Advanced Node.js

Software requirements

* Node.js (LTS)
* VS Code

Node.js:

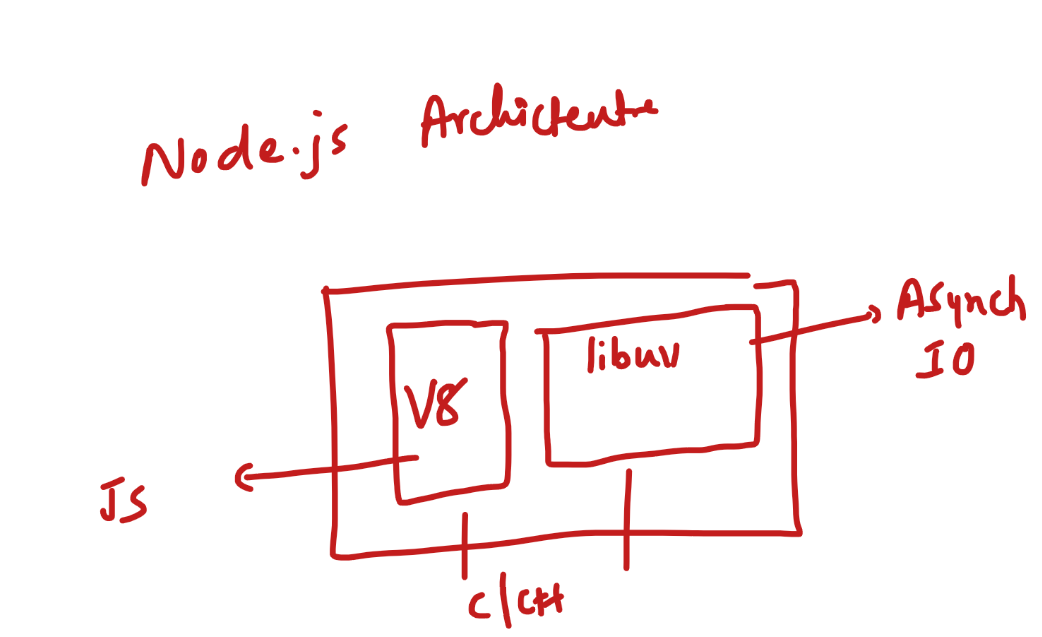
It is a runtime environment to run the javascript outside the browser

It helps the web developers who already have idea on Javascript to use it to implement various backend programs

1. Handling File Systems
2. Handling OS Resources
3. Accessing the Database
4. Networking

>> node file-name (or) node file-name.js

Node.js Architecture

V8 Engine: It takes care of running the Javascript code, V8 engine was part of Google Chrome

libuv: It takes care of Asynchronous IO operations

What are IO Operations

IO operations are those which performs CPU intensive tasks like:-

1. Accessing Network (TCP/UDP)
2. Accessing DB
3. Accessing files system
4. Timer

Synchronous vs Asynchronous

1. Synchronous: The logics are executed in sequence, where current statement depends on previous statement execution
2. Asynchronous: The logics are executed independently without blocking the primary program execution flow, to handle the results from the asynchronous operations we use some mechanisms in Javascript like callback functions

V8 engine & libuv both are implemented in C/C++ programs,

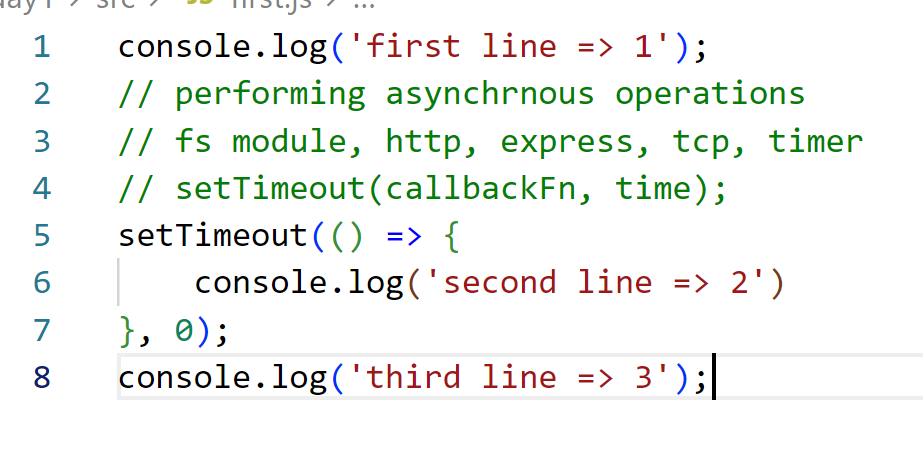
C++ addson: it provides a facility for a Javascript to run the C++ code by providing the API’s in Javascript

V8 engine will have callstack – which runs your javascript code

libuv will have event loop, event queue & other libraries to handle asynchronous operations.

