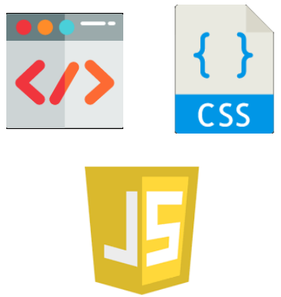
Understanding the World of JavaScript

simplilearn

You Already Know

Course(s):

A Front-End Web Developer MasterClass Using HTML, CSS, and JavaScript



Recap

- Get introduced to web development
  - Web development introduction
  - Front-end web development

- Explain the fundamentals of HTML and HTML tags
  - HTML boilerplate
  - HTML tags

- Explain the fundamentals of CSS
  - Advanced CSS
  - CSS pseudo classes

Recap

- Build a web page layout using CSS
  - Sidemenu links hover CSS
  - CSS website tweak

- Explain JavaScript coding essentials
  - Prompt and alert JavaScript
  - JavaScript DOM
  - Document query selectors
  - Event listeners click events

A Day in the Life of a MEAN Stack Developer

Joe has 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 have 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 primitives and objects
- Work with IIFEs, callbacks, closures, and functions
- Explain maps and classes
- Explain Fetch and Promises
- Explain the importance of Babel

Functions and Prototyping

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:

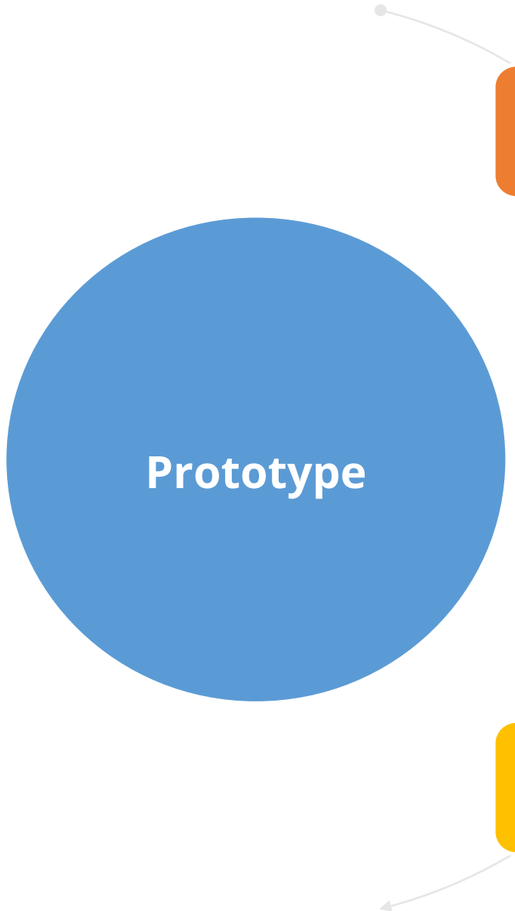
- Creates a new empty anonymous object
- Uses that object within the call
- Implicitly returns the new object at the end of the call

