javascript tricks

or

closures functions

github.com/egonelbre/jstricks
git.io/asDSQw

Sometimes,

the elegant implementation is a function.

Not a method.

Not a class.

Not a framework.

Just a function.

- John Carmack

many names for similar things

lexical closure

lambda

anonymous function

first-class function

high-order function

delegate

blocks

...

applicable elsewhere...

```
lisp     - (lambda (x)(x*x))
C++11    - [](int x){return x*x}
python   - lambda x: x*x
C#       - x => x*x
Obj.-C   - ^(int x){return x*x}
ruby     - lambda{|x| x*x}
haskell  - \x -> x*x
java 8   - x => x*x
PHP      - create_function("$x","return $x*$x")
...
```

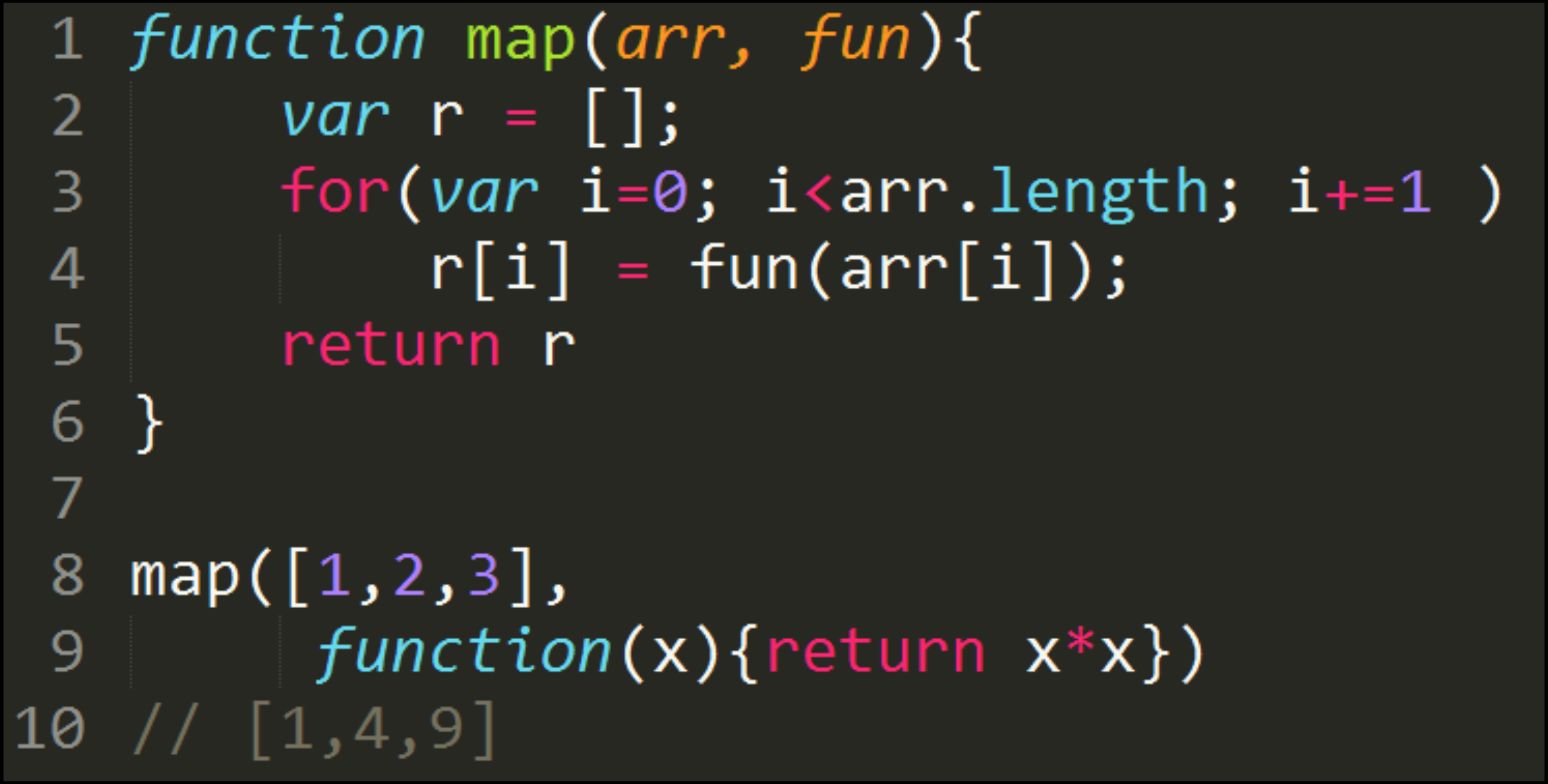
WARNING!

I call those things "tricks" for a reason.

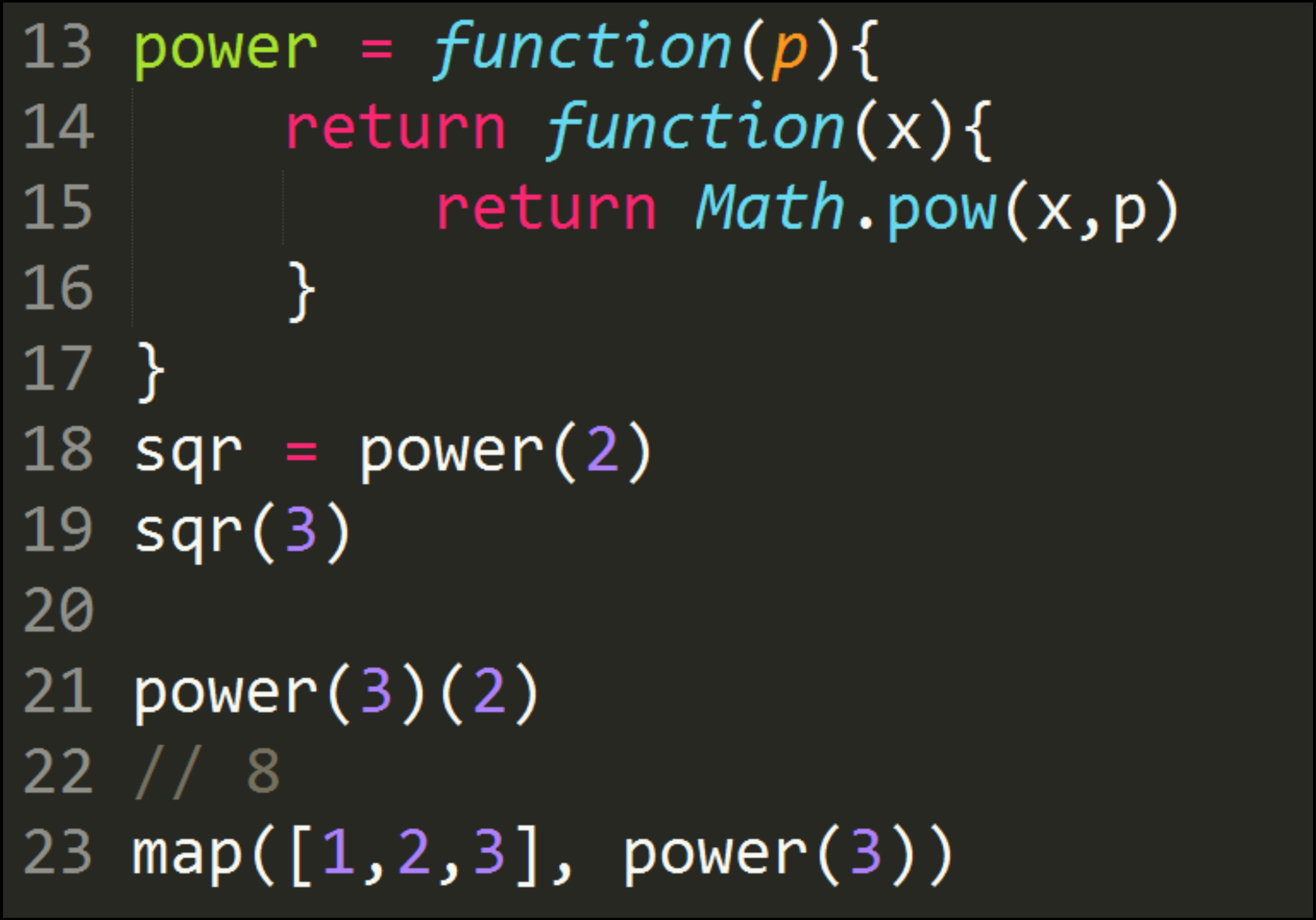
With great power comes great responsibility.

Meaning, before using those "tricks" in a production environment verify that you can debug them.

If possible, use already existing libraries.



```javascript
function map(arr, fun){
    var r = [];
    for(var i=0; i<arr.length; i+=1 )
        r[i] = fun(arr[i]);
    return r
}

map([1,2,3],
     function(x){return x*x})
// [1,4,9]
```



```
power = function(p){
    return function(x){
        return Math.pow(x,p)
    }
}
sqr = power(2)
sqr(3)

power(3)(2)
// 8
map([1,2,3], power(3))
```

so... let's start simple...
counting....

