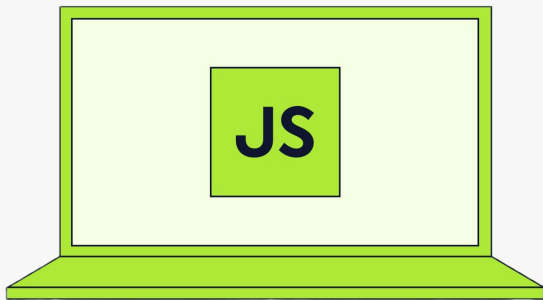


The Complete Javascript Course



Lecture 4: Functions

-Vishal Sharma

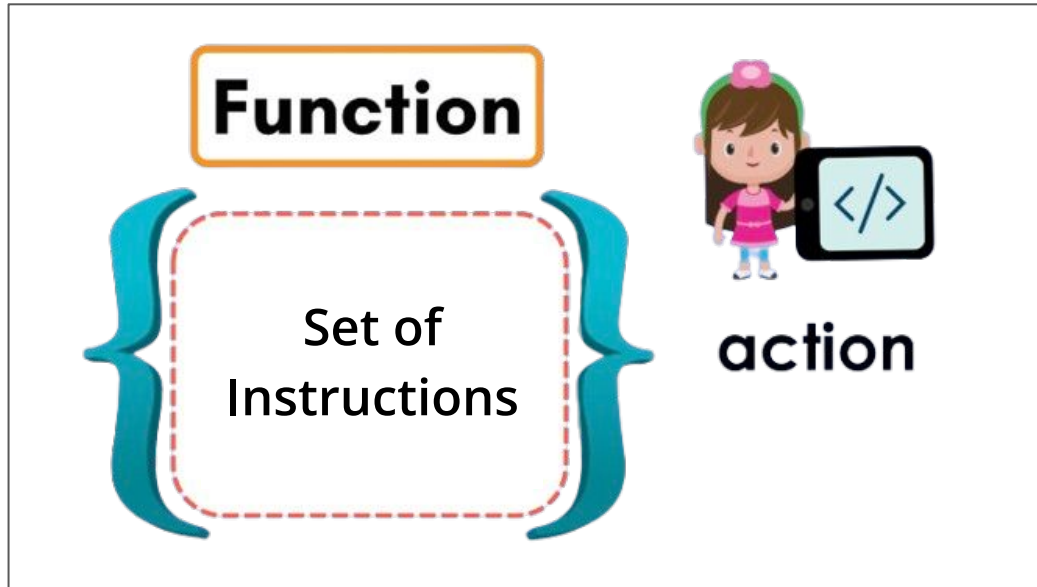
Table of Contents

- Function Declaration and Syntax
- Parameters vs Arguments
- Spread and Rest Operator
- Function Return Values
- Higher Order Functions and Callback Functions

Function Declaration and Syntax

What is a Function?

Function is a block of code made out of set of steps which result in a single specific action.



What does it mean??
Let's understand with an
analogy

Functions: Like Tying your shoes

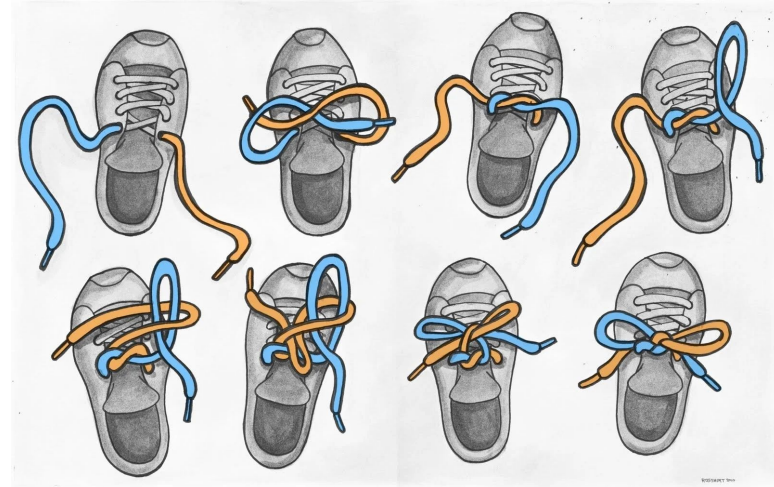
A function is like tying your shoes—once you learn how, you can repeat the steps whenever needed without rethinking the process