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

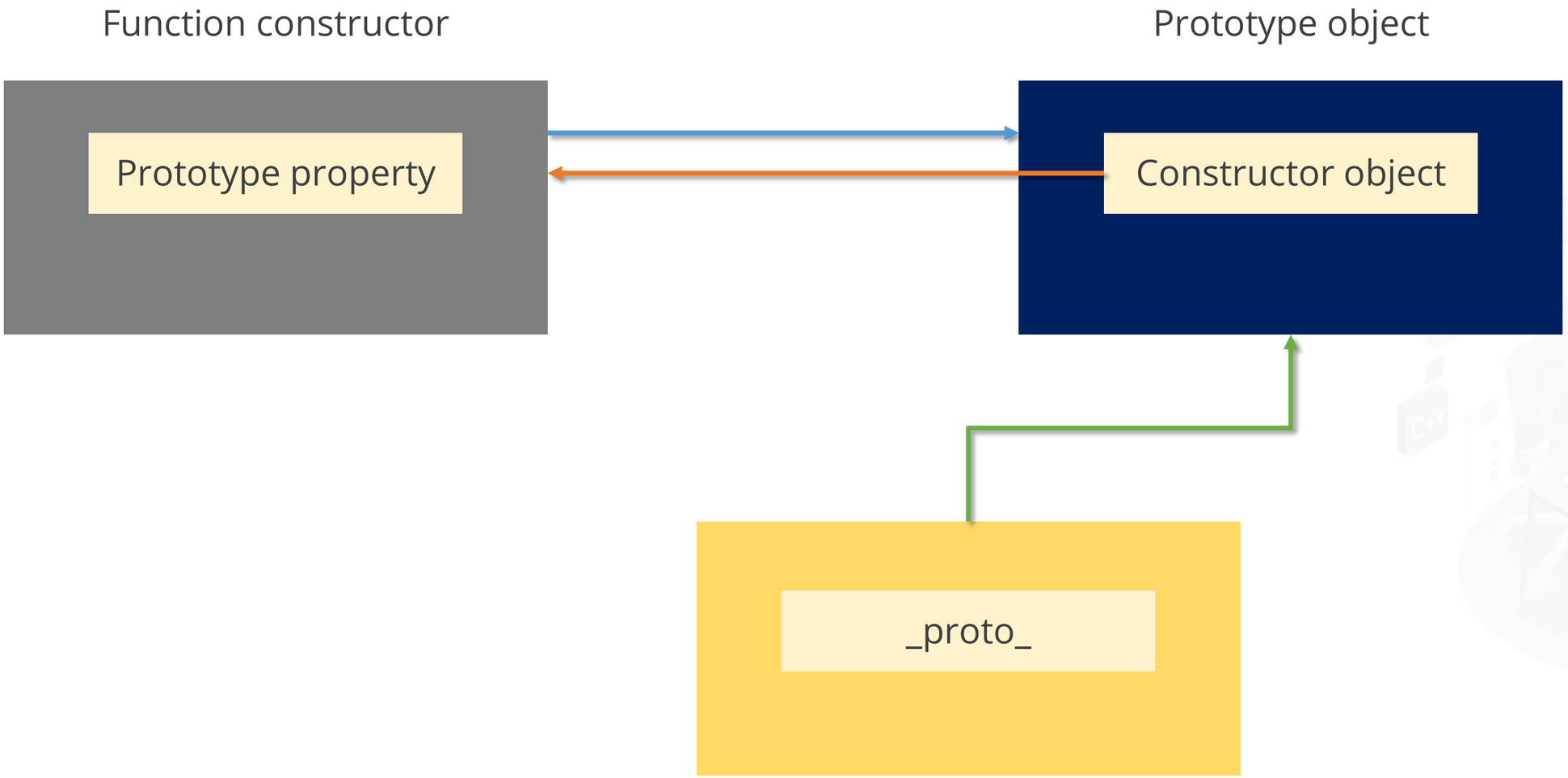


All JavaScript objects inherit properties and methods from a prototype

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

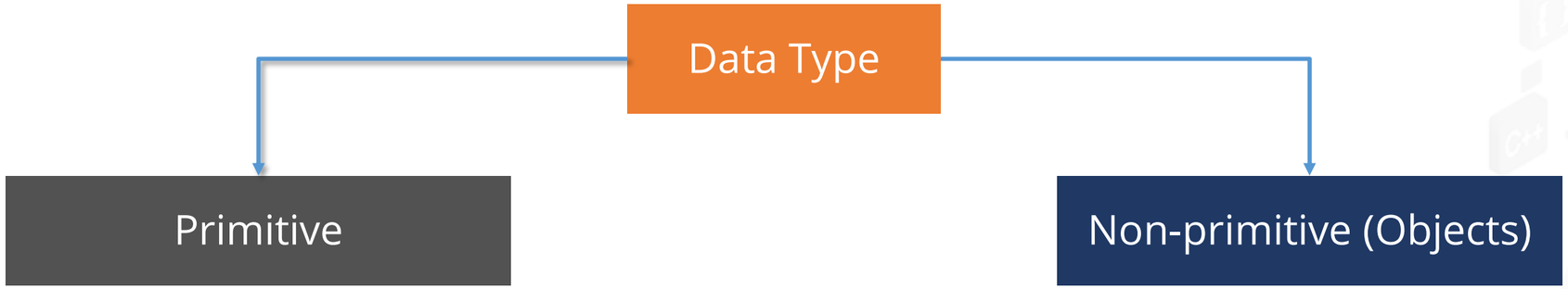
Assisted Practice: Guidelines to Demonstrate Function Prototype

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 repository.

Primitives and Objects

Data Types in JavaScript

A data type is a classification that specifies which type of value a variable has and what type of mathematical, relational, or logical operations can be applied to it without any error.



Primitive Data Type

A primitive is a data that is not an object and has no methods.

1 **Numbers**

2 **Strings**

“Hello World”

3 **Boolean**

4 **Null**

{ }

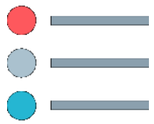
5 **Undefined**

Non-Primitive Data Type

Non-primitive data types (Objects) are not defined by the programming language but can be created by the programmer.

1 Objects

2 Arrays



3 Functions

4 Date

5 Regex

Primitives and Objects

Duration: 20 min.

Problem Statement:

You are given a project to demonstrate the use of primitives and objects in JavaScript.

ASSISTED PRACTICE

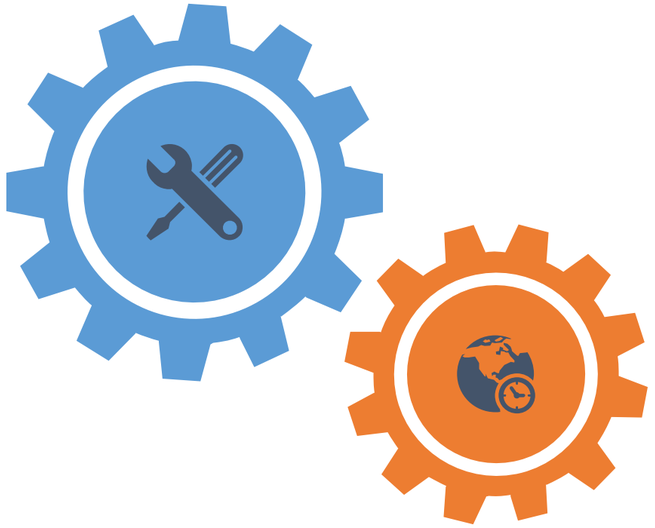
Assisted Practice: Guidelines to Demonstrate Primitives and Objects

1. Create a JavaScript project in your IDE.
2. Write a program to demonstrate different data types used in JavaScript.
3. Initialize the .git file.
4. Add and commit the program files.
5. Push the code to your GitHub repository.

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

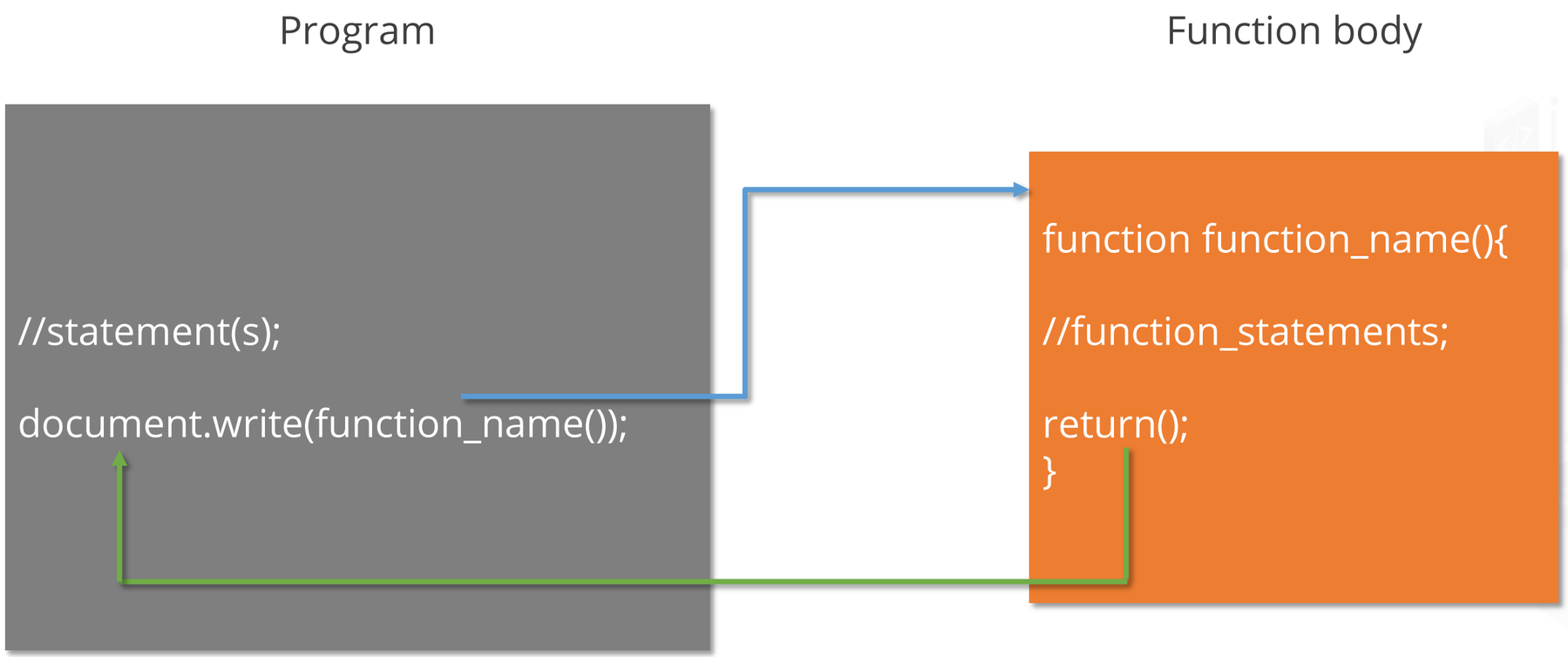
- The function definition is sometimes mentioned as ***function declaration*** or ***function statement***
- Every function definition should begin with ***function*** keyword. User-defined function name should be unique
- Function parameters are enclosed within parenthesis, separated by commas
- The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