Event loop checks whether the callstack is empty or not, if its empty it pushes the task present in the queue for execution.

first.js



Here the V8 runs the console.log(1) & console.log(3) first because setTimeout is handled by libuv, which adds the callback to the queue, since event loop waits for callstack to be empty, the callback in the queue is executed only after console.log(3) even though the timer is 0s.

Modules in Node.js

Reusable functionalities which you can export & import in your javascript, it could be variables, functions or classes

a.js

let module.exports.done = true;

b.js

let a = require(“./a.js”); // a.js module will be included

console.log(a.done);

Understanding how to include modules & require()

a.js

exports.done = false;

let b = require(“./b.js”);

b.js

let a = require(“./a.js”);

// some statements

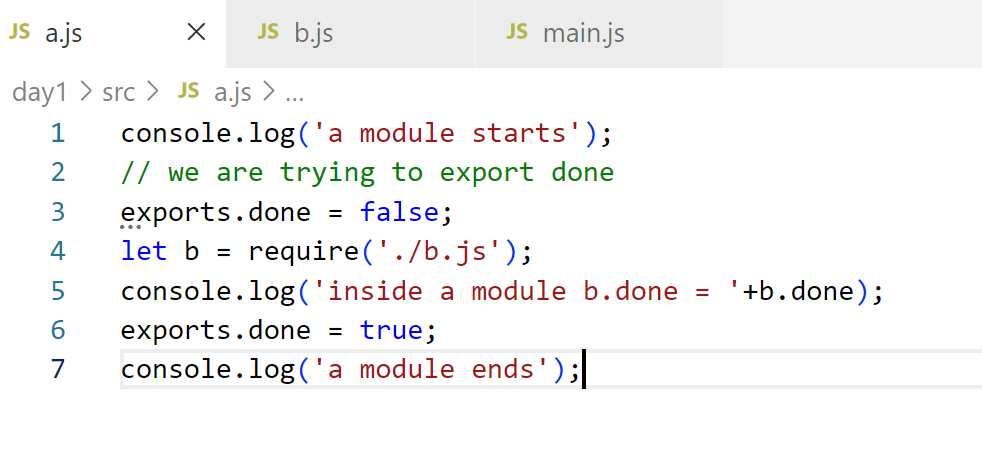
main.js

let a = require(“./a.js”);  
let b = require(“./b.js”);

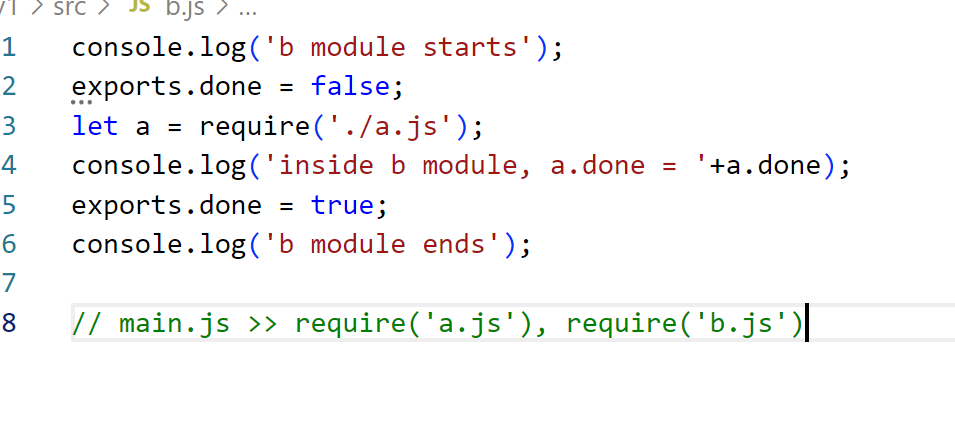
console.log(a.done);