Functions

=

Tying your shoes

- 1 Gather laces
- 2 Knot
- 3 Loop
- 4 Swoop
- 5 Pull tight



Similarly we use functions in our programs

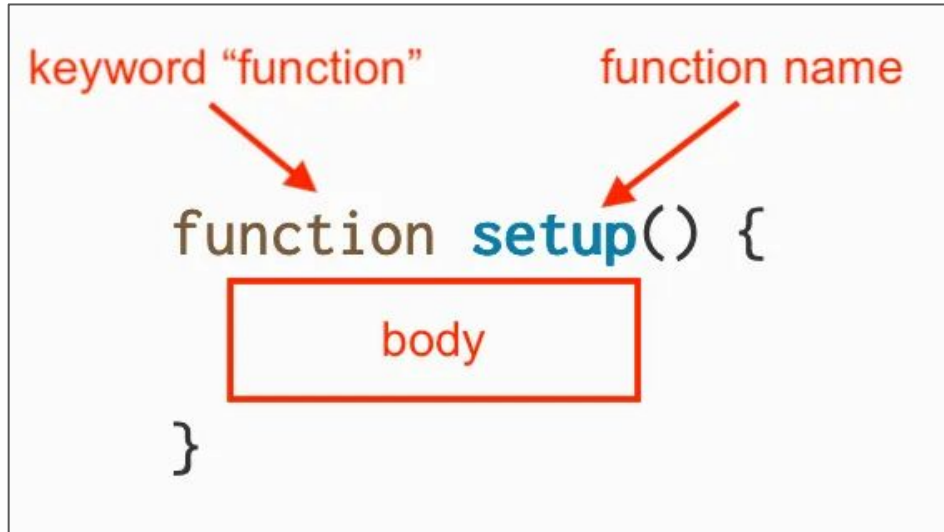
Let us define a tie shoes function and reuse it multiple times:-

```
1 // Function to tie your shoes
2 function tieShoes() {
3     console.log("Cross the laces.");
4     console.log("Make a loop with one lace.");
5     console.log("Wrap the other lace around the loop.");
6     console.log("Pull through and tighten.");
7 }
8
9 // Reusing the function multiple times
10 tieShoes(); // First time
11 tieShoes(); // Second time
```



Naming a function

A programmer can give function a name and be used again and again.



Challenge:

**Write a Function to Display
Hello World**

Were you able to do it??

Let's start writing:-



```
1  function sayHello() {  
2      console.log("Hello, World!");  
3  }  
4  
5      sayHello();  
6  // Output: Hello, World!
```

How simple it is!!


Think of a more complex task/function

Your dad wants you to calculate your daily expenses but you find it difficult to do manually, you are crying for help!!



Your Rescue: Write an add function

Don't worry functions are here for rescue.



```
1  function add(){  
2      // your code here  
3  }
```

But how to tell the function
what to add??

Function: parameters

We can define parameters in function to receive values.