2

Function Calling

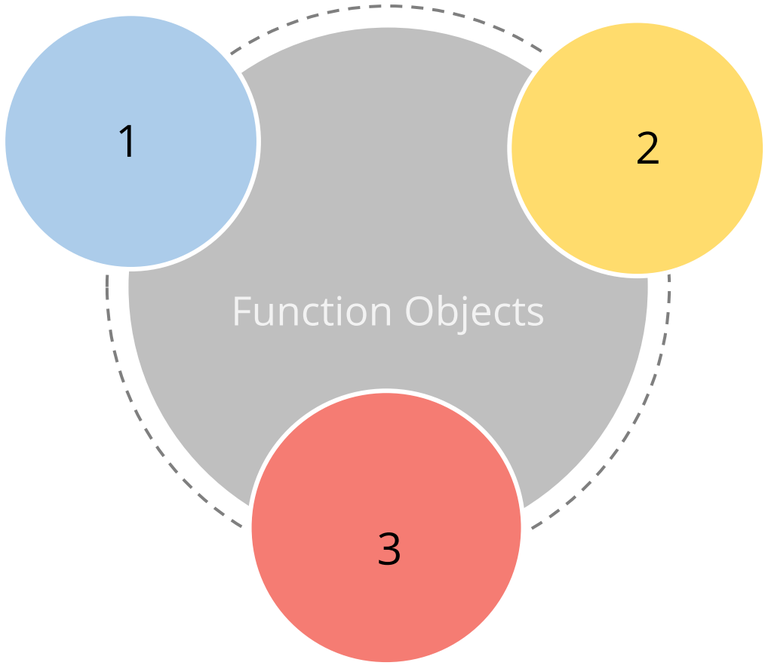
- Calling a function: You call a function by using the ***function name,*** separated by the value of parameters enclosed between parenthesis and a semicolon at the end
- Returning value to the function: There are many cases when you need to return a value from the function after performing a few operations.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

Function Execution



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: 20 min.

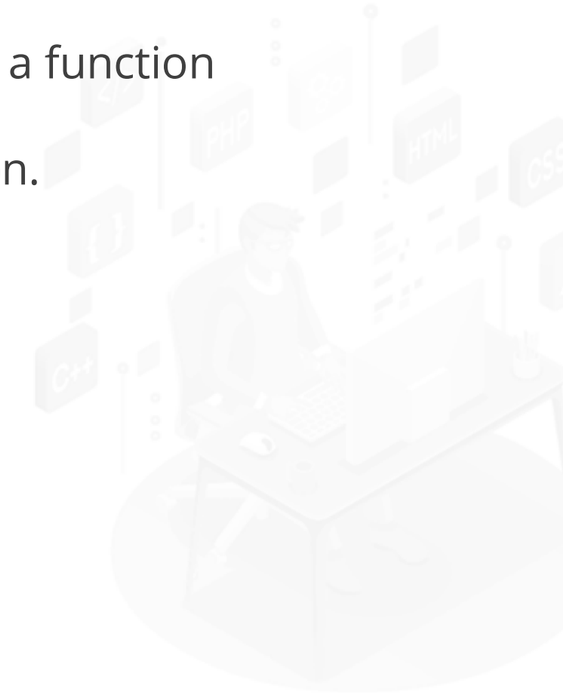
Problem Statement:

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

ASSISTED PRACTICE

Assisted Practice: Guidelines to Work with Functions

1. Create a JavaScript project in your IDE.
2. Write a program in JavaScript to demonstrate how 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 repository.



IIFEs, Callbacks, and Closures

IIFEs (Immediately Invoked Function Expressions)

Syntax:
(function() {
/* */
}) ();

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

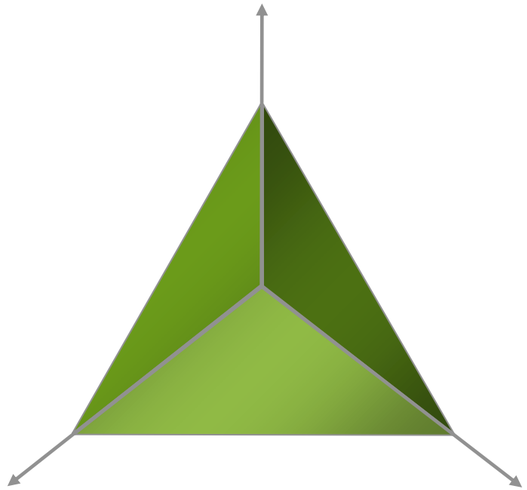
IIFE

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

Queued functions (Timers)

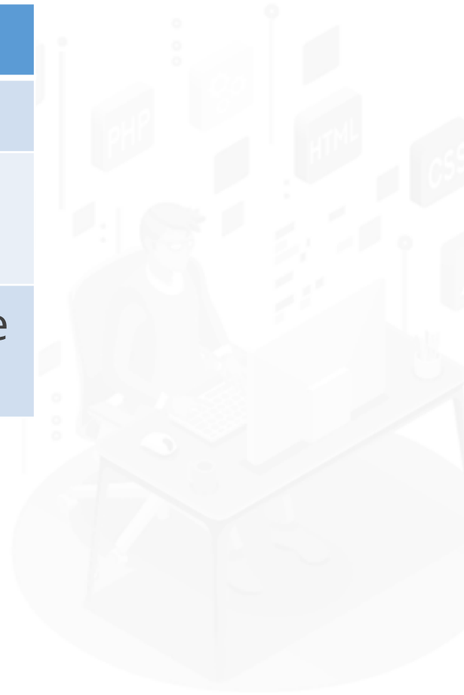
Encapsulation

Event handling

Callbacks

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: 30 min.

Problem Statement:

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

ASSISTED PRACTICE

Assisted Practice: Guidelines 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 employees of an organization.
3. Initialize the .git file.
4. Add and commit the program files.
5. Push the code to your GitHub repository.