0

1

2

3

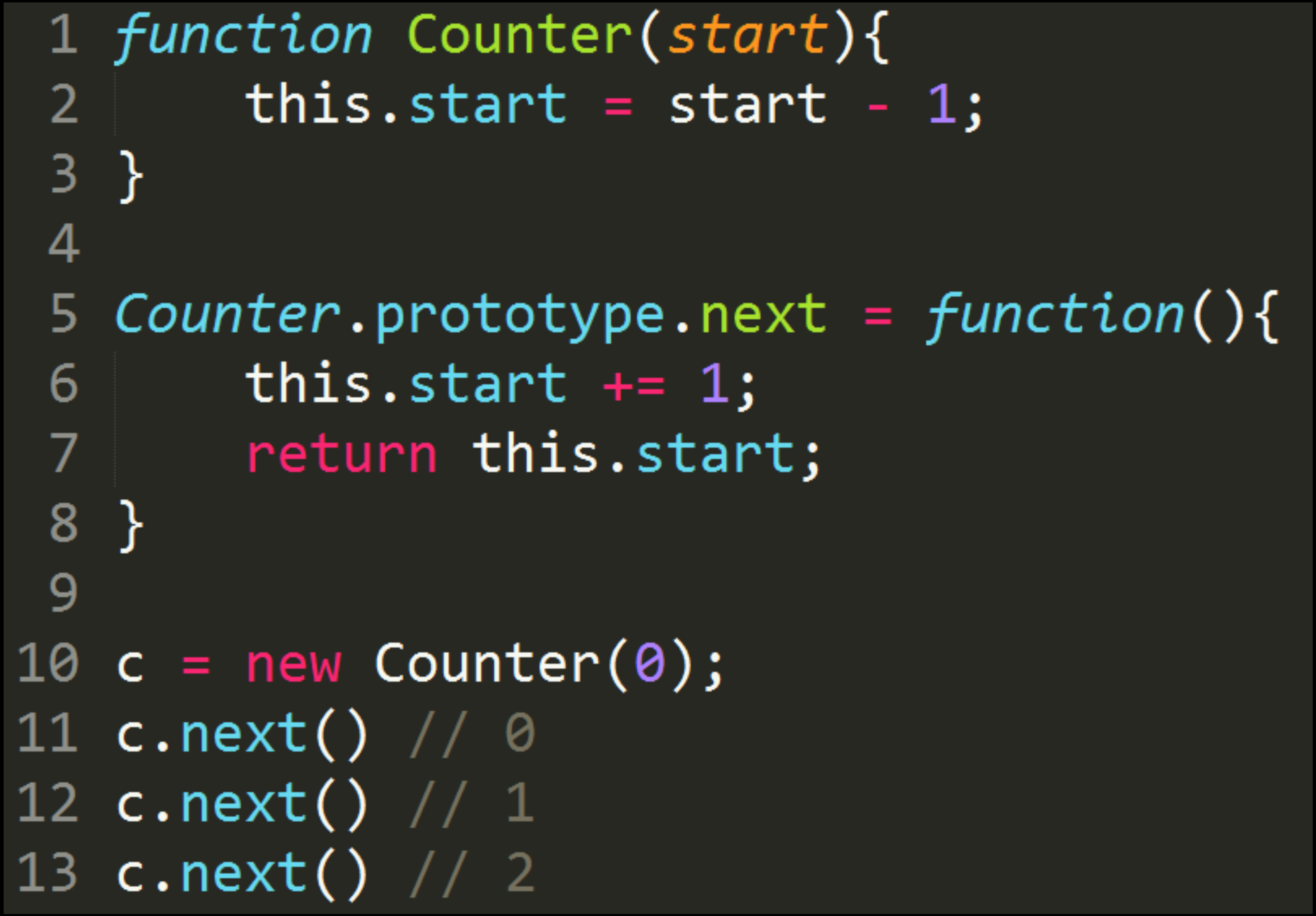
4

5

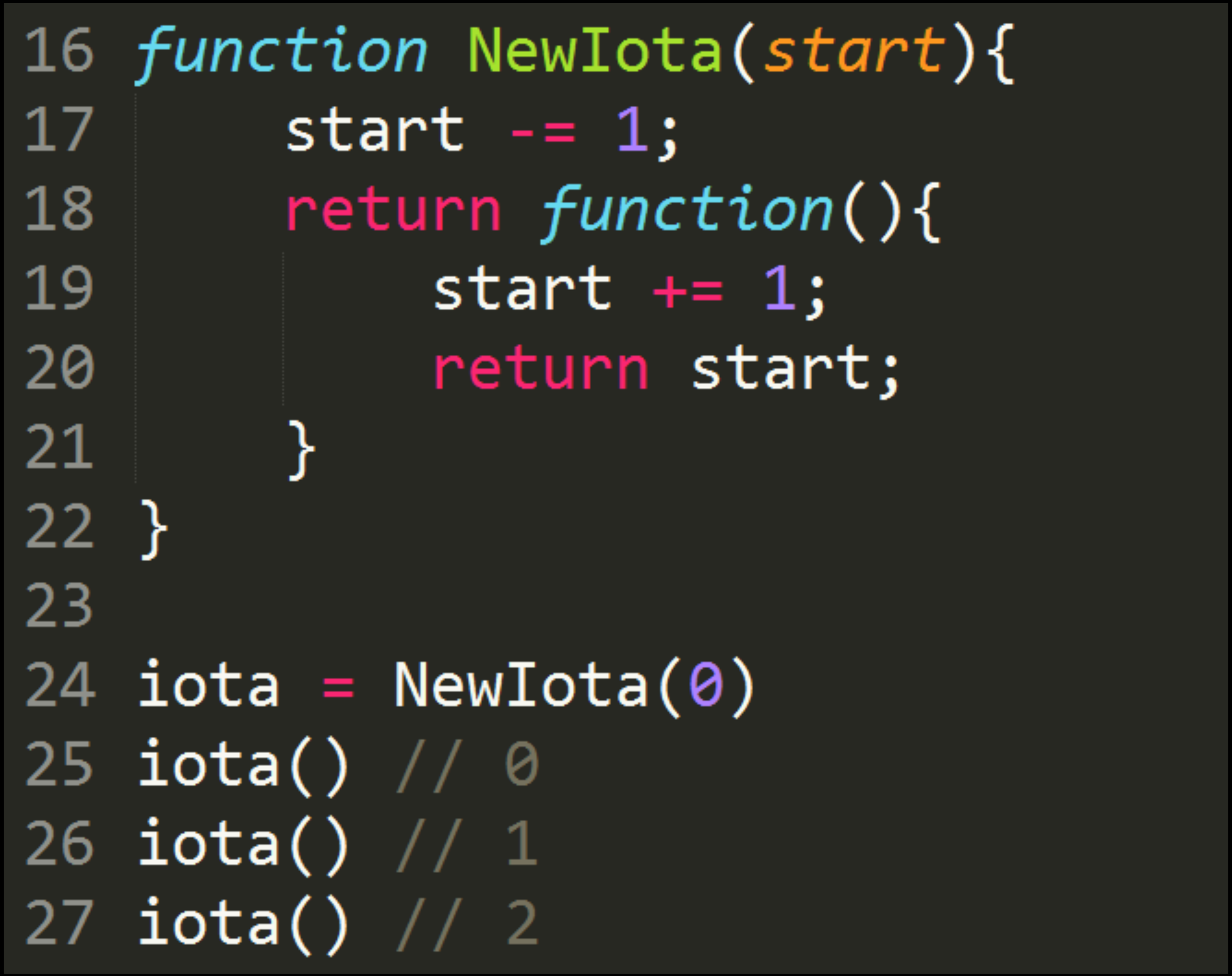
6

7

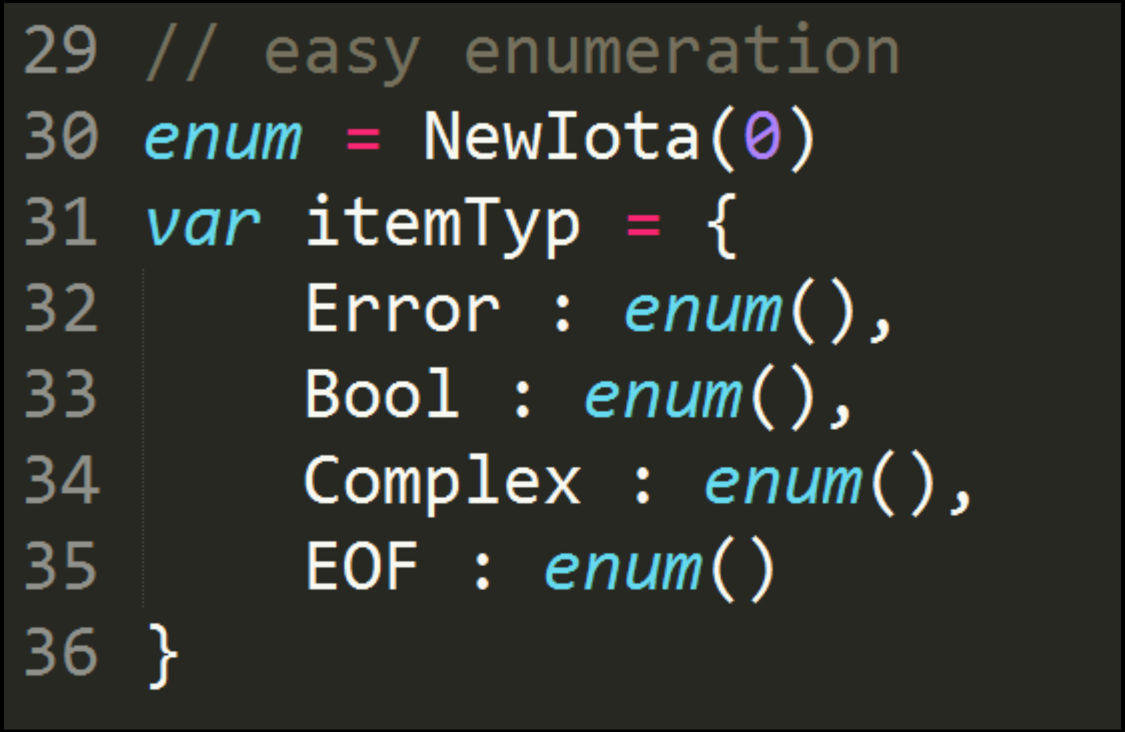
...



```javascript
function Counter(start){
    this.start = start - 1;
}

Counter.prototype.next = function(){
    this.start += 1;
    return this.start;
}

c = new Counter(0);
c.next() // 0
c.next() // 1
c.next() // 2
```

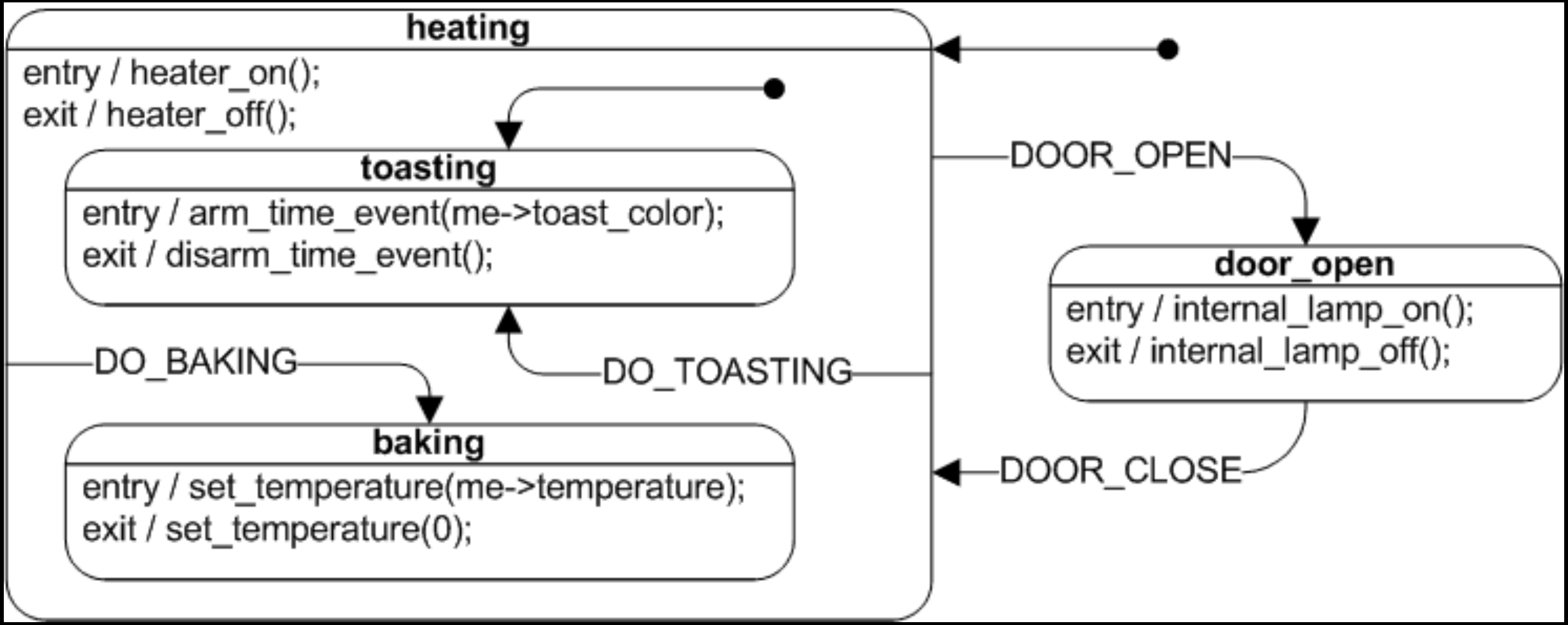


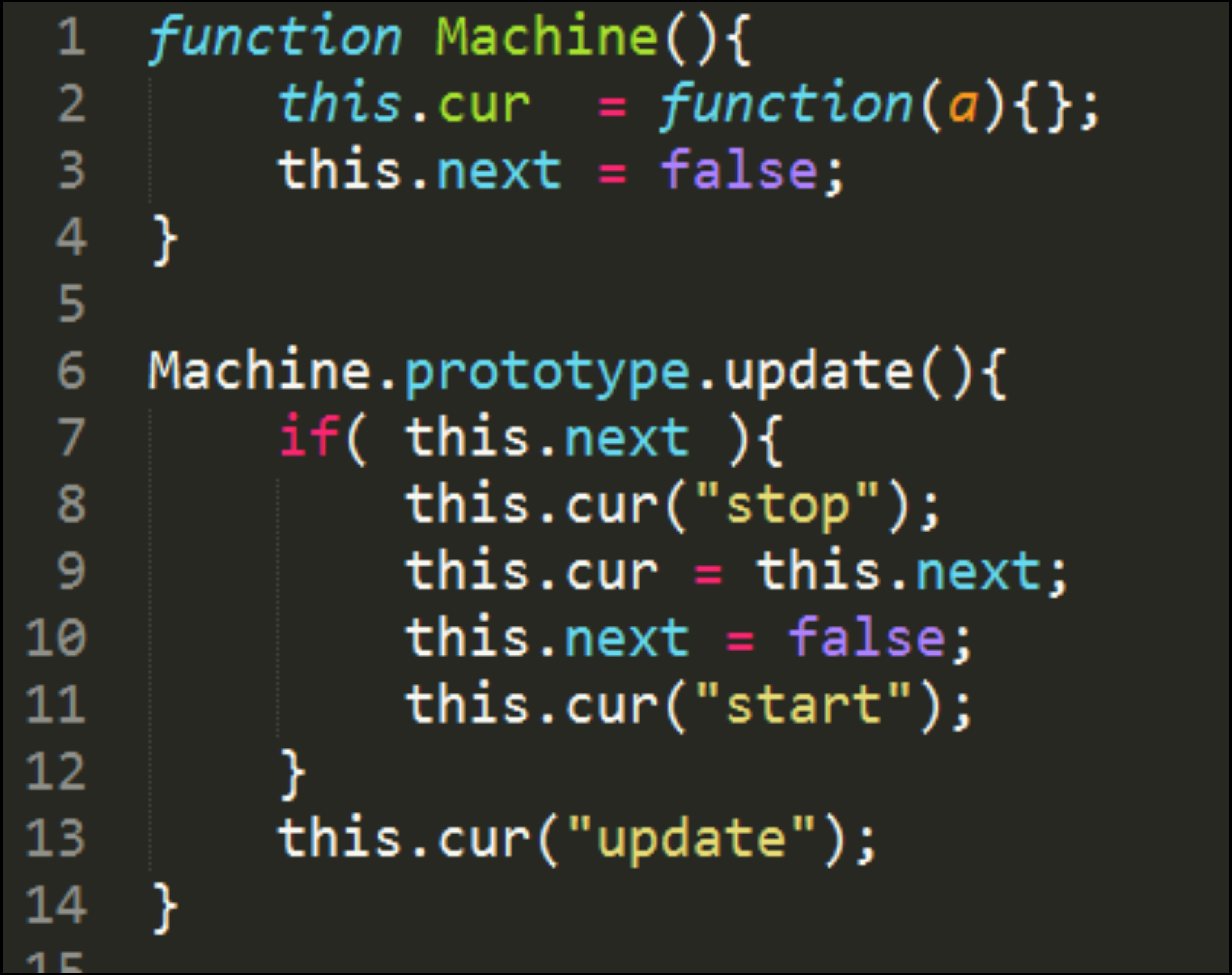
```
function NewIota(start){
    start -= 1;
    return function(){
        start += 1;
        return start;
    }
}

iota = NewIota(0)
iota() // 0
iota() // 1
iota() // 2
```



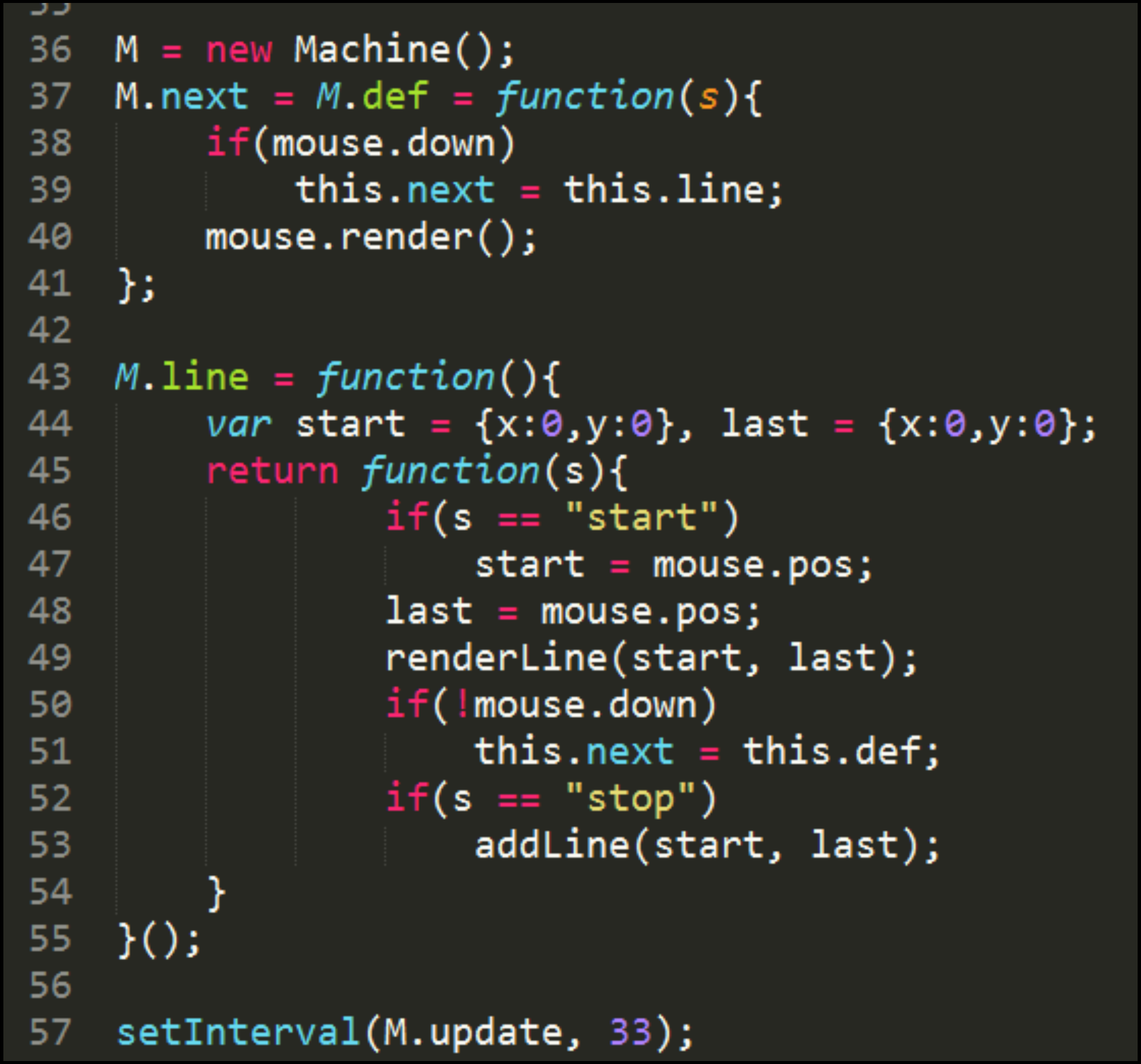
```
// easy enumeration
enum = NewIota(0)
var itemTyp = {
        Error : enum(),
        Bool : enum(),
        Complex : enum(),
        EOF : enum()
}
```

