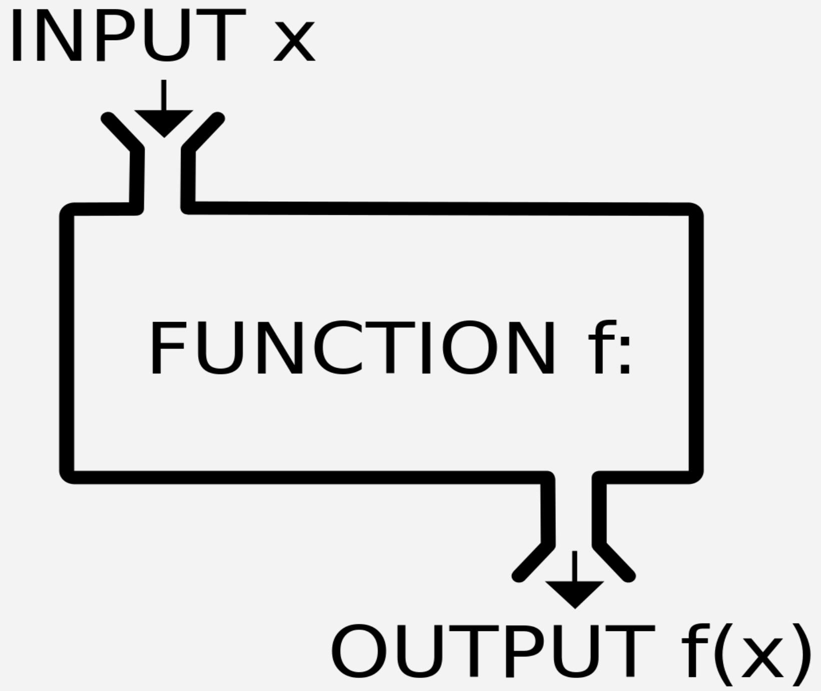
Functions

What is a function


INPUT x
FUNCTION f:
OUTPUT f(x)

Why?

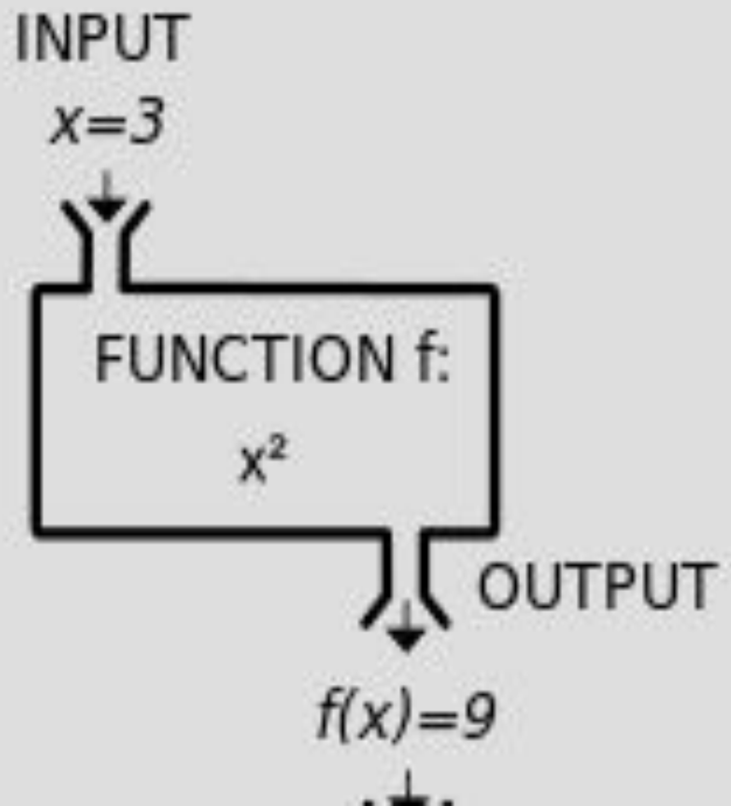
When you are executing behavior more than one time

You can avoid repeating yourself

You can code reuse complex behaviour without needing to recreate the code

To break problems down into smaller parts

Mathematical



A relation between a set of inputs and a set of outputs where each input is related to exactly one output.

The New York Code + Design Academy

Programming

Can take any number of inputs (including 0)

Not necessarily used to return a value (When?)

Each input is not necessarily related to exactly one output (Why?)

