- Angular is a powerful, open-source, front-end JavaScript (JS) framework for building dynamic and modern applications, such as mobile, desktop, and web.
- This platform befits the needs of developers by providing declarative templates, end-to-end tooling, dependency injection, and integrated best practices for solving challenges in development.
- Angular development environment uses the Angular CLI tool and create lighter and faster applications.

Exploring the Need of Angular

- Angular allows creating dynamic and interactive applications that live on the web with attractive and flexible user interfaces.
- Angular eliminates the need of keeping unnecessary lines of code by separating the Model and the View using the transparent change detection system.
- A change in the Model is automatically checked, synchronized, and reflected in the View.
- Angular selects only the part of the Document Object Model (DOM) data structures that has been modified based on only the data that has been updated.
- It uses advanced development environment, such as the Command Line Interface (CLI) tool.
- With CLI, you can create a project, add necessary files, and perform a range of ongoing development processes, such as bundling, testing, and deployment.
- Angular is popular among developers for the following reasons:
    - Building Single Page Applications (SPA)
    - Receiving continuous support from Google
    - Securing applications using TypeScript
    - Defining declarative UIs
    - Enabling object manipulation using Plain Old JavaScript Object (POJO)
- Angular applications, without the server, can work like any native application by maintaining states completely at the client side.

Creating the Initial Application

- You can verify the versions of Node.js and npm at the command prompt by using the following command:
- Node -v and npm-v
- The command to install Angular CLI is npm install –g @angular/cli, where the -g flag denotes that the CLI tool is installed globally.
- Now, after setting up a development environment, you need to create an Angular application using the CLI tool.
- To create a project, you use the following command:

- ng new any-project-name
- The command to install Angular CLI is npm install –g @angular/cli, where the -g flag denotes that the CLI tool is installed globally.
- Now, after setting up a development environment, you need to create an Angular application using the CLI tool.
- To create a project, you use the following command:
- ng new any-project-name
- After the application is created, you need to serve your application on the browser.
- Before launching the server, you need to go to the project directory and give the following command at the command prompt:
- cd any-project-name
- To launch the server and open the browser automatically, you need to give the following command at the command prompt:
- ng serve --open
- The ng serve command launches the server, watches the files, and rebuilds the application if you make any changes in your files at any point in time.

Features of Angular

- The basic features of Angular include:
    - Cross-platform
        - Angular facilitates creating client applications, such as mobile, desktop, or web apps using HTML and TypeScript.
        - These apps require zero-step installation and are available offline due to less or no dependency on the server.
        - The desktop apps can be created across multiple platforms, such as Mac, Windows, and Linux.
    - Performance and speed
        - Angular applications take less time to load with the new Component Router that allows automatic code splitting.
        - This makes it easy for users to only load the code that is required for rendering the view they requested.
    - Productivity
        - Angular uses HTML for quickly creating the UIs of the application.
        - With HTML, the developers do not have to spend a lot of time on checking program flows and deciding which module loads first.

        - Also, using Angular, there is no need to include additional getter and setter methods.
    - Simplified MVC pattern
        - The Angular framework is embedded in the MVC architecture but it does not embed MVC completely per the established standards.
        - The improved variant does not ask developers to split the application into different MVC components and then build code to reunite them.
        - The variant asks the developers to divide the app and then takes care of everything else.

