Module 1: Introduction to WebX.0

1. Explain Evolution of Web

Web 1.0:

- Static Pages
- Content is served from server's file system
- Pages built using Server Side / SSI/ CGI
- Frames and tables are used to structure pages

Web 2.0:

- Dynamic
- Interoperability of user
- AJAX & Javascript
- Server-side Scripting
- People could sort information
- Retrieve and classify data

Web 3.0:

- Altering data/web according to user
- Upgradation of the backend of the website can be done by the user
- Data isn't owned but shared
- Services show different views of the same data
- Semantic web
- Provides frameworks like SOA, WebOS
- Saas Business Model.
- Open-source software platform.
- Web Personalization.
- Resource Pooling.
- Intelligent Web.

Web 1.0	Web 2.0	Web 3.0
Mostly Read-Only	Wildly Read-Write	Portable & Personal
Company Focus	Community Focus	Individual Focus
Home Pages	Blogs / Wikis	Lifestreams / Waves
Owning Content	Sharing Content	Consolidating Content
Web Forms	Web Applications	Smart Applications
Directories	Tagging	User Behavior
Page Views	Cost Per Click	User Engagement
Banner Advertising	Interactive Advertising	Behavioral Advertising
Britannica Online	Wikipedia	The Semantic Web
HTML/ Portals	XML / RSS	RDF / RDFS / OWL

Web 4.0:

- Symbiotic web(Amazon, eBay): Common framework that allows data to be shared and reused across applications, enterprises, and communities.
- ubiquitous web:
- based on wireless communication (mobile devices or computers) connecting people and objects whenever and wherever in the physical or virtual world in real-time

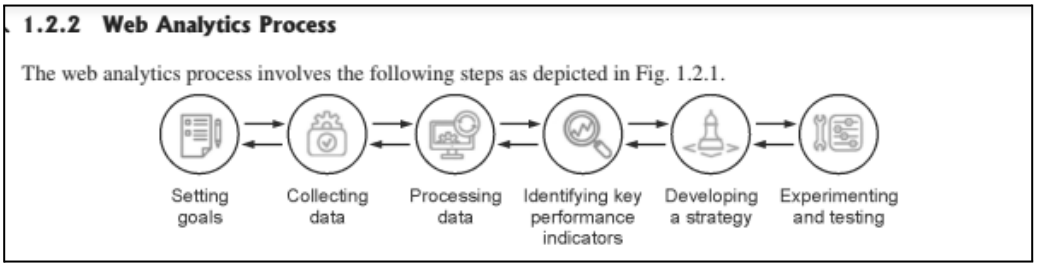
Web 5.0:

- Decentralized, Personal level independence
- based on emotional association with humans
- functions like a personal assistant (Intelligent Personal Agents)
- Example: Emotional Web

2. Explain Web Analytics 2.0 in details

Web analytics 2.0 is:

- **the analysis of qualitative and quantitative data from your website**
- process of analyzing the behavior of visitors to a website.
- involves tracking, reviewing and reporting data to measure web activity,
- Data collected may include traffic sources, referring sites, page views, paths taken and conversion rates.
- enables a business to retain customers, attract more visitors and increase the dollar volume each customer spends.
- The objective of web analytics is to serve as a business metric for promoting specific products to the customers



Types of Web Analytics:

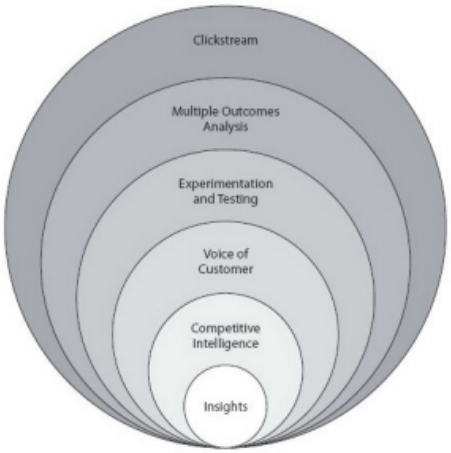
- **Off-site web analytics:** monitoring visitor activity outside of an organization's website. gives insight into how a business is performing in comparison to competitors.
- **On-site web analytics:** uses analytics to track the activity of visitors to a specific site to see how the site is performing.
- The data gathered is usually more relevant to a site's owner. Include **log file analysis(log management) and page tagging(adding snippets of code into a website's HTML to track users)**

Tools Used:

- Google Analytics : tracks page views, unique visitors, bounce rates, referral Uniform Resource Locators, average time on-site, page abandonment, new vs. returning visitors and demographic data.
- Optimizely : helps businesses test and optimize their online experiences and marketing efforts, including conversion rate optimization.
- Kissmetrics : gathers website data and presents it in an easy-to-read format.
- Crazy Egg : Tracks where customers click on a page.

Clickstream : collection, storage, processing, and analysis of click- level data on your website. measurement of pages and campaigns, as well as the analysis of all types of site behaviour:

- Bounce Rate, Sources, Visits, Visitors, Time on Site,Page Views, and many more.



(1A3)**Fig. 1.2.2 : Web Analytics 2.0 Model**

Multiple Outcome Analysis : entails linking customer behaviour to the company's objectives.
Website must Boost revenue; cut costs; and improve customer satisfaction and loyalty.

Experimentation and Testing : improve by running live experiments on the websites.
The services and products can be launched offline based on the customer response.
With experimentation and testing, there is less chance of failure.

Voice of Customer : In this scenario, the customer's voice is important.
Customers' feedback can be gathered **using an online survey**.

Competitive Intelligence : analysing competitors by assessing their strengths and weaknesses. Comparing your performance to that of your competitors allows you to develop your services and create unique and relevant features and functions for your customers.

3. Discuss Clickstream Analysis

- process of collecting, analyzing and reporting aggregate data regarding **pages a user visits on a website and the order in which they visit** them.
- The **path a user takes through a website is called the click stream**.
- There are two levels of clickstream analysis :
    - **1 Traffic Analytics :** operates at the **server level** and **tracks how many pages are served to the user, how long it takes each page to load, how often the user hits the browser's back or stop button and how much data is transmitted before the user moves on**.
    - **2. Ecommerce Analytics :** E-commerce-based analysis **uses clickstream data to determine the effectiveness of the site** in conversions/ transactions.
- Benefits of click stream include :
    - 1. Identify customer trends
    - 2. Discover new mediums
    - 3. Increase conversion
    - 4. Understand user behaviour

4. Discuss the strategy to choose web analytics tool.

- **1. Define your business needs**
  - The first thing one need to do is **determine what their business needs are**.
  - **Decide what exactly it is they want to track and improve in their business**.
  - Be sure to choose a web analytics software **that can provide the data you need for their business.**
  - In addition, the **solution should be scalable to accommodate the future growth and expansion of their business**.
- **2. Ease of use**
  - Find web analytics software that comes with an **easy- to-use interface**.
  - Make sure the **reports generated are easy to read and interpret**.
  - To save time, **find a solution that allows one to capture all the vital analytics data