cyclic require

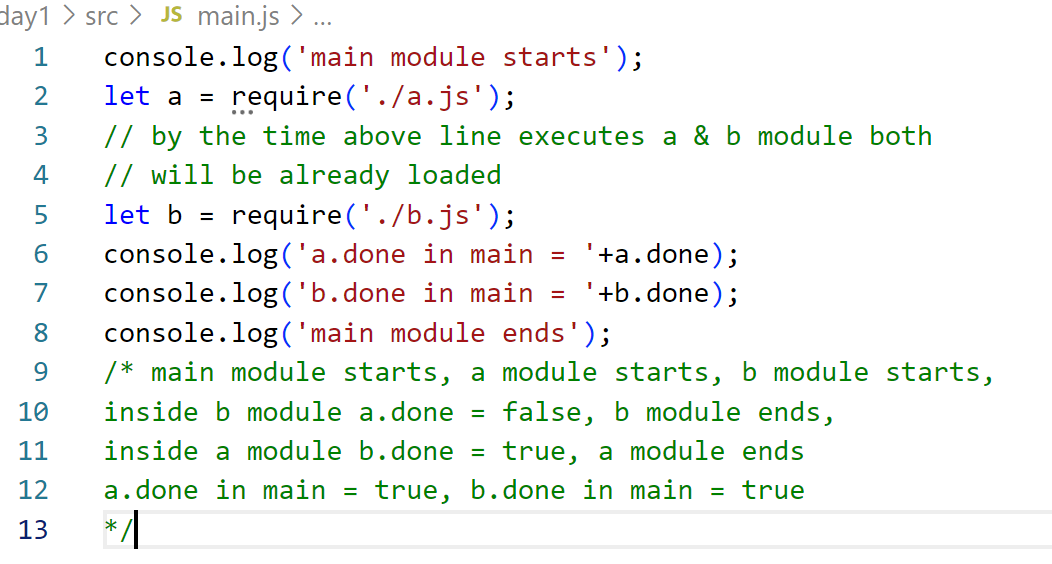
a.js



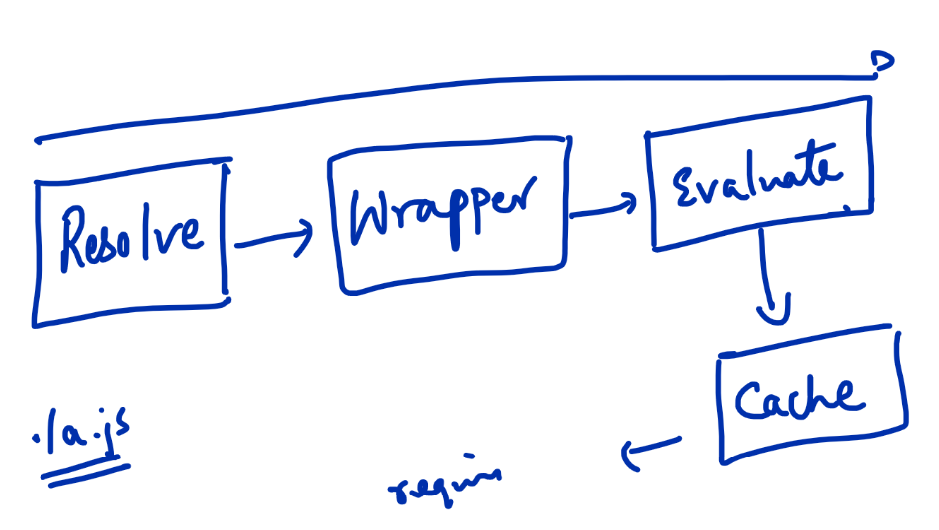
b.js



main.js



How the require works



How to use exports for functions & classes

circle.js

module.exports.area = function(r) {  
 // write some code  
}

module.exports.Circle = class {  
 //  
}

In Javascript you have constructor keyword

class Employee {   
 constructor(id, name, salary) {   
 this.id = id; // initializes id property for employee object  
 this.name = name; // initializes name property for employee