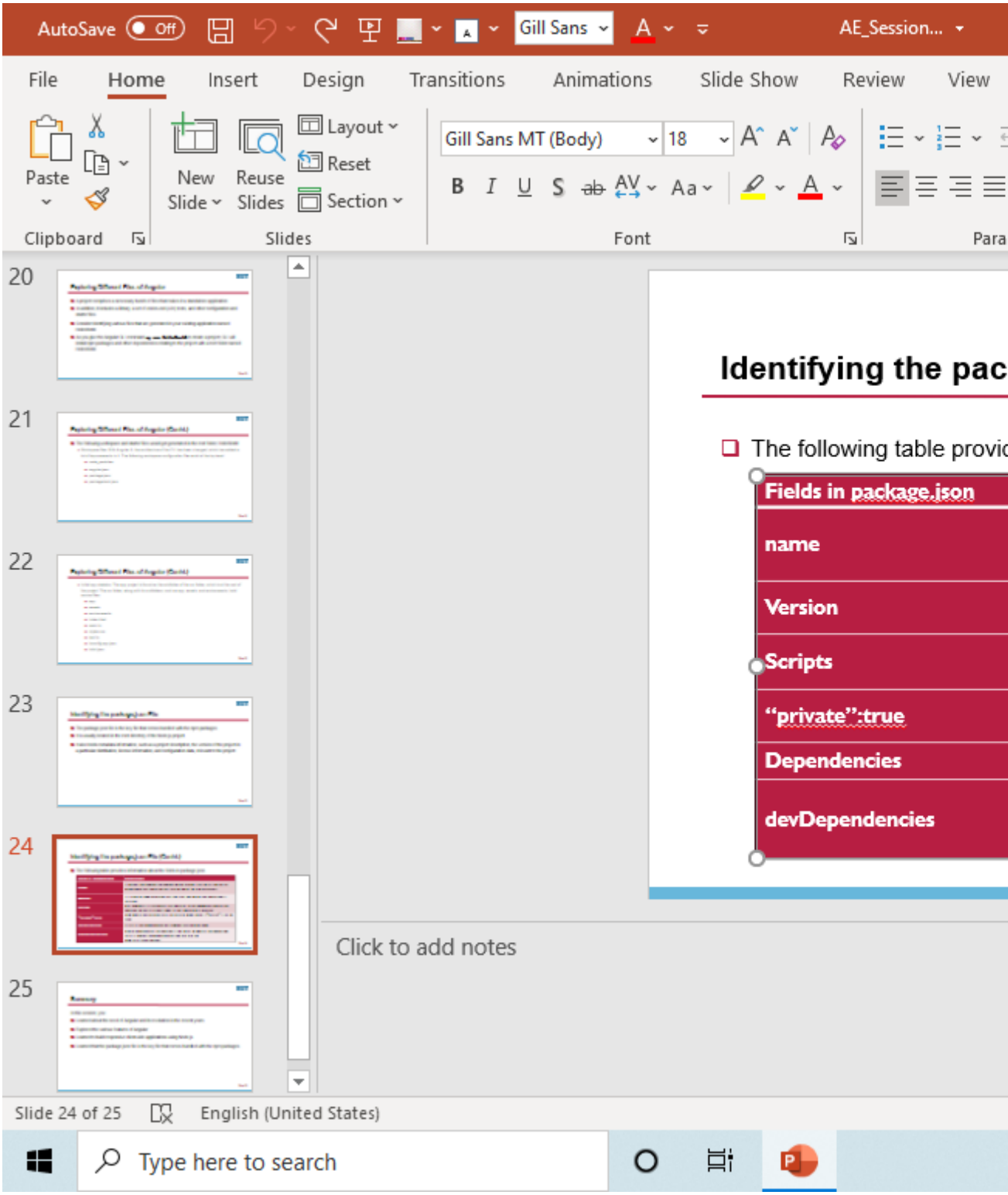
- ❑ The following workspace and starter files would get generated in the root folder, HelloWorld:
    - Workspace files: With Angular 6, the architecture of the CLI has been changed, which has added a lot of improvements in it. The following workspace configuration files exist at the top level:
        - node_modules
        - angular.json
        - package.json

    package-lock.json
    - Initial app skeleton: The app project is found as the subfolder of the src folder, which is at the root of the project. The src folder, along with its subfolders, such as app, assets, and environments, hold source files.
        - app
        - assets
        - environments
        - index.html
        - main.ts
        - styles.css
        - test.ts
        - tsconfig.app.json

    Identifying the package.json File

- ❑ The package.json file is the key file that comes bundled with the npm packages.
- ❑ It is usually located in the root directory of the Node.js project.
- ❑ It also holds metadata information, such as a project description, the version of the project in a particular distribution, license information, and configuration data, relevant to the project.

- The following table provides information about the fields in package.json.



Identifying TypeScript

- TypeScript is an object-oriented and open-source JavaScript developed and maintained by Microsoft.

