**Week3**

**Lab Session: Functions in JavaScript**

**1. JavaScript Function**

A JavaScript function is a block of reusable code designed to perform a specific task. Functions allow you to write modular, maintainable, and efficient code. They can accept inputs (parameters), process those inputs, and return a value.

**2. How to Write Functions in JavaScript**

**a. Function Declaration**

function greet() {

console.log("Hello, World!");

}

This defines a function named greet that prints "Hello, World!" to the console when called.

To call the function:

greet();

**b. Function with Parameters**

You can pass parameters to make functions more flexible.

function addNumbers(a, b) {

return a + b;

}

console.log(addNumbers(5, 10)); // Output: 15

**c. Function with Default Parameters**

Functions can have default parameter values.

function multiply(a, b = 1) {

return a \* b;

}

console.log(multiply(5)); // Output: 5

**3. How to Export a Function with Arguments**

Exporting a function is useful when you want to use it in another file/module.

**Named Export**

export function greet(name) {

console.log(`Hello, ${name}!`);

}

**Default Export**

function greet(name) {

console.log(`Hello, ${name}!`);

}

export default greet;

**Importing an Exported Function**

For named export:

import { greet } from './filename.js';

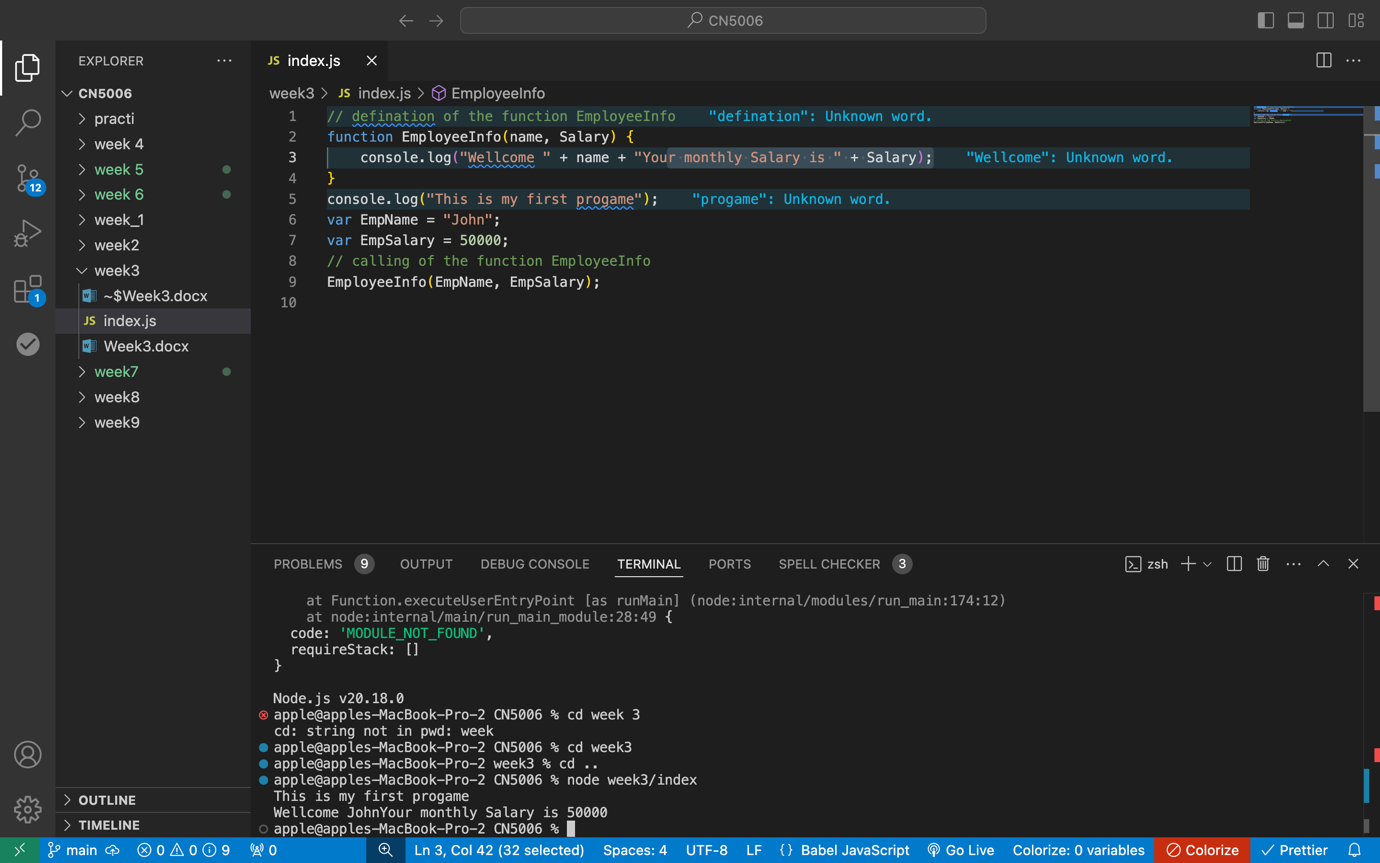
greet("Alice");

For default export:

import greet from './filename.js';

greet("Alice");

**Tutorial exercise 1:**



**4. Defining an Arrow Function**

Arrow functions provide a shorter syntax for writing functions.