display() {   
 console.log(this.id); // console.log(id) doesn’t work

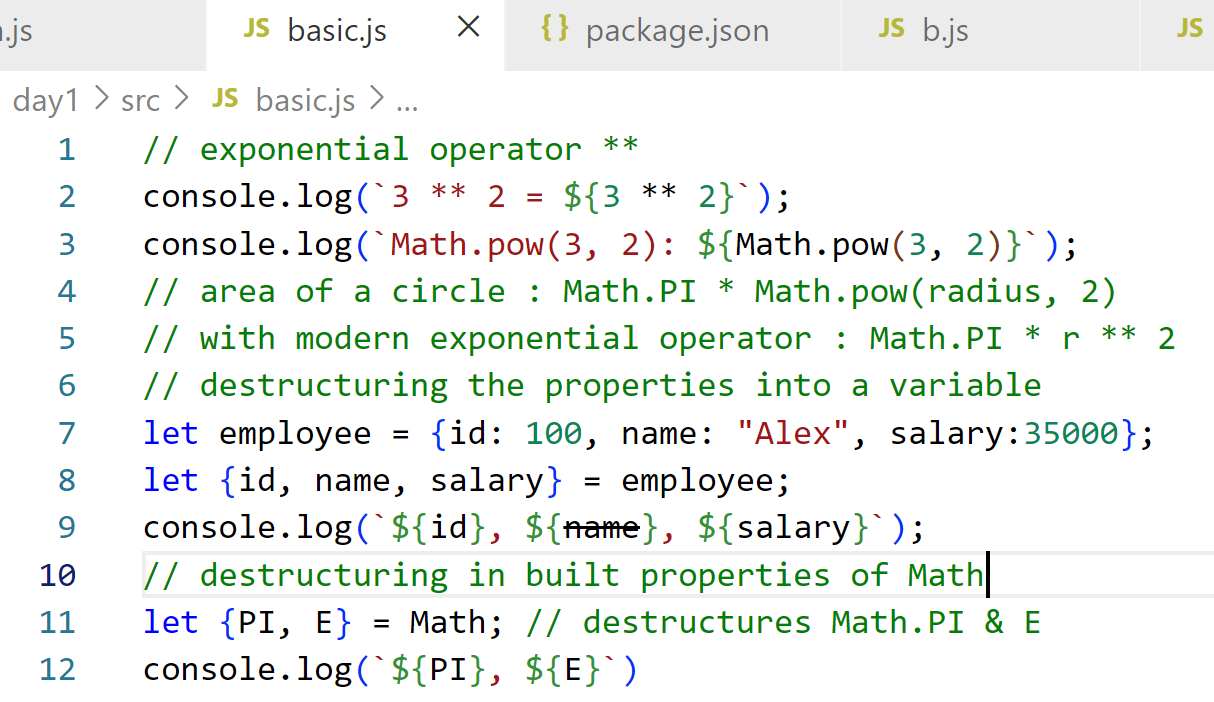
console.log(‘id = ‘+this.id+’, name = ‘+this.name+’, salary = ‘+this.salary);