- ❑ It is a strongly typed, object-oriented, and well-accepted compiled language.
- ❑ It is designed for building large applications and transcompiles to pure JavaScript.
- ❑ TypeScript is accepted as a full-fledged server-side technology and uses the standard specifications of a scripting language, ECMAScript.
- ❑ It takes its basic language features from ES5 and other language features, such as modules and class-based orientation, from ES6.
- ❑ Angular is predominantly used with TypeScript because it has the support of JavaScript as its execution environment.
- ❑ The Language Service component of TypeScript provides typical editor operations, such as statement completion, colorization, code formatting, and so on.
- ❑ Installing Node.js and TypeScript
    - Install Node.js by downloading the .msi file from the following link:

      https://nodejs.org/en
    - Then install TypeScript into the local system using the following command:

      npm install -g typescript
    - To execute the TypeScript code, you need to execute the following two commands:

      // Process of transpilation

      tsc any-program-name.ts

      The preceding command is used to transpile the TypeScript code to JavaScript

      Identifying TypeScript
- ❑ TypeScript is an object-oriented and open-source JavaScript developed and maintained by Microsoft.
- ❑ It is a strongly typed, object-oriented, and well-accepted compiled language.
- ❑ It is designed for building large applications and transcompiles to pure JavaScript.
- ❑ TypeScript is accepted as a full-fledged server-side technology and uses the standard specifications of a scripting language, ECMAScript.
- ❑ It takes its basic language features from ES5 and other language features, such as modules and class-based orientation, from ES6.
- ❑ Angular is predominantly used with TypeScript because it has the support of JavaScript as its execution environment.
- ❑ TypeScript compiler
    - ❑ The TypeScript compiler (tsc) converts all the instructions written in TypeScript into its equivalent JavaScript.

- This process of converting the TypeScript (.ts) code into its equivalent JavaScript (.js) code is known as transpilation.
- The TypeScript compiler only accepts the .ts or .d.ts files and rejects any raw JavaScript files passed to it.

Features of TypeScript

- The following features distinguish TypeScript from JavaScript:
  - TypeScript is portable
    - It can run across browsers, operating systems, and devices.
    - It can easily adapt to an environment that JavaScript already runs on.
  - TypeScript reuses JavaScript libraries
    - Unlike JavaScript, TypeScript needs type definitions for variables, which calls for the rewriting of JavaScript libraries.
    - It reuses JavaScript libraries by defining types in the declaration files with the extension .d.ts.
    - The declaration file is similar to header files in C/C++, offers intellisense for types and function calls, and provides variable support for JavaScript libraries.
  - TypeScript is only JavaScript
    - TypeScript begins with and ends at JavaScript.
    - For learning TypeScript, you only need to know JavaScript.
  - JavaScript is TypeScript
    - The JavaScript code, which is the .js file, can be renamed to TypeScript, which is the .ts file.

Identifiers in TypeScript

- The following rules are defined for naming an identifier, which are like variables and functions in C/C++:
  - It comprises both characters and digits. It can begin with a character but not a digit.
  - It must not include spaces and special characters other than underscore (_) or a dollar symbol ($) while naming itself.
  - It is case-sensitive and does not use predefined keywords.

TypeScript Keywords, Data Types, and Variables

Keywords	Keywords	Keywords
as	any	break

const	catch	continue
else	enum	export
for	false	finally
get	if	implements
in	instanceof	let
null	new	number
public	private	return
string	static	switch
try	throw	this
typeof	var	void
yield		

TypeScript Keywords, Data Types, and Variables (Contd.)

- Data types
  - These are the following three data types supported by the language:
    - The any type: It is a super type of all the types defined in the TypeScript. If you are using the any type, you can carefully skip using type checking for a variable.
    - The built-in type: These are the pre-existing data types in the system. The following table displays the built-in types used for TypeScript programs.

Data Types	Keywords	Description
Boolean	boolean	It denotes the logical values, true a
Null	null	It denotes an intentional absence o
Number	number	It represents double-precision, 64-b values, which hold both integers an values.
String	string	It denotes a sequence of Unicode c
Undefined	undefined	It represents the value given to all u variables.
Void	void	It is used with functions representin non-returning functions.

- Variables
  - A variable is a name given to a space in memory to hold a value that is to be referred to later.

- It acts like a container for the values in a program.
- A variable must be declared in a program before it is used.
- The keyword used to declare a variable is var.

  // Declaring a variable

  var [identifier-name] : [type-annotation] = value;

- You can declare a variable using the following four options:

  1. Declare a variable and its value in a single statement:

  var [identifier-name] : [type-annotation] = value;

  1. Declare a variable's type but no value:

  var [identifier-name] : [type-annotation];

  In the preceding line, a variable will be set to "undefined".

  1. Declare a variable's value but no type:

  var [identifier-name] = value;

  In the preceding line, the variable data type will be set to "any".

  1. Declare neither a variable's value nor its type:

  var [identifier-name];

- TypeScript variable scopes
  - There are three scopes associated with a variable:
    1. Local scope: These variables are called local variables. They are declared and defined within methods or loops.
    2. Class scope: These variables are also called as fields or class variables. They are declared and defined within the class but outside the constructs.
    3. Global scope: These variables are also called global variables. They are declared outside the programming constructs but can be defined with the programming constructs.

Functions in TypeScript

- A function comprises a set of statements/instructions that needs to be executed repetitively in order to perform a specific task.
- It modularizes the code into separate functionalities, which can be called again and reused as many times as required by a program.
- A function declaration informs the compiler about a function's name, return type, and parameters.
- The following syntax is used to create a function:

function function-name (param1:datatype[=value], param2:datatype[=value]):return-type{}

- It is mandatory to use the built-in keyword function in order to create a function.
- If a function is defined with parameter(s), it is called a parameterized function; otherwise, it is called a parameter-less function.
- Any function can have optional and default parameters passed to it.
- An optional parameter is identified when it is marked with a question mark symbol "?" appending to its name.
- The optional parameter is always set as the last argument in a function.
- The default parameters are assigned values by default. However, the default parameters can also be explicitly passed with values.
- The following syntax depicts the optional and default parameters passed to a function:

```
// Syntax for optional parameter

function  function-name(param 1[:datatype], param 2?[:datatype])

// Syntax for default parameter

function  function-name(param 1[:datatype], param 2[:datatype]=default value)
```

Lambda Functions and Function Overloading

- Functions are further extended as anonymous functions or lambda functions and can be overloaded.
- Lambda functions
  - The lambda functions represent anonymous functions.
  - The anonymous functions are those functions that are not bound to any function name.
  - The anonymous functions accept inputs, returns output, and can be assigned to a variable.
  - They are dynamically created at runtime and not accessible after their initial creation.
  - The following example depicts an anonymous function:

    ```
    var displaymessage=function([arguments]) {…}
    ```

  - Lambda functions are also called arrow functions and have the following three parts associated with them:
    1. Parameters: A function may optionally have parameters.
    2. The fat arrow (=>) / lambda notation: It is also called as the "goes to" operator.

3. Statements: It refers to the set of instructions in a function.

- You can also use a single letter parameter to define a precise function declaration.
- A lambda expression is the anonymous function expression that points to a single line of code.

- Function overloading
  - A process where a program that has multiple methods with the same name and same purpose but different implementations based on the inputs is known as function overloading.
  - To overload a function, you need to perform the following three steps:
    1. The successful implementation of the concept called function overloading largely depends on its function signature. The following three things are evaluated in a function signature:
       - The data type of the parameters
       - The number of parameters
       - The sequence or order of the parameters
    2. The declaration of overloaded functions must be immediately followed by function definition. The types of the parameters passed should be set to "any" if the parameter types are different during overloading.
    3. A call should be made to the overloaded functions by passing values to it.

++

Sample syntax

++

```
function add(p:string, q:string):string;


function add(x:number, y:number): number;


function add(num1: any, num2:any): any {
   return num1 + num2 ;


}


var concat = add("TypeScript ", "Function Ovelorading"); // returns "TypeScript Function Overloading"
```

```
console.log(concat);


var sum = add(100, 150); // returns 250

console.log(sum);
```

- Which one of the following data types supported by TypeScript is also depicted as a dynamic type?

    1. null
    2. undefined
    3. any
    4. arrays

    Answer: 3. any

    Exploring the Object-Oriented Features of TypeScript

- TypeScript with Object-Oriented Programming (OOP) specifications provides a familiarity with using OOP powerful features, such as classes, interfaces, inheritance, and so on, when writing TypeScript code.
- JavaScript is also adapted to use OOP features.
- With ES6, TypeScript allows developers to build their applications by following the object-oriented class-based approach and then compile the code down to pure JavaScript.
- The compiled code can then be accessed across browsers, hosts and devices, or operating systems without waiting for the next JavaScript version.

    TypeScript Tuples, Classes, Enums, Generics, and Inheritance

- Tuples
    - A tuple in TypeScript stores a heterogeneous collection of elements with varied data types.
    - TypeScript offers a tuple as a data type.
    - The following code represents the syntax for declaring a tuple:

      ```
      // Syntax for declaring a tuple

      var tuple_name = [value 1, value 2], …value n];
      ```

    - Accessing tuple values
        - To access each item of a tuple, you can use the following code snippet:

      ```
      // Syntax for accessing tuple item

      tuple_name[index];
      ```

    - Operations on tuple

- You can perform the following operations on a tuple:
- Push: You can call the push() method when it is required to push or append a new item

to the tuple.

- Pop: You can call the pop() method when it is required to remove an item from the tuple.

Sample

```
var tuple_of_player_details = [30,"Justin"];

console.log("Count of items before push: "+tuple_of_player_details.length); // returns the tuple size

tuple_of_player_details.push("Middon"); // append player's last name to the tuple

console.log("Count of items after push: "+tuple_of_player_details.length);

console.log("Count of items before pop: "+tuple_of_player_details.length);
console.log(tuple_of_player_details.pop()+" has been popped out from the tuple");// removes

console.log("Count of items after pop: "+tuple_of_player_details.length);
```

- Classes
  - A class is a blueprint that encapsulates the data of an object.
  - To declare a class, you need to use the keyword class followed by the class name.
  - The following syntax is used to declare a class:

```
// Syntax for declaring a class

class class_name

{

    …

    …

    …

}
```

  - A basic class structure includes the following three components:
    - Fields: It is a variable that stores a value pertaining to an object.
    - Constructors: It is a special function that is used to initialize an object of a class by allocating a memory to it.
    - Functions: It is a block of code that can be called repetitively in order to execute a given set of instructions in it.

- After you create a class, you need to create an object/instance of a class. An object of a class is created using the new keyword.
- To access the data members (attributes and functions) of a class, you need to use the object of the class with a dot (.) notation.

  The dot notation is also called period.

- Generics
  - Generics builds components that efficiently works with the data of today as well as keeps scope for the data of the future.
  - It works over a range of types rather than being limited to a specific type.
  - It is this feature of generics that makes it stand out from the rest and allows users to consume components made using generics and create their own types.
  - The following syntax is used for defining generics:

```
…

function function_name<T>(param1: T, param2: T): T

{

//return values

return args1;

}

…==========================================================
```

- Inheritance
  - The concept of inheritance is implemented when one class inherits the properties of another class.
  - The class that is being inherited is called the base, super, or parent class.
  - The class that is inheriting is called a child, sub, or derived class.

    The following syntax is used to represent the concept of inheritance

```
class class_parent{

// Body of the parent class

}

class class_child extends class_parent{

// Body of the child class as well properties of parent class

}
```

- Interface

- It includes properties and method declarations inside it using either a function or an arrow function.
- Classes can be derived from an interface and classes must implement the structure provided by an interface.
- The implementing classes can have their own properties and methods but they must define all the members of an interface.
- To define an interface, you need to use the interface keyword.
- A class implementing an interface must use the implements keyword.
- To extend an interface from other interfaces, you need to use the extends keyword.
- The following syntax is used for defining an interface:

```
interface IName {
//properties
//methods declarations

}==================================
```

- Duck-typing
  - In duck-typing, the two objects are considered of the same type if both the objects share the same properties.
  - The TypeScript compiler allows the creation of duck-typing objects on the go and offers a strong type-check.
  - Consider an example, VehicleDuckTyping, to understand the concept of duck-typing:

```
class Car {
   no_of_wheels = 4;
}
class Truck {
   no_of_wheels = 4;
}
class Aeroplane {
   no_of_wheels = 3;
   fly(){
      console.log("Aeroplane can fly!");
   }
```

```
}

var car: Car = new Truck(); // substitutes

var truck: Truck = new Car(); // substitutes

var carTwo: Car = new Aeroplane();
```

- Which one of the following keywords can you use to access a super class in inheritance?

    1. implements
    2. extends
    3. super
    4. new

    Answer: 3. super