Pure Functions

Functions constructed to behave like a mathematical function

Function definition

```
keyword     name        parameters
    ^           ^               ^
function     adder     (firstNum,  secondNum)  {
    return firstNum + secondNum;
}

      v

   keyword
```

Key Ideas

Container to store lines of code in

Defined functions may be invoked any number of times

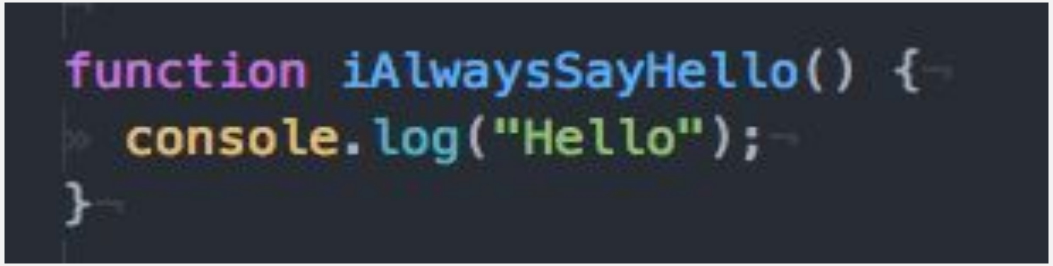
The New York Code + Design Academy

Definitions

- <u>function name</u> the name you create for your function to reference later
- <u>parameter</u> the expected inputs defined when creating a function,
  you can define a function with no parameters
- <u>return value</u> the value that the function outputs
  - you can have a function without a return value/statement (will still return undefined)
  - you can have multiple return statements within a function, but once one of them is read, the function will stop executing and return that value

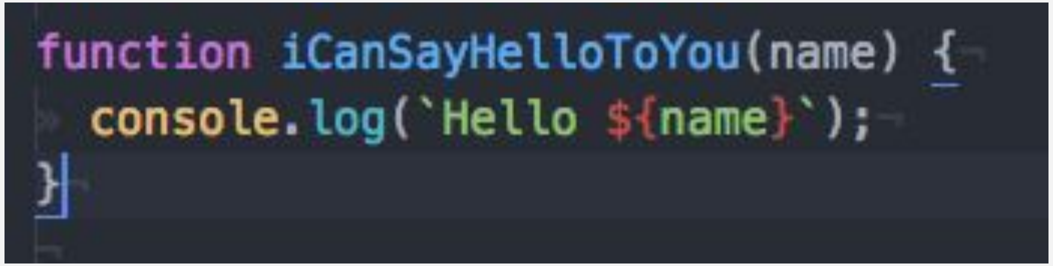
Examples

No parameters



```
function iAlwaysSayHello() {
  console.log("Hello");
}
```

Parameters

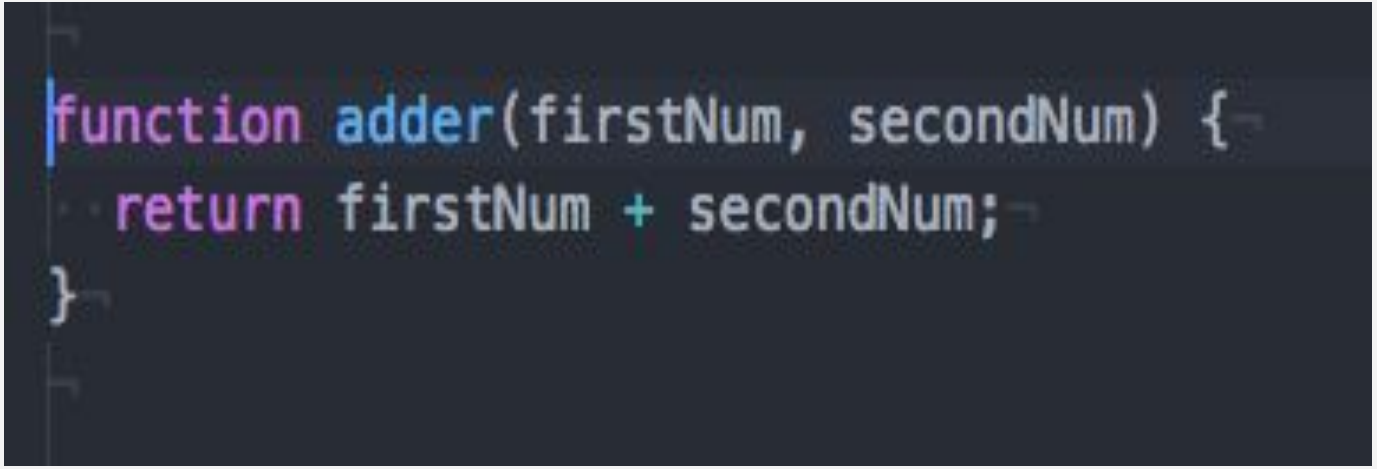


```
function iCanSayHelloToYou(name) {
  console.log(`Hello ${name}`);
}
```