Function Parameters

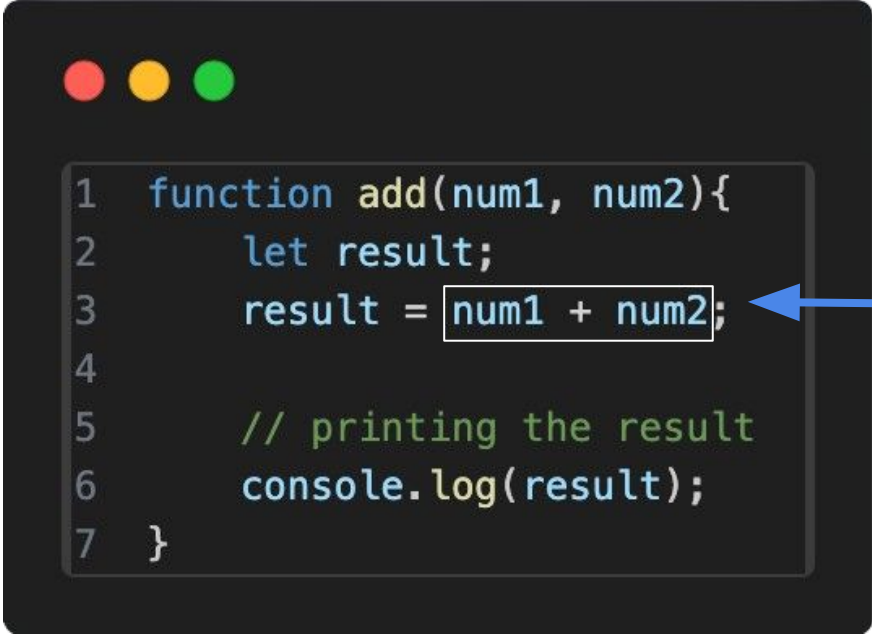


```
1 function add(num1, num2){  
2     // your code here  
3 }
```


Here “*num1*” and
“*num2*” are
function
parameters.

Function: using parameters to add

Let's use parameters to calculate the sum. And print it:-



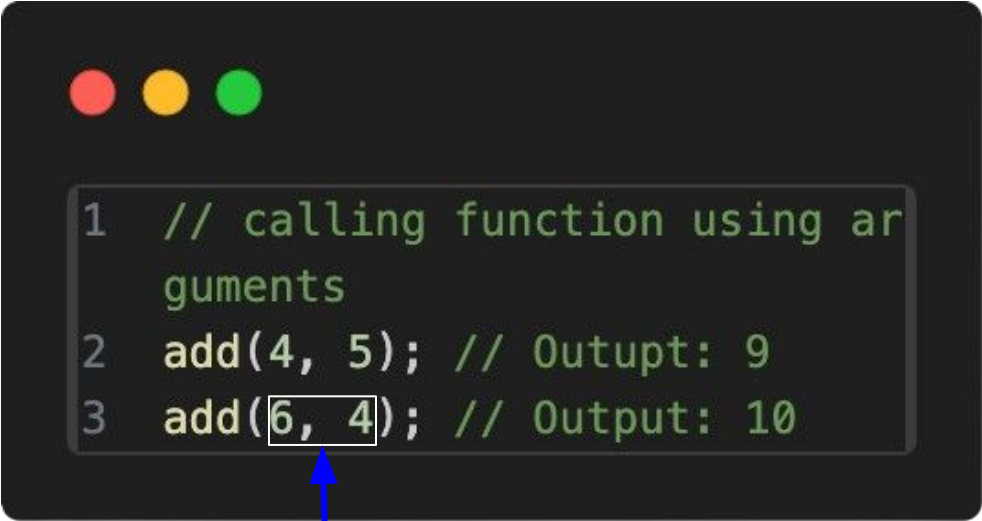
```
1 function add(num1, num2){  
2   let result;  
3   result = num1 + num2;  
4  
5   // printing the result  
6   console.log(result);  
7 }
```



Using parameters to
perform addition

Function: calling using arguments

Let's just call the function by passing values inside it:-



```
1 // calling function using ar  
  guments  
2 add(4, 5); // Outupt: 9  
3 add(6, 4); // Output: 10
```

A blue arrow points from the text box below to the number 6 in the third line of code.

Arguments are the values passed to a function and assigned to its parameters.