console.log(`id = ${this.id}, name = ${this.name}`)  
 }  
 }   
}

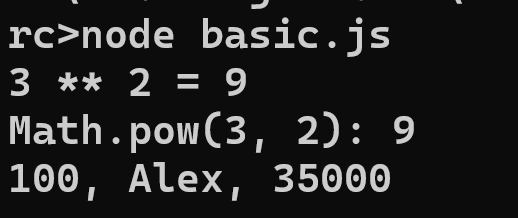
In many case back tick is useful

let id = 100;   
let name = “Alex”;  
let url = `http://ip:port:/${id}/${name}`

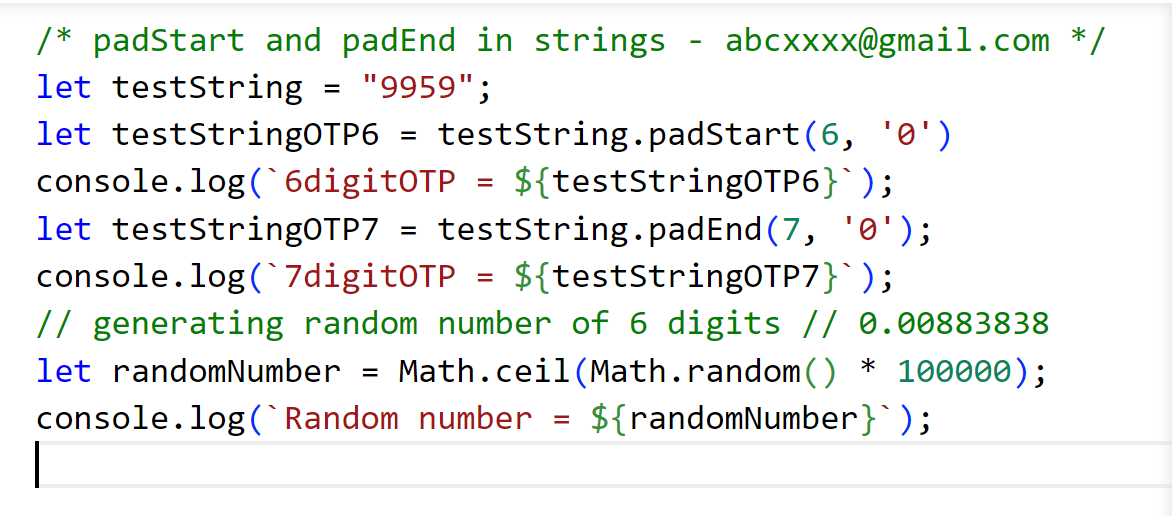
Understanding the basics



Output:



padStart & padEnd: This is used on strings when you want to add extra characters either in the beginning or at the end



Array methods:

filter, map & sort: These methods have an internal iteration

filter: It filters each element based on some condition and returns a new array

let filteredItems = array.filter(item => item % 2 == 0);

map: It transforms each element into another element & stores in a new array

sort: It sorts the elements and returns a new array

Callbacks, Promises & Async/await.

Callbacks: These are the functions which are supplied as an argument to a function which is going to execute later

Drawbacks of callbacks

If you keep nesting the callbacks it becomes difficult to understand the code.

Callbacks written in some places like interacting with SQL/No Sql databases

mongodb.connection(url, ( err, client ) => {   
 // run some queries using the client

client.query(“inserting data”, (err, result) => {   
 // here the nesting starts to grow when there dependent results we need obtain  
 });  
}) ;

Earlier people used to write callbacks when they want to interact with the apis

ex: XMLHttpRequest was the object people used to call the APIs,

Promises: It is introduced to simplify writing nested callbacks, it will have 2 states

1. Pending state
2. Settled State – Resolved/Success & Rejected/Failed

To settle the promise state we need to call some functions like then(callbackFn) or catch(callbackFn)

callbackFn of then() & catch() are called when promise is settled, the then() invokes its callbackFn, when the promise is resolved/successful, the catch() invokes its callbackFn, when the promise is rejected/failed.

Newer APIs in Javascript use promise to perform asynchronous operations.

ex: fetch(), axios libraries, mongodb libraries

Async/Await: It is used to make asynchronous operation synchronous, i.e., when you are dependent on the result of the previous asynchronous operation

async function testAsync() {  
 let res = await fetch(url); // blocked until promise is settled  
 let value = await res.json(); //blocked until promise is settled  
}

All three programs in an html file

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <div>

        <button onclick = 'fetchData()'>CallApi</button>

        <br />

        <button onclick = 'test()'>PromiseButton</button>

        <br />

        <button onclick = 'testAsync()'>AsyncAwaitButton</button>

    </div>

    <div id = 'users'></div>

    <script type = 'text/javascript'>

        async function testAsync() {

            let url = 'https://jsonplaceholder.typicode.com/users';

            let res = await fetch(url);  // waits promise to settle

            let value = await res.json();// waits promise to settle

            document.querySelector('#users')

                .textContent = JSON.stringify(value);

        }

        function test() {

            let url = 'https://jsonplaceholder.typicode.com/users';

            fetch(url)

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

                .then(value => {

                    let ele = document.querySelector('#users');

                    ele.textContent = JSON.stringify(value);

                });

        }

        function fetchData() {

         let url = 'https://jsonplaceholder.typicode.com/users';

         let xhr = new XMLHttpRequest();

         xhr.open('GET', url, true);

         xhr.send(); // sends the HTTP GET Request

         // onreadystatechange callback is executed aysnc

         // readyState 1 to 4

         xhr.onreadystatechange = () => {

            if(xhr.readyState == 4) {

                // responseText will have resonseContent

                let response = xhr.responseText;

                let ele = document

                        .querySelector('#users');

                ele.textContent = response;