Neither has a return statement

Examples

Single return statement

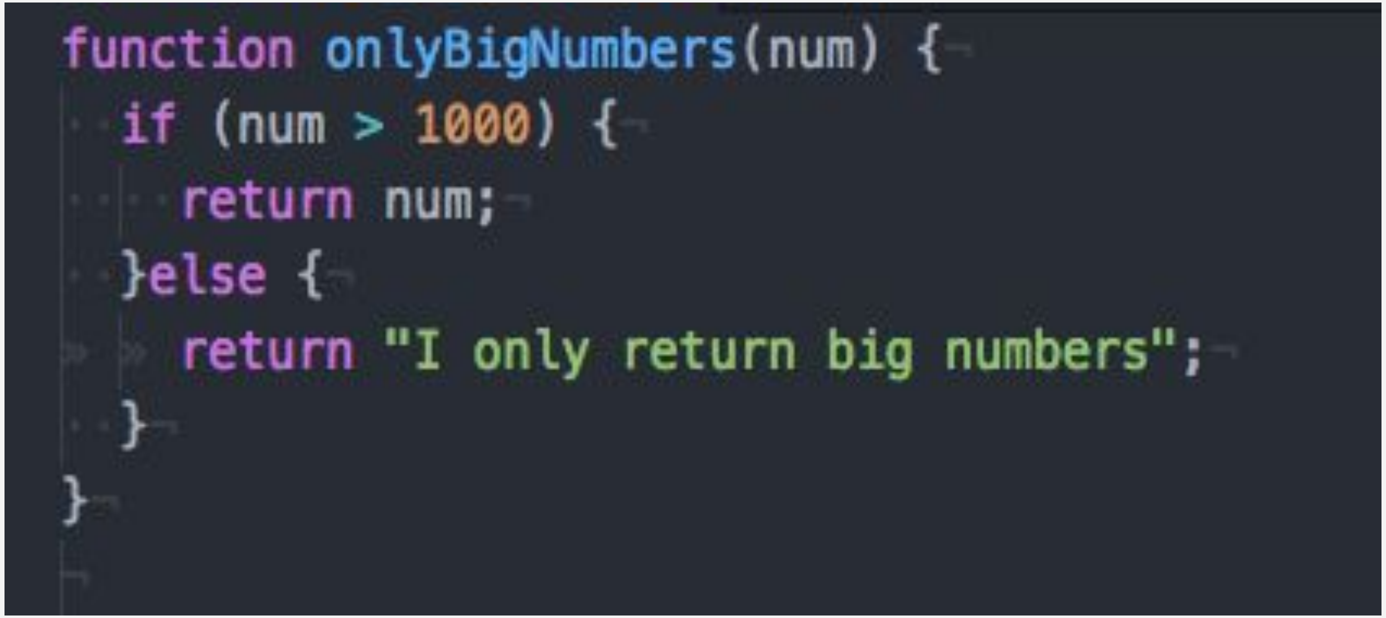


```
function adder(firstNum, secondNum) {
  return firstNum + secondNum;
}
```

The New York Code + Design Academy

Examples

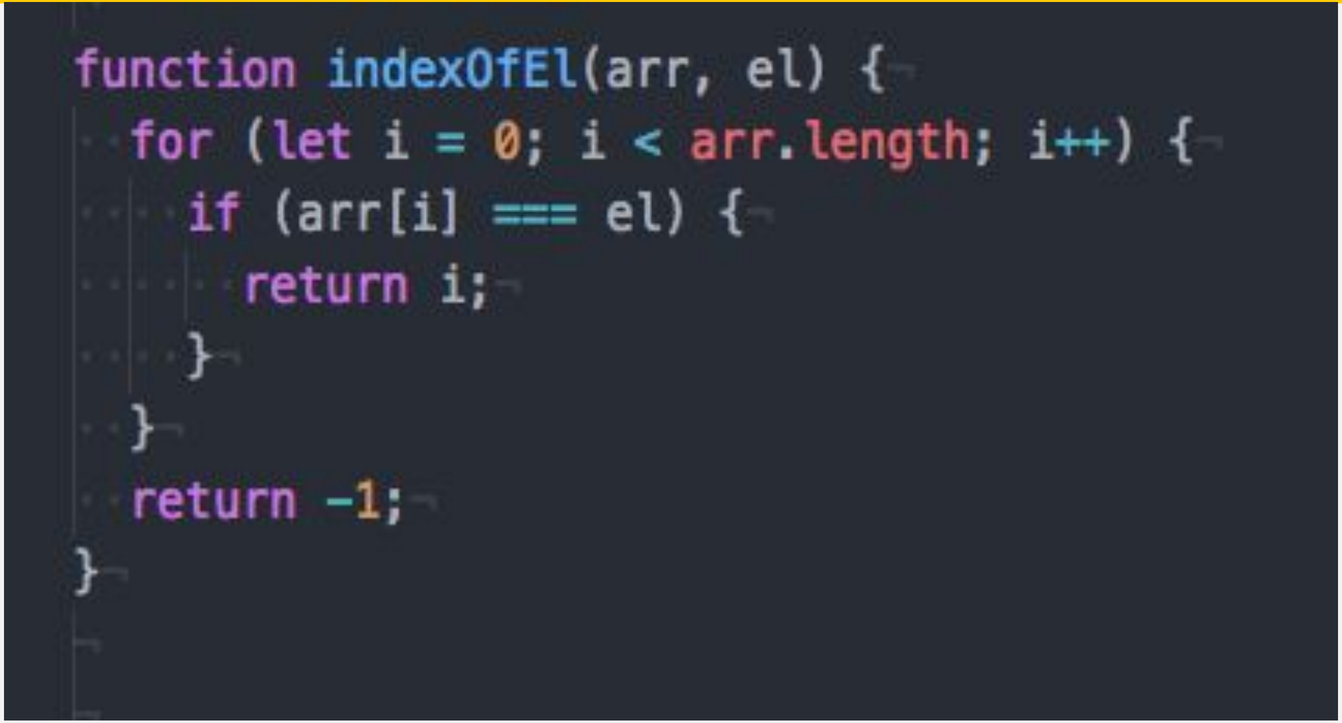
Multiple return statements



```
function onlyBigNumbers(num) {
  if (num > 1000) {
    return num;
  }else {
    return "I only return big numbers";
  }
}
```