State Machine...


heating
entry / heater_on();
exit / heater_off();
toasting
entry / arm_time_event(me->toast_color);
exit / disarm_time_event();
DO_BAKING
DO_TOASTING
baking
entry / set_temperature(me->temperature);
exit / set_temperature(0);
DOOR_OPEN
door_open
entry / internal_lamp_on();
exit / internal_lamp_off();
DOOR_CLOSE



```
function Machine(){
    this.cur  = function(a){};
    this.next = false;
}

Machine.prototype.update(){
    if( this.next ){
        this.cur("stop");
        this.cur = this.next;
        this.next = false;
        this.cur("start");
    }
    this.cur("update");
}
```

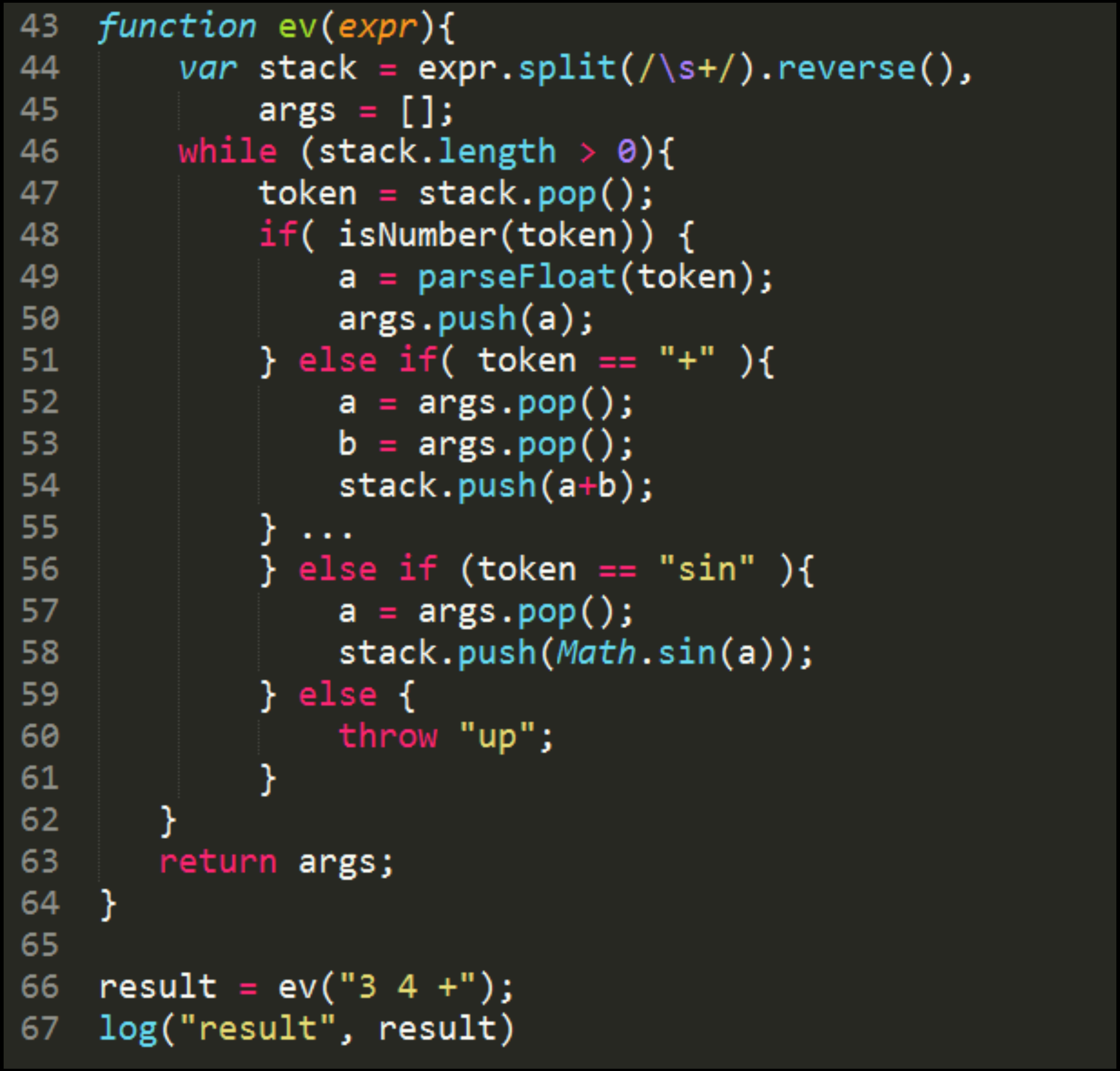


```
M = new Machine();
M.next = M.def = function(s){
    if(mouse.down)
        this.next = this.line;
    mouse.render();
};

M.line = function(){
    var start = {x:0,y:0}, last = {x:0,y:0};
    return function(s){
            if(s == "start")
                start = mouse.pos;
            last = mouse.pos;
            renderLine(start, last);
            if(!mouse.down)
                this.next = this.def;
            if(s == "stop")
                addLine(start, last);
    }
}();

setInterval(M.update, 33);
```

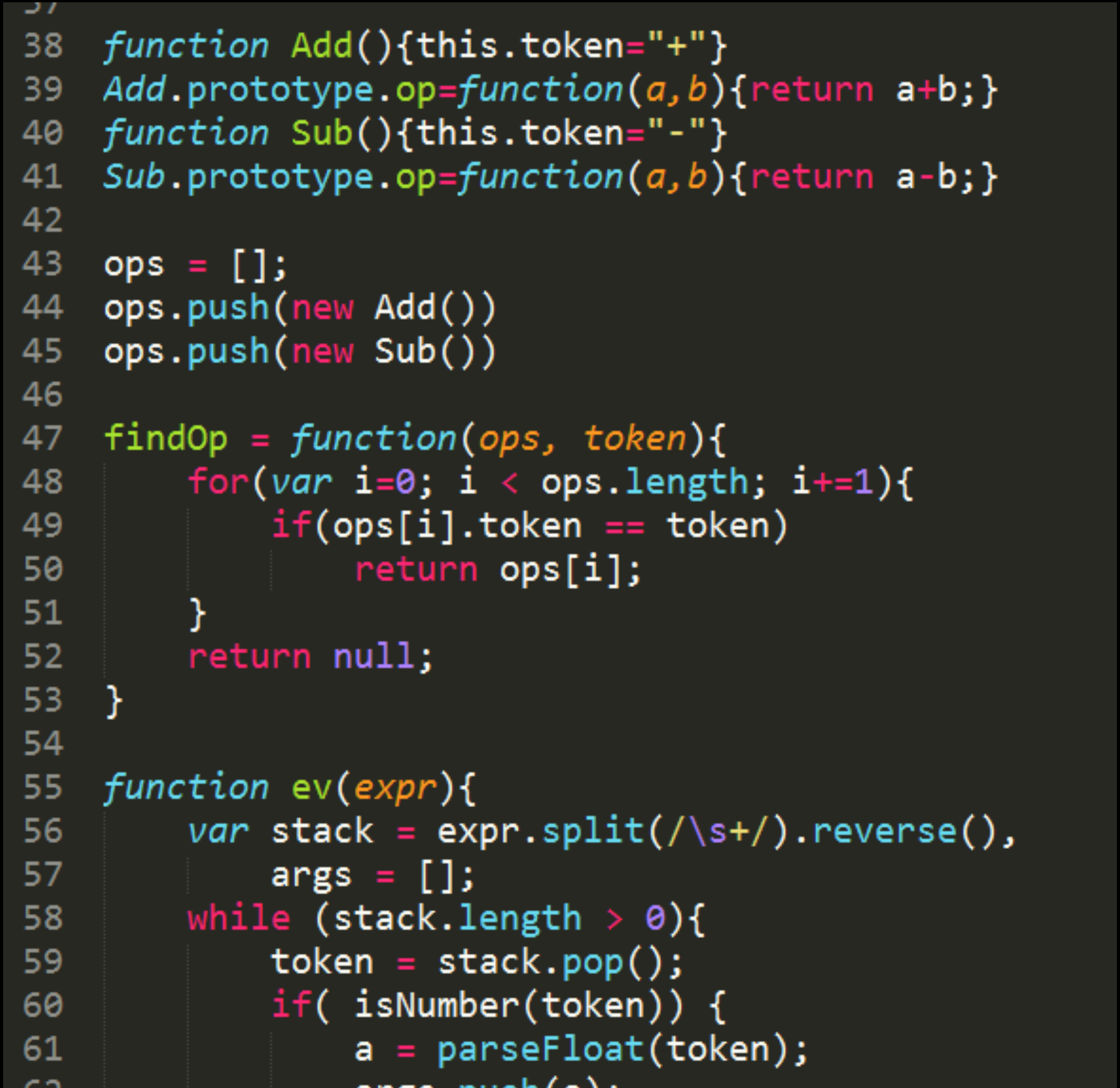
RPN calculator

"3 4 + sin" == (3 4 +)sin

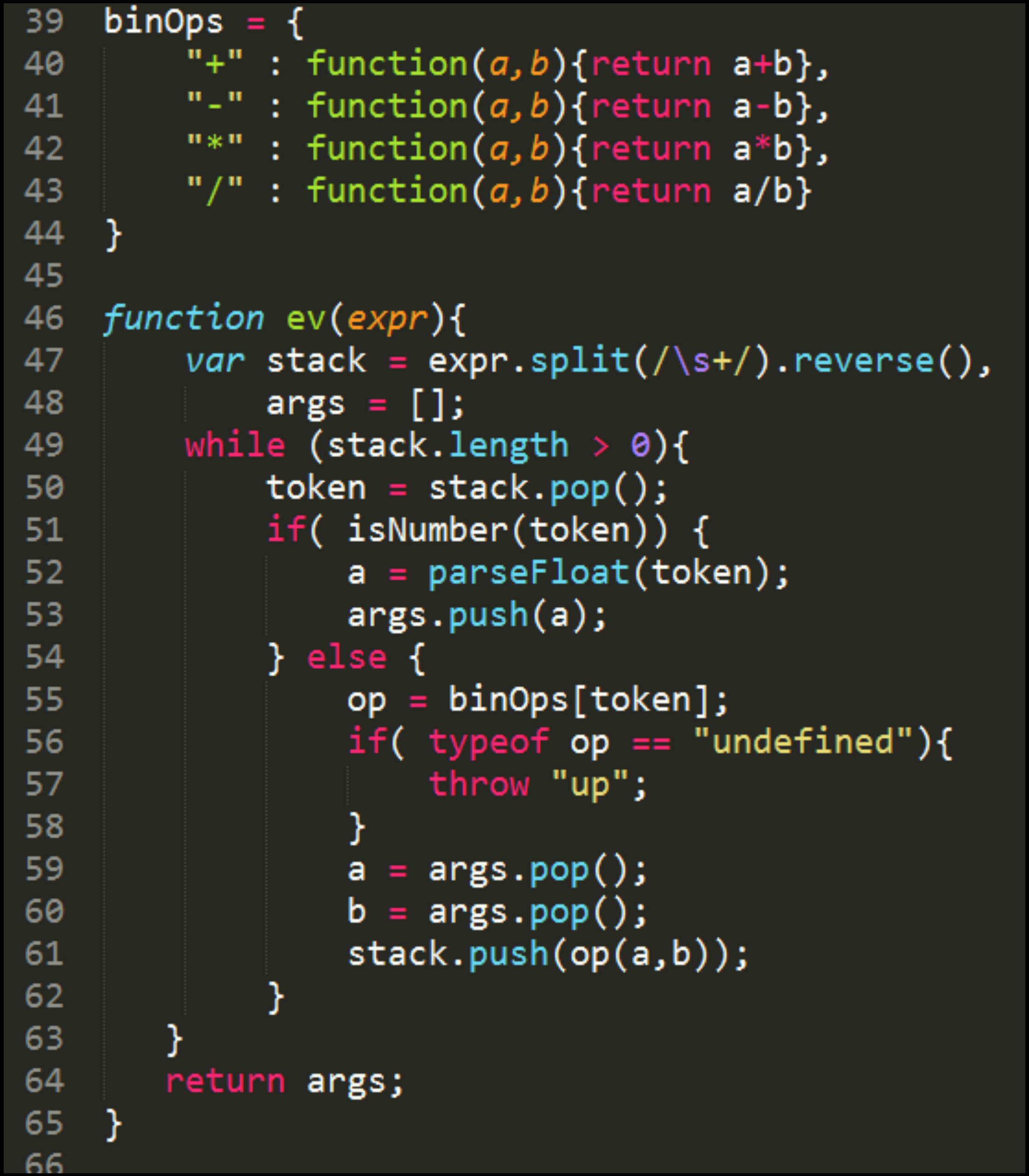
"3 4 5 * −" == (3 (4 5 *) −)



