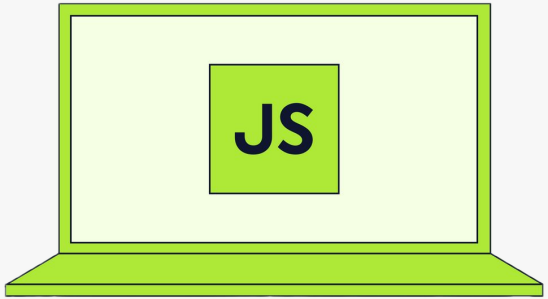
Newton School of Technology

The Complete Javascript Course



@newtonschool

Lecture 4: Functions

-Vishal Sharma

JS

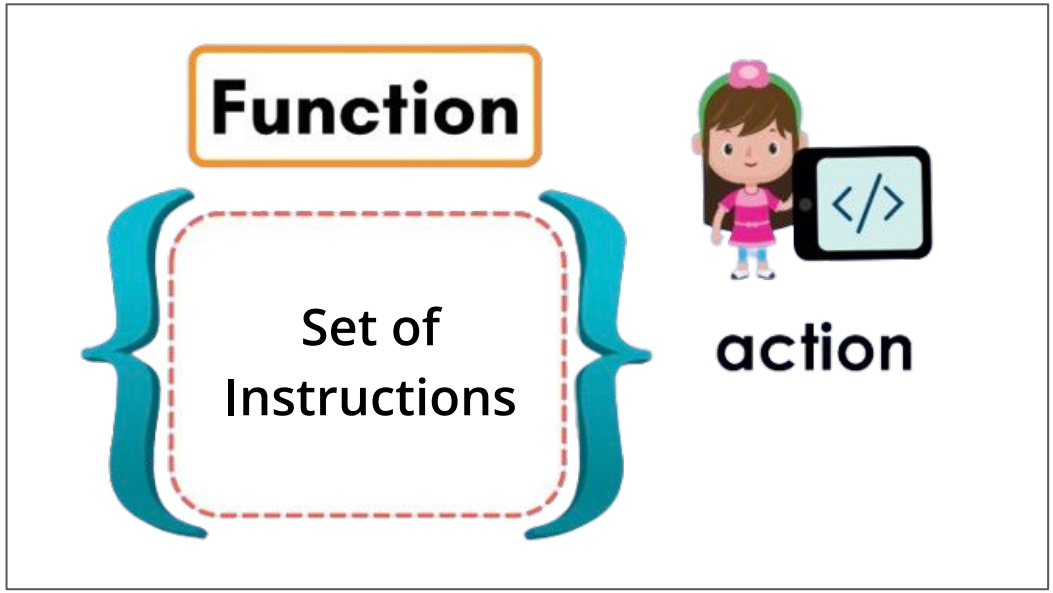
Table of Contents

Newton School of Technology

Function Declaration and Syntax

What is a Function?

A function is a block of code made out of a set of steps that result in a single specific action.



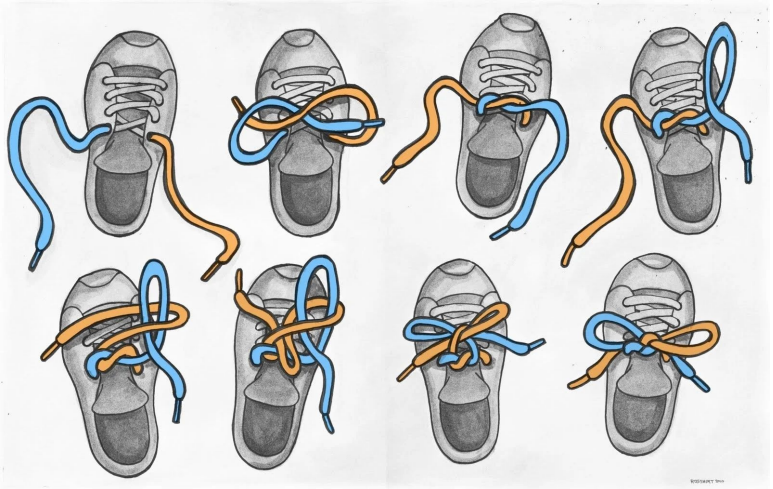
What does it mean??
Let's understand with an analogy

