FULL STACK

Understanding the World of JavaScript

simplilearn

You Already Know

Course(s):

Full JavaScript Masterclass Course: ES6 Modern Development

Recap

- Setup JavaScript on Windows, Mac OS X, and Linux
  - Setup on Windows
  - Setup on Mac OS X
  - Setup on Linux

- Explain the basic features of javaScript programming language
  - Comments
  - Variables
  - Functions
  - Operators and Loops

Recap

- Describe the objects used in JavaScript
  - Numbers
  - Boolean
  - String
  - Date
  - Array
  - Math

- Demonstrate how to create classes in JavaScript
  - Class inheritance
  - Method overriding
  - Super keyword

A Day in the Life of a Full Stack Developer

Joe had performed remarkably in the last sprint. Based on his expertise, the company has asked Joe to develop an expense tracker for an e-commerce company.

In this sprint, he has to develop a website where the program managers of a specific team will add the details of the professional deals they want to make with the vendors. The finance team will check the expenses of those teams and will decide their annual budget.

In this lesson, we will learn how to solve this real-world scenario to help Joe complete his task effectively and quickly.

Learning Objectives

By the end of this lesson, you will be able to:

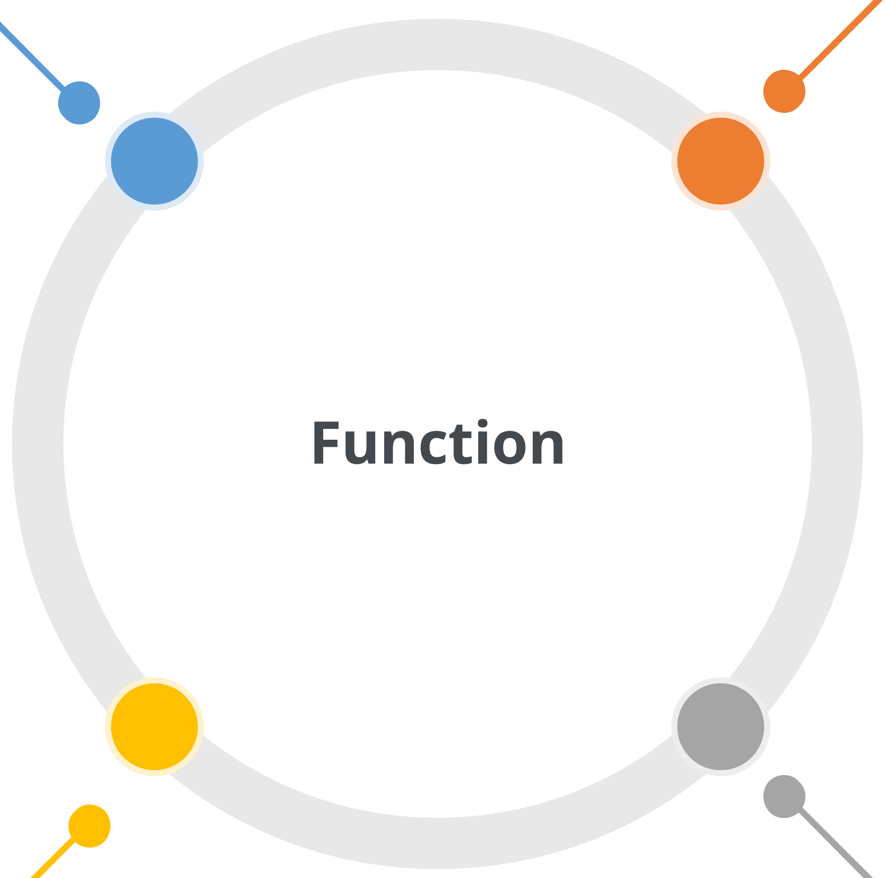
- Create prototypes of functions
- Demonstrate working with functions
- Work with IIFEs, callbacks, closures, and functions
- Explain maps and classes

FULL STACK

Functions and Prototyping

simplilearn

Functions



It can be predefined or user defined

It is an object and a subprogram designed to perform a specific task

It gets executed when called and always returns a value

Keyword is used to create a function. Function name can contain letters, digits, underscores, and dollar signs

Function Constructors

When any function is called with a new keyword, JavaScript:

- Creates a new empty anonymous object
- Uses that object within the call
- Implicitly returns the new object at the end of the call

In JavaScript, any function can be called a constructor

All the global classes such as number or string are functions acting as constructors containing useful properties