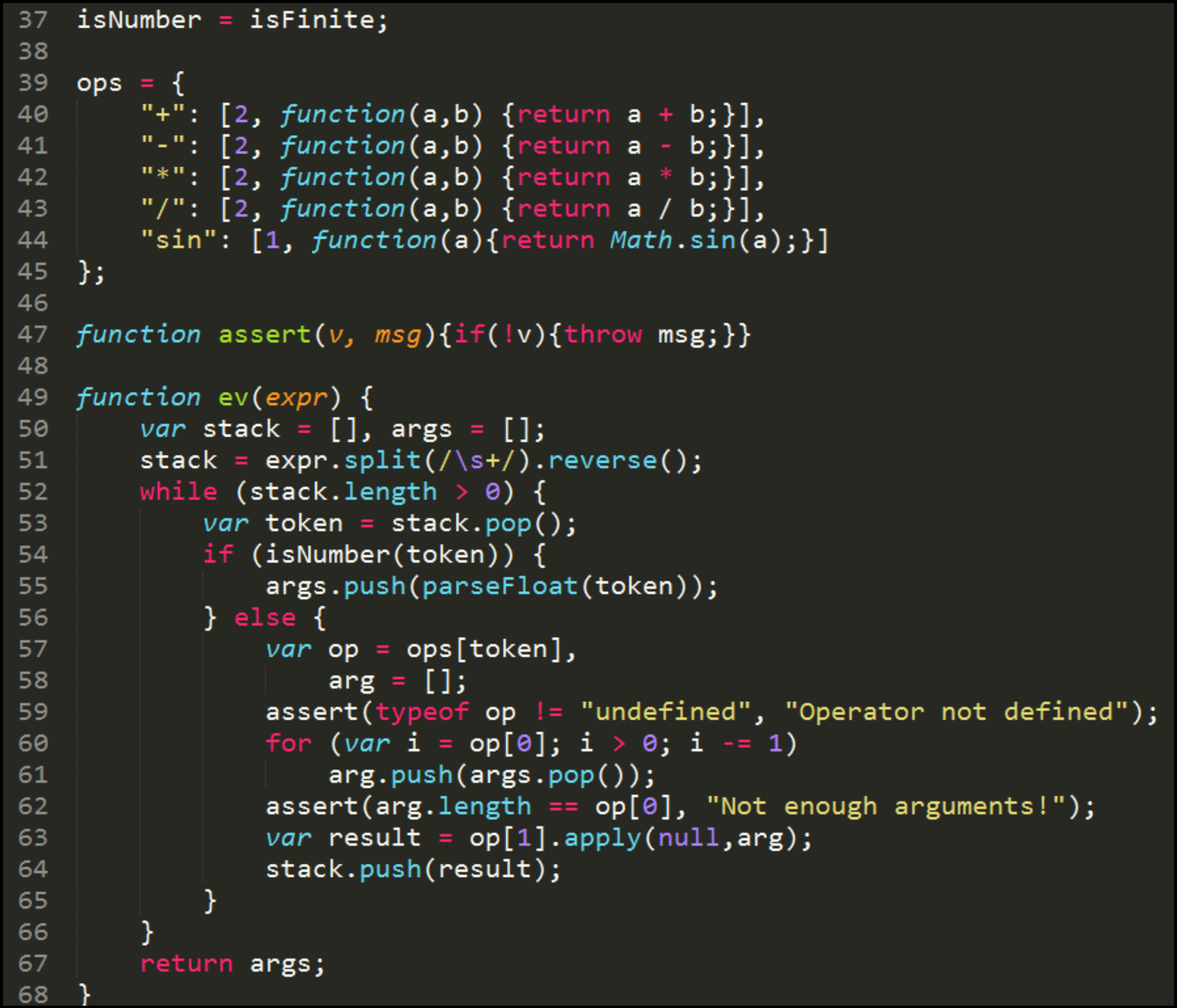
```
function ev(expr){
    var stack = expr.split(/\s+/).reverse(),
        args = [];
    while (stack.length > 0){
        token = stack.pop();
        if( isNumber(token)) {
            a = parseFloat(token);
            args.push(a);
        } else if( token == "+" ){
            a = args.pop();
            b = args.pop();
            stack.push(a+b);
        } ...
        } else if (token == "sin" ){
            a = args.pop();
            stack.push(Math.sin(a));
        } else {
            throw "up";
        }
   }
   return args;
}

result = ev("3 4 +");
log("result", result)
```



```
function Add(){this.token="+"}
Add.prototype.op=function(a,b){return a+b;}
function Sub(){this.token="-"}
Sub.prototype.op=function(a,b){return a-b;}

ops = [];
ops.push(new Add())
ops.push(new Sub())

findOp = function(ops, token){
    for(var i=0; i < ops.length; i+=1){
        if(ops[i].token == token)
            return ops[i];
    }
    return null;
}

function ev(expr){
    var stack = expr.split(/\s+/).reverse(),
        args = [];
    while (stack.length > 0){
        token = stack.pop();
        if( isNumber(token)) {
            a = parseFloat(token);
```

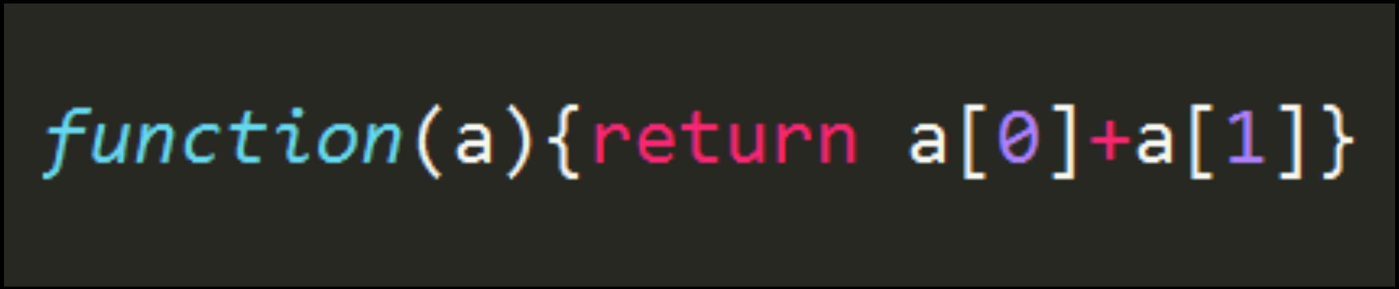


```
binOps = {
    "+" : function(a,b){return a+b},
    "-" : function(a,b){return a-b},
    "*" : function(a,b){return a*b},
    "/" : function(a,b){return a/b}
}

function ev(expr){
    var stack = expr.split(/\s+/).reverse(),
        args = [];
    while (stack.length > 0){
        token = stack.pop();
        if( isNumber(token)) {
            a = parseFloat(token);
            args.push(a);
        } else {
            op = binOps[token];
            if( typeof op == "undefined"){
                throw "up";
            }
            a = args.pop();
            b = args.pop();
            stack.push(op(a,b));
        }
   }
   return args;
}
```



```javascript
isNumber = isFinite;

ops = {
    "+": [2, function(a,b) {return a + b;}],
    "-": [2, function(a,b) {return a - b;}],
    "*": [2, function(a,b) {return a * b;}],
    "/": [2, function(a,b) {return a / b;}],
    "sin": [1, function(a){return Math.sin(a);}]
};

function assert(v, msg){if(!v){throw msg;}}

function ev(expr) {
    var stack = [], args = [];
    stack = expr.split(/\s+/).reverse();
    while (stack.length > 0) {
        var token = stack.pop();
        if (isNumber(token)) {
            args.push(parseFloat(token));
        } else {
            var op = ops[token],
              arg = [];
            assert(typeof op != "undefined", "Operator not defined");
            for (var i = op[0]; i > 0; i -= 1)
                arg.push(args.pop());
            assert(arg.length == op[0], "Not enough arguments!");
            var result = op[1].apply(null,arg);
            stack.push(result);
        }
    }
    return args;
}
```

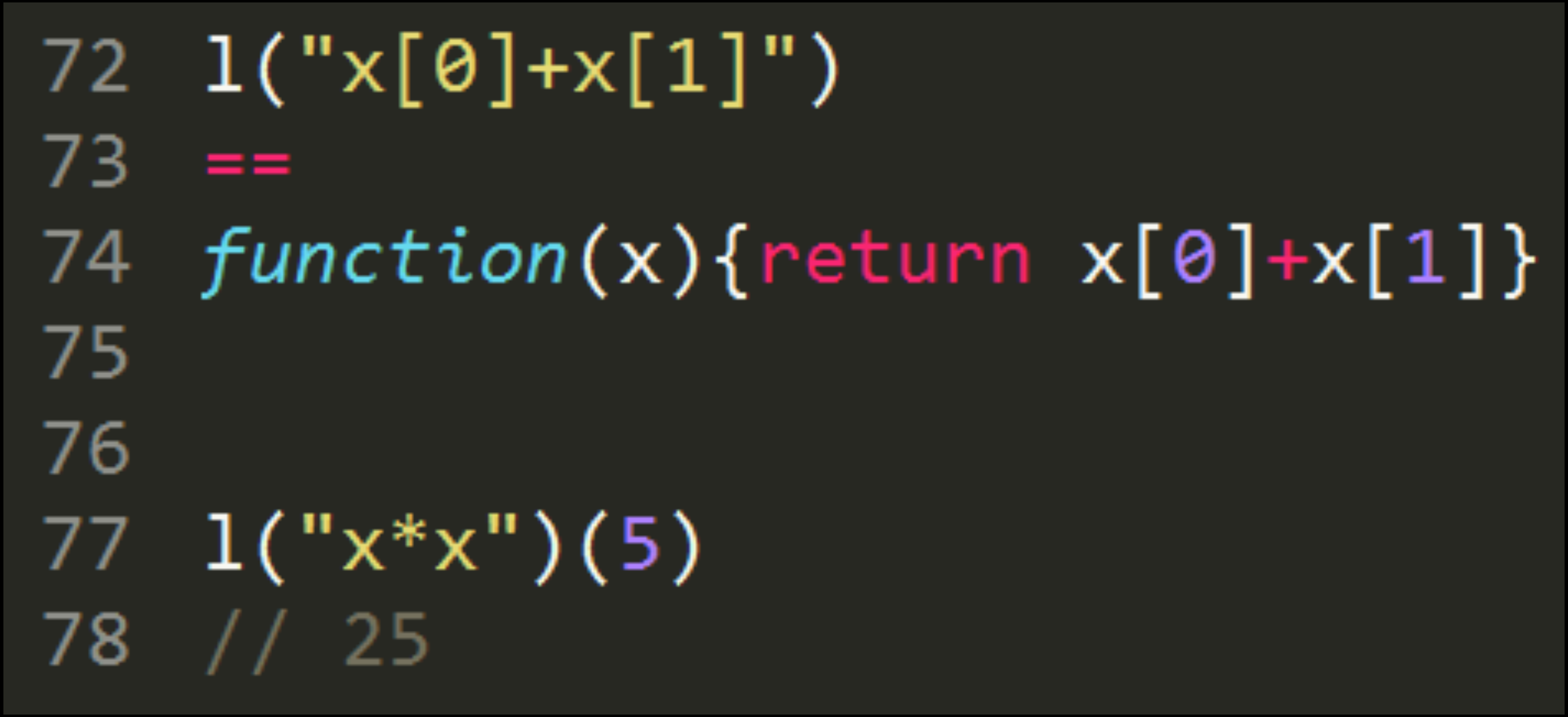
....



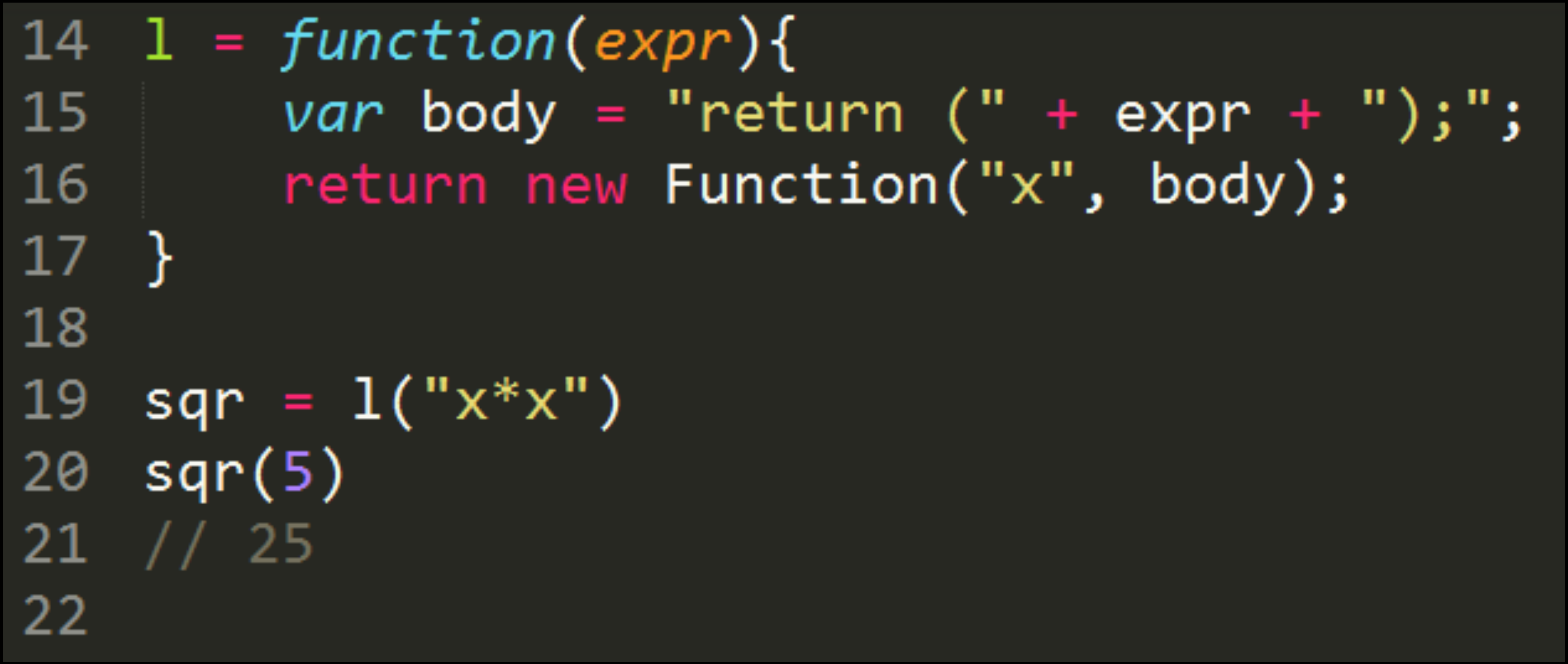
```
function(a){return a[0]+a[1]}
```

too much typing....

let's try to something like...