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

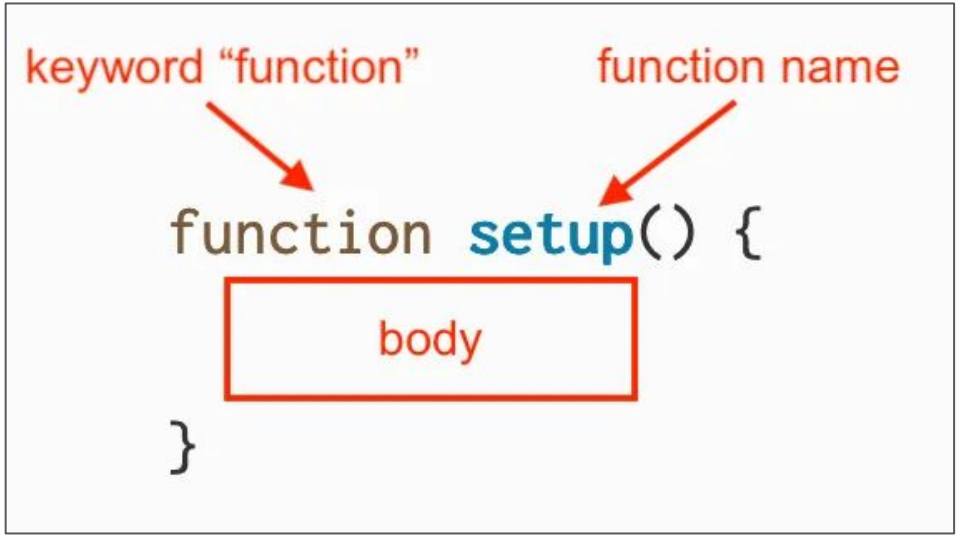
Newton School of Technology

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

```javascript
// Function to tie your shoes
function tieShoes() {
    console.log("Cross the laces.");
    console.log("Make a loop with one lace.");
    console.log("Wrap the other lace around the loop.");
    console.log("Pull through and tighten.");
}

// Reusing the function multiple times
tieShoes(); // First time
tieShoes(); // Second time
```

Naming a function

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



Newton School of Technology

Challenge:

Write a Function to Display Hello World

Were you able to do it??

Newton School of Technology

Let’s start writing:

```
function sayHello() {
    console.log("Hello, World!");
  }

  sayHello();
  // 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

Newton School of Technology

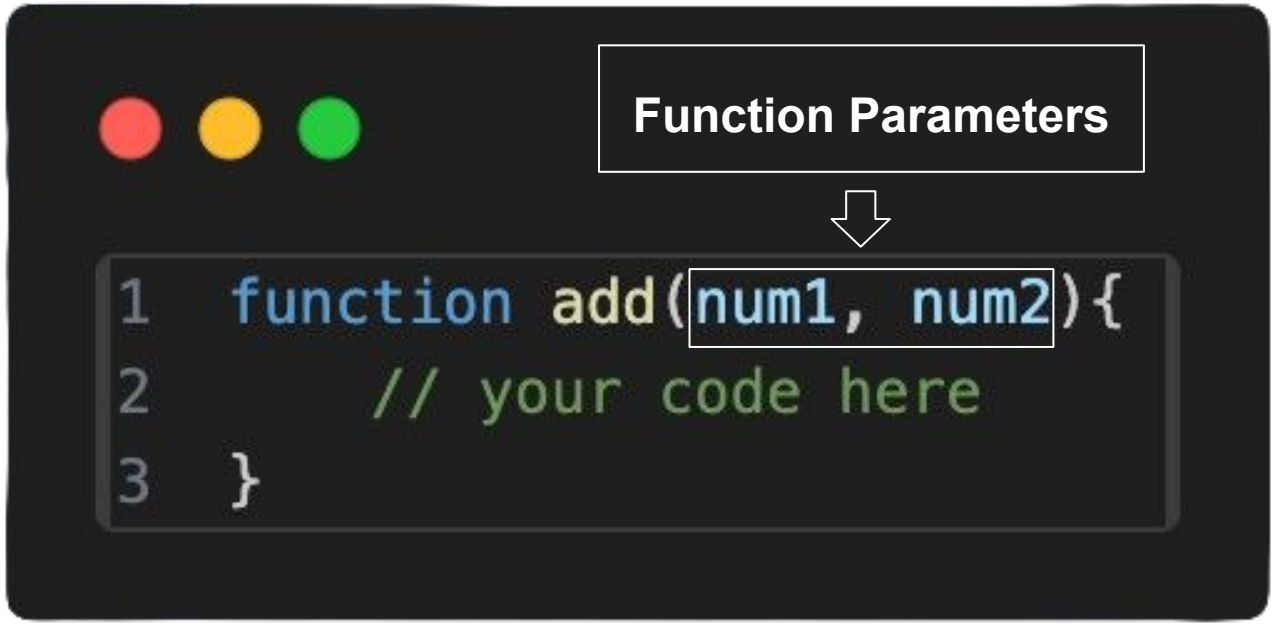
Don’t worry, functions are here for rescue.