The New York Code + Design Academy

Examples



```
function indexOfEl(arr, el) {
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] === el) {
      return i;
    }
  }
  return -1;
}
```

The New York Code + Design Academy

Function invocation

We have previously only been defining functions in our examples. In order to actually run the code we must invoke our functions.

Invocation = calling = executing

We’ve been invoking functions all along!

console.log(“This is function invocation”)

Invocation

Function definition

```
function adder (firstNum,  secondNum)  {
        return firstNum + secondNum;
}
```

Invocation

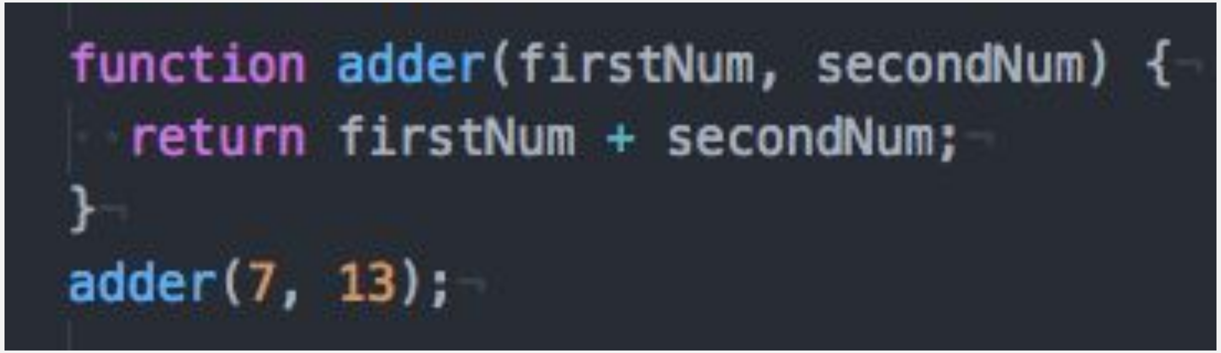
```
name   arguments
  ^        ^
adder(4, 5);
```

The New York Code + Design Academy

Arguments vs Parameters

Think of parameters as variable names that you're assigning for your function during function definition. Refer to your parameters inside the function block just as you would with declared variables.