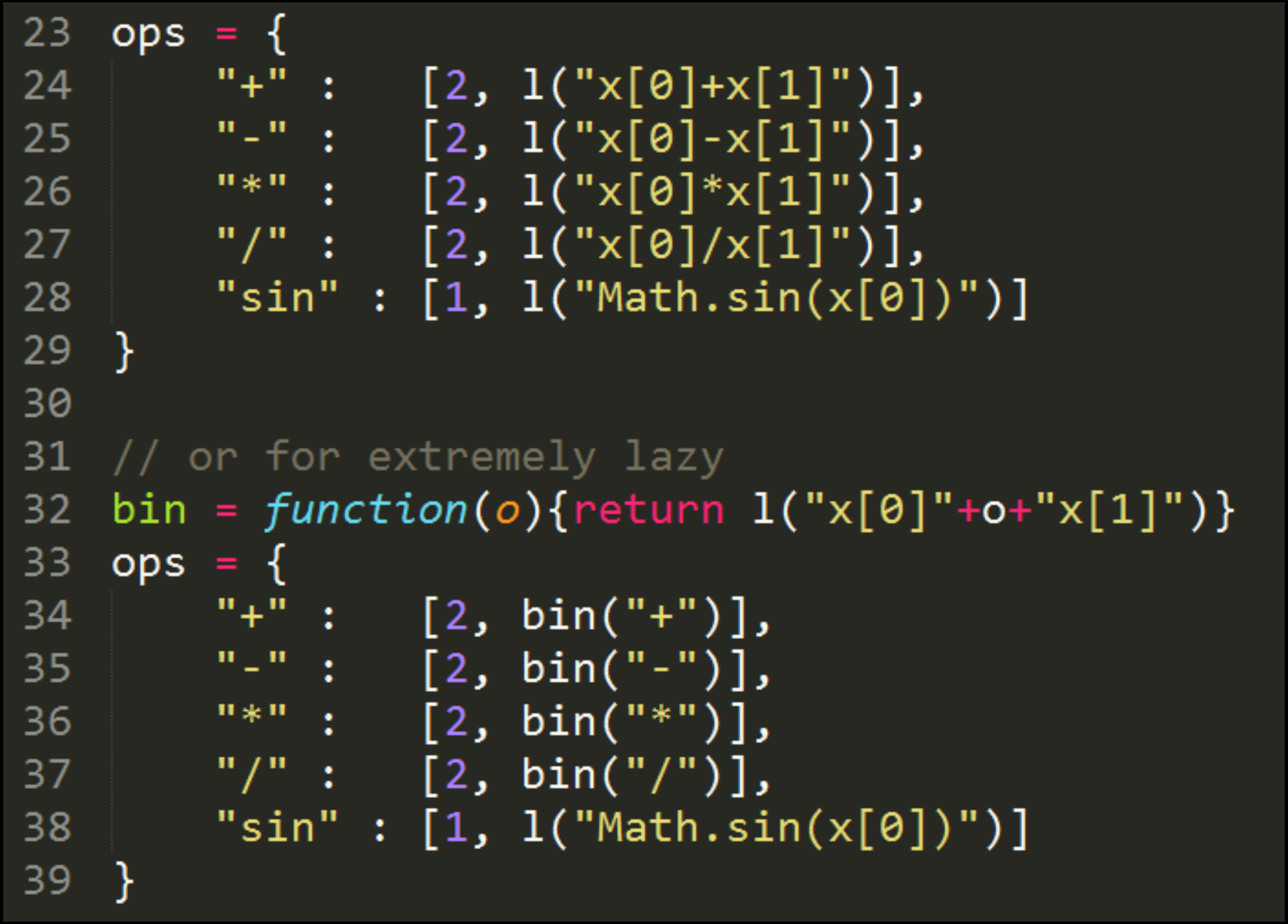


```
l("x[0]+x[1]")
==
function(x){return x[0]+x[1]}


l("x*x")(5)
// 25
```



```
l = function(expr){
    var body = "return (" + expr + ");";
    return new Function("x", body);
}

sqr = l("x*x")
sqr(5)
// 25

```

(of course this can be made smarter:
    http://osteele.com/sources/javascript/functional/)

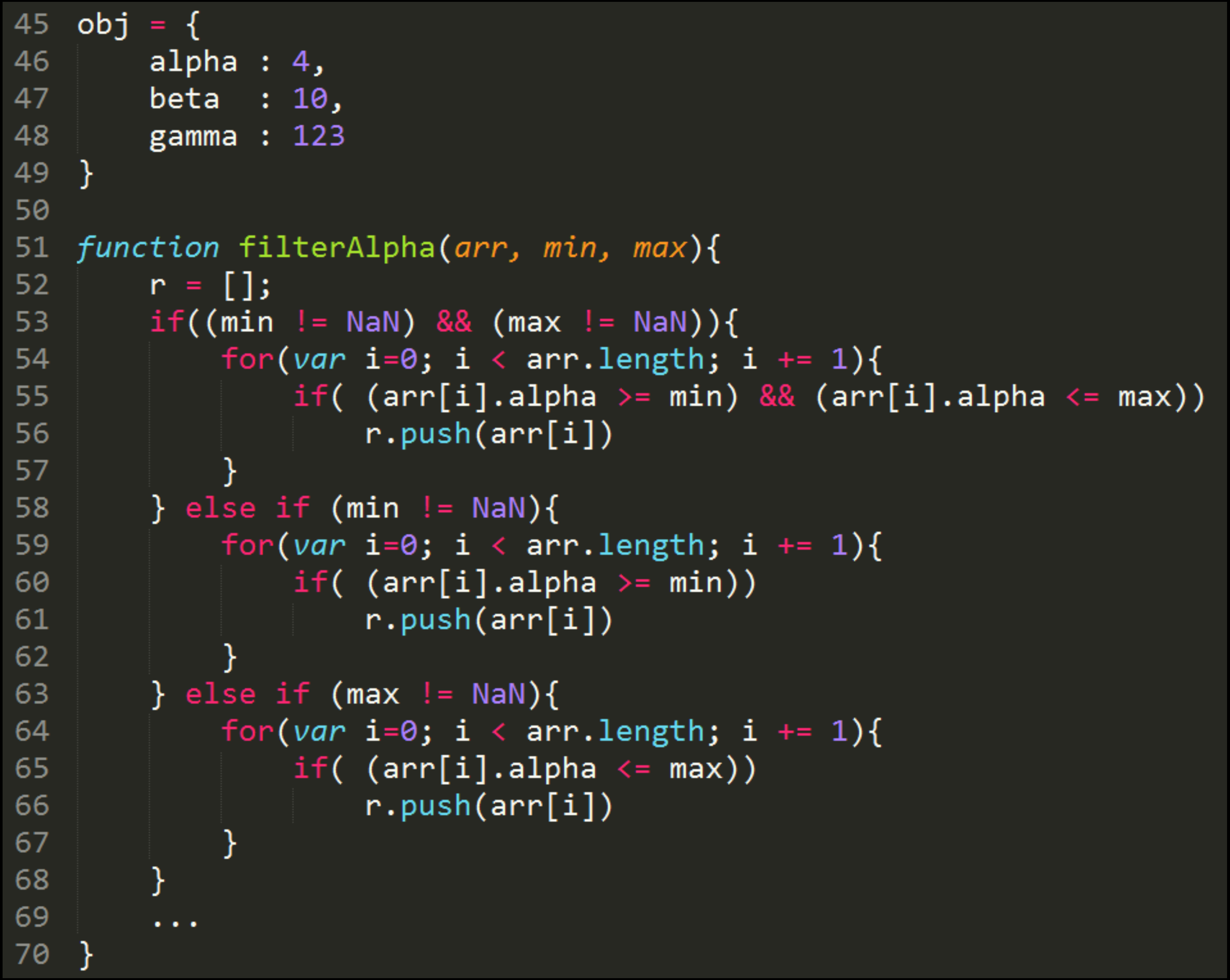


```
ops = {
    "+" :   [2, l("x[0]+x[1]")],
    "-" :   [2, l("x[0]-x[1]")],
    "*" :   [2, l("x[0]*x[1]")],
    "/" :   [2, l("x[0]/x[1]")],
    "sin" : [1, l("Math.sin(x[0])")]
}

// or for extremely lazy
bin = function(o){return l("x[0]"+o+"x[1]")}
ops = {
    "+" :   [2, bin("+")],
    "-" :   [2, bin("-")],
    "*" :   [2, bin("*")],
    "/" :   [2, bin("/")],
    "sin" : [1, l("Math.sin(x[0])")]
}
```

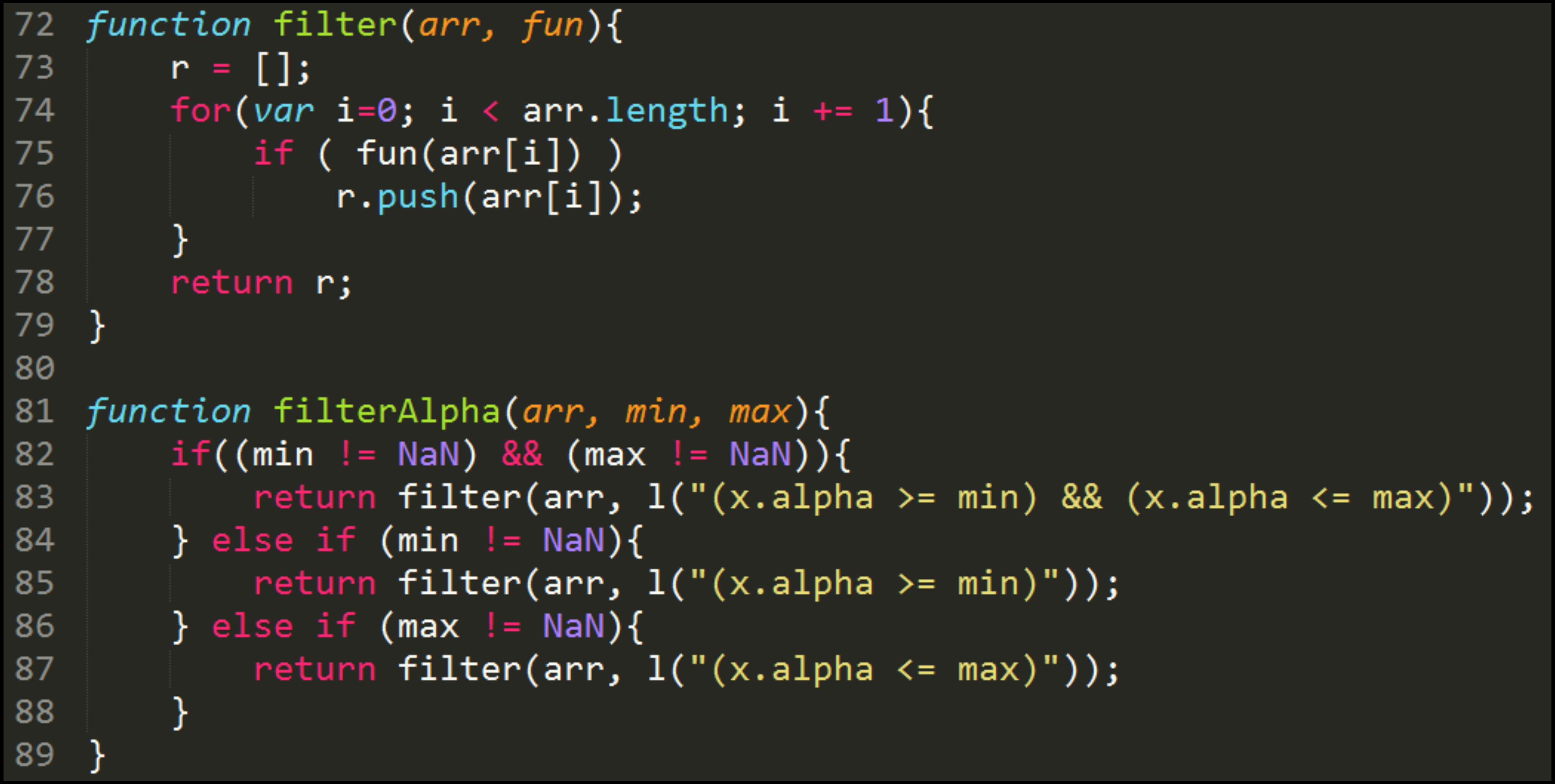
but all languages don't have "eval"?

... we can still combine functions ...

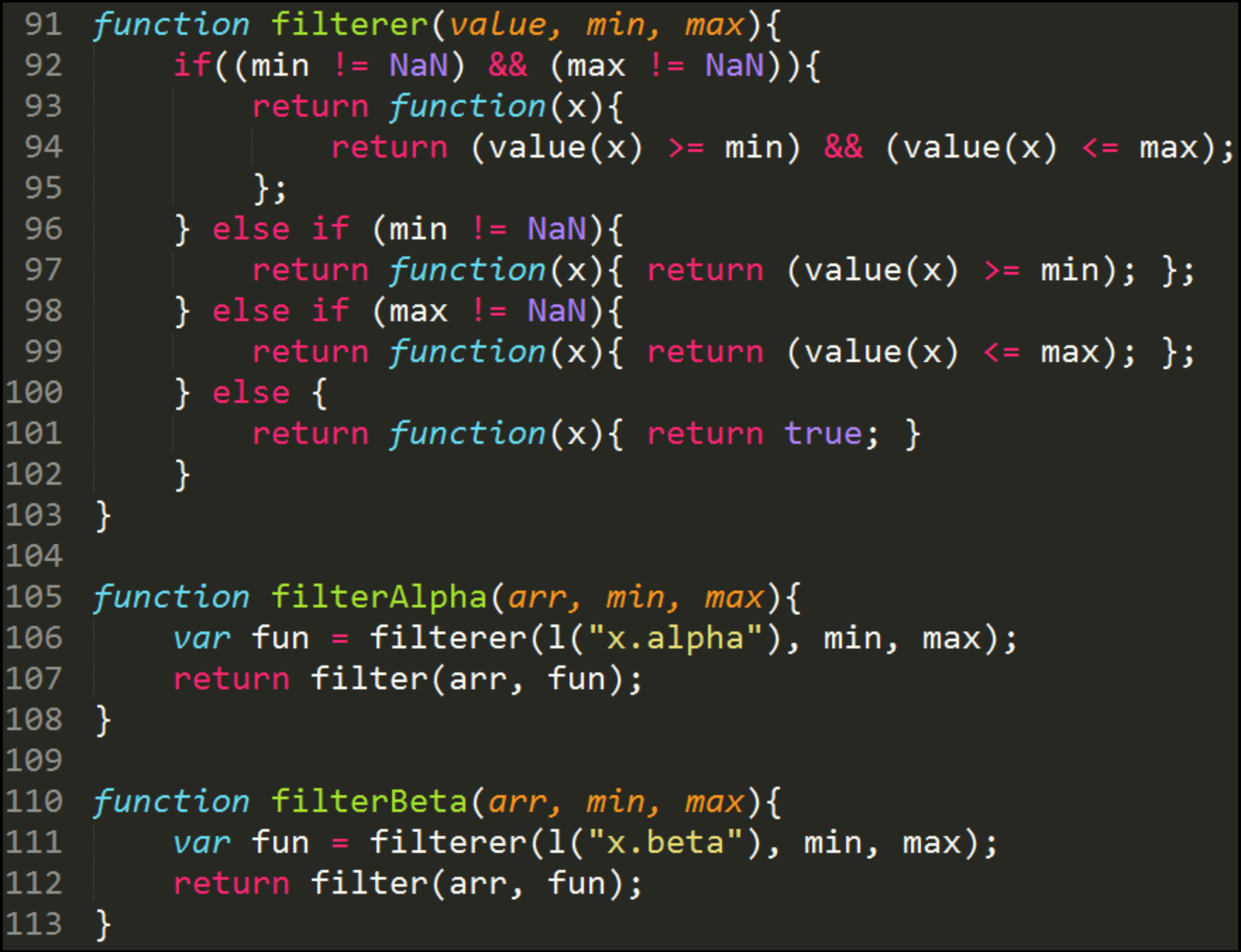
https://github.com/egonelbre/spexs/blob/master/src/spxs/filters.go