**Basic Arrow Function**

const greet = () => {

console.log("Hello, World!");

};

greet();

**Arrow Function with Parameters**

const addNumbers = (a, b) => a + b;

console.log(addNumbers(5, 10)); // Output: 15

**Single Parameter Arrow Function**

If there's a single parameter, parentheses are optional.

const square = n => n \* n;

console.log(square(4)); // Output: 16

**Arrow Function with Default Parameters**

const multiply = (a, b = 1) => a \* b;

console.log(multiply(5)); // Output: 5

**Arrow Function and this Context**

Arrow functions don’t have their own this. They inherit this from their enclosing scope.

function Person(name) {

this.name = name;

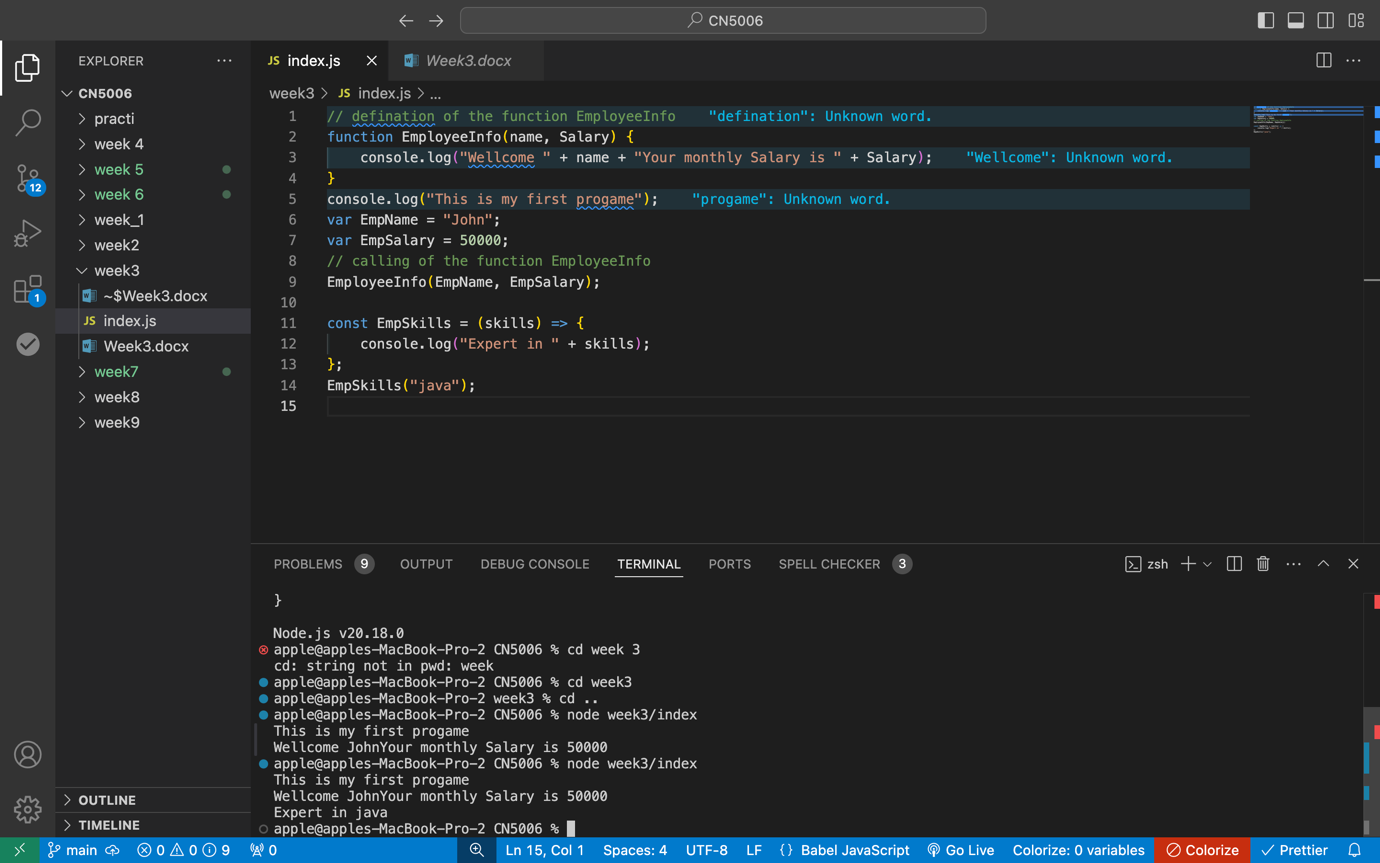
this.greet = () => console.log(`Hi, I'm ${this.name}`);

}

const alice = new Person("Alice");

alice.greet(); // Output: Hi, I'm Alice

**Tutorial exercise 2**



**Creating Local Modules**

A local module is a JavaScript file used to organize and reuse functionality. Write your logic in a file, then export the required functions or variables using module.exports.