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

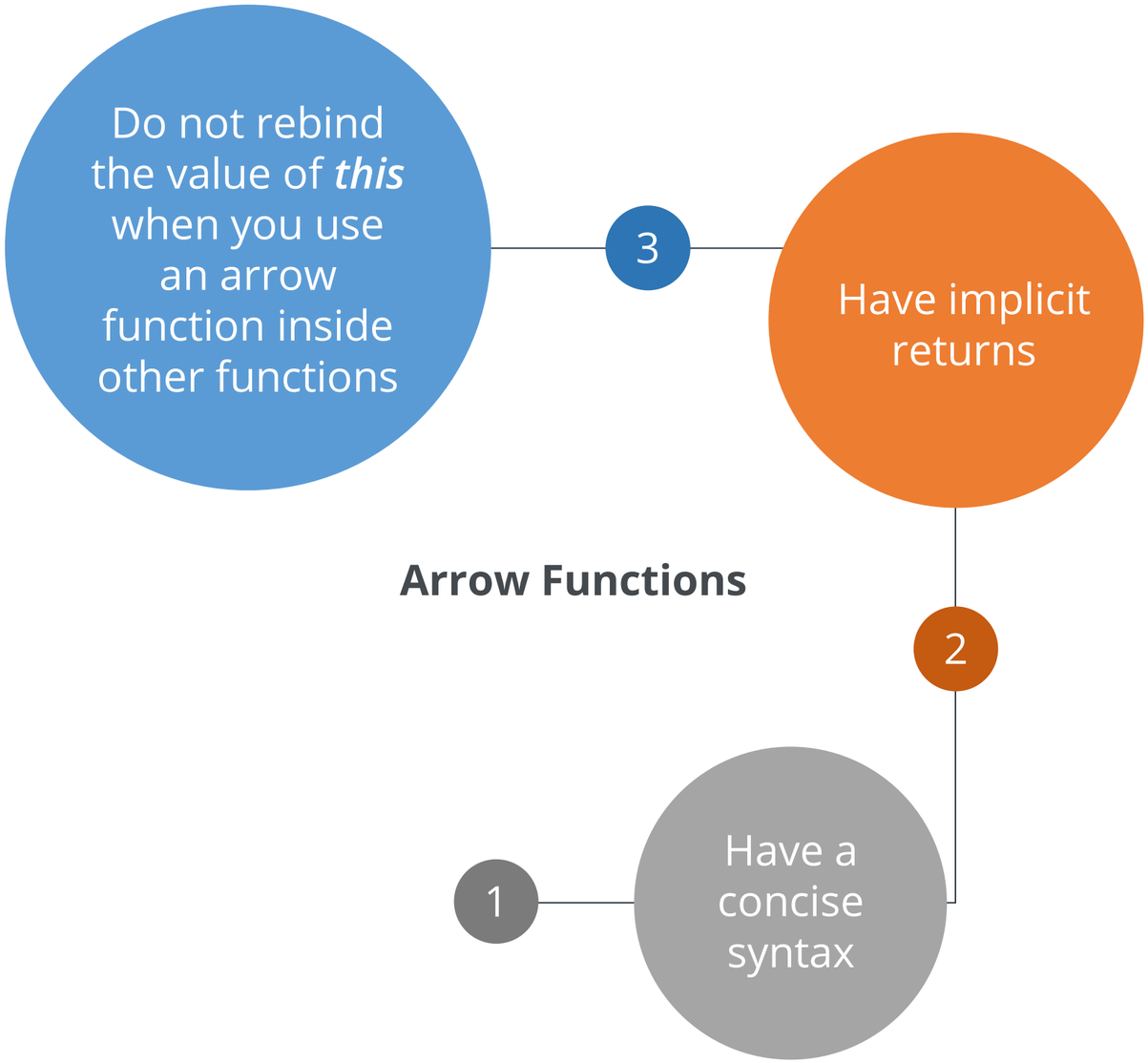
```
function display(){
    if(true){
        var item1 = 'apple';        //exist in function scope
        const item2 = ‘ball';       //exist in block scope
        let item3 = ‘cloud';        //exist in block scope

    }
    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



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: 35 min.

Problem Statement:

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

ASSISTED PRACTICE

Assisted Practice: Guidelines 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 repository.

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

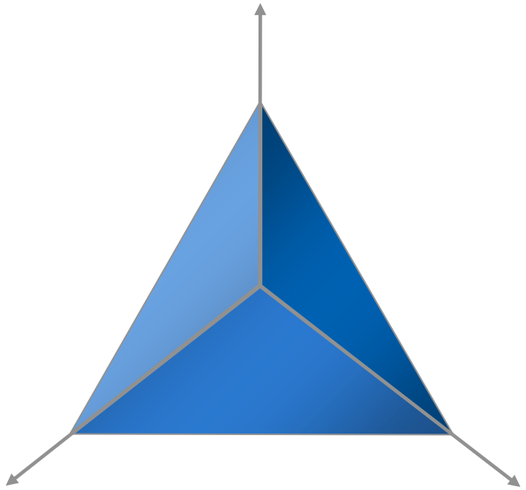
- Any parameter with a default value is considered to be optional
- Default values can be set to parameters that appear before arguments without default values

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

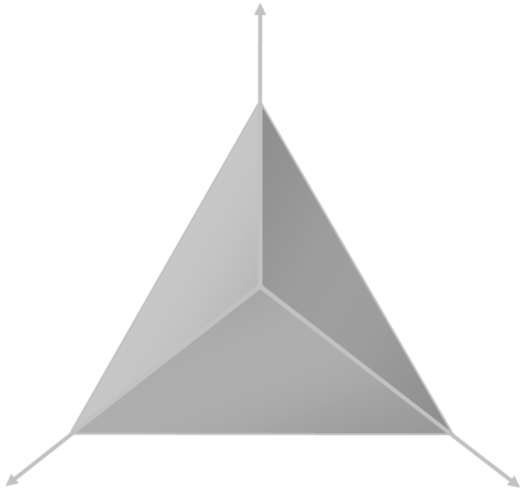
A map can hold both objects and primitive values as either keys or values

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: 35 min.

Problem Statement:

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

ASSISTED PRACTICE