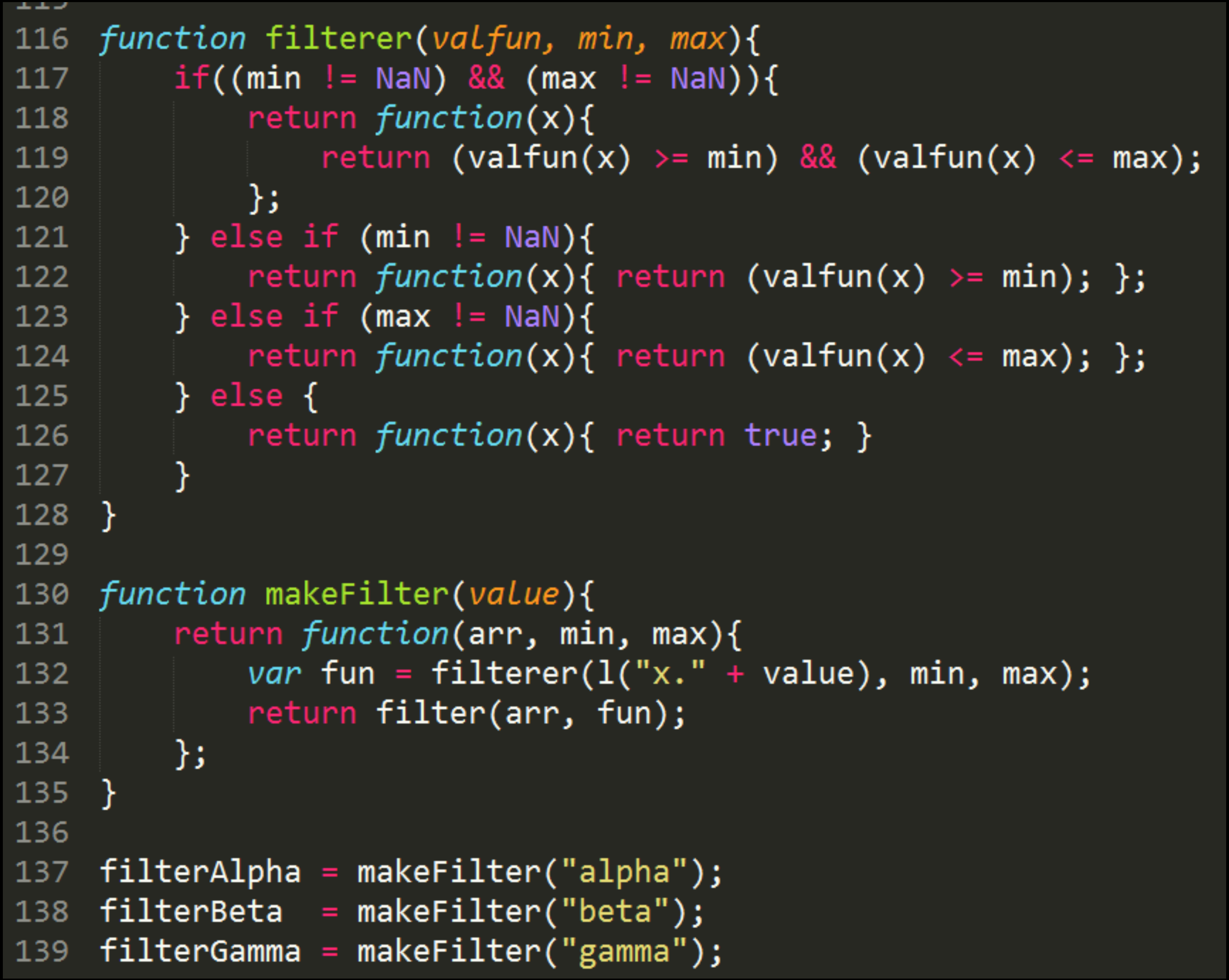
```javascript
obj = {
    alpha : 4,
    beta  : 10,
    gamma : 123
}

function filterAlpha(arr, min, max){
    r = [];
    if((min != NaN) && (max != NaN)){
        for(var i=0; i < arr.length; i += 1){
            if( (arr[i].alpha >= min) && (arr[i].alpha <= max))
                r.push(arr[i])
        }
    } else if (min != NaN){
        for(var i=0; i < arr.length; i += 1){
            if( (arr[i].alpha >= min))
                r.push(arr[i])
        }
    } else if (max != NaN){
        for(var i=0; i < arr.length; i += 1){
            if( (arr[i].alpha <= max))
                r.push(arr[i])
        }
    }
    ...
}
```



```javascript
function filter(arr, fun){
    r = [];
    for(var i=0; i < arr.length; i += 1){
        if ( fun(arr[i]) )
            r.push(arr[i]);
    }
    return r;
}

function filterAlpha(arr, min, max){
    if((min != NaN) && (max != NaN)){
        return filter(arr, l("(x.alpha >= min) && (x.alpha <= max)"));
    } else if (min != NaN){
        return filter(arr, l("(x.alpha >= min)"));
    } else if (max != NaN){
        return filter(arr, l("(x.alpha <= max)"));
    }
}
```



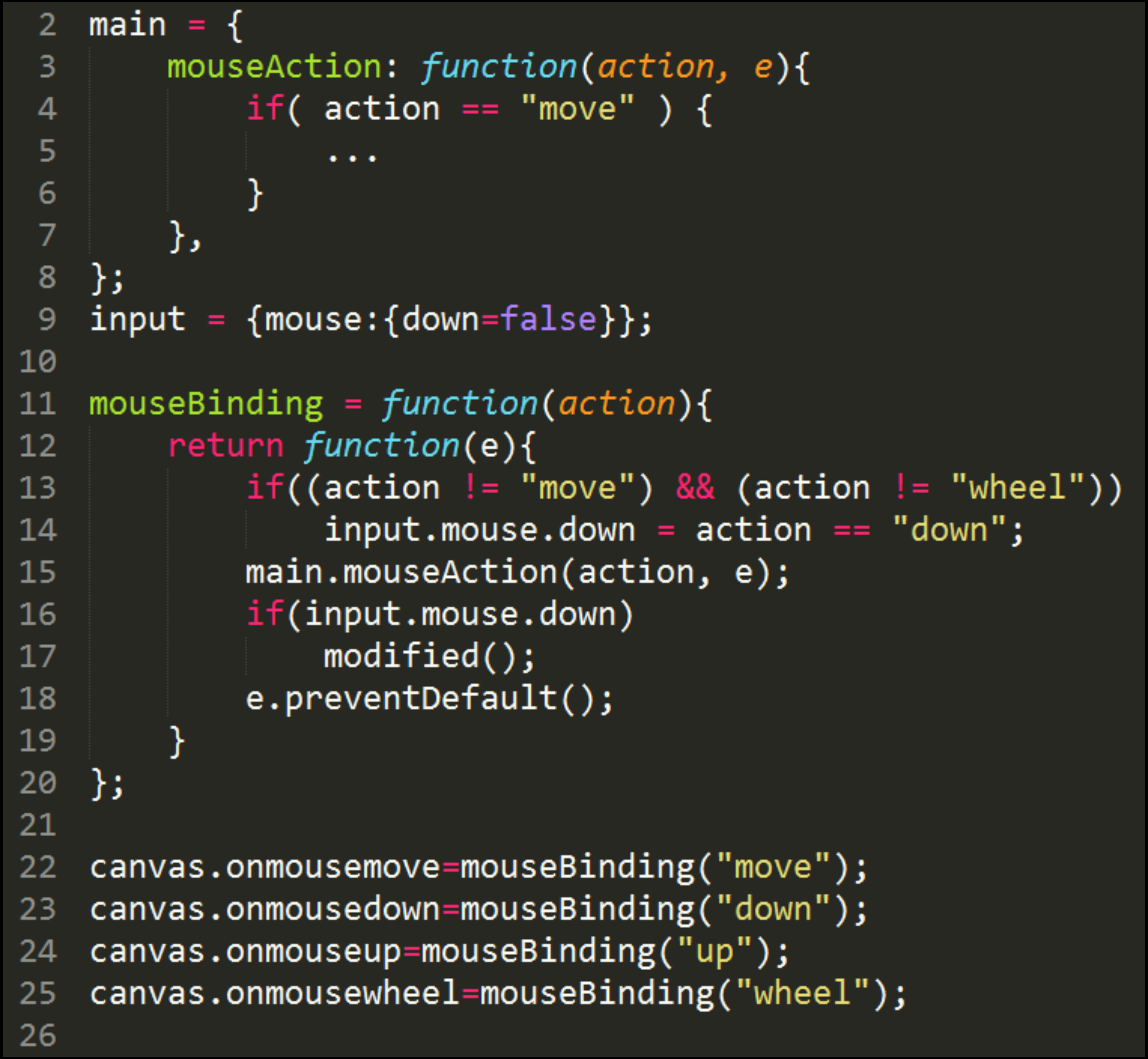
```
function filterer(value, min, max){
    if((min != NaN) && (max != NaN)){
        return function(x){
            return (value(x) >= min) && (value(x) <= max);
        };
    } else if (min != NaN){
        return function(x){ return (value(x) >= min); };
    } else if (max != NaN){
        return function(x){ return (value(x) <= max); };
    } else {
        return function(x){ return true; }
    }
}

function filterAlpha(arr, min, max){
    var fun = filterer(l("x.alpha"), min, max);
    return filter(arr, fun);
}

function filterBeta(arr, min, max){
    var fun = filterer(l("x.beta"), min, max);
    return filter(arr, fun);
}
```



```javascript
function filterer(valfun, min, max){
    if((min != NaN) && (max != NaN)){
        return function(x){
            return (valfun(x) >= min) && (valfun(x) <= max);
        };
    } else if (min != NaN){
        return function(x){ return (valfun(x) >= min); };
    } else if (max != NaN){
        return function(x){ return (valfun(x) <= max); };
    } else {
        return function(x){ return true; }
    }
}

function makeFilter(value){
    return function(arr, min, max){
        var fun = filterer(l("x." + value), min, max);
        return filter(arr, fun);
    };
}

filterAlpha = makeFilter("alpha");
filterBeta  = makeFilter("beta");
filterGamma = makeFilter("gamma");
```

same principles...

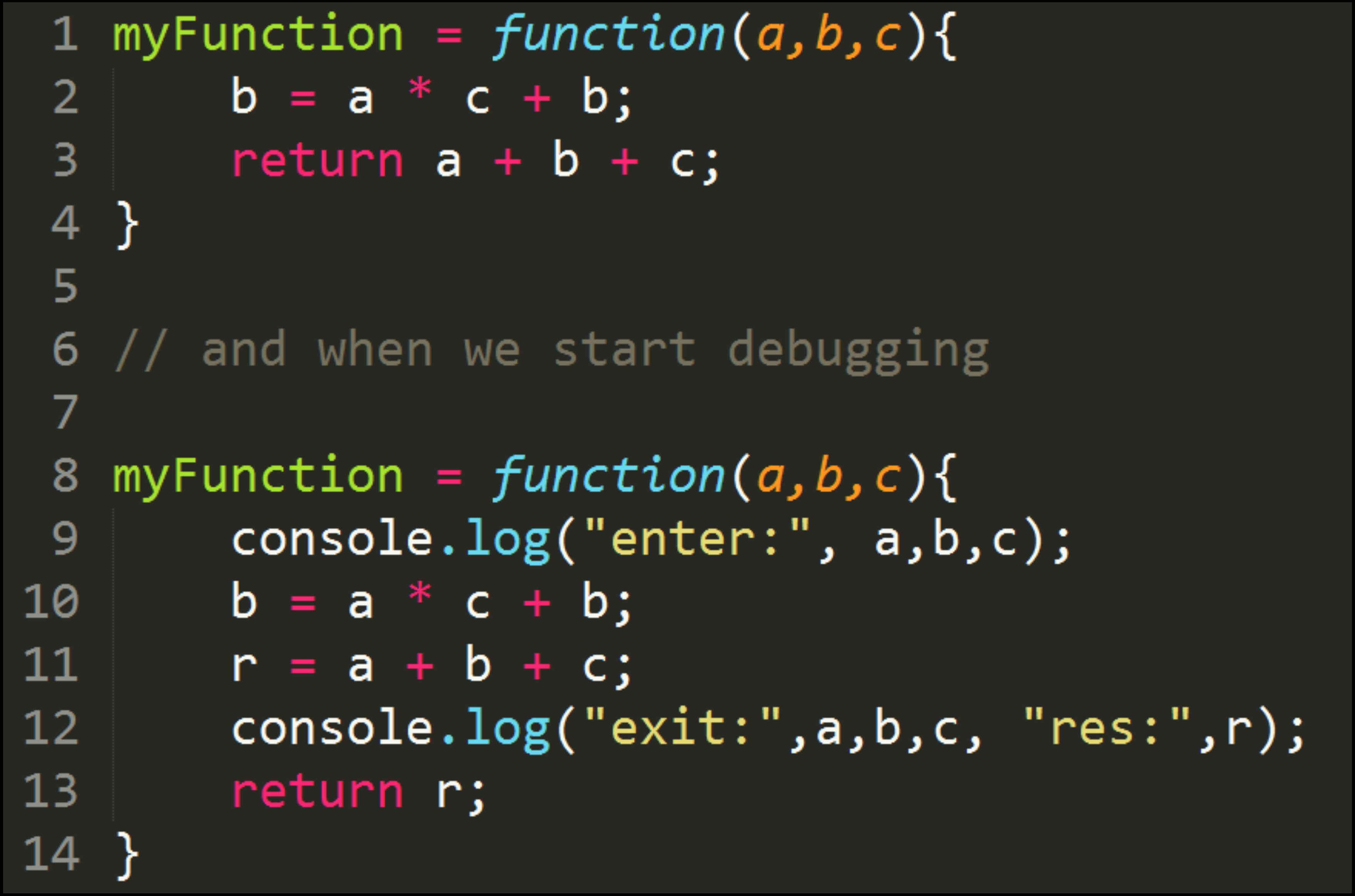
https://github.com/egonelbre/fuse-ed



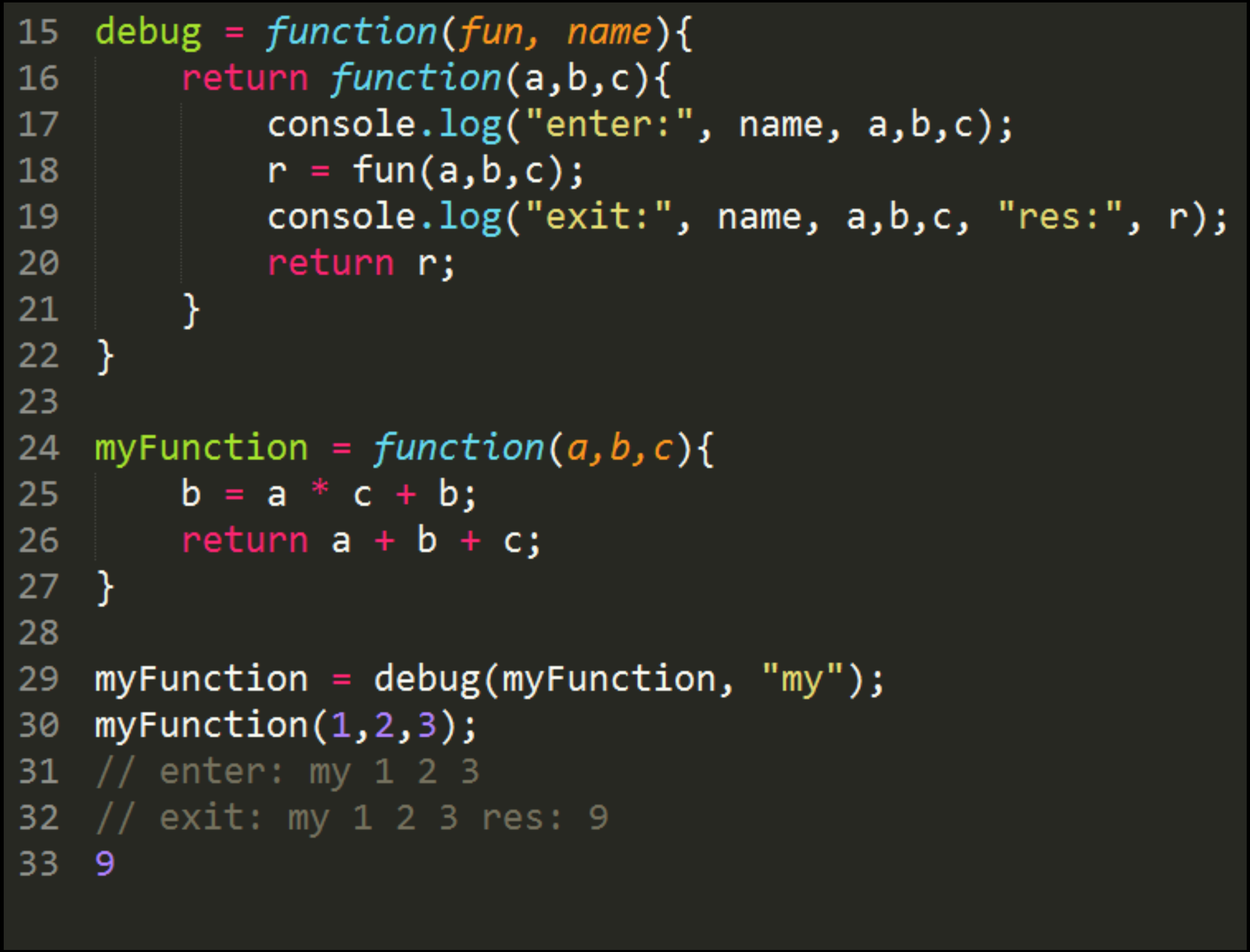
```
main = {
    mouseAction: function(action, e){
        if( action == "move" ) {
            ...
        }
    },
};
input = {mouse:{down=false}};

mouseBinding = function(action){
    return function(e){
        if((action != "move") && (action != "wheel"))
            input.mouse.down = action == "down";
        main.mouseAction(action, e);
        if(input.mouse.down)
            modified();
        e.preventDefault();
    }
};

canvas.onmousemove=mouseBinding("move");
canvas.onmousedown=mouseBinding("down");
canvas.onmouseup=mouseBinding("up");
canvas.onmousewheel=mouseBinding("wheel");

```

debugging

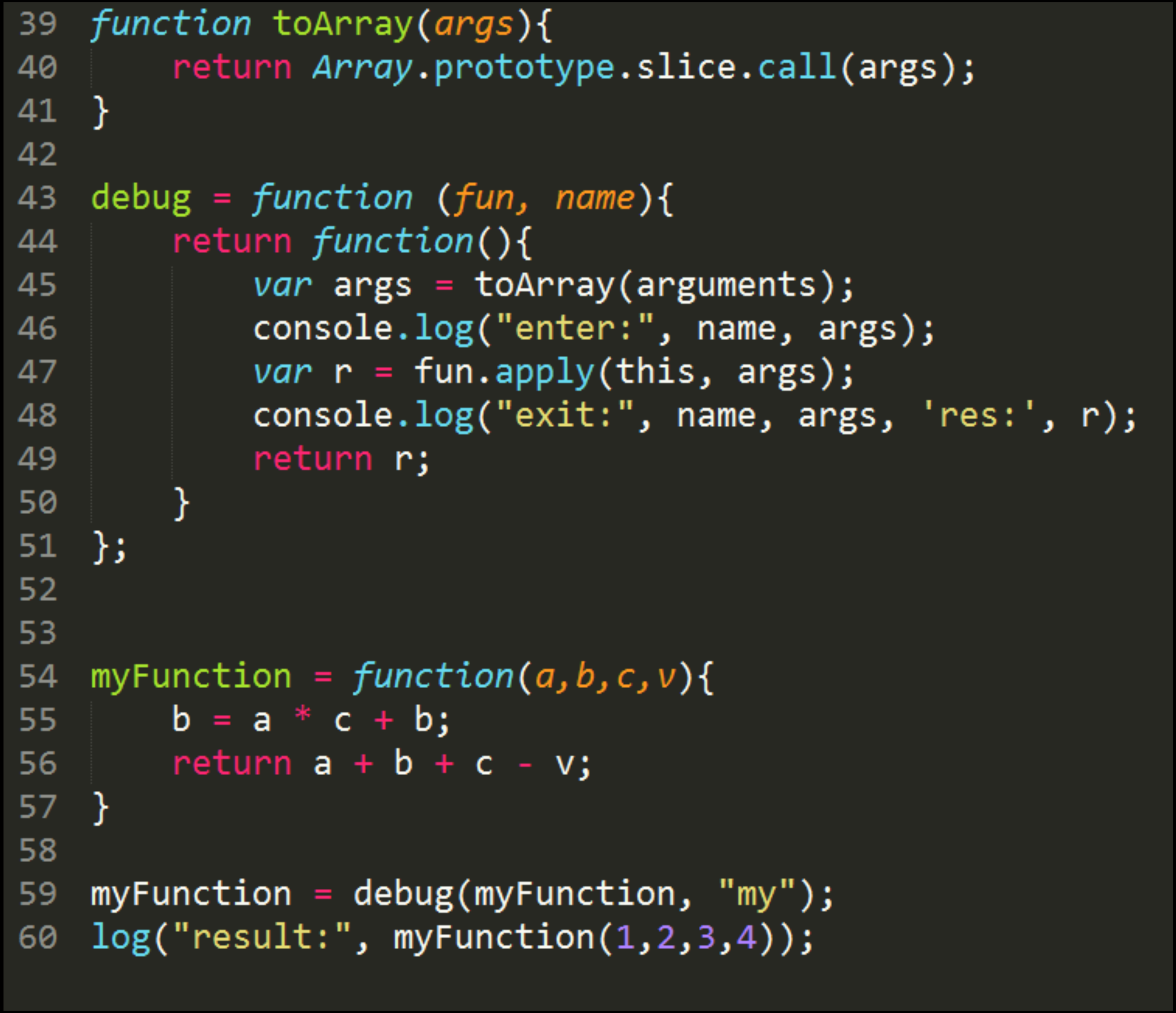
https://github.com/egonelbre/spexs/tree/master/src/debugger



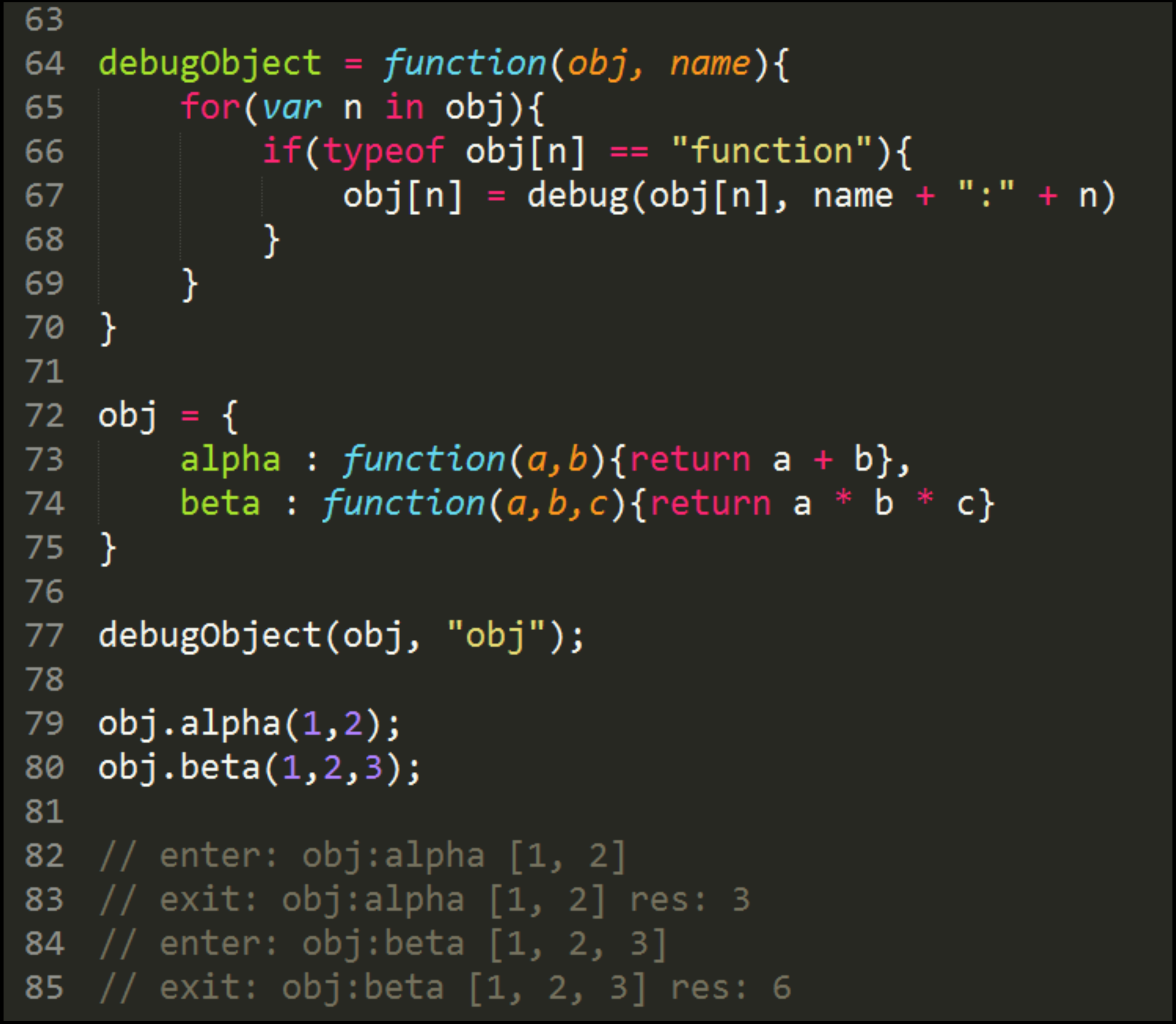
```
myFunction = function(a,b,c){
    b = a * c + b;
    return a + b + c;
}

// and when we start debugging

myFunction = function(a,b,c){
    console.log("enter:", a,b,c);
    b = a * c + b;
    r = a + b + c;
    console.log("exit:",a,b,c, "res:",r);
    return r;
}
```



```javascript
debug = function(fun, name){
    return function(a,b,c){
        console.log("enter:", name, a,b,c);
        r = fun(a,b,c);
        console.log("exit:", name, a,b,c, "res:", r);
        return r;
    }
}

myFunction = function(a,b,c){
    b = a * c + b;
    return a + b + c;
}

myFunction = debug(myFunction, "my");
myFunction(1,2,3);
// enter: my 1 2 3
// exit: my 1 2 3 res: 9
9
```

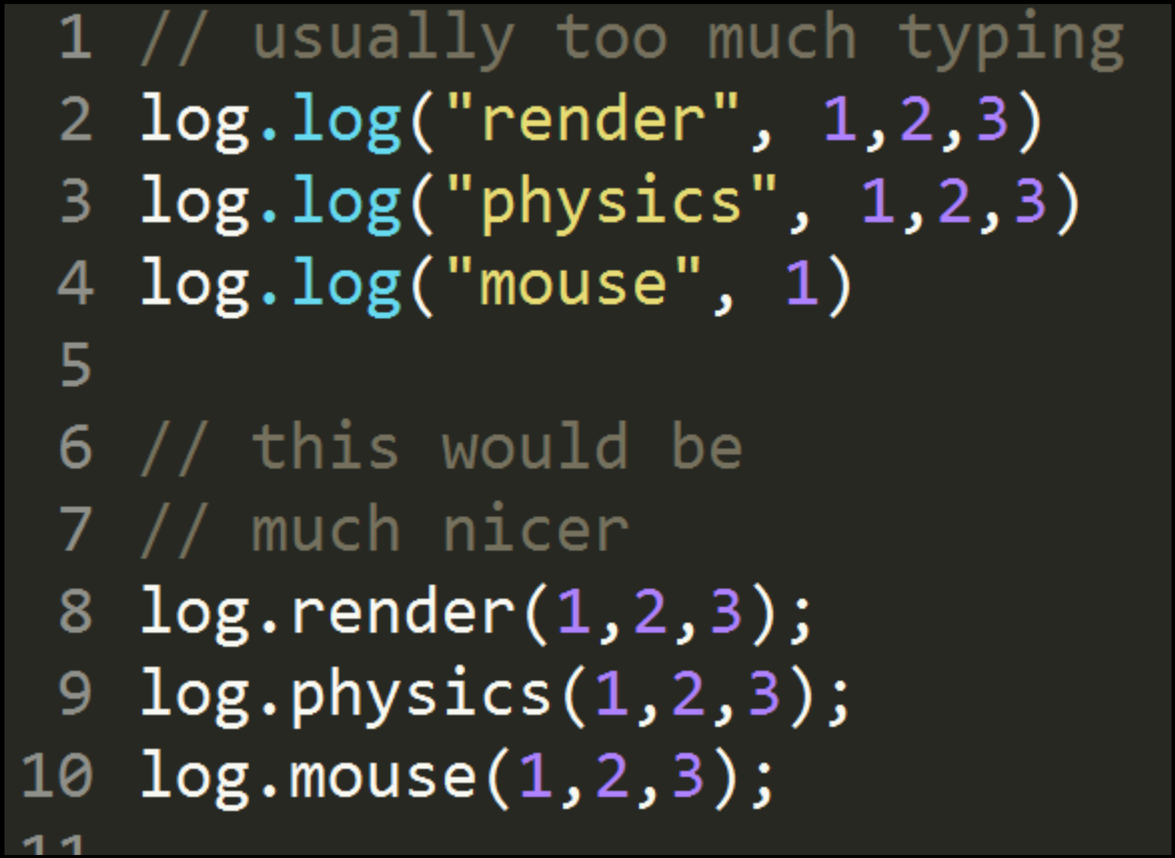


```javascript
function toArray(args){
    return Array.prototype.slice.call(args);
}

debug = function (fun, name){
    return function(){
        var args = toArray(arguments);
        console.log("enter:", name, args);
        var r = fun.apply(this, args);
        console.log("exit:", name, args, 'res:', r);
        return r;
    }
};


myFunction = function(a,b,c,v){
    b = a * c + b;
    return a + b + c - v;
}

myFunction = debug(myFunction, "my");
log("result:", myFunction(1,2,3,4));
```

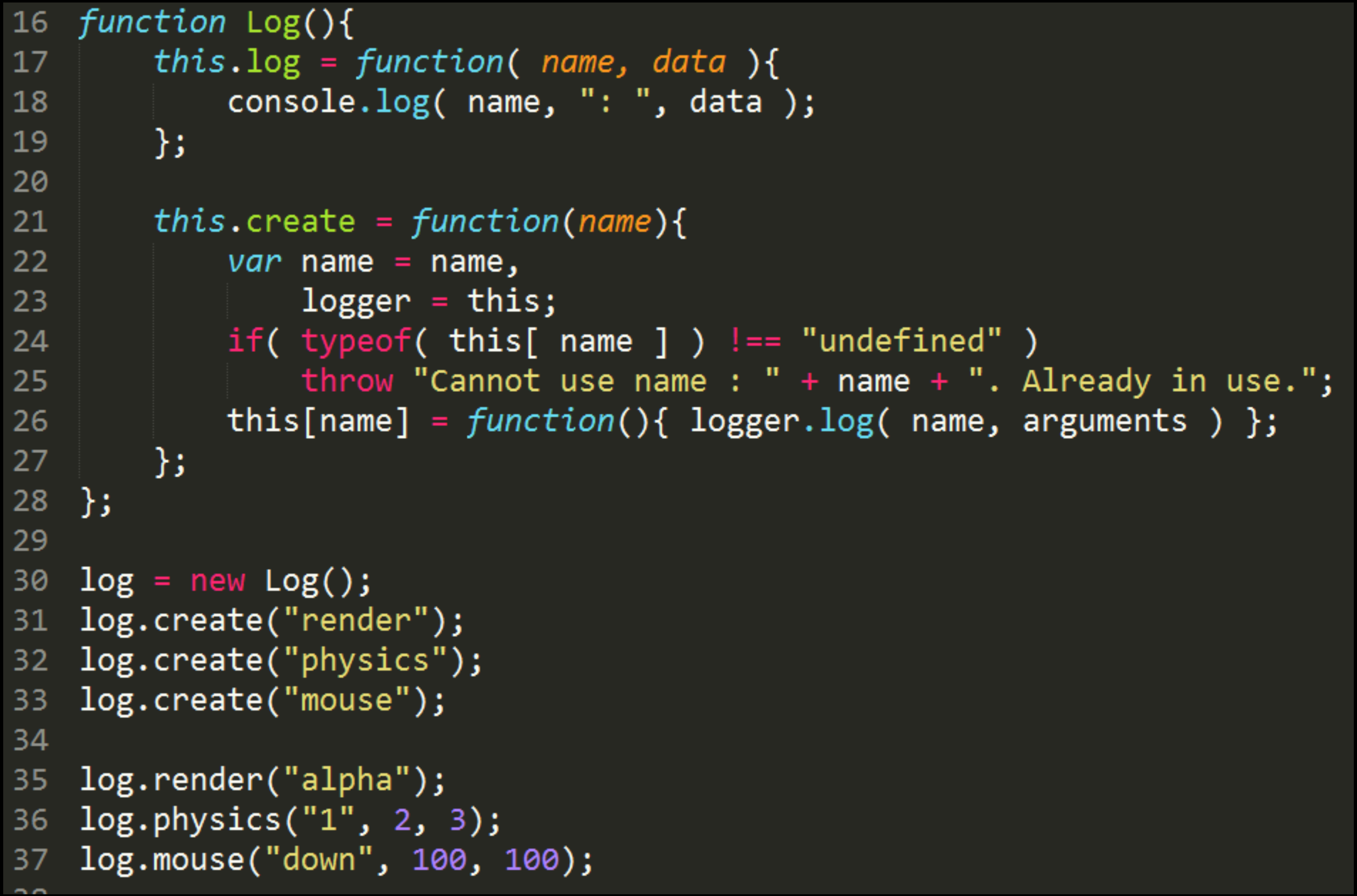


```javascript

debugObject = function(obj, name){
    for(var n in obj){
        if(typeof obj[n] == "function"){
            obj[n] = debug(obj[n], name + ":" + n)
        }
    }
}

obj = {
    alpha : function(a,b){return a + b},
    beta : function(a,b,c){return a * b * c}
}

debugObject(obj, "obj");

obj.alpha(1,2);
obj.beta(1,2,3);

// enter: obj:alpha [1, 2]
// exit: obj:alpha [1, 2] res: 3
// enter: obj:beta [1, 2, 3]
// exit: obj:beta [1, 2, 3] res: 6
```

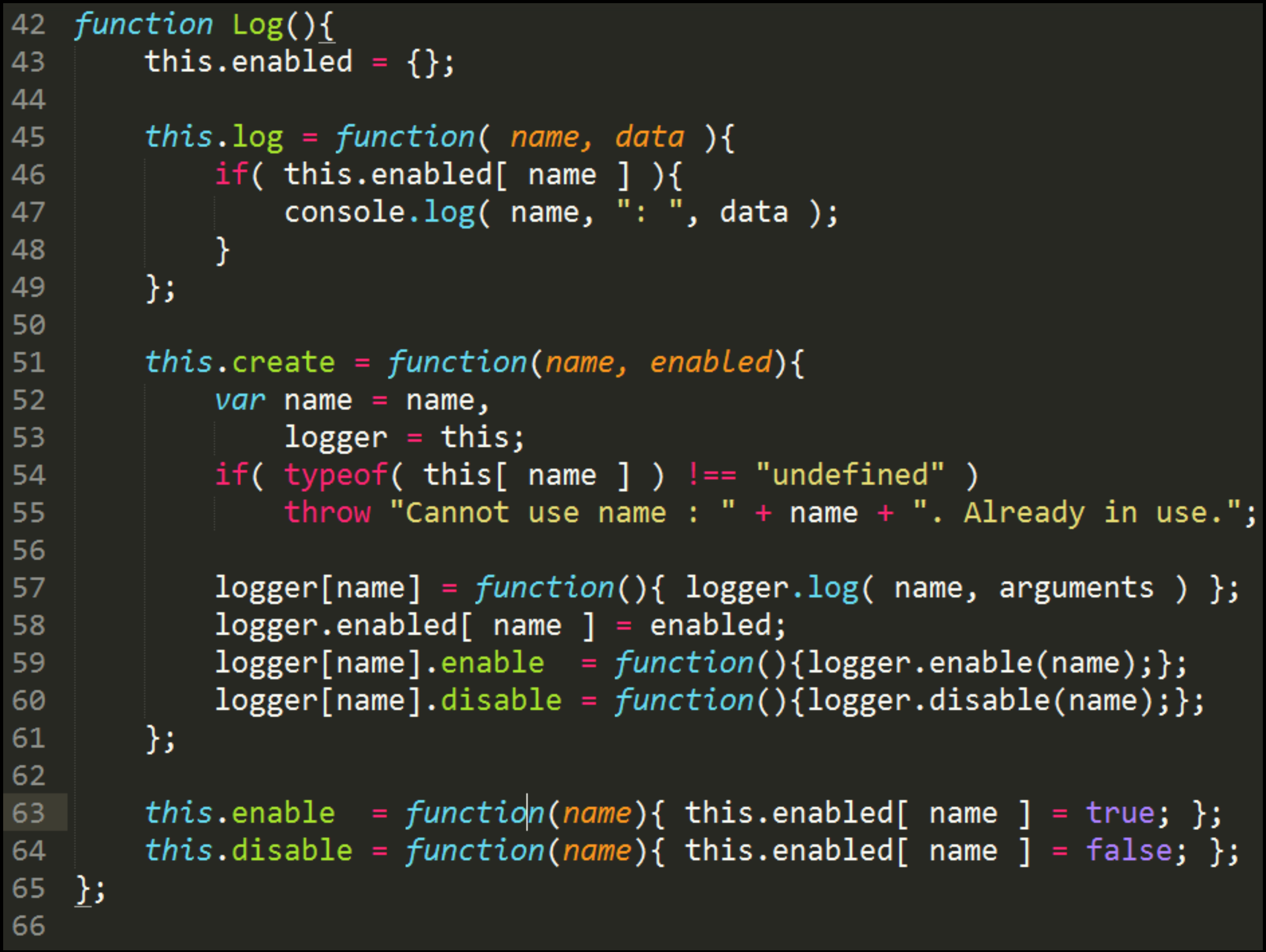
simple logging



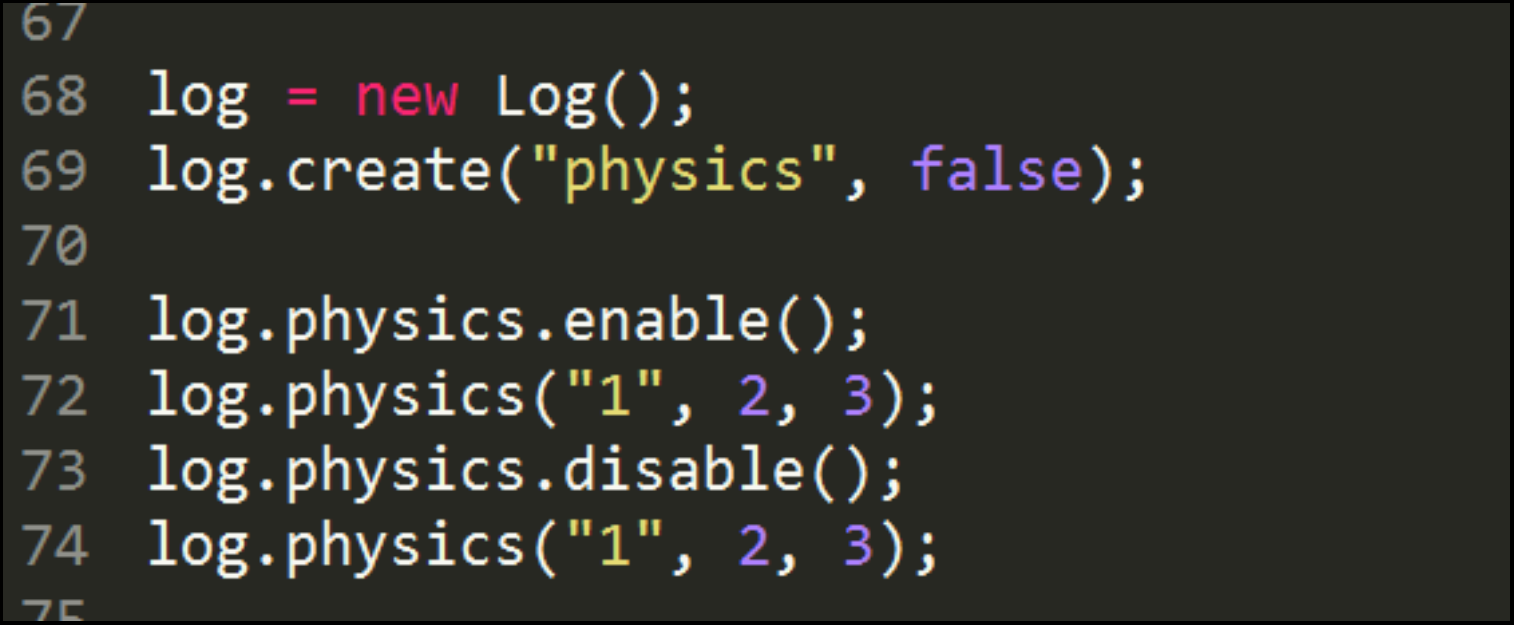
```
// usually too much typing
log.log("render", 1,2,3)
log.log("physics", 1,2,3)
log.log("mouse", 1)

// this would be
// much nicer
log.render(1,2,3);
log.physics(1,2,3);
log.mouse(1,2,3);
```



```javascript
function Log(){
    this.log = function( name, data ){
        console.log( name, ": ", data );
    };

    this.create = function(name){
        var name = name,
            logger = this;
        if( typeof( this[ name ] ) !== "undefined" )
            throw "Cannot use name : " + name + ". Already in use.";
        this[name] = function(){ logger.log( name, arguments ) };
    };
};

log = new Log();
log.create("render");
log.create("physics");
log.create("mouse");

log.render("alpha");
log.physics("1", 2, 3);
log.mouse("down", 100, 100);
```



```javascript
function Log(){
    this.enabled = {};

    this.log = function( name, data ){
        if( this.enabled[ name ] ){
            console.log( name, ": ", data );
        }
    };

    this.create = function(name, enabled){
        var name = name,
            logger = this;
        if( typeof( this[ name ] ) !== "undefined" )
            throw "Cannot use name : " + name + ". Already in use.";

        logger[name] = function(){ logger.log( name, arguments ) };
        logger.enabled[ name ] = enabled;
        logger[name].enable  = function(){logger.enable(name);};
        logger[name].disable = function(){logger.disable(name);};
    };

    this.enable  = function(name){ this.enabled[ name ] = true; };
    this.disable = function(name){ this.enabled[ name ] = false; };
};
```



```
log = new Log();
log.create("physics", false);

log.physics.enable();
log.physics("1", 2, 3);
log.physics.disable();
log.physics("1", 2, 3);
```

Finally

Function manipulation and composition can make code simpler and shorter.

You won't learn it by reading slides.

...so...

GO TRY IN YOUR LANGUAGE