            }

         }

        }

    </script>

</body>

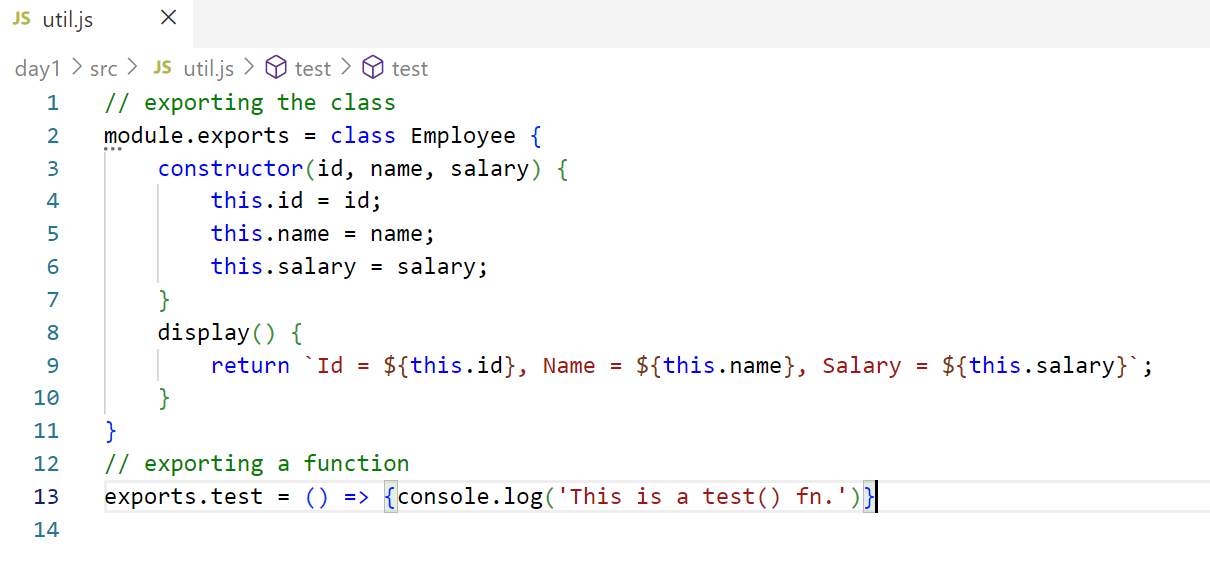
</html>

How to use the classes & functions as a module & include

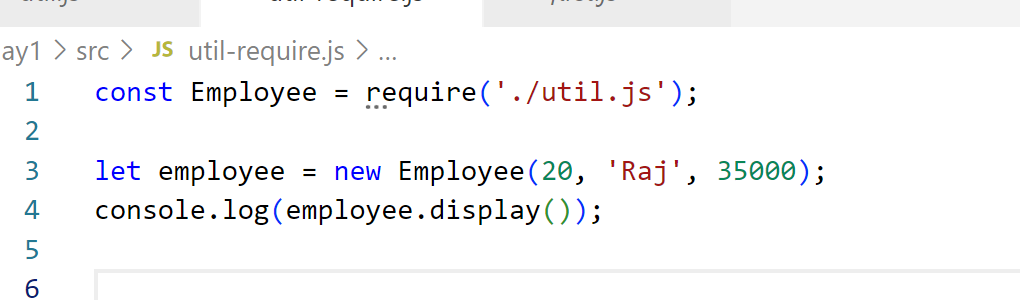
module.exports = class Employee {   
 // constructors & functions goes here  
}

exports.area = function() {  
 // area code goes here  
}

util.js



util-require.js



Modern syntax to import & export

export class Employee { } // import {Employee} from ‘./a.js’

export function test() { } //import {Employee,test} from ‘./a.js’;

Note: This modular type doesn’t work by default, we need to update package.json with “type”:”module”

util.js

// exporting the class

export class Employee {

    constructor(id, name, salary) {

        this.id = id;

        this.name = name;

        this.salary = salary;

    }

    display() {

        return `Id = ${this.id}, Name = ${this.name}, Salary = ${this.salary}`;

    }

}

// exporting a function

export function test() {

    console.log('This is a test() fn.')