By convention, the constructor name is in uppercase

Prototype

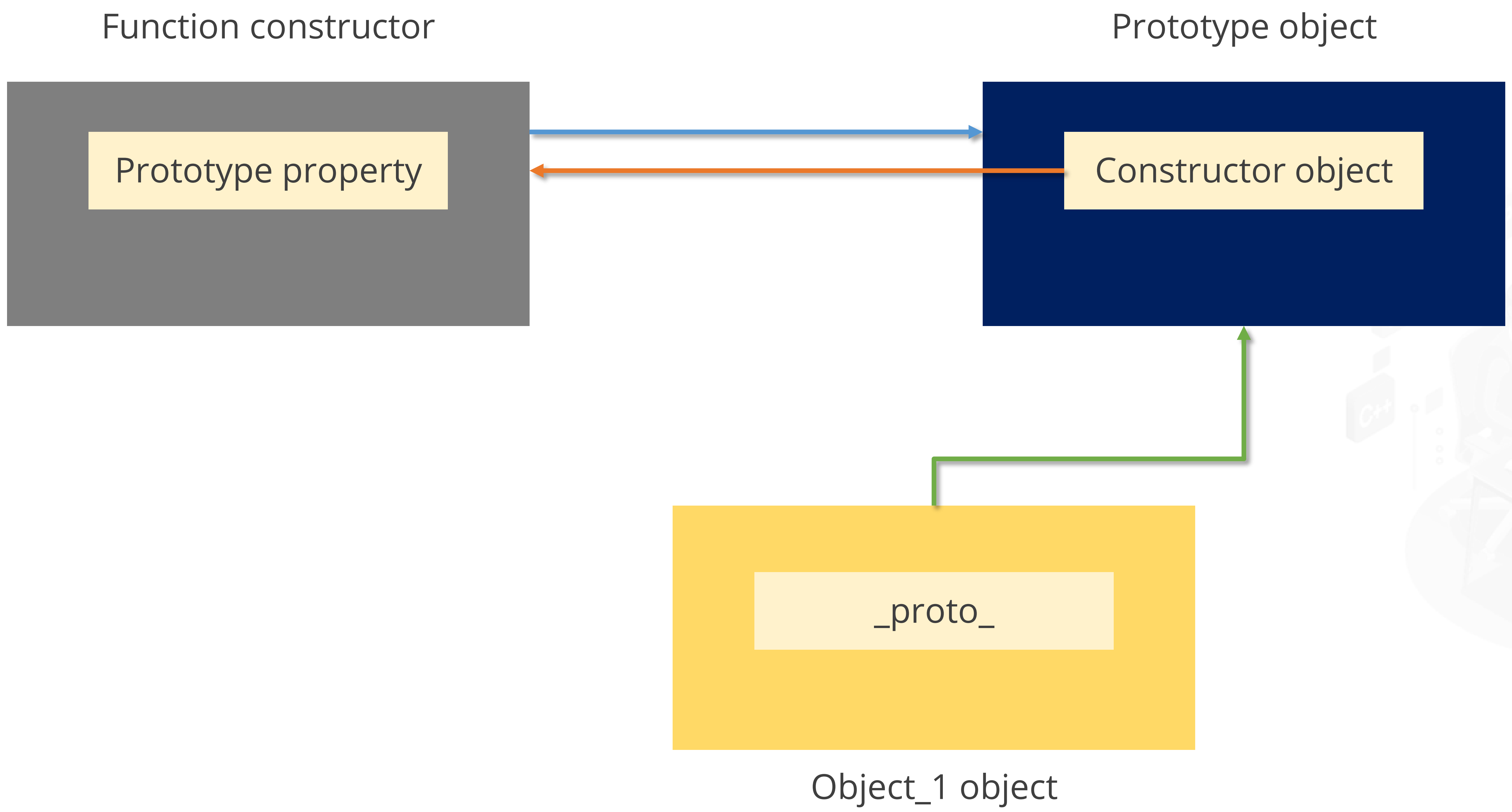
Prototype

All JavaScript objects inherit properties and methods from a prototype

Function’s prototype property is modifiable, but object’s prototype property is not visible

Every function includes prototype object by default

Prototype Chaining



Prototype: Properties and Methods

Properties

- prototype.constructor
- prototype._proto_

Methods

- .prototype.hasOwnProperty()
- prototype.isPrototypeOf()
- prootype.toLocalString()
- prototype.toString()
- prototype.valueOf

Dot notation (.) provides access to an object's properties

Functions and Prototyping

Duration: 20 min.

Problem Statement:

You are given a project to demonstrate the use of functions and prototypes in JavaScript.

Assisted Practice: Guidelines

Steps to demonstrate function prototype:

1. Create a JavaScript project in your IDE

2. Write a program in JavaScript using prototypes of functions to display the employee information of an organization

3. Initialize the .git file

4. Add and commit the program files

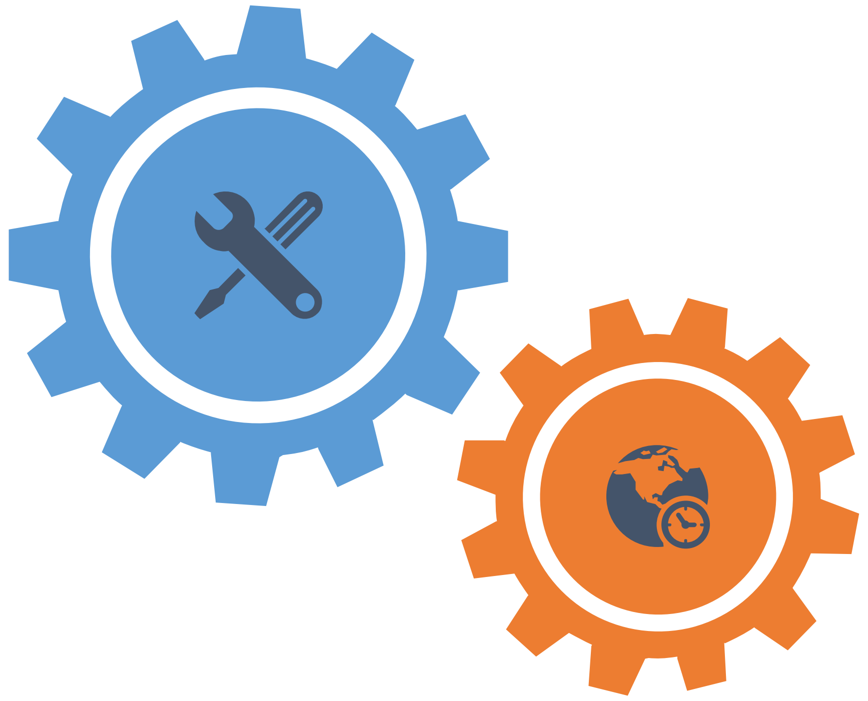
5. Push the code to your GitHub repositories

FULL STACK

Working with Functions

Why Use Functions?