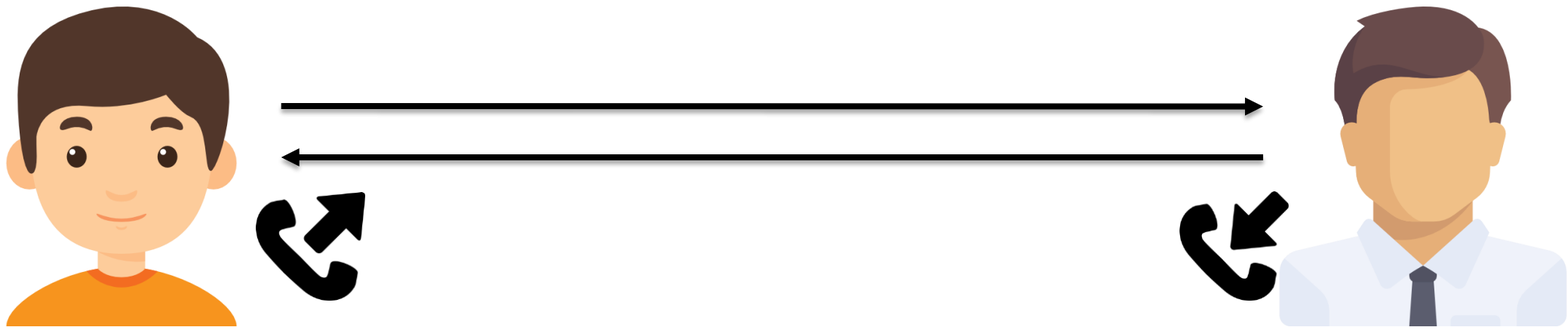
Assisted Practice: Guidelines to Demonstrate Maps and Classes

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 repository.

Promises and Async

JavaScript: As a Synchronous Approach

- Joe, a student, has subscribed to an online training program
- Joe calls up his mentor to clarify his doubts
- Chandler, Joe's mentor, answers Joe's call as soon as his phone starts ringing
- Joe hangs up once the conversation is over

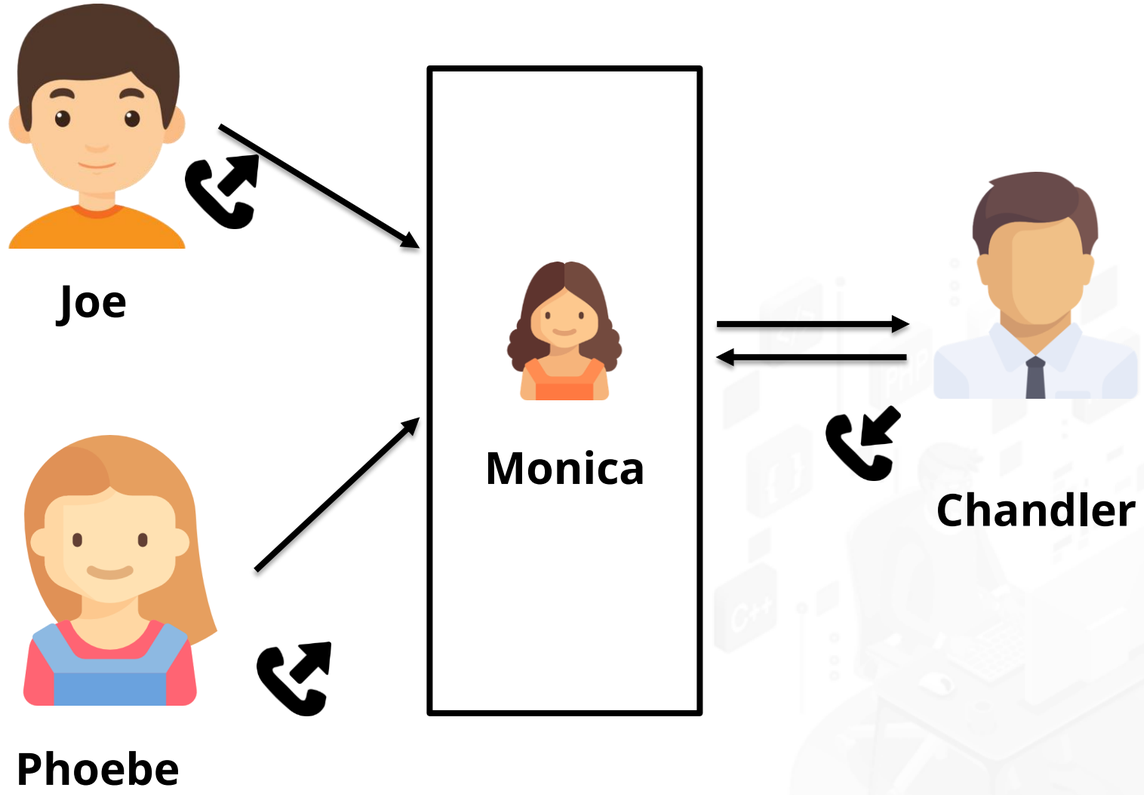


Joe **Chandler**

JavaScript is a synchronous, blocking, single-threaded language. Programs are executed line by line, one line at a time.

JavaScript: As an Asynchronous Approach

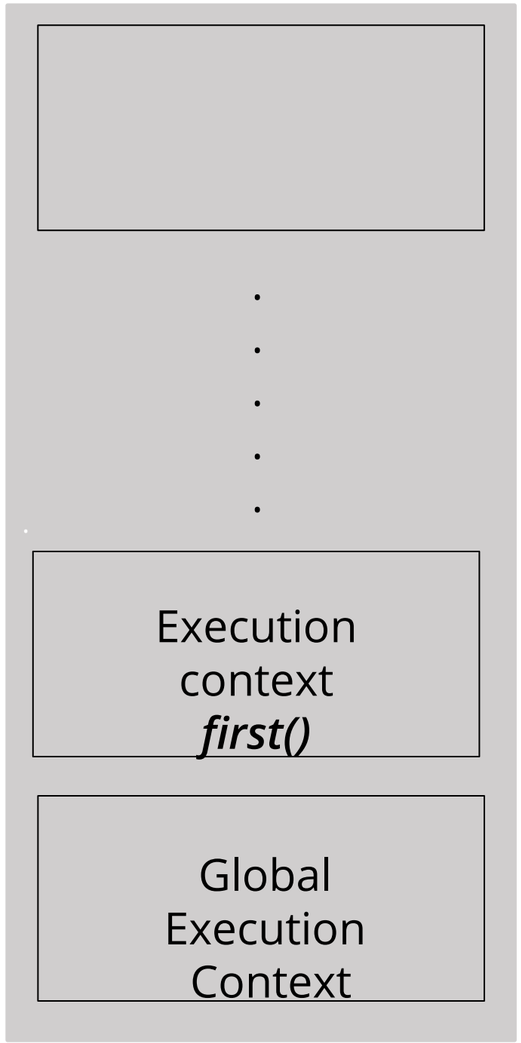
- Phoebe subscribed to the program, same as Joe
- Phoebe has to clarify her doubts with Chandler
- At the same time, Joe wants to talk to Chandler
- Since it is not possible to answer both at the same time, Chandler hired Monica to manage the incoming calls

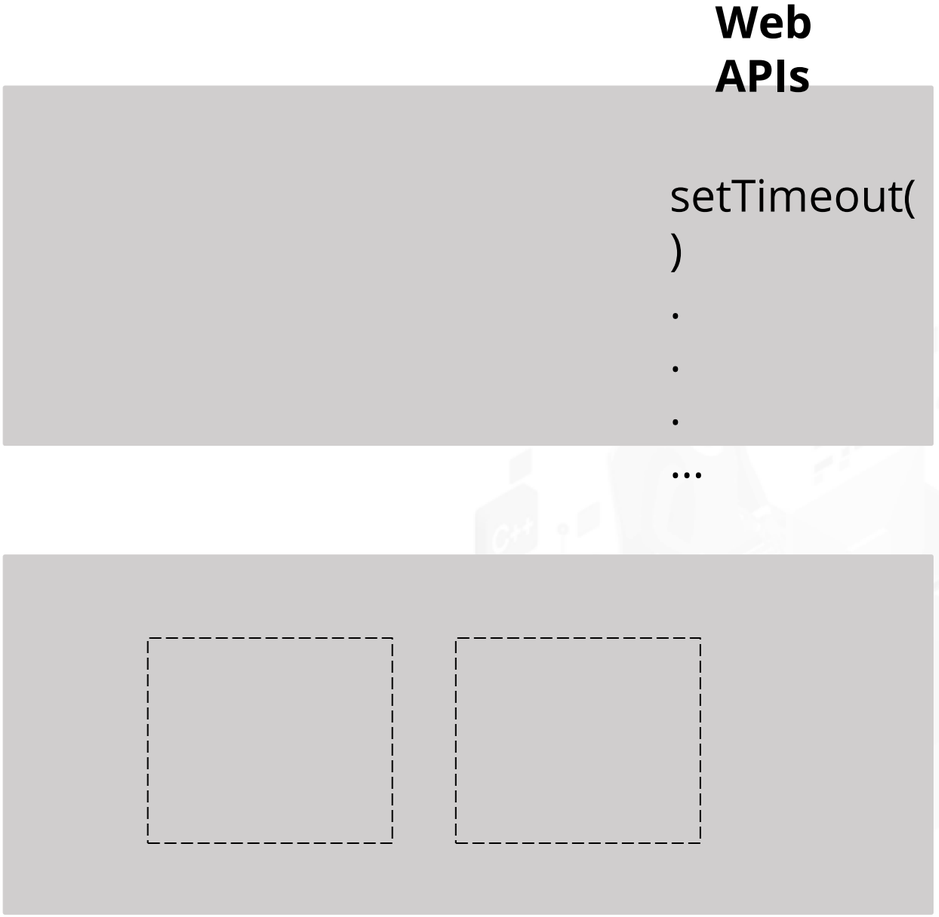


Asynchronous JavaScript can be used to perform long network requests without blocking the main thread via async callback requests.

JavaScript: Event Loop

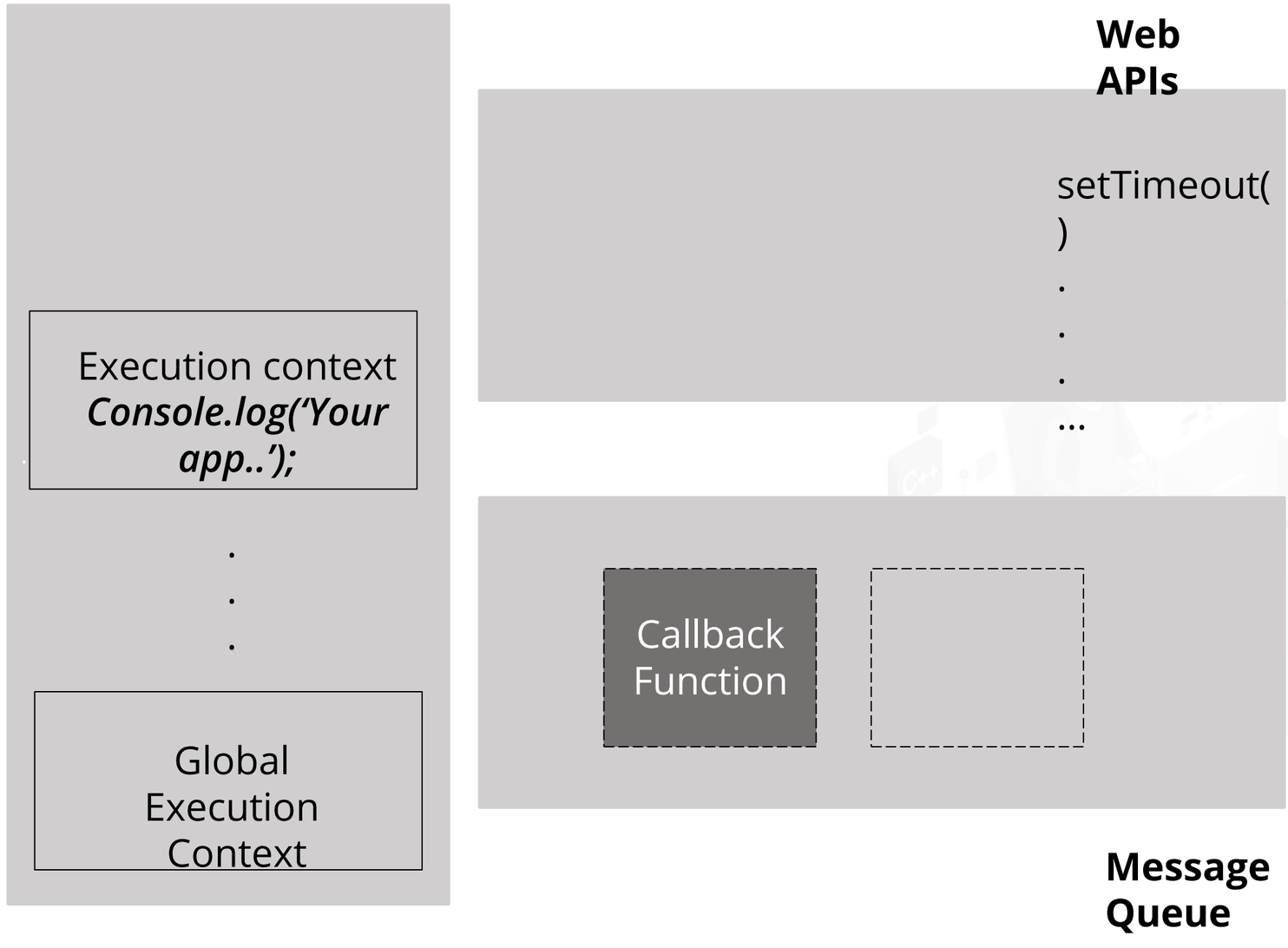
```
const first = () => {