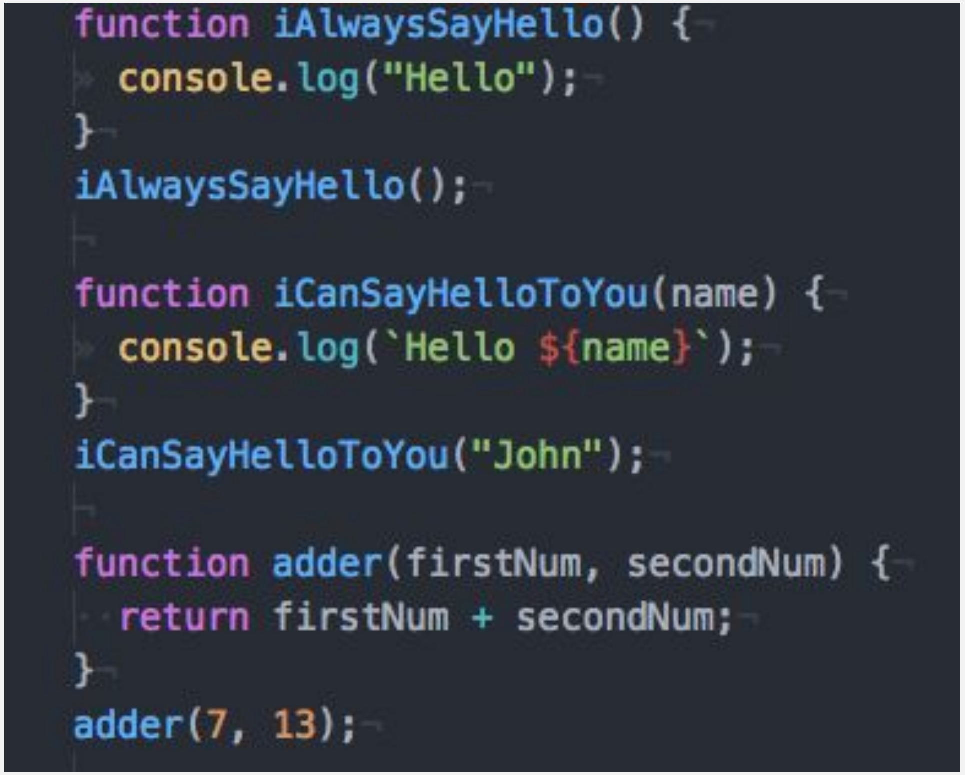
When we invoke our function we assign values to our parameters through corresponding arguments.



```
function adder(firstNum, secondNum) {
  return firstNum + secondNum;
}
adder(7, 13);
```

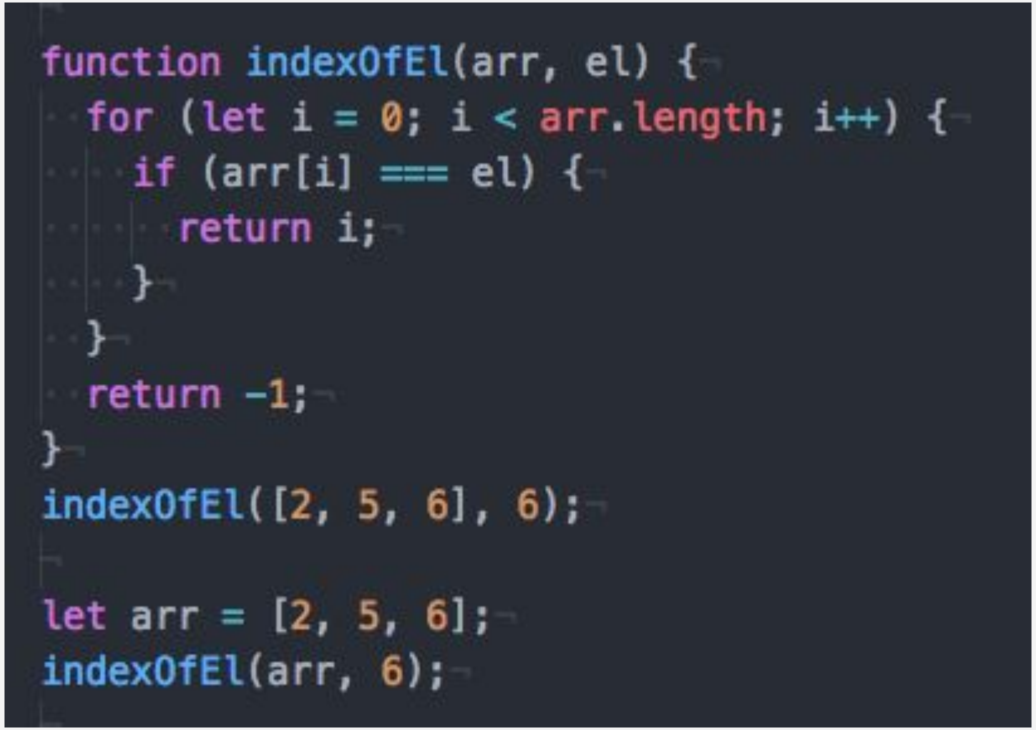
Note - The terms argument and parameter are often used interchangeably.

Function invocation examples


```
function iAlwaysSayHello() {
  console.log("Hello");
}
iAlwaysSayHello();

function iCanSayHelloToYou(name) {
  console.log(`Hello ${name}`);
}
iCanSayHelloToYou("John");

function adder(firstNum, secondNum) {
  return firstNum + secondNum;
}
adder(7, 13);
```