```
function add(){
    // your code here
}
```

But how to tell the function what to add??

Function: parameters

We can define parameters in a function to receive values.



Here "*num1*" and "*num2*" are function parameters.

Function: using parameters to add

Newton School of Technology

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

```
function add(num1, num2){
    let result;
    result = num1 + num2;

    // printing the result
    console.log(result);
}
```

Using parameters to perform addition

Function: calling using arguments

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

```
// calling function using arguments
add(4, 5); // Outupt: 9
add(6, 4); // Output: 10
```

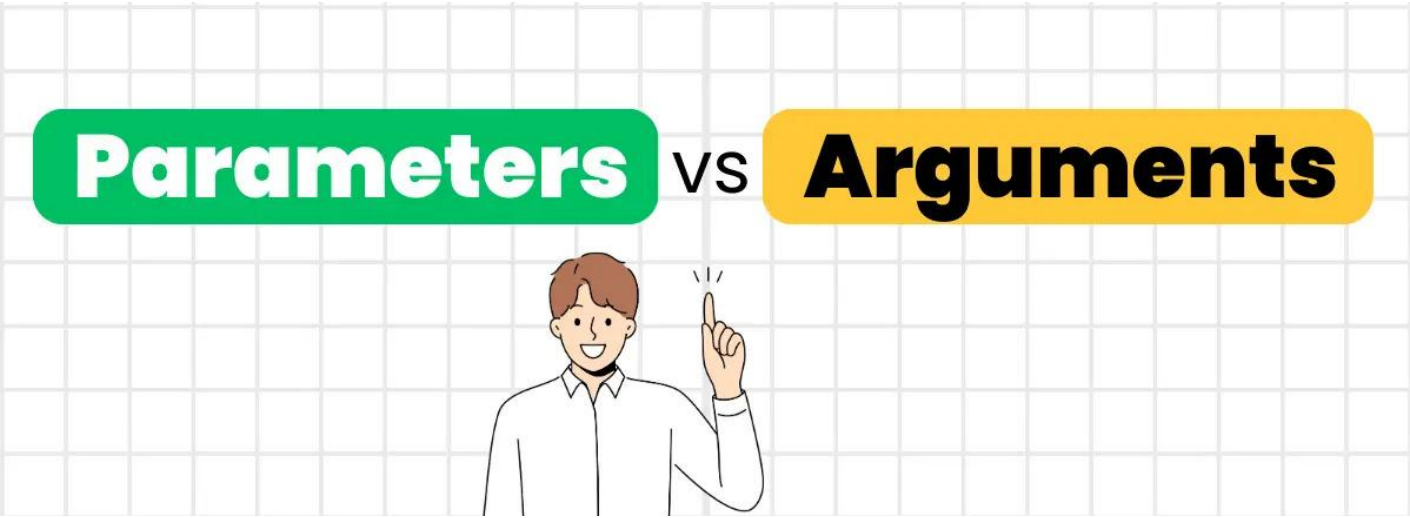
Arguments are the values passed to a function and assigned to its parameters.

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

Newton School of Technology

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

Parameters

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

sum(5, 6);
```

