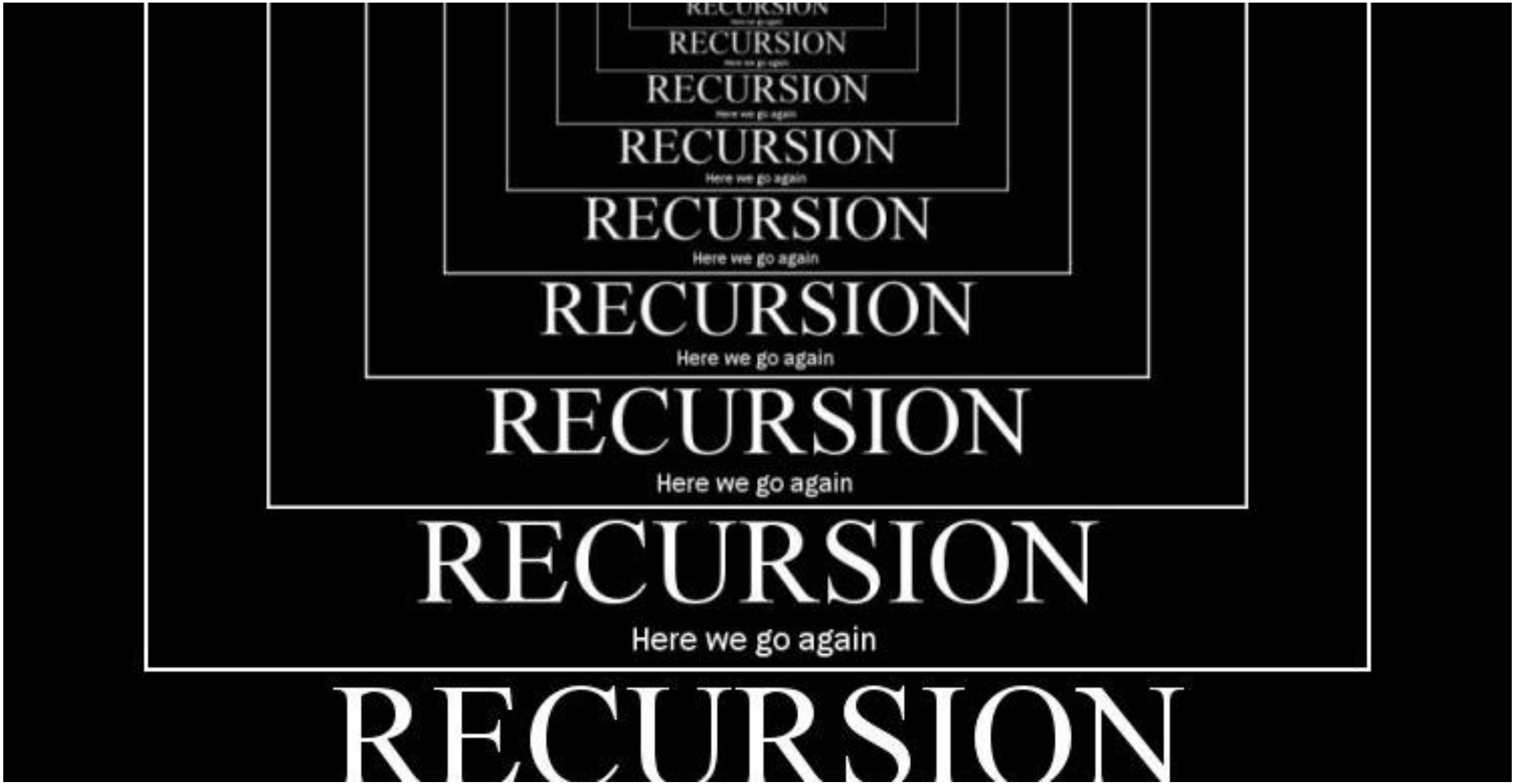
```
function f() {
    let count = 0;

    return function() {
        return count++;
    };
}

let x = f();

console.log(x());
console.log(x());
```

Curry

```
function s(x) {
    return function(y) {
        return x + y;
    };
}

let sum = s(5);

console.log(sum(1));
console.log(sum(1));
console.log(sum(2));
```

Curry

```
function s(x, y) {
    return function() {
        return x + y;
    };
}

let sum = s(5, 1);

console.log(sum());
console.log(sum());
```


RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION

Recursion

```
function s() {
    var result = 0;
    for (var i = 0; i < arguments.length; i++) {
        result += arguments[i];
    }

    return result;
}

console.log(s(1, 2, 3));
```

Recursion

```
function s() {
    var args = [].slice.call(arguments);
    if (args.length <= 2) {
        return args[0] + args[1];
    }

    return args[0] + s.apply(null, args.slice(1));
}

console.log(s(1, 2, 3));
```

Recursion

```
function s(...args) {
    if (args.length <= 2) {
        return args[0] + args[1];
    }

    return args[0] + s(...args.slice(1));
}

console.log(s(1, 2, 3));
```