console.log ('Hello User!');
timeout();
console.log('Your app is ready!');
};

const timeout()=>{

setTimeout(()=>{
    console.log('Please wait...');
},2000);

};

first();
```





JavaScript: Event Loop

```
const first = () => {

console.log ('Hello User!');
timeout();
console.log('Your app is ready!');
};

const timeout()=>{

setTimeout(()=>{
    console.log('Please wait...');
},2000);

};

first();
```



JavaScript: Callbacks

A callback is referred to a function passed into another function as an argument, which is invoked inside the outer function to complete a task.

```
function welcome(name){
alert('hello '+name);
};

function UserEntry(callback){

var name= prompt('Please enter your
name:');
Callback(name);
}

UserEntry(welcome);
```

JavaScript: Promises

A promise is an object that represents the completion of an event of an asynchronous operation and its result.

A promise:

- Improves code readability
- Handles asynchronous operations
- Handles errors

A promise has the following states:

- **fulfilled:** When the promise is a success
- **rejected:** When the promise is a failure
- **pending:** When the promise is in a pending state
- **settled:** When the promise is complete

```
var promise = new Promise(function(resolve,reject){

});
```

```
var promise = new Promise(function(resolve,reject){

Resolve('JavaScript Promises');  });

promise.then(function(successMessage){

console.log(successMessage);