}

// exporting an object

export const message = {status:200, result:"HELLO"};

// exporting an array

const users = [

    {id:1, name:"Virat", phone:9999},

    {id:2, name:"Rohit", phone:8888}

];

util-imports.js

import {Employee, message, test, users} from './util.js';

// use their names as it is to work with it

let emp = new Employee(123, 'Raj', 90000);

console.log(emp.display());

// using arrays

let arr = users;

arr.forEach(item => console.log(item));

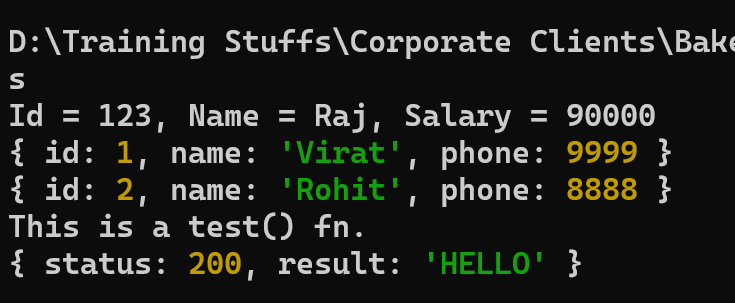
// calling test

test();

// using message without assignment

console.log(message);

Output:



Types of exports & imports

1. Named exports & imports: It must be imported with the same name
2. Default exports & imports : It can be imported with any name, a file can have maximum one default exports

// default export

export default class App {   
 //connect(), store(), retrieve(), update();  
}

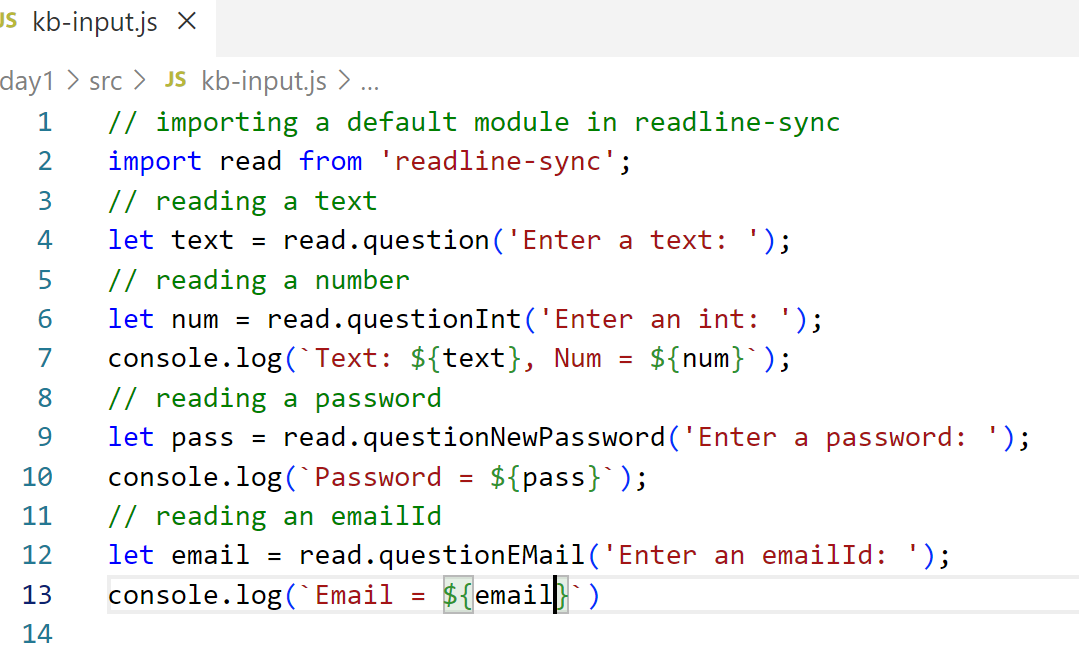
// named exports  
export class Xyz { }   
export class Demo { }

Importing the above modules  
import DB, { Xyz, Demo } from ‘./util.js’;  
DB.store(), DB.retrieve();

How to read input from the keyboard

npm install readline-sync // this is a third party module which is used to take input from the keyboard, it has a default module which provides methods to read strings, numbers, password, email-ids

Syntax: import readline from ‘readline-sync’;



Output:

