Q 1. What is NPM and REPL with example

NPM (Node Package Manager) is a package manager for JavaScript, primarily used for managing packages and dependencies for Node.js projects. It allows developers to easily install, manage, and share code packages, making it simpler to build complex applications by leveraging existing libraries and tools.

Here's a brief overview of both NPM and REPL, along with examples:

1. **NPM (Node Package Manager):**
   * NPM is primarily used to install and manage packages for Node.js projects.
   * It comes bundled with Node.js installation.
   * It maintains a registry of over a million packages of JavaScript code.
   * NPM provides a command-line interface (CLI) to interact with the package manager.

Ex :

Let's say you want to install the popular lodash library using NPM. You can do so by running the following command in your terminal or command prompt:

bashCopy code

npm install lodash

This command will download and install the lodash package into your project's **node\_modules** directory and add it to your project's **package.json** file as a dependency.

REPL :

* + REPL is a programming environment that allows you to enter and execute commands interactively.
  + It reads input from the user, evaluates it, prints the result, and then loops back to read the next input.
  + It's useful for experimenting with code, testing small code snippets, and debugging.

Example:

Node.js comes with a built-in REPL environment. You can launch the Node.js REPL by simply typing **node** in your terminal or command prompt and pressing Enter.

Cmd :

* Node

2.

After launching the REPL, you'll see a prompt (**>** or **...**) where you can enter JavaScript code. For example, you can perform simple arithmetic operations:

> 2 + 3

5

3.

You can define variables and functions, and execute them directly in the REPL:

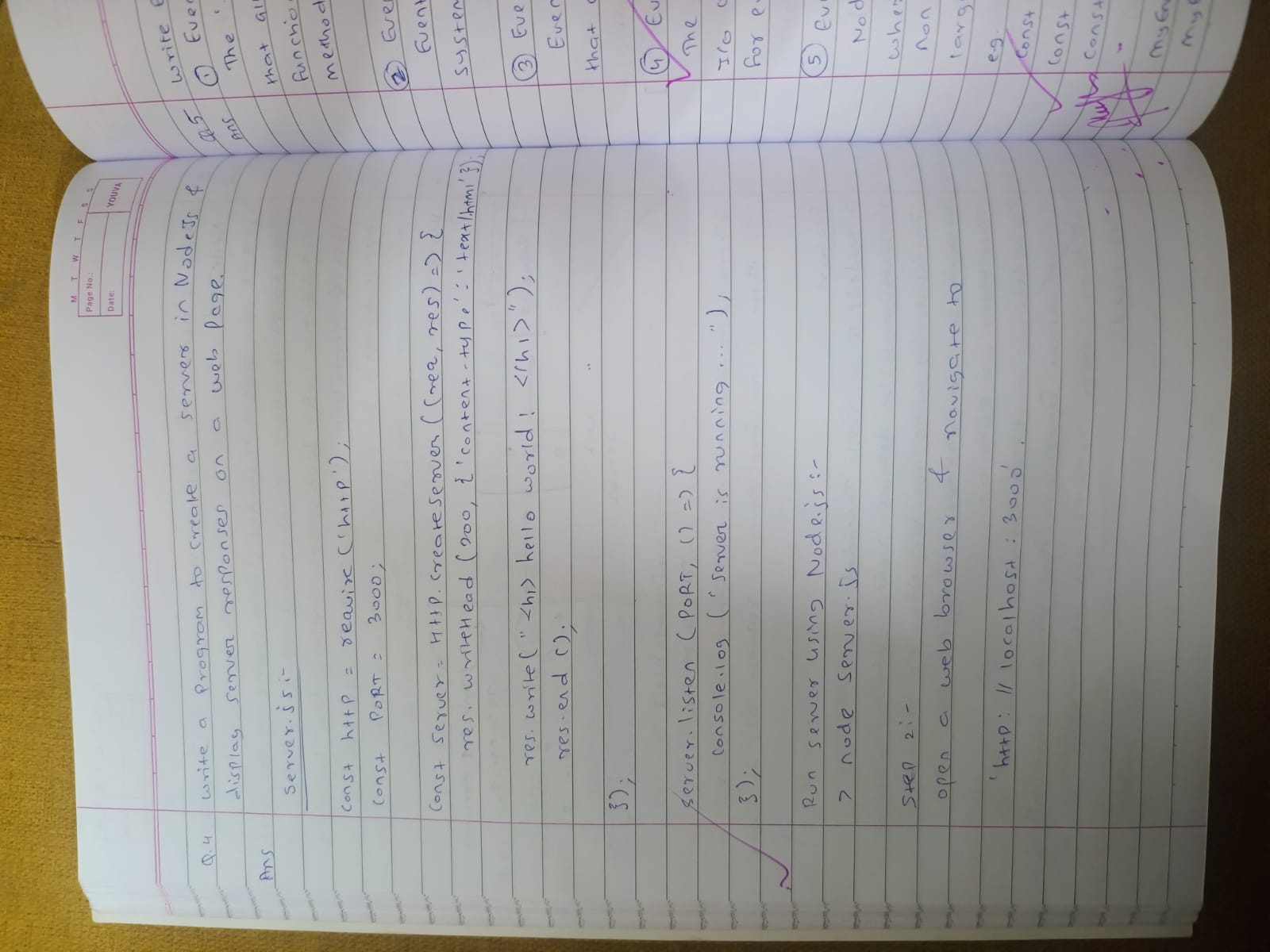
> let greet = name => `Hello, ${name}!`;

undefined

> greet("John")

'Hello, John!'

Q.2 What are the steps to create a web server in nodejs



ANGULAR :

Q 1 . What are the different components used in angular

Angular is a popular front-end framework for building dynamic web applications. It comprises several key components that work together to facilitate the development process. Here are the main components used in Angular:

1. **Modules:** Modules are containers for different parts of your application. They help organize the application into cohesive blocks of functionality. Angular applications are typically divided into multiple modules, such as the root module (**AppModule**) and feature modules.
2. **Components:** Components are the building blocks of Angular applications. They encapsulate the template, data, and behavior of a view. Each component consists of a TypeScript class that defines the component's behavior and properties, along with an HTML template that defines the component's layout.
3. **Templates:** Templates are HTML files that define the UI (User Interface) of an Angular component. Templates can include Angular-specific syntax and binding expressions that allow you to display dynamic data and respond to user input.
4. **Directives:** Directives are markers on a DOM (Document Object Model) element that tell Angular to do something with that element. Angular provides several built-in directives, such as **ngIf**, **ngFor**, and **ngModel**, which enable you to add behavior and manipulate the DOM dynamically.
5. **Services:** Services are reusable pieces of code that perform specific tasks or provide functionality to components. They are typically used to encapsulate business logic, data access, or utility functions. Services are often injected into components or other services using Angular's dependency injection system.
6. **Dependency Injection (DI):** Dependency Injection is a design pattern used in Angular to manage the dependencies between different parts of an application. Angular's DI system allows you to declare dependencies in a component's constructor, and Angular will automatically provide the required dependencies when the component is created.
7. **Modules:** Modules are containers for organizing related components, directives, pipes, and services. Angular applications typically have a root module (**AppModule**) and may include additional feature modules to modularize the application's functionality.
8. **Pipes:** Pipes are used to transform data before displaying it in the UI. Angular provides several built-in pipes for common transformations, such as formatting dates, numbers, and currency, as well as creating custom pipes for specific requirements.

Q 2. What are the different services in angular explain with example

In Angular, services are singleton objects that are instantiated only once during the lifetime of an application. They are used to encapsulate and share functionality across different parts of an application, such as components, directives, and other services. Here are some common types of services in Angular, along with examples:

* **HTTP Service:** The HTTP service is used to perform HTTP requests and handle responses. It allows Angular applications to communicate with backend servers to fetch or send data. This service is typically used to implement RESTful APIs.
* **Authentication Service:** An authentication service is used to manage user authentication and authorization. It typically handles tasks such as user login, logout, and token management.
* **Logging Service:** A logging service is used to log messages and errors to the console or a server. It helps in debugging and monitoring the application.
* **Data Sharing Service:** A data sharing service is used to share data between different components that are not directly related. It can act as a central repository for storing and retrieving shared data.

Q 3 . What is data binding and explain two way binding with example

Data binding is a powerful feature in Angular that establishes a connection between the application's data (stored in components) and the UI (User Interface) elements, such as HTML templates. It allows you to synchronize the state of the application's data with what the user sees and interacts with in the UI. There are several types of data binding in Angular, including interpolation, property binding, event binding, and two-way binding.

Two-way binding is a type of data binding that enables automatic synchronization of data between the component class and the template. It allows changes in the UI to update the component's data, and vice versa, without the need for manual event handling.

Here's how two-way binding works with an example:

* **Template (HTML):** In the HTML template, you use the **[(ngModel)]** directive to establish two-way data binding between an input element and a component property.

<input type="text" [(ngModel)]="name">

<p>Hello, {{name}}!</p>

* **Component Class (TypeScript):** In the component class, you define a property (**name** in this case) and initialize it with an initial value.

import { Component } from '@angular/core';

@Component({

selector: 'app-example',

templateUrl: './example.component.html',

styleUrls: ['./example.component.css']

})

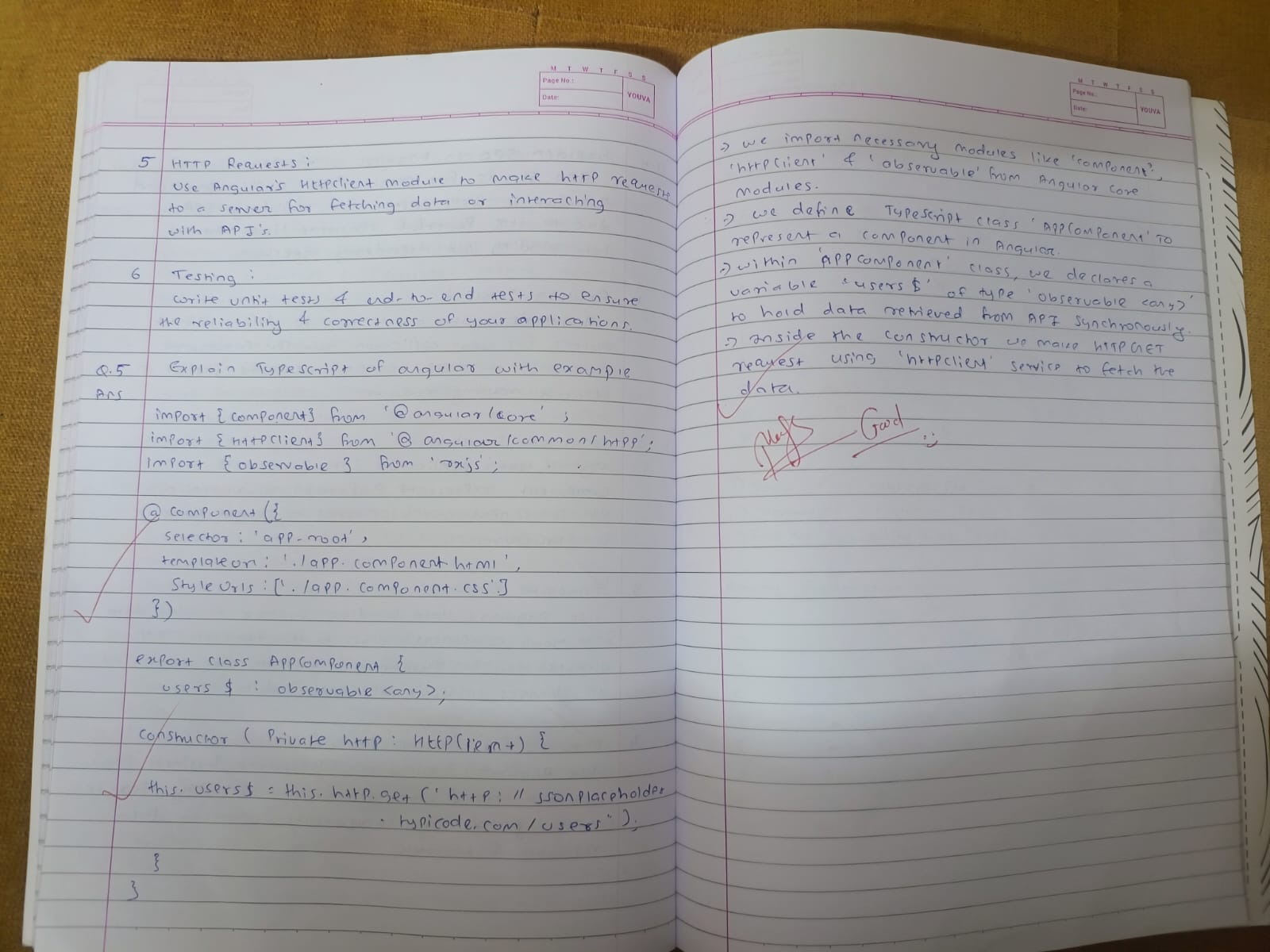
export class ExampleComponent {

name: string = 'John';

}

Here's a step-by-step explanation of how two-way binding works in this example:

* When the user types something into the input field, Angular captures the input event and updates the value of the **name** property in the component class.
* Since the **name** property is bound to the input field using **[(ngModel)]**, any changes to the **name** property will automatically update the input field.
* Additionally, Angular's change detection mechanism ensures that any changes to the **name** property in the component class will be reflected in the corresponding interpolation (**{{name}}**) in the template, updating the displayed text.

Q 4. What is type script and e, 

Q 5 .plain routing with example

Navigation is an important aspect of web applications. A single-page application (**SPA**) does not have multiple-page concepts, and it moves from one view (expense list) to another view.

It provides clear and understandable navigation elements decide the success of an application.

Angular provides a comprehensive set of navigation features to accommodate simple scenarios in a complex environment.

The process of defining the navigation element and associated view is called the routing in Angular. Angular provides a separate module, the Router module, for setting up navigation in an Angular application.

Configure Routing

Angular CLI provides full support for setting up routing during the application build process and working on an application. Let's create a new application with router enabled using the command below -

ng new routing-app

Angular CLI generate a new module, AppRoutingModule for routing purpose.

1. **import** { NgModule } from '@angular/core';
2. **import** { Routes, RouterModule } from '@angular/router';
3. **const** routes: Routes = [];
4. @NgModule({
5. imports: [RouterModule.forRoot(routes)],
6. exports: [RouterModule]
7. })
8. export **class** AppRoutingModule { }

### Accessing routes

1. <router-outlet></router-outlet>

Use **routerLink** property in the required place.

1. <a routerLink="/about" routerLinkActive="active">First Component</a>