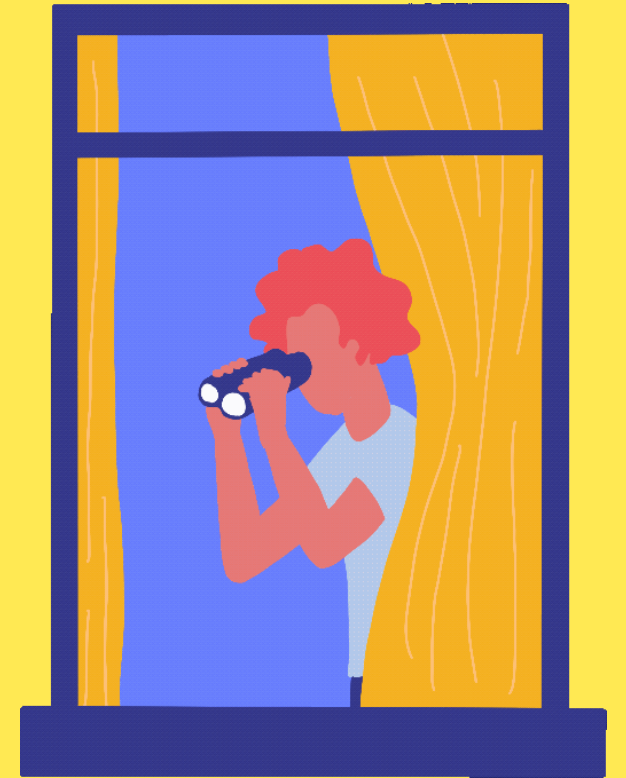# FUNCTIONS IN DETAIL

Important things you should know about functions

SCOPE

Variable "visibility"

The location where a variable is defined dictates **where we have access to that variable**

FUNCTION SCOPE

```javascript
function helpMe(){

    let msg = "I'm on fire!";

    msg; //"I'm on fire";
}

msg; //NOT DEFINED!
```

msg is scoped to the *helpMe* function

FUNCTION SCOPE

```
let bird = 'mandarin duck';

function birdWatch(){

    let bird = 'golden pheasant';
    bird; //'golden pheasant'
}


bird; //'mandarin duck'
```

bird is scoped to
birdWatch function

BLOCK SCOPE

```javascript
let radius = 8;

if(radius > 0){

    const PI = 3.14;

    let circ = 2 * PI * radius;

}

console.log(radius); //8
console.log(PI); //NOT DEFINED
console.log(circ); //NOT DEFINED
```

PI & *circ* are scoped to the BLOCK

LEXICAL SCOPE

```javascript
function outer() {
  let hero = "Black Panther";

  function inner() {
    let cryForHelp = `${hero}, please save me!`
    console.log(cryForHelp);
  }

  inner();
}
```

嵌套函数或内部函数可以访问与父函数或祖父函数相同的内容, 反过来不是

FUNCTION EXPRESSIONS

```javascript
const square = function (num) {
  return num * num;
}
square(7); //49
```

FUNCTIONS ARE... OBJECTS!

HIGHER ORDER FUNCTIONS

HIGHER ORDER FUNCTIONS

Functions that operate on/with other functions.
They can:

- Accept other functions as arguments
- Return a function
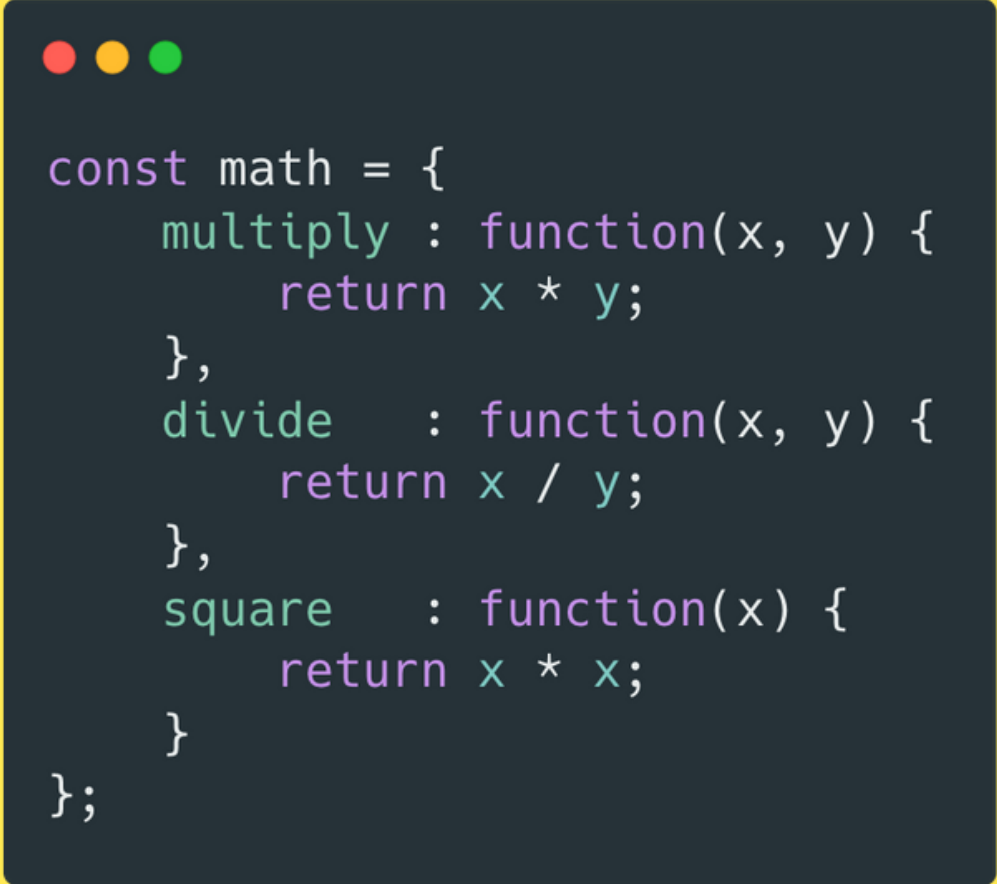
FUNCTIONS AS ARGUMENTS

```javascript
function callTwice(func) {
    func();
    func();
}


function laugh() {
    console.log("HAHAHAHAHAHAHAHHAHAHAHAH");
}
callTwice(laugh) //pass a function as an arg!
//"HAHAHAHAHAHAHAHHAHAHAHAH"
//"HAHAHAHAHAHAHAHHAHAHAHAH"
```

RETURNING FUNCTIONS

```javascript
function makeBetweenFunc(min, max) {
  return function (val) {
    return val >= min && val <= max;
  }
}


const inAgeRange = makeBetweenFunc(18, 100);


inAgeRange(17); //false
inAgeRange(68); //true
```

METHODS

```javascript
const math = {
    multiply : function(x, y) {
        return x * y;
    },
    divide   : function(x, y) {
        return x / y;
    },
    square   : function(x) {
        return x * x;
    }
};
```

We can add functions as properties on objects.

We call them **methods**!

SHORTHAND

```
const math = {
  blah: 'Hi!',
  add(x, y) {
    return x + y;
  },
  multiply(x, y) {
    return x * y;
  }
}
math.add(50, 60) //110
```

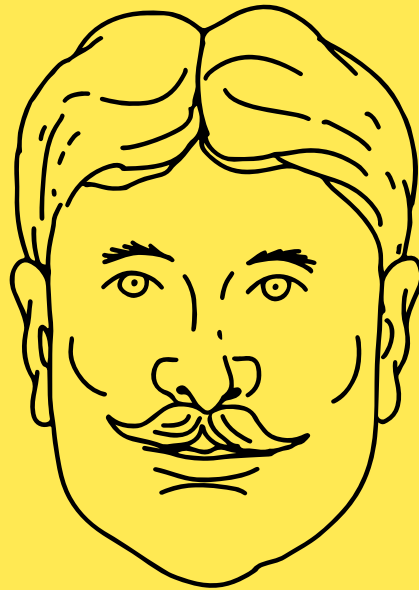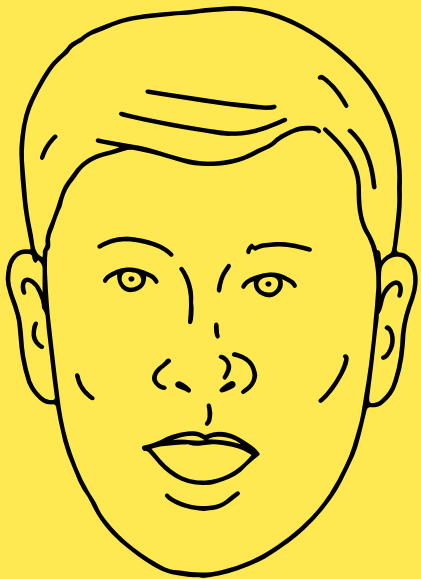We do this so often that there's a new shorthand way of adding methods.

'THIS' IN METHODS

Use the keyword **this** to access other properties on the same object.

```javascript
const person = {
  first: 'Robert',
  last: 'Herjavec',
  fullName() {
    return `${this.first} ${this.last}`
  }
}
person.fullName(); //"Robert Herjavec"
person.last = "Plant";
person.fullName(); //"Robert Plant"
```

The value of *this* depends on the invocation context of the function it is used in.

SAME FUNCTION

```
person.fullName();
//"Robert Herjavec"
```

```
const person = {
  first: 'Robert',
  last: 'Herjavec',
  fullName() {
    return `${this.first} ${this.last}`
  }
}
```

DIFFERENT RESULT???

```
const func = person.fullName;
func()
//"undefined undefined"
```

The value of *this* depends on the **invocation context** of the function it is used in.