Arguments

Function Return Values

Function: return values

Newton School of Technology

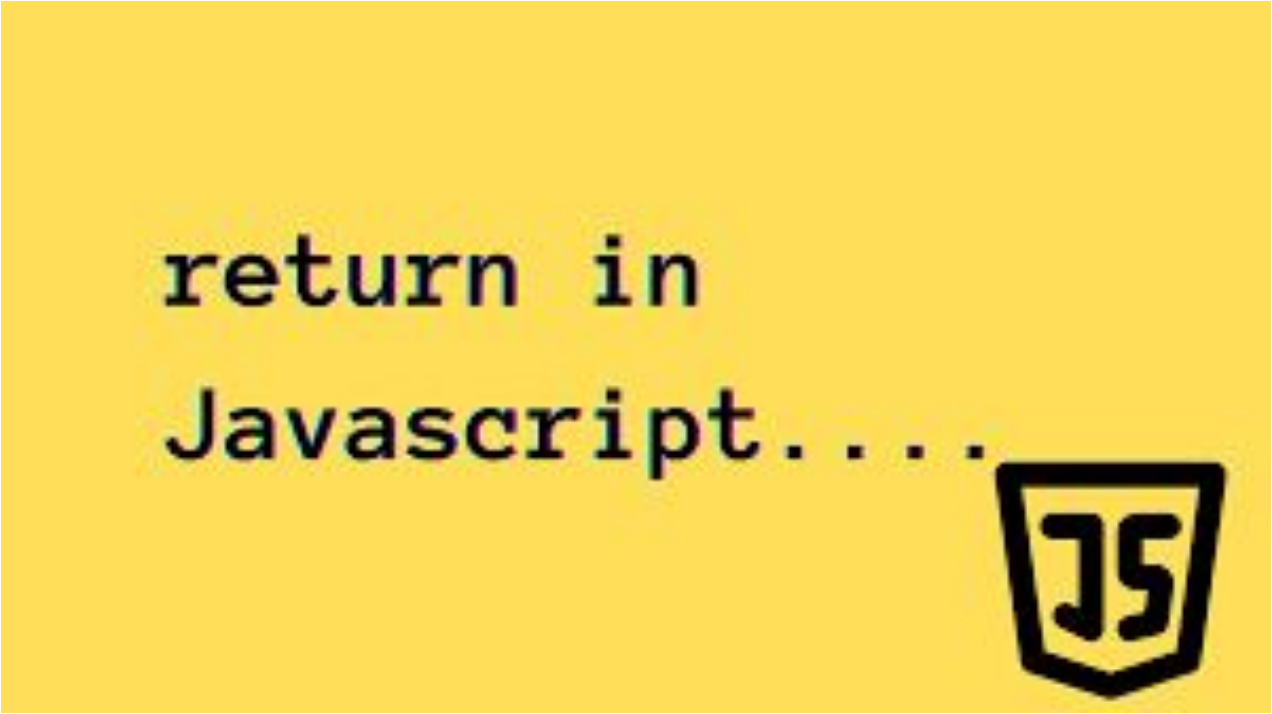
You expect someone to return something whenever you give them a task; this could be the task's status or final result, among other things.

Return Values

Function: return values

Newton School of Technology

Similarly, functions do return values on completion.



Function: return value syntax

Similarly, functions 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 the result value to where the function was called from.

Function: return values with Example

Newton School of Technology

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

```
function add(num1, num2){
    let result;
    result = num1 + num2;

    // printing the result
    console.log(result);
}
```

```
function add(num1, num2){
    let result;
    result = num1 + num2;

    // returning the result
    return result;
}
```

Function: return values with Example

Newton School of Technology

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

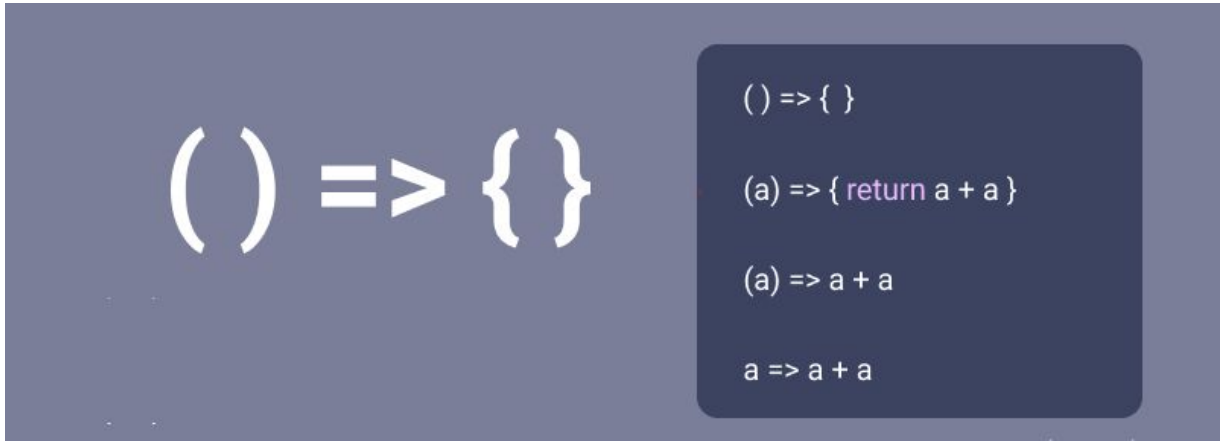
```
function add(num1, num2){
    let result;
    result = num1 + num2;

    // returning the result
    return result;
}

let sum = add(2, 3);
console.log(sum); // Output: 5
```

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

Functions: Arrow Functions

Newton School of Technology

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



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

Functions: Arrow Function example

Newton School of Technology

Let’s look at an example:

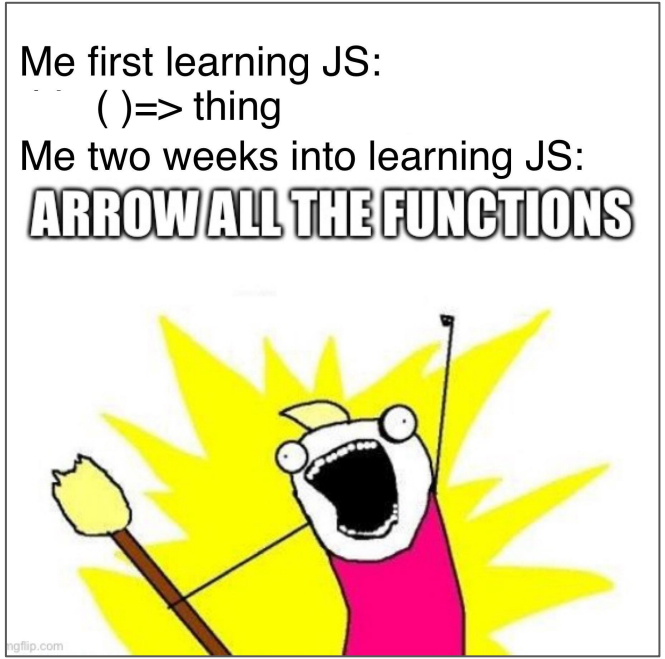
```
// Traditional function
function multiply(a, b) {
    return a * b;
}

// Arrow function
const multiply = (a, b) => a * b;
```

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

Arrow Functions: Shortcut to Simplicity

Newton School of Technology

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



Feeling awesome, right??

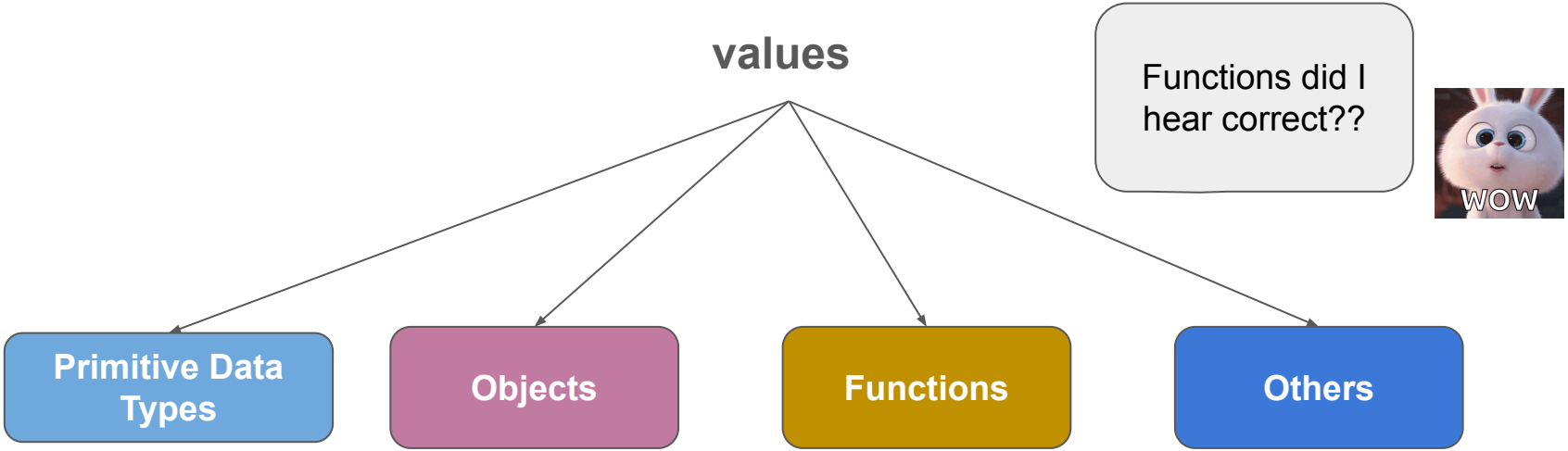
Newton School of Technology

Higher Order Functions and Callbacks

Values we can pass/return in a function

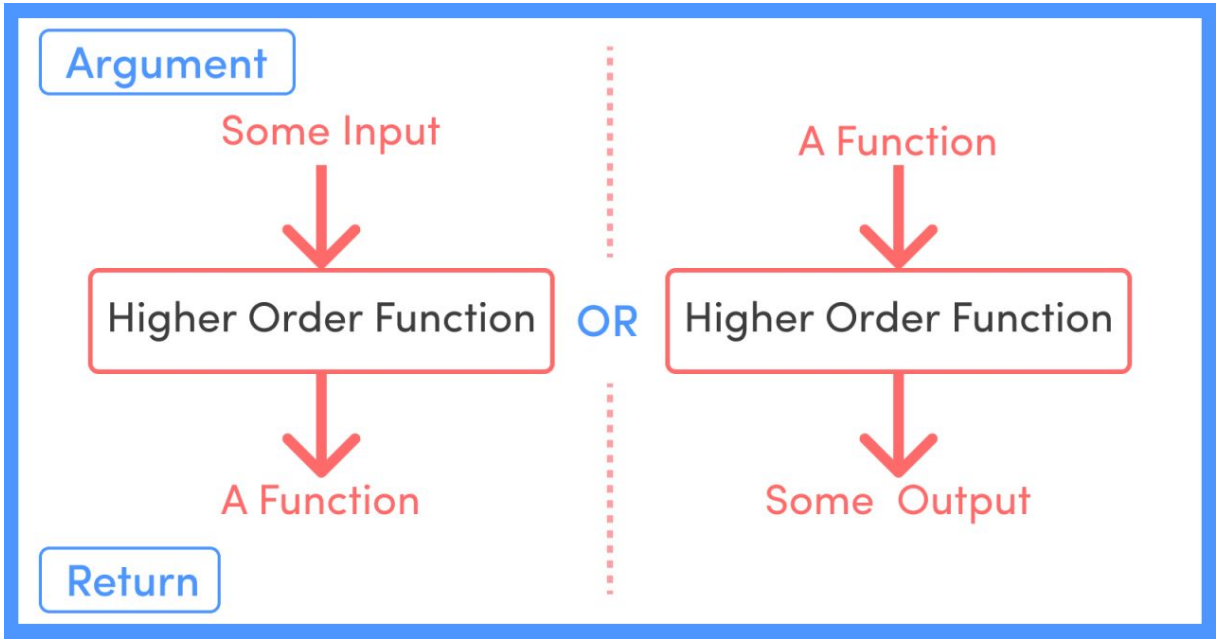
Newton School of Technology

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



Higher-Order Functions

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

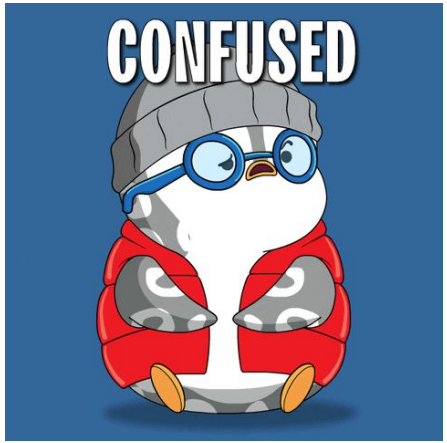


But why do we need function to receive and accept values??