**Exporting Functions and Variables**

You can export items to make them accessible in other files:

* Export multiple functions or variables as an object.
* Export a single function or variable directly.

**Using a Module Declared in a Separate File**

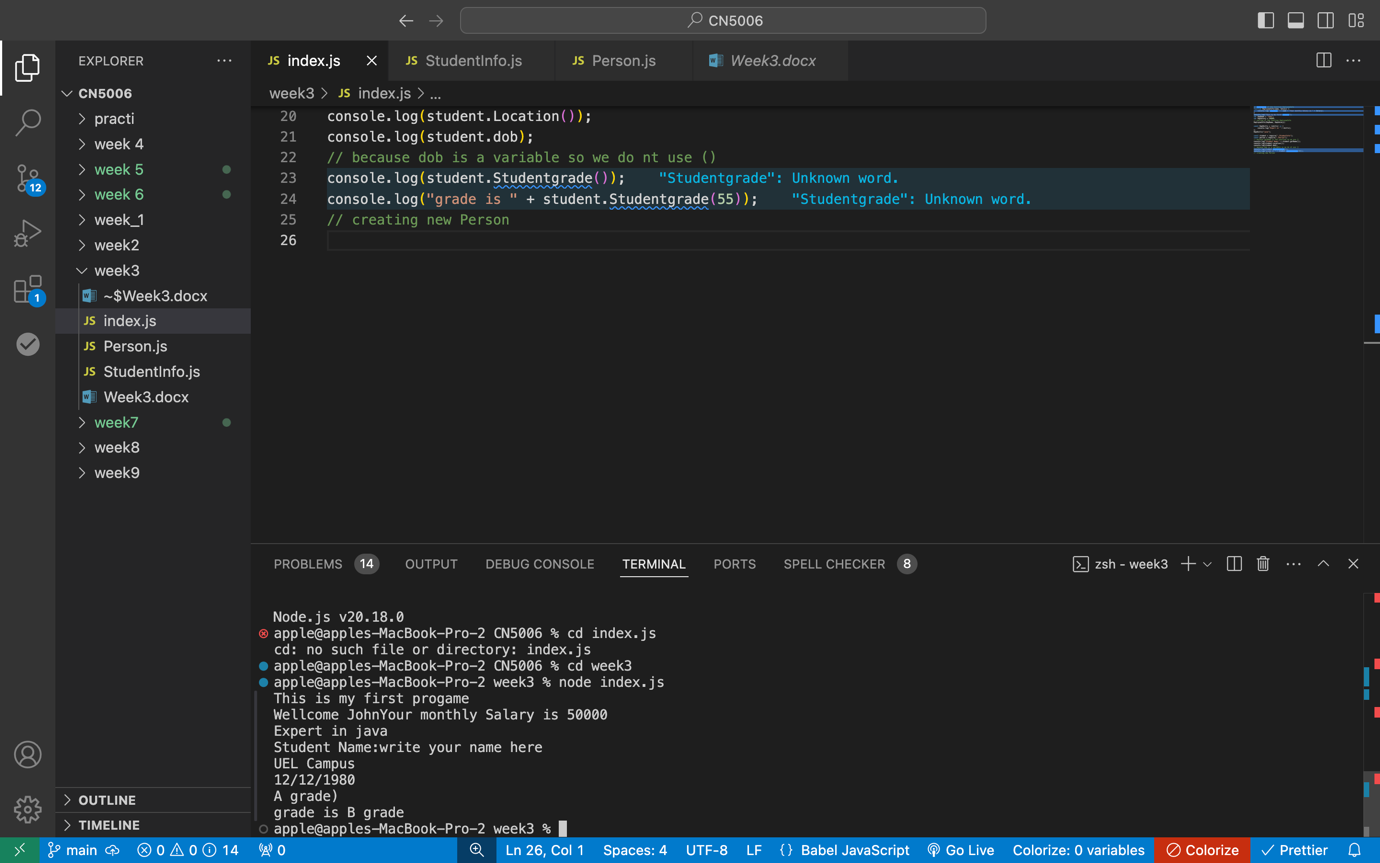
To use a module, import it with the require() function, specifying the relative path to the file. Imported functions or variables can then be used in the current file.

**Accessing Node.js Core Modules**

Node.js provides built-in core modules like fs, http, path, and os for server-side functionality. These modules are imported directly using require() without additional installation. They offer utilities for file handling, creating servers, working with file paths, and retrieving system information.

In summary, local modules and core modules in Node.js allow for modular code organization and efficient utilization of built-in server-side tools.

Tutorial 3



Tutorial exercise 4