Code reusability:
You can define the code once and can use it multiple times



Different results:
You can use the same code multiple times with different arguments and can get different results

Function Execution Steps

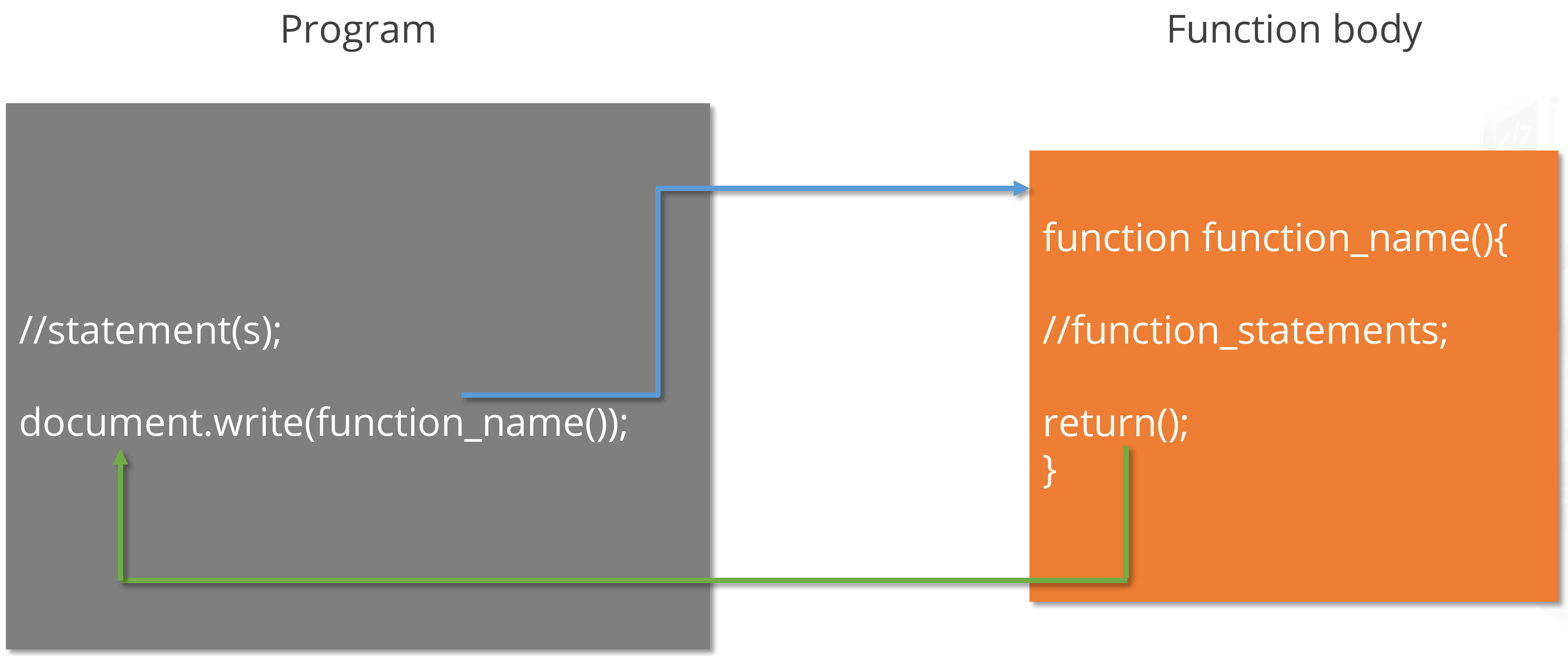
1 Function Definition

- The function definition is sometimes mentioned as function declaration or function statement
- Every function definition should begin with function keyword. User-defined function name should be unique
- Function parameters are enclosed within parentheses, separated by commas
- The function body is enclosed within curly braces {}

2 Function Calling

- Calling a function: You call a function by using the *function name,* separated by the value of parameters enclosed between parenthesis and a semicolon at the end
- Returning value to the function: There are many cases when you need to return a value from the function after performing a few operations. In these cases, ***return*** statements are used

Function Execution

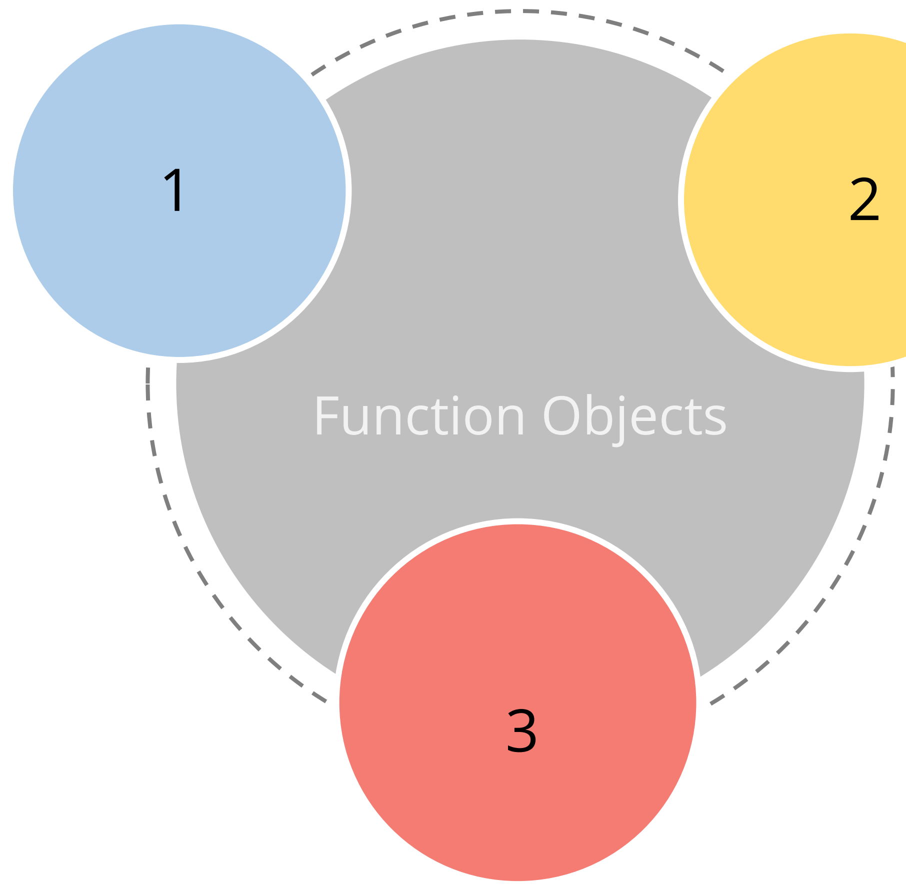


```
//statement(s);

document.write(function_name());
```

```
function function_name(){

//function_statements;

return();
}
```

Function Objects

Function constructors are used to create function objects



They globally execute the code

Syntax:
new Function ([arg1[, arg2[, ...argn]],] function_body)

Passing Functions as Arguments

```
function functionOne(x) { alert(x); }

function functionTwo(var1, callback) { callback(var1); }

functionTwo(2, functionOne);
```

- Functions can be variables in JavaScript. So, you can pass a function as an argument to the other function
- The function passed in can also be called a *Callback* function
- In the example provided, function One takes in an argument and issues an alert with **x** as its argument. Function Two takes in an argument and a function and then passes the argument to the function. Function One is the callback function in this case

Function Returning Function

```
function sqr()  {
        return function cal(x)  {return x * x; }
}

function functionTwo(var1, callback) { callback(var1); }

var ans=sqr();
ans(5);
```

- Return statement passes information from inside a function back to the point in the main program where the function was called
- Returning a function is useful when you are using a prototype-based object model
- We can return a sub-function to main function as shown in the example

Working with Functions

Duration: 15 min.

Problem Statement:

You are given a project to demonstrate how to work with functions.

Assisted Practice: Guidelines

Steps to work with functions:

1. Create a JavaScript project in your IDE

2. Write a program in JavaScript to demonstrate how a function works, how to pass a function as an argument to the other function, and how to return a function to a function

3. Initialize the .git file

4. Add and commit the program files

5. Push the code to your GitHub repositories

IIFEs, Callbacks, and Closures

IIFEs (Immediately Invoked Function Expressions)

Syntax:

```
( function() {
/*  */
} ) ();
```

IIFE is a way to execute functions as soon as they are created

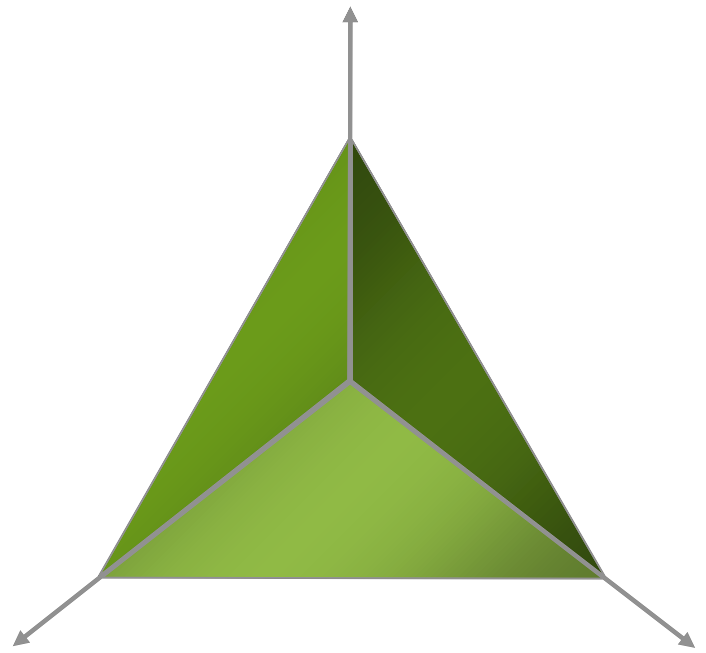
IIFE

It is also known as a regular function

IIFE is a simple way to isolate variable declarations and is used to achieve data privacy

Callback Function

It is to be executed after another function has finished executing



JavaScript will keep executing other events while listening a response from a particular event which takes more time

It is used while handling an asynchronous operation

Closures

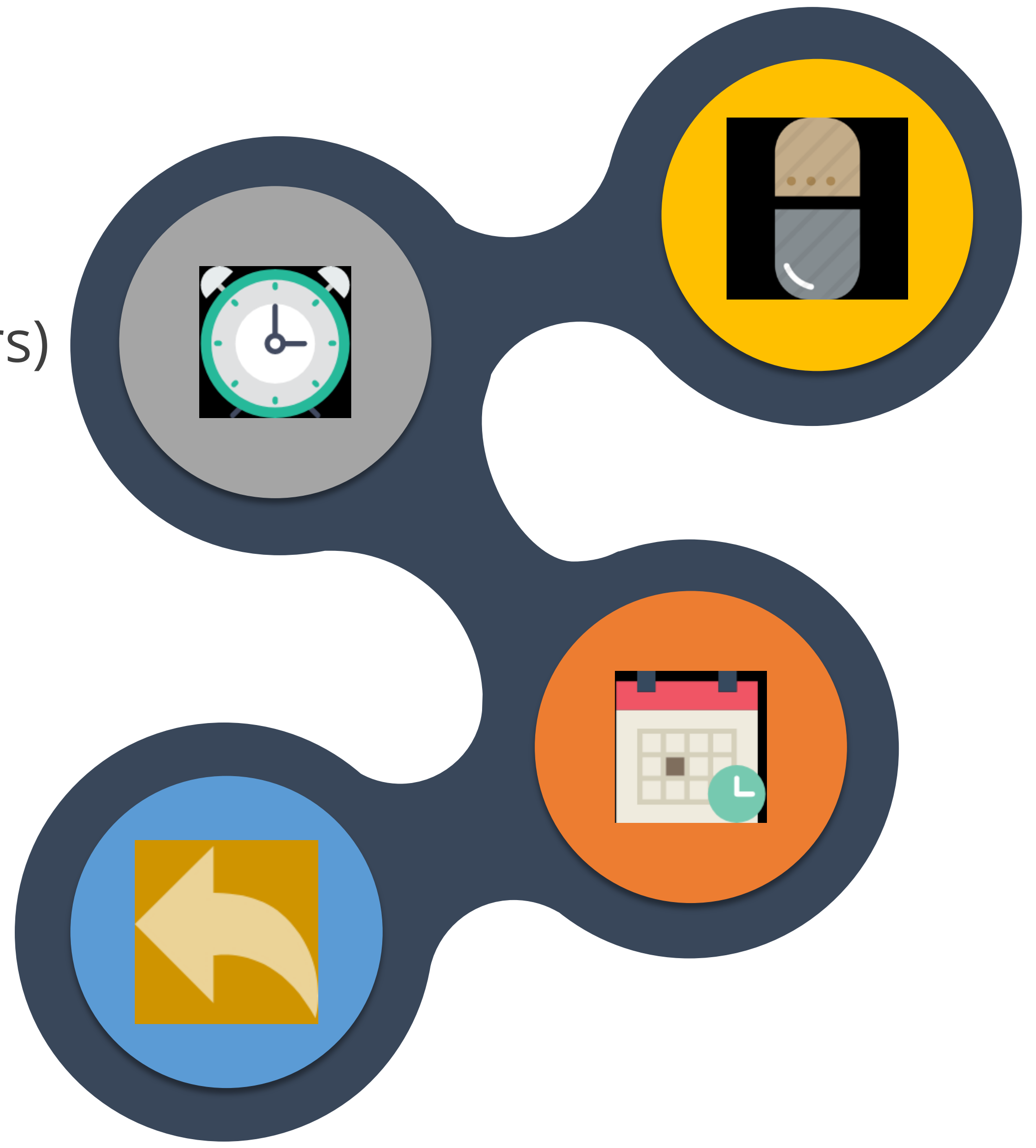
Practically, any function can be considered a closure. A function can refer or have access to:

- Variables and parameters in their own function scope
- Variables and parameters of outer (Parent) functions
- Variables from the global scope

Global variables can be made local with closures

Closures carry the scope with them at the time of their invocation

Uses of Closures



Bind(), Call(), and Apply()

Method Name	Description
bind()	Used to create a new function
call()	Used to call a function that contains *this* value and an argument list
apply()	Used to call a function that contains *this* value and a single array of arguments

IIFEs, Callbacks, and Closures

Duration: 15 min.

Problem Statement:

You are given a project to demonstrate the functionality of IIFEs, callbacks, and closures in JavaScript.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate IIFEs, Callbacks, and Closures:

1. Create a JavaScript project in your IDE

2. Write a program in JavaScript to demonstrate how IIFEs, closures, and callbacks can be used to allot specific employee IDs to the employees of an organization

3. Initialize the .git file

4. Add and commit the program files

5. Push the code to your GitHub repositories

FULL STACK

IIFEs and Functions

Use of let and const

let

- let is used when you need to reassign a variable
- It declares a local variable in a block scope
- let can be used for loops or mathematical operations

const

- const means that the identifier cannot be reassigned
- The scope of const statement is similar to the scope of let statement

Blocks

- A *block statement* is a group of zero or more statements
- Identifiers declared with *let* and *const* do have block scope

let

```
let x=1;
{
let x=5;
}

console.log(x); //answer will be 1
```

const

```
const y=20;
{
const y=40;
}

console.log(y); //answer will be 20
```

Block Scope

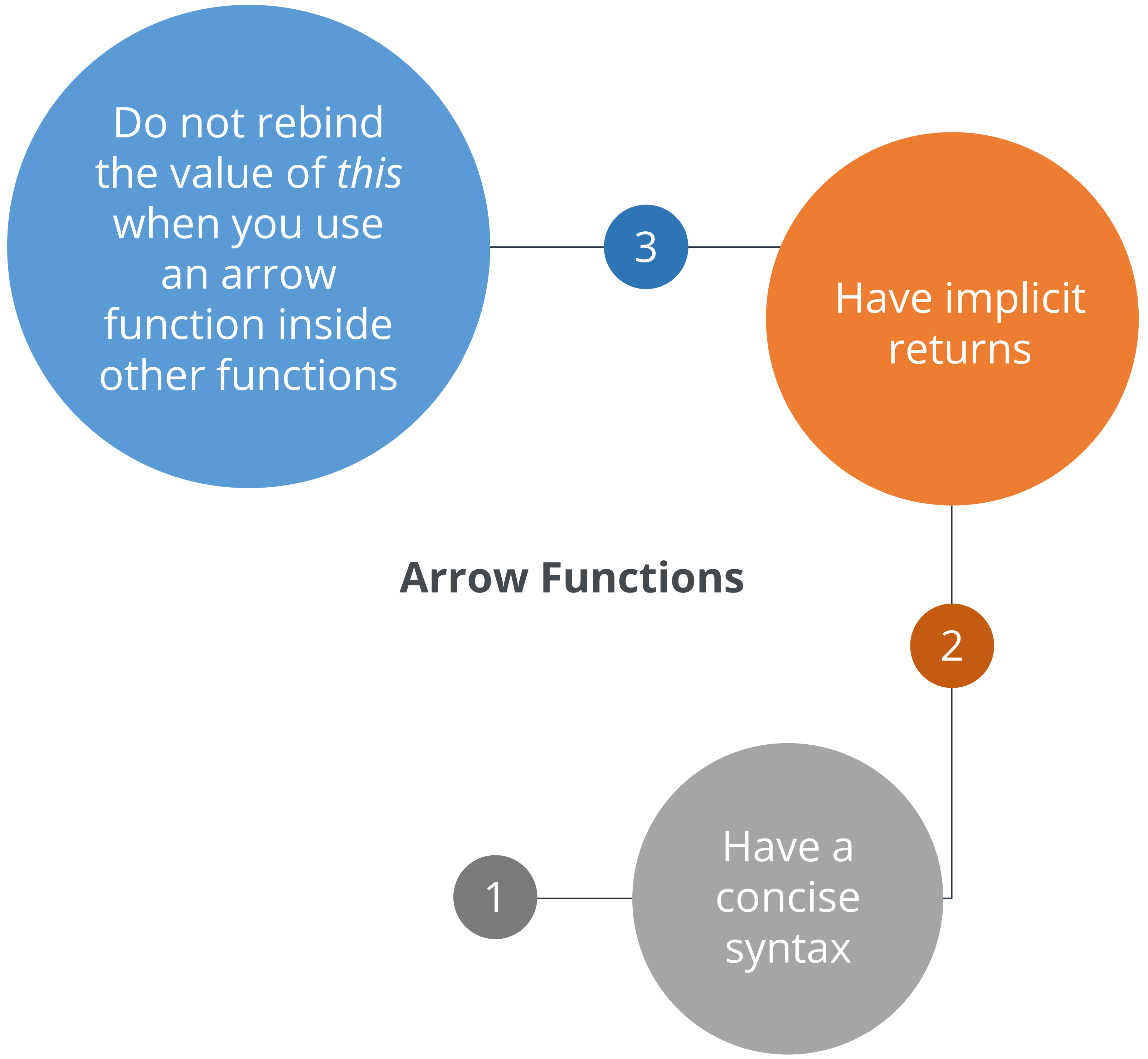
```
function display(){
    if(true){
        var item1 = 'apple';        //exist in function scope
        const item2 = ‘ball';       //exist in block scope
        let item3 = ‘cloud';        //exist in block scope

    }
    console.log(item1);
    console.log(item2);
    console.log(item3);

}

display();
//result:
//apple
//error: item2 is not defined
//error: item3 is not defined
```

String Functions

- A string is a series of characters enclosed within single or double quotes
- String indexes are zero based

Method	Description
charAt()	It returns the character at the specified index
endsWith()	It checks whether a string ends with specified string or characters
includes()	It checks whether a string contains the specified string or characters
slice()	It extracts a part of a string and returns a new string
split()	It splits a string into an array of substrings
subString()	It extracts the characters from a string between two specified indices
toLowerCase()	It converts a string to lowercase letters
toString()	It returns the value of a String object
valueOf()	It returns the primitive value of a String object

Arrow Functions


Do not rebind the value of *this* when you use an arrow function inside other functions
3
Have implicit returns
Arrow Functions
2
1
Have a concise syntax

Example:

```
const welcome = () => 'Hello World'

welcome() //'Hello World'
```

IIFE with Arrow Functions

```
let x;

(x = () => {

console.log(“ This is the example of Arrow function”);
 }) ();

// Output will be This is the example of arrow function
```

IIFEs and Functions

Duration: 20 min.

Problem Statement:

You are given a project to demonstrate the use of blocks, string functions, and arrow functions.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate IIFEs and Functions:

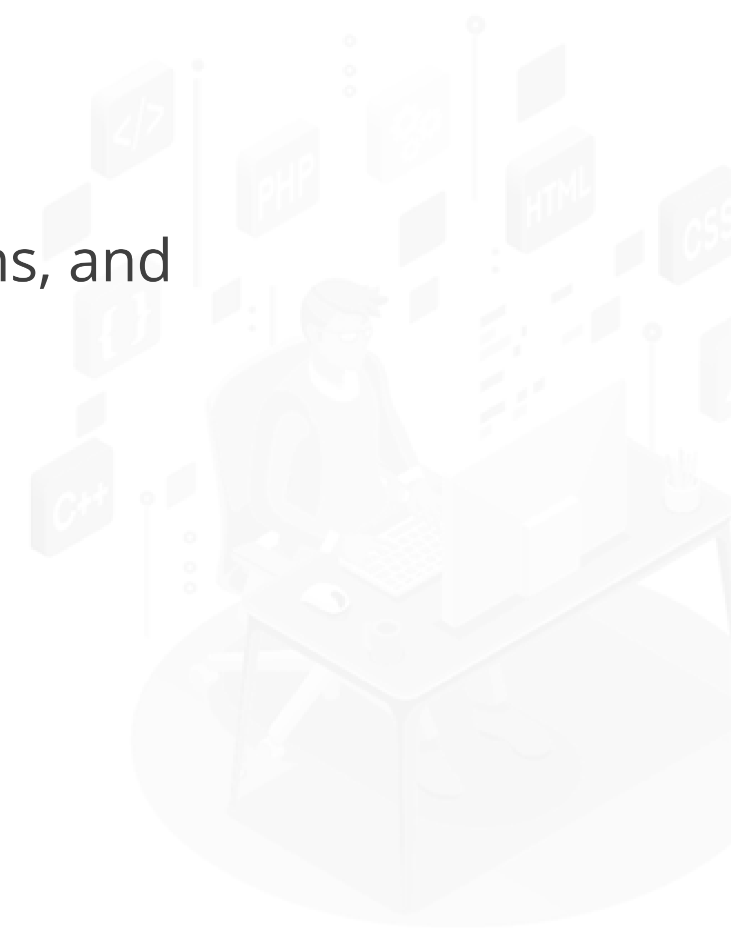
1. Create a JavaScript project in your IDE

2. Write a program in JavaScript to develop a calculator using blocks, string functions, and arrow functions

3. Initialize the .git file

4. Add and commit the program files

5. Push the code to your GitHub repositories



Maps and Classes

Arrays

An array is a collection of elements stored at contiguous memory locations. It is index based. The first element refers to index 0.

Array Declaration

```
let a= new Array(“Orange”, “Apple”, “Banana”, “Grapes”, “Mango”);
```

or

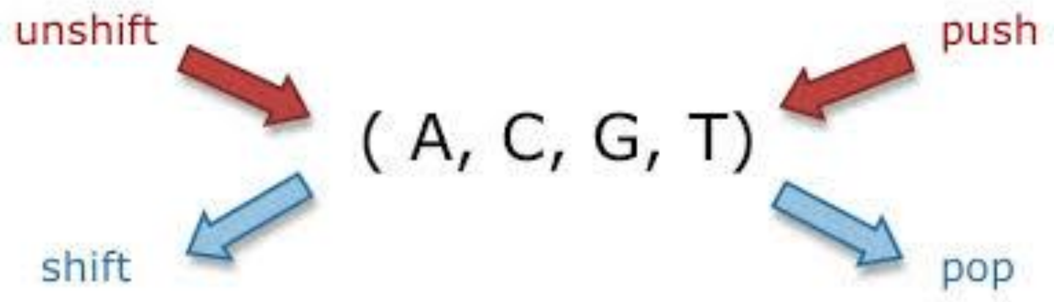
```
let a=[ ];
```

a[0]	a[1]	a[2]	a[3]	a[4]

Arrays Operations

Array Operations

- **Popping**: The *pop()* method is used to remove the last element from an array
- **Pushing**: The *push()* method is used to add a new element to an array
- **Shifting**: The *shift()* method removes the first array element and shifts all other elements to lower index
- **Unshifting**: The *unshift()* method adds a new element to an array and unshifts all other (older) elements



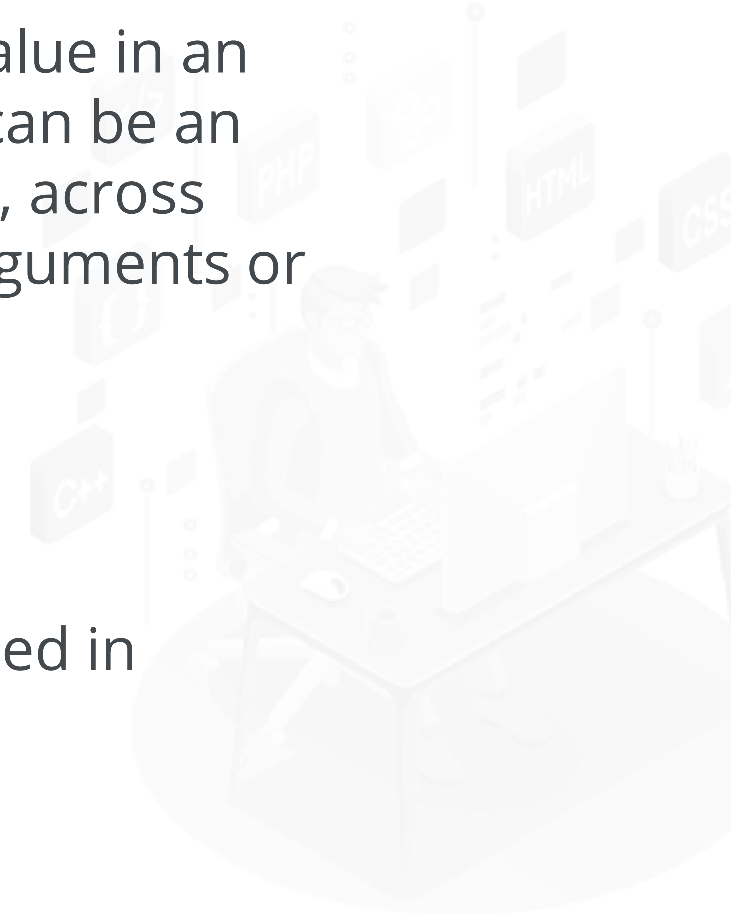
Spread Operators

Syntax:
var variable_name=[...value];

It spreads the value in an iterable, which can be an array or a string, across zero or more arguments or elements.

It is mostly used in a variable array when more than one value is expected there.

It can also be used in function calls.



Rest and Default Parameters

Rest Parameters

- Rest parameters allow us to work in a clean and easy way with an indefinite number of parameters
- They are indicated by three dots (**…**) preceding a parameter
- They should be at the end

Default Parameters

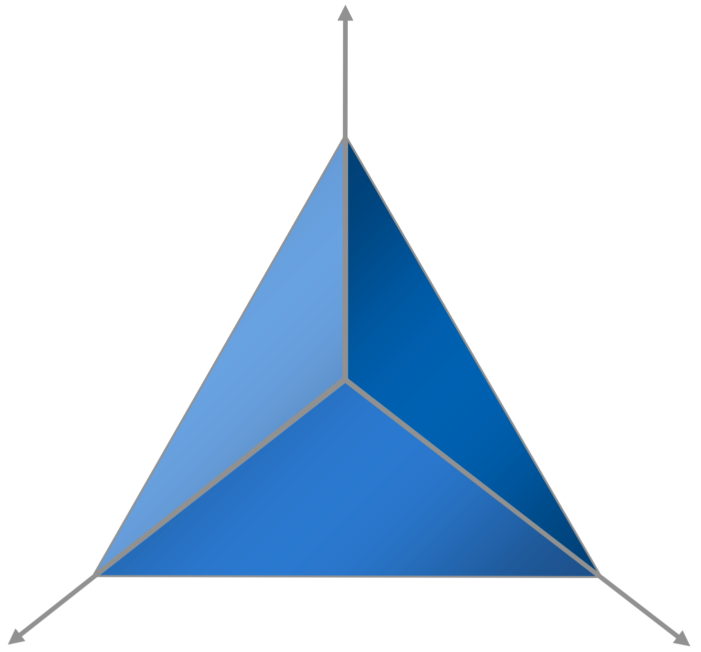
- Any parameter with a default value is considered to be optional
- Default values can be set to parameters that appear before arguments without default values

Map in JavaScript

A map is a collection of elements in which each element is stored in a key-value pair

Syntax:

new map ([iterable]);



A map object iterates its elements in an insertion order that returns an array of [key, value] for each iteration

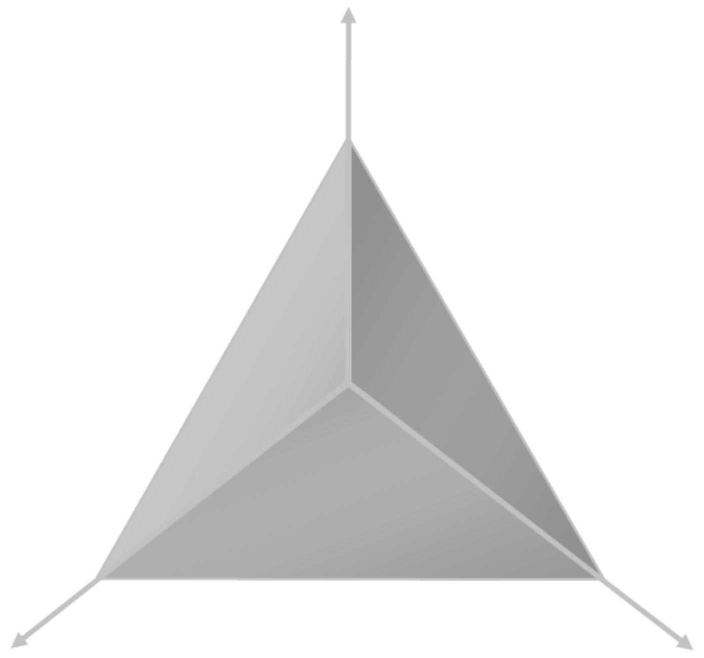
A map can hold both objects and primitive values as either keys or values

Map: Methods

Method	Description
Map.prototype.set()	Adds key and value to a map object
Map.prototype.has()	Returns a boolean value depending on presence of the specified key
Map.prototype.get()	Returns the value of the corresponding key
Map.prototype.delete()	Deletes both the key and the value from the map
Map.prototype.clear()	Removes all elements from the map object
Map.prototype.entries()	Returns an iterator object that contains a key-value pair for each element present in the map object
Map.prototype.keys()	Returns an iterator object which contains all the keys present in the map object
Map.prototype.values()	Returns an iterator object which contains all the values present in the map object
Map.prototype.forEach()	Executes callback function once for each key-value pair in the map in an insertion order
Map.prototype[@@iterator]()	Returns a map iterator function which is the entries() method of map object by default

Classes in JavaScript

JavaScript classes are different than Java classes. Classes are special functions, just like function expressions and function declarations



Classes do not allow property value assignments like constructor functions or object literals

Class syntax has two components: class expressions and class declarations

Features of Classes

Subclassing:
This is the way you can implement inheritance in JavaScript

Constructor:
It is a special function in class declaration and defines a function that represents that class

Getter and Setter:
Getter and setter are used to get and set the property value

Static methods:
These are functions of classes and not of their prototypes. These methods are declared using the *static* keyword

Maps and Classes

Duration: 15 min.

Problem Statement:

You are given a project to demonstrate how to use maps and classes in JavaScript.

Assisted Practice: Guidelines

Steps to demonstrate maps and classes:

1. Create a JavaScript project in your IDE

2. Write a program in JavaScript to work with maps and classes

3. Initialize the .git file

4. Add and commit the program files

5. Push the code to your GitHub repositories

Key Takeaways

- Function's prototype property is modifiable, but object's prototype property is not visible.
- A map is a collection of elements in which each element is stored in a key-value pair.
- JavaScript classes are different than Java classes. Classes are special functions, just like function expressions and function declarations.

Team Budget Planner

Duration: 70 min.

Problem Statement:

You need to develop a website where program managers of a specific team will add details of professional deals they want to make with vendors. The finance team will check expenses of those teams and will decide their annual budget.

Before the Next Class

Course(s):

- An Introduction to TypeScript
- Angular Training Course
- MongoDB Developer and Administrator Certification Training

Before the Next Class

You should be able to:

- Setup your environment for TypeScript
- Demonstrate Primitive and Non-Primitive data types

  Explain object-oriented TypeScript functionality
- Describe Generics
- Build Angular components
- Understand Bootstrap
- Explain binding and events
- Understand the basics of MongoDB
- Perform CRUD operations
- Explain indexing and aggregation
- Explain replication and sharding