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

Newton School of Technology

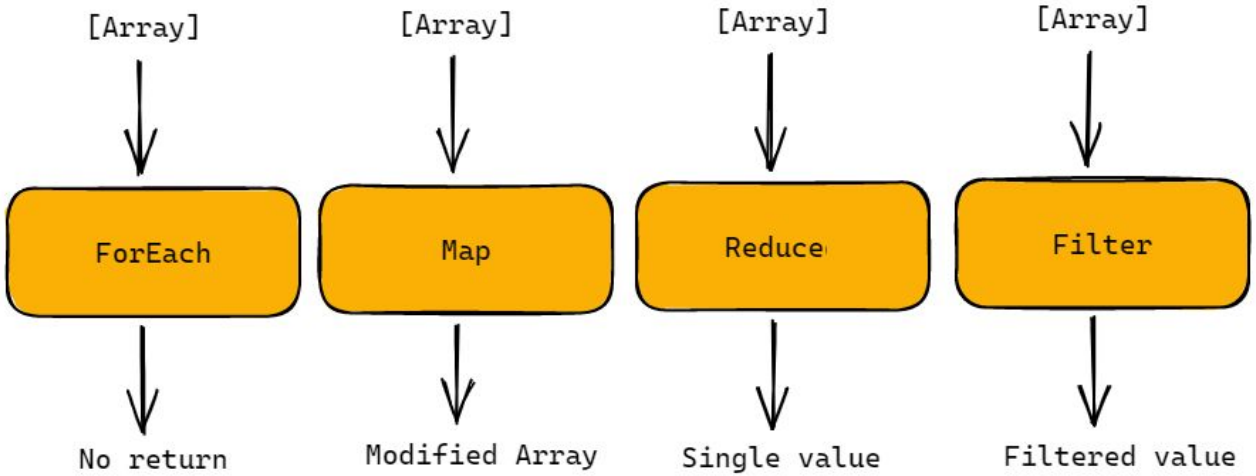
Let’s create a greet function which takes name and function as input and returns a greeting.

```
// Higher-order function
function greet(name, callback) {
    return callback(name);
    // The callback function is applied to the name
}

// Simple functions that adds a greeting
function sayHello(name) {
    return `Hello, ${name}!`;
}

function goodMorning(name){
    return `Good Morning, ${name}`;
}

// Using the higher-order function
const greeting1 = greet('Rohan', sayHello);
const greeting2 = greet('Sohan', goodMorning);

console.log(greeting1); // Output: Hello, Rohan!
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

Newton School of Technology

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

```
// Higher-order function
function greet(name, callback) {
    return callback(name);

// The callback function is applied to the name
}
```

Rest and Spread Operator

Rest operator: team player of javascript

Newton School of Technology

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

Newton School of Technology

Let's write a code for it:

```
function formCricketTeam(
        captain, ...team
) {
    console.log("Captain:", captain);
    console.log("Team:", team);
}
```

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

Newton School of Technology

Let’s pass values to the function:-

```
// Example usage:
formCricketTeam(
    // Captain
    "Virat Kohli",

    // Team
    "Jasprit Bumrah",
    "Hardik Pandya",
    "Ravindra Jadeja",
    "others"
);
```

Team members are grouped together in receiving function using rest operator

Spread Operator: The Team Promoter

Newton School of Technology

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

Player 1 | Player 2 | Player 3 | Player 4 | Player 5 | Others

Spread Operator: The Team Promoter

Newton School of Technology

Let’s understand it with a code:-

```
// Using spread operator
const [
        player1,
        player2,
        player3,
        player4,
        player5
] = [...rcbTeam];

console.log('Player 1:', player1);
console.log('Player 2:', player2);
console.log('Player 3:', player3);
console.log('Player 4:', player4);
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

Newton School of Technology

Thanks for watching!