Actual values passed to function are called ***arguments***

Parameters vs Arguments

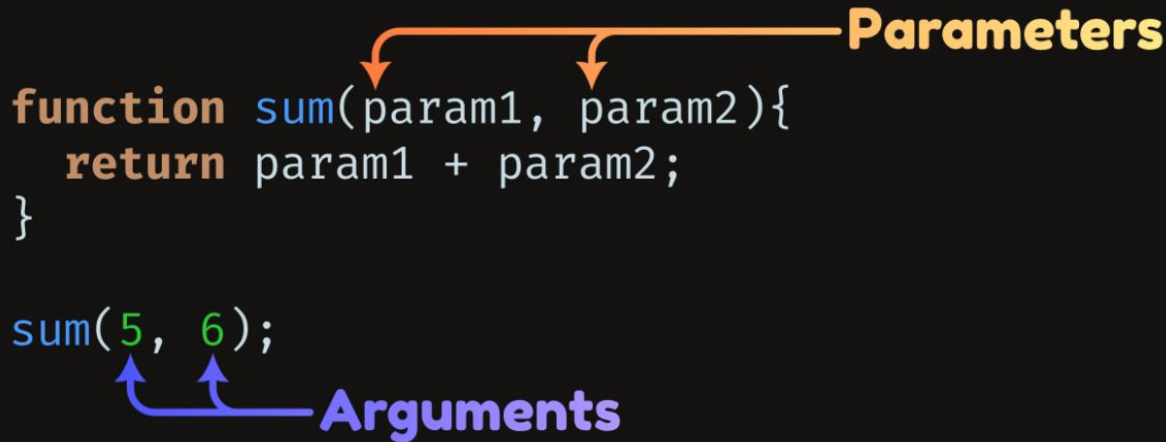
Difference: parameters and arguments

Now can you tell me the difference between these two??



Difference: parameters and arguments

Parameters are placeholders in a function declaration, while arguments are the actual values passed during a function call.



```
function sum(param1, param2){  
    return param1 + param2;  
}  
  
sum(5, 6);
```

Parameters

Arguments

Function Return Values

Function: return values

Wherever you assign some task to someone, you expect them to return something which is could be final outcome/status of task etc.



Return Values

Function: return values


Similarly function do return values on completion.

return in
Javascript.....




Function: return value syntax

Similarly function do return values on completion.




```
1  function findSquare(num) {  
2      let result = num * num;  
3      return result;  
4  }  
5  
6  let square = findSquare(3);
```




We want to return result value to where function was called from.

Function: return values with Example

Let's reuse the add function again but this time we will return values instead of printing.



```
1 function add(num1, num2){  
2     let result;  
3     result = num1 + num2;  
4  
5     // printing the result  
6     console.log(result);  
7 }
```



```
1 function add(num1, num2){  
2     let result;  
3     result = num1 + num2;  
4  
5     // returning the result  
6     return result;  
7 }
```



Function: return values with Example

Displaying returned values by calling the function and storing result in a variable.

```
1 function add(num1, num2){  
2     let result;  
3     result = num1 + num2;  
4  
5     // returning the result  
6     return result;  
7 }  
8  
9 let sum = add(2, 3);  
10 console.log(sum); // Output: 5
```

Returning values is preferred over displaying the result inside the function.

Functions: Arrow Functions

Arrow functions are a shorter syntax for writing functions in JavaScript. They use the `=>` syntax and are often more concise than traditional function expressions.

`() => { }`

```
() => { }
```

```
(a) => { return a + a }
```


```
(a) => a + a
```

```
a => a + a
```

They not only make function easier to write but also easier to read and debug.

Functions: Arrow Function example

Let's look at an example:-

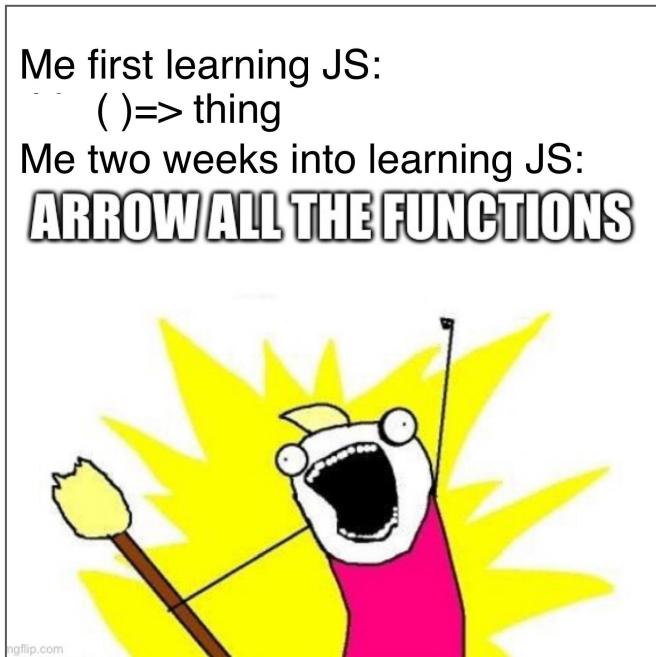


```
1 // Traditional function
2 function multiply(a, b) {
3     return a * b;
4 }
5
6 // Arrow function
7 const multiply = (a, b) => a * b;
```

Can you observe that arrow function syntax is simpler, easier and more readable compared to traditional function.

Arrow Functions: Shortcut to Simplicity

Arrow functions make coding faster and cleaner, like taking a shortcut.

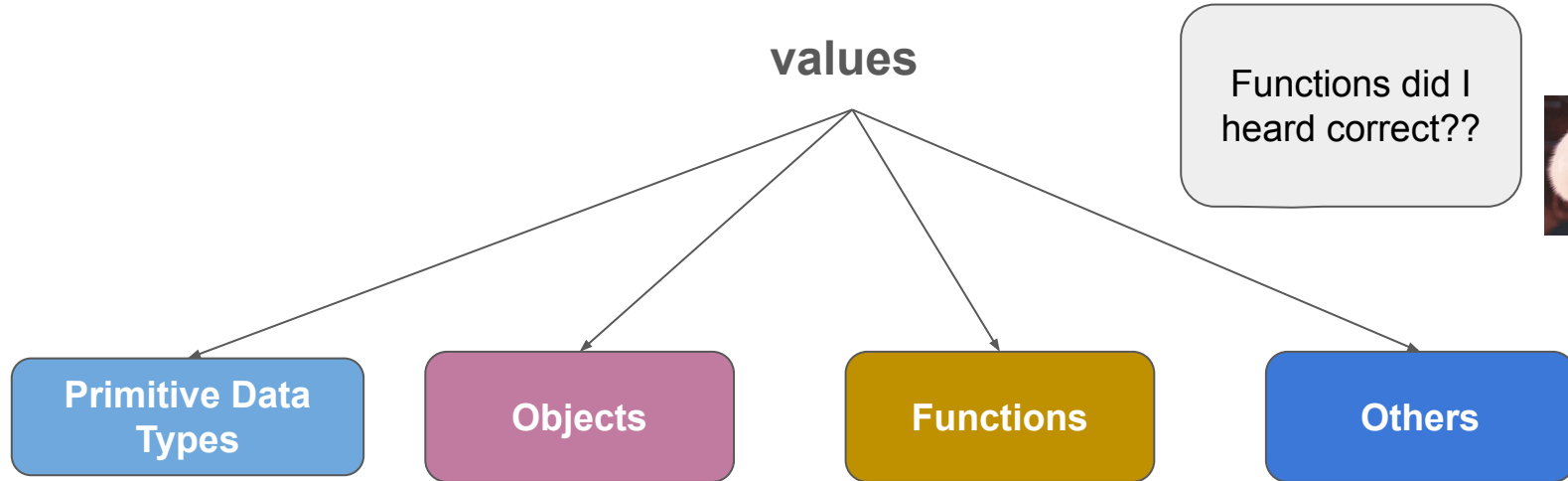


Feeling awesome,
right??

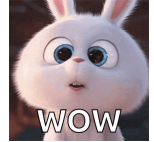
Higher Order Functions and Callbacks

Values we can pass/return in a function

You can pass variety of values in a function. Let's have a look:-

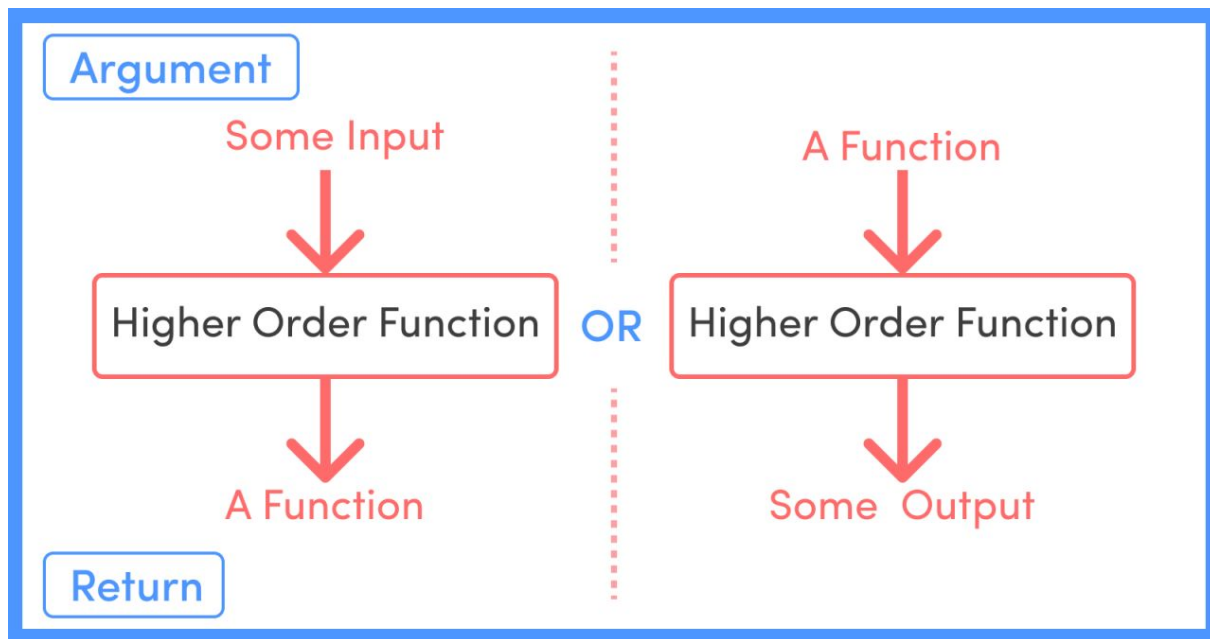


Functions did I
heard correct??



Higher-Order Functions

A **higher-order function** is a function either accepts a function as an argument or returns a function.

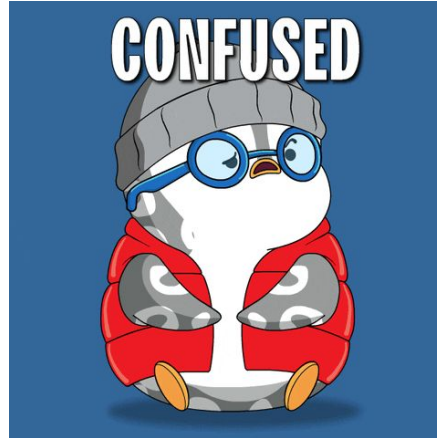


But why do we
need function to
receive and accept
values??

Advantage of Higher-Order Functions

We receive or return functions as values to make our code flexible and reusable, allowing us to define behavior dynamically and build more modular, maintainable programs.

Understood??



Don't worry? Let's understand it with an example in next slides.

Understanding HOF with example

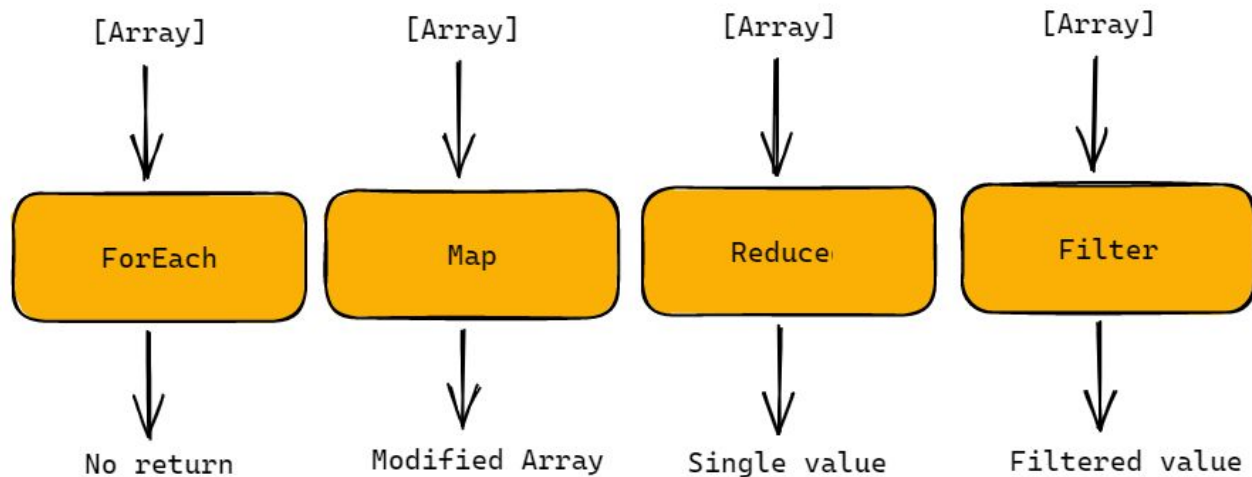
Let's create a greet function which takes name and function as input and returns a greeting.

```
1 // Higher-order function
2 function greet(name, callback) {
3     return callback(name);
4     // The callback function is applied to the name
5 }
6
7 // Simple functions that adds a greeting
8 function sayHello(name) {
9     return `Hello, ${name}!`;
10 }
11
12 function goodMorning(name){
13     return `Good Morning, ${name}`;
14 }
15
16 // Using the higher-order function
17 const greeting1 = greet('Rohan', sayHello);
18 const greeting2 = greet('Sohan', goodMorning);
19
20 console.log(greeting1); // Output: Hello, Rohan!
21 console.log(greeting2); // Output: Good Morning, Sohan
```

This might seem of no use, but when we will read arrays methods like map, filter etc, you will get to know helpful the higher order functions are.

Few popular higher order functions

Here are few most popular higher order functions:-

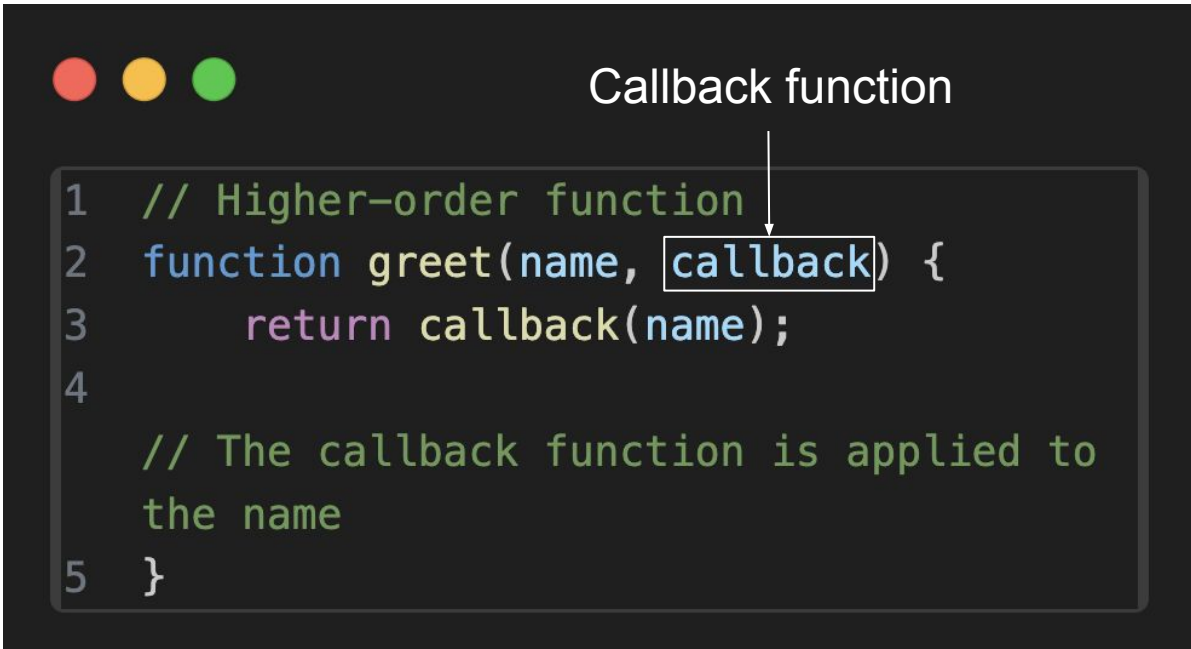


Don't worry we will learn these higher order functions in later lectures.

Understanding Callbacks

A **callback** is a function passed as an argument to another function, which is then executed inside that function. You can name callback function as you wish.

Callback function



```
1 // Higher-order function
2 function greet(name, callback) {
3     return callback(name);
4
5     // The callback function is applied to
6     the name
7 }
8 }
```

Rest and Spread Operator

Rest operator: team player of javascript

You are tasked to form a cricket team. First, pick your favourite duo to open the batting, then select the captain. The rest make up the team.



Rest operator: team player of javascript

Let's write a code for it:-

```
1 function formCricketTeam(  
2     captain, ...team  
3 ) {  
4     console.log("Captain:", captain);  
5     console.log("Team:", team);  
6 }
```

rest operator
grouping rest of
the values in
variable team

Output:

Captain: Virat Kohli
Team: ["Jasprit Bumrah", "Hardik
Pandya", "Ravindra Jadeja",
"Others"]

Rest operator: team player of javascript

Let's pass values to the function:-

```
1 // Example usage:
2 formCricketTeam(
3     // Captain
4     "Virat Kohli",
5
6     // Team
7     "Jasprit Bumrah",
8     "Hardik Pandya",
9     "Ravindra Jadeja",
10    "others"
11 );
```

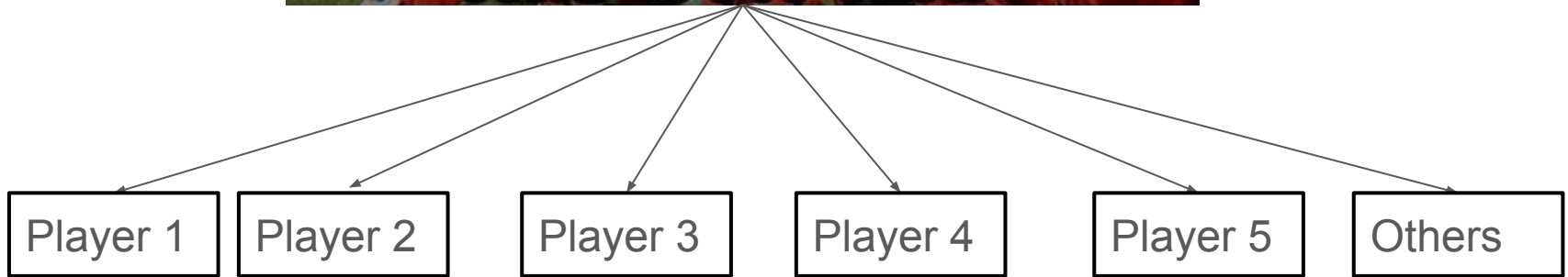
Team members are grouped together in receiving function using rest operator

Spread Operator: The Team Promoter

The spread operator (`...`) takes values from a group (like an array or object) and spreads them out into individual pieces, just like a promoter showing off each team member one by one.



Spread Operator: The Team Promoter



Spread Operator: The Team Promoter

Let's understand it with a code:-

```
1 // Using spread operator
2 const [
3     player1,
4     player2,
5     player3,
6     player4,
7     player5
8 ] = [...rcbTeam];
9
10 console.log('Player 1:', player1);
11 console.log('Player 2:', player2);
12 console.log('Player 3:', player3);
13 console.log('Player 4:', player4);
14 console.log('Player 5:', player5);
```

Here We are using spread operator to spread out individual player names in player 1, player 2, ...etc.



In Class Questions

**Thanks
for
watching!**