}, function(errorMessage){

console.log(errorMessage); })
```

Promises and Async

Duration: 35 min.

Problem Statement:

You are given a project to demonstrate the use of event loops, callbacks, and promises in JavaScript.

ASSISTED PRACTICE

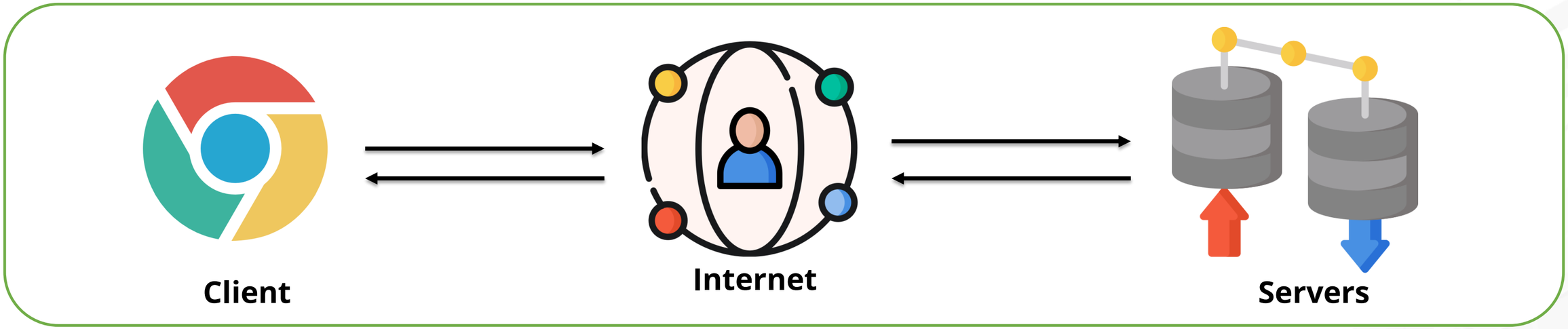
Assisted Practice: Guidelines to Demonstrate Promises and Async

1. Create a JavaScript project in your IDE.
2. Write a program in JavaScript to display the possibilities after flipping a coin using promises and async.
3. Initialize the .git file.
4. Add and commit the program files.
5. Push the code to your GitHub repository.

AJAX Calls

AJAX

AJAX stands for **A**synchronous **JA**vaScript and **X**ML. It helps in developing better, faster, and more interactive web applications with XML, HTML, CSS, and JavaScript.

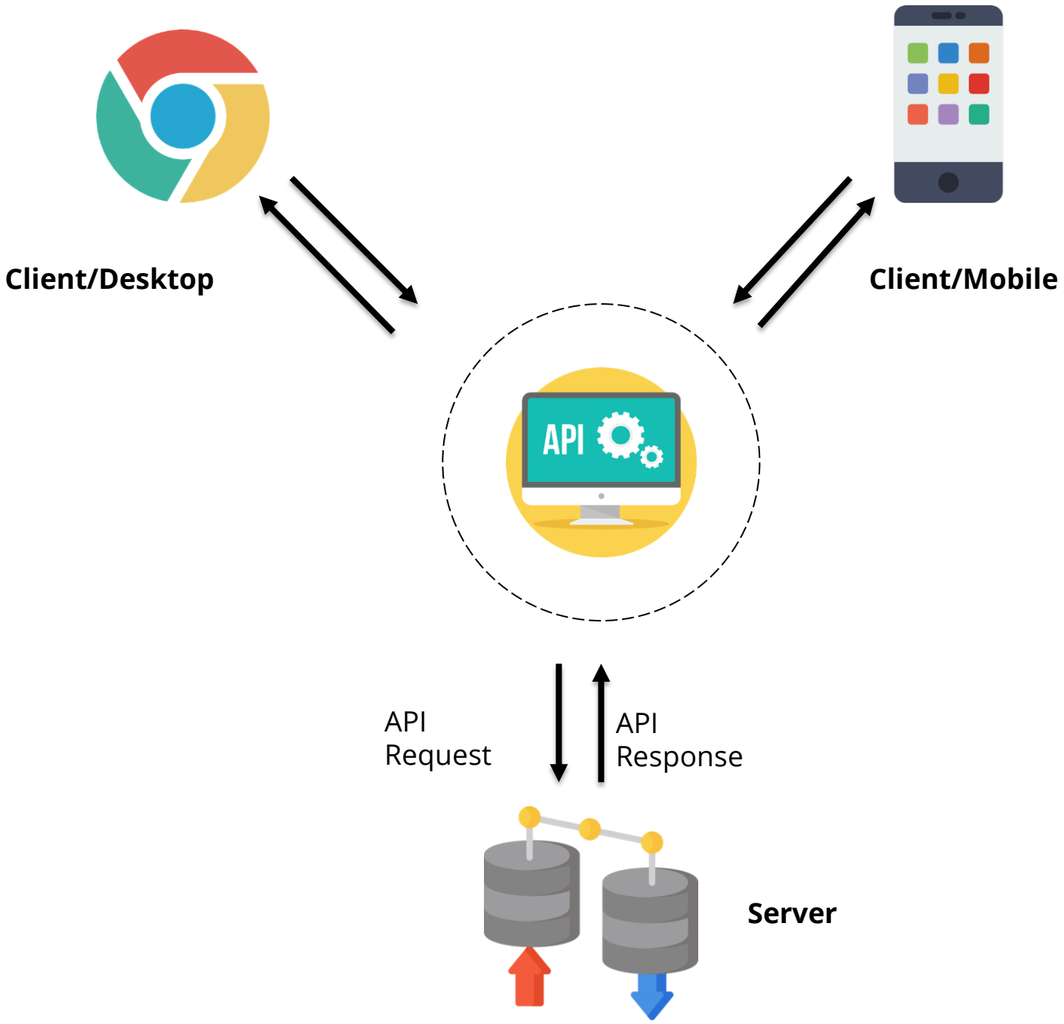


AJAX is based on the following standards:

- Browser-based presentation
- Data is fetched from servers and stored in XML format
- Data is fetched using **XMLHttpRequest** objects

APIs

API stands for **A**pplication **P**rogramming **I**nterface.



Third-party APIs:

- Google Maps
- YouTube Videos
- Weather Data
- Movies Data
- Top 10 Downloaded Apps

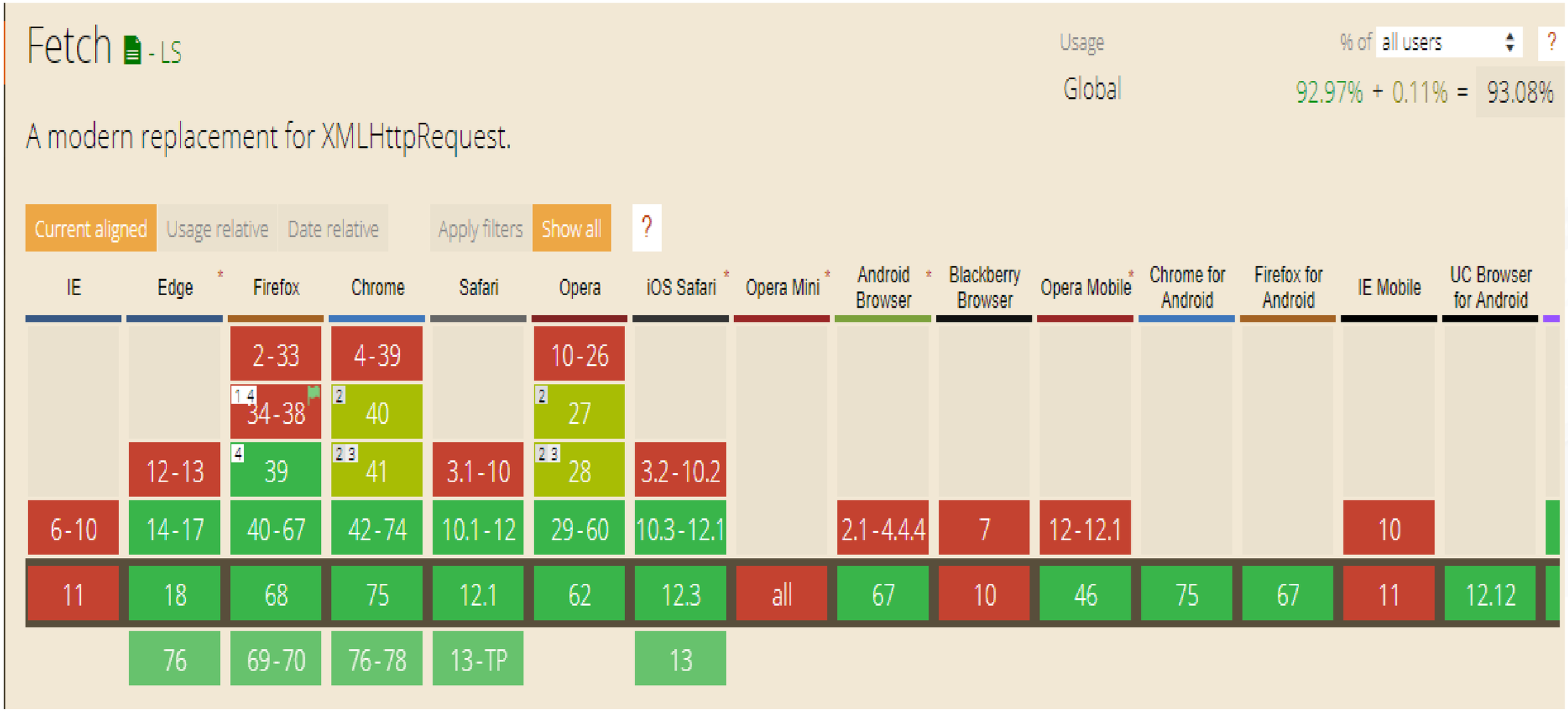
AJAX with Fetch API

The **Fetch API** provides an interface to fetch resources across networks. It helps in defining the HTTP-related concepts such as extensions to HTTP. It is widely used in progressive web app service workers.

Example:

```
fetch('<URL>', {method: 'GET'})

.then(response=>response.json())

.then(json=>console.log(json))

.catch(error=>console.log('error:',error)

);
```

Browser Support for Fetch API



Working with Fetch API

- **Cookieless by default:** Application's authentication could fail, since all implementations of Fetch API may not send cookies.
- **Errors are not rejected:** Rejections only occur if a request cannot be completed; therefore, error trapping is complicated to implement.
- **Timeouts are not supported:** Browsers will continue to run until and unless they are stopped.
- **Aborting a Fetch:** Fetch can be aborted by calling ***controller.abort();***.

AJAX with Promise

Let's assume two functions, **welcome()** and **userProfile().** The **userProfile()** function will not work as it depends on the **welcome()** function.

Ajax without Promise

```
function welcome(){
$.ajax({
url:<some_URL>,
type:'POST',
data:{ //some data },
success: userProfile()
})
}
```

Ajax with Promise

```
function welcome(){
return new Promise((resolve,
reject)=>{
$.ajax({
url:<some_URL>,
type:'POST',
data:{ //some data },
success: userProfile() }) }) }
```

AJAX Calls

Duration: 35 min.

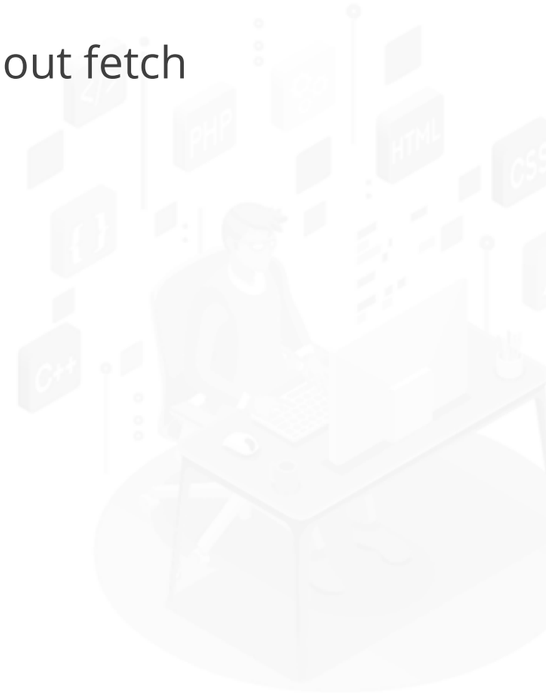
Problem Statement:

You are given a project to demonstrate the implementation of fetch and promises in JavaScript via AJAX.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Demonstrate AJAX Calls

1. Create a JavaScript project in your IDE.
2. Write a program in JavaScript to demonstrate how AJAX calls work with and without fetch API.
3. Initialize the .git file.
4. Add and commit the program files.
5. Push the code to your GitHub repository.



Webpack and Modern JavaScript

Modern JavaScript: Overview

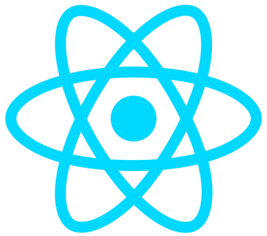
Modern JavaScript is a safe, secure, and reliable programming language. It can execute in the browsers as well as on servers. The browsers have an embedded engine to execute the scripts.

Engine Workflow:

- The engine reads the script
- It converts the JavaScript to machine code
- The machine code is then executed

JavaScript Supports:

Angular



React

Vue.js

Node.js

Modern JavaScript: Compatibility

JavaScript is able to:

- Add a new HTML page, change the existing content, and modify the appearance
- Store the data on the client side
- React to the user inputs

JavaScript is unable to:

- Directly access the operating system functions

New languages **transpiled** to JavaScript before execution:

CoffeeScript

Flow by Facebook

TypeScript by Microsoft

Dart by Google

NPM and Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.

V8 is Google's open source, high-performance JavaScript and WebAssembly engine. It is written in C++.

Perks of Node.js:

- Performance and scalability
- Combination of Node.js with microservices pattern
- Cross-functional team building
- The ***npm*** enterprise
- Long-term support
- Cross-platform development

Webpack

Webpack is a static module bundler for modern JavaScript applications. It helps in mapping every module of the project requirements by building a dependency graph. It can generate more than one bundle. Webpack is used to quickly develop an application.

Webpack module dependencies can be implemented in any one of the following ways:

- An ES6 ***import*** statement
- A commonJS ***require()*** statement
- An ***@import*** statement inside of a CSS/SASS file
- An image URL in a stylesheet or an HTML file

Webpack *devServer*

devServer: It is an object used to change the behavior of the application.

webpack.config.js

```
var path = require('path');

module.exports = {
  //...
  devServer: {
    contentBase: path.join(__dirname, 'dist'),
    compress: true,
    port: 9000
  }
};
```

Output Status:

```
http://localhost:9000/
webpack output is served from /build/
Content not from webpack is served from /path/to/dist/
```

Webpack and Modern JavaScript

Duration: 30 min.

Problem Statement:

You are given a project to demonstrate implementation of the following:

- NPM and node.js setup with respect to JavaScript
- Webpack configuration and bundling modules

ASSISTED PRACTICE

Assisted Practice: Guidelines to Demonstrate Webpack and Modern JavaScript

1. Create a JavaScript project in your IDE.
2. Write a program in JavaScript to work with node.js and webpack.
3. Initialize the .git file.
4. Add and commit the program files.
5. Push the code to your GitHub repository.

Working with Babel

Babel: Overview

Babel is an open source JavaScript compiler used to convert ES6+ code into a backward compatible version of JavaScript.

Popular uses of Babel:

- Transforming syntax
- Adding polyfill features
- Transforming source code

Example

Arrow function as input

```
[10,30,21].map((n)=>n+1)
;
```

Arrow function as output

```
[10,30,21].map(function(
n){
return n+1;
});
```

Working with Babel

Duration: 20 min.

Problem Statement:

You are given a project to implement Babel and ensure generation of backward code compatibility.

ASSISTED PRACTICE

Assisted Practice: Guidelines to Demonstrate Working with Babel

1. Create a JavaScript project in your IDE.
2. Write a program in JavaScript to work with Babel.
3. Initialize the .git file.
4. Add and commit the program files.
5. Push the code to your GitHub repository.

Key Takeaways

- JavaScript is light-weight, cross-platform, and object-oriented programming language used to create client-side dynamic pages
- There are two types of data types in JavaScript: primitive data type and non-primitive (object) data type
- IIFE is a way to execute functions as soon as they are created and is also known as regular function
- JavaScript Map object is used to store each element as key-value pair
- A promise is an object that represents the completion of an event of an asynchronous operation and its result. Promises handle errors, handle asynchronous operations, and improves code readability
- Babel is a free and open-source JavaScript compiler and configurable transpiler used for web development

Team Budget Planner

Problem Statement:

Duration: 60 min.

Write a JavaScript program to track the expenses of an e-commerce company.

LESSON-END PROJECT