The New York Code + Design Academy

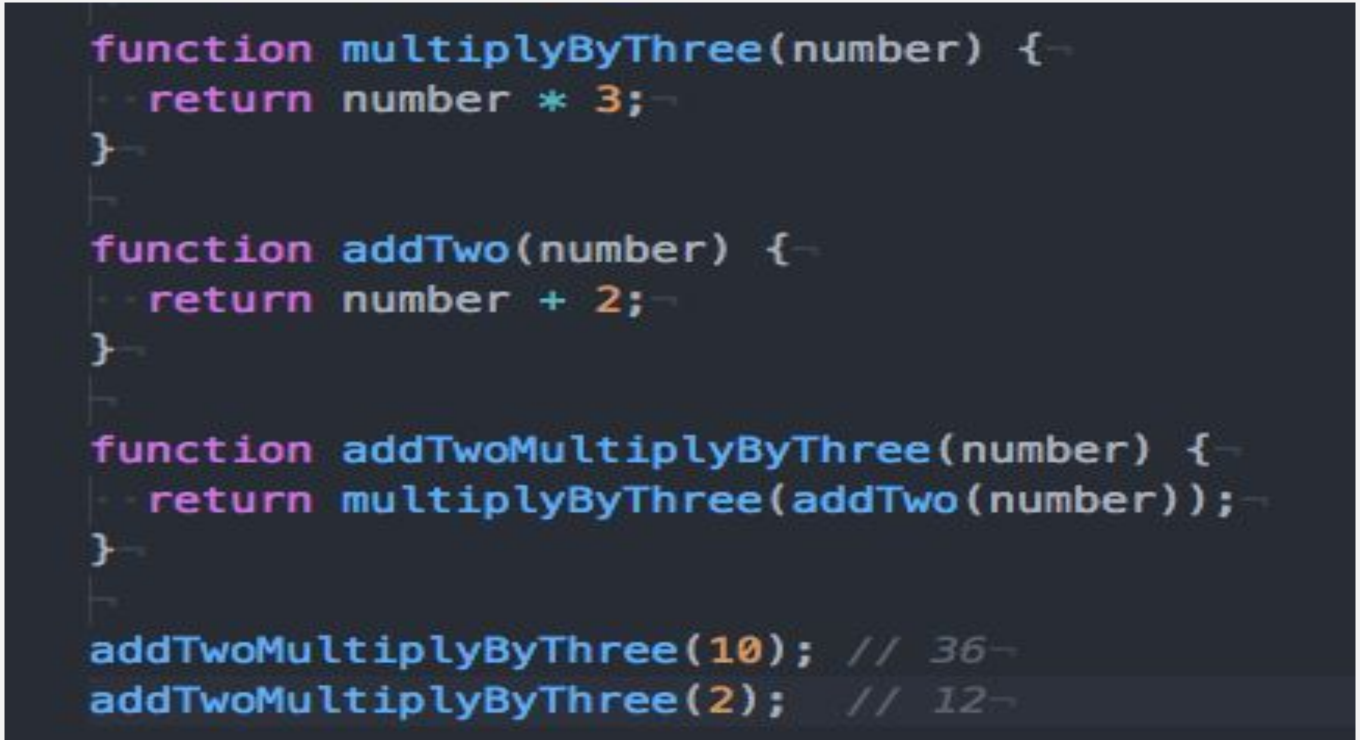
Function invocation examples



```
function indexOfEl(arr, el) {
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] === el) {
      return i;
    }
  }
  return -1;
}
indexOfEl([2, 5, 6], 6);

let arr = [2, 5, 6];
indexOfEl(arr, 6);
```

Functions within functions

You can invoke a function within another function.



```
function multiplyByThree(number) {
  return number * 3;
}

function addTwo(number) {
  return number + 2;
}

function addTwoMultiplyByThree(number) {
  return multiplyByThree(addTwo(number));
}

addTwoMultiplyByThree(10); // 36
addTwoMultiplyByThree(2);  // 12
```

The New York Code + Design Academy

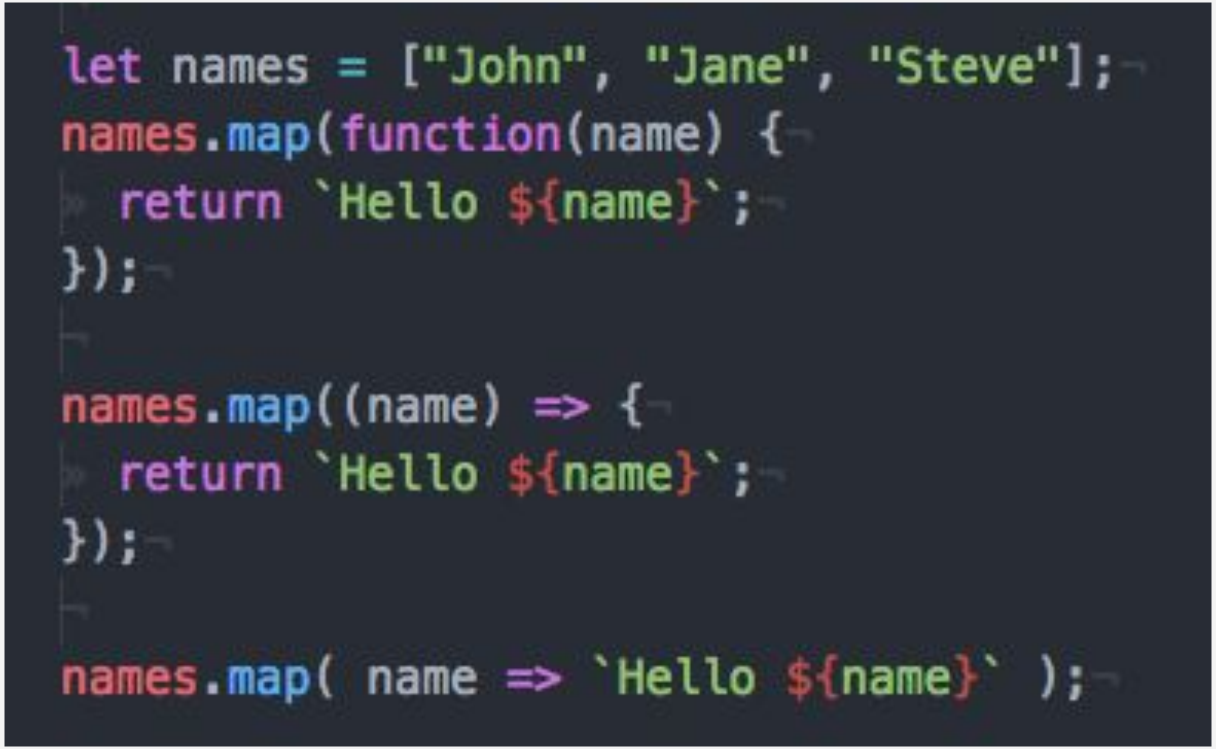
Callbacks and anonymous functions

Callback A function, passed to another function, as an argument

Anonymous function A function without a function name

.map

Map



```
let names = ["John", "Jane", "Steve"];
names.map(function(name) {
  return `Hello ${name}`;
});

names.map((name) => {
  return `Hello ${name}`;
});

names.map( name => `Hello ${name}` );
```

The New York
Code + Design
Academy

Recursion!!!?!?!?!?!?!?

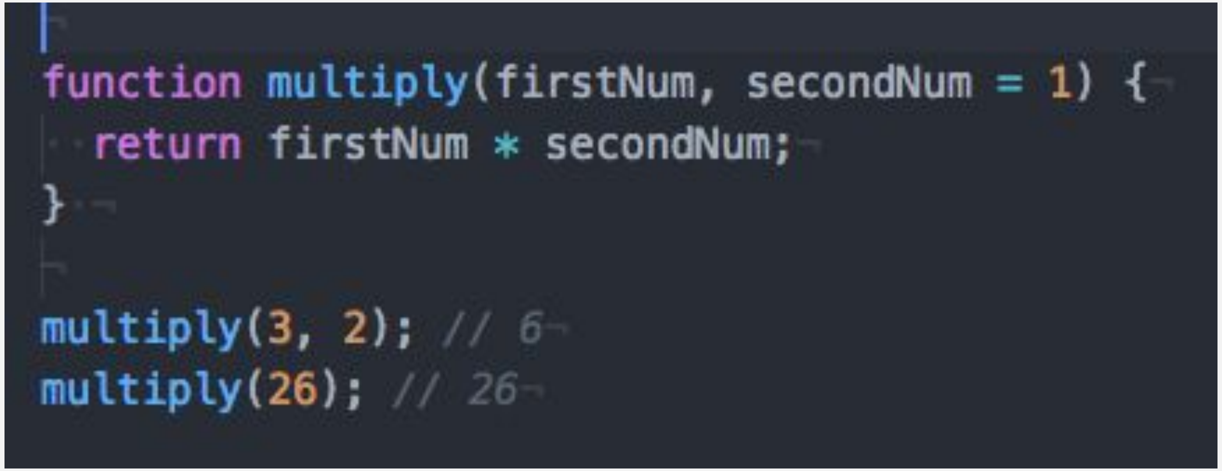
Recursion A function that calls itself

3 Steps

1. Base Case
2. Reduce Input
3. Recursive Step

Default parameters

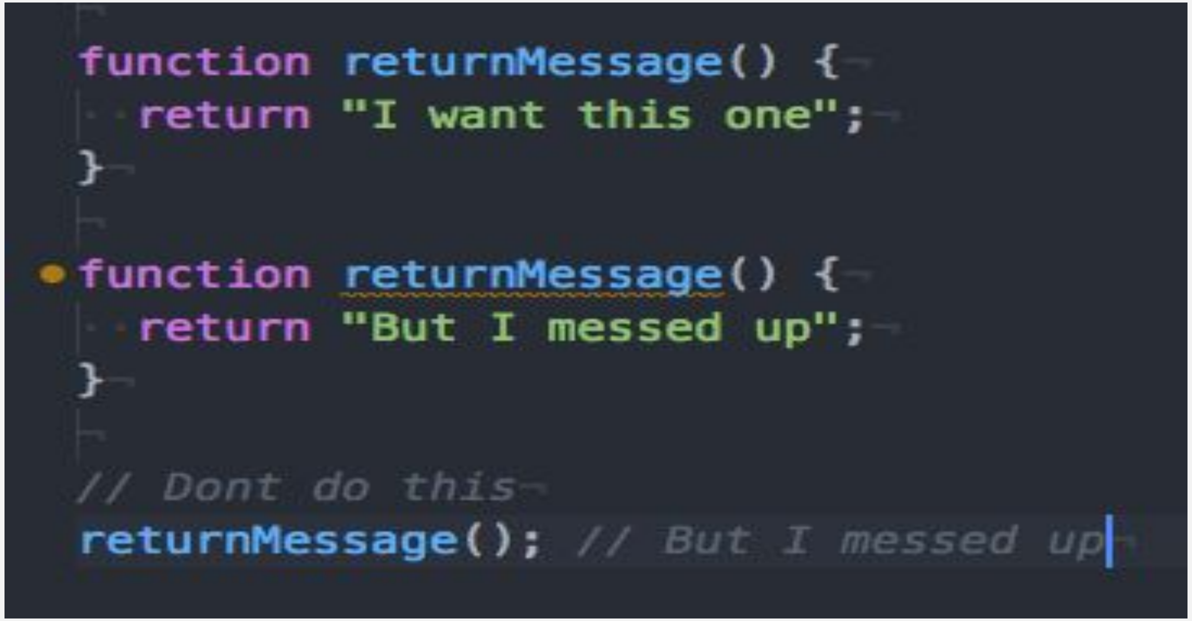
You can provide default values for a parameter, if you are unsure how many arguments your function will be passed.



```
function multiply(firstNum, secondNum = 1) {
  return firstNum * secondNum;
}

multiply(3, 2); // 6
multiply(26); // 26
```

Name Collision

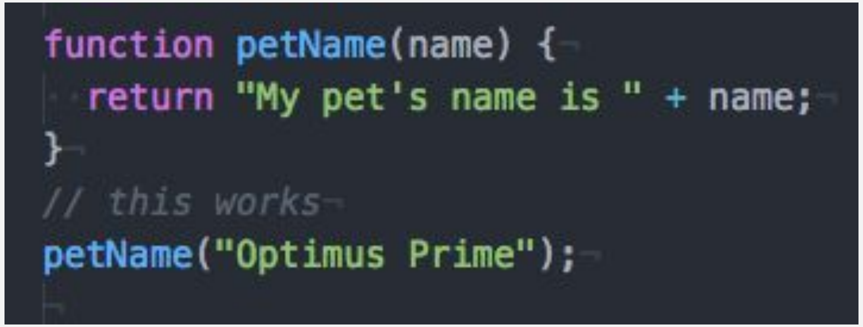
If you use the same function name twice, the second one will take precedence.



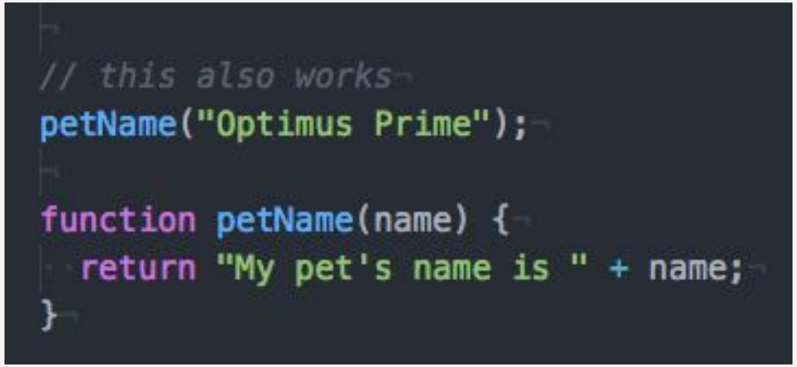
```
function returnMessage() {
    return "I want this one";
}

function returnMessage() {
    return "But I messed up";
}

// Dont do this
returnMessage(); // But I messed up
```

The New York Code + Design Academy

Hoisting



```
function petName(name) {
  return "My pet's name is " + name;
}
// this works
petName("Optimus Prime");
```



```

// this also works
petName("Optimus Prime");

function petName(name) {
  return "My pet's name is " + name;
}
```

[MDN - Hoisting]

{The New York Code + Design Academy}

Additional Topics

[Pass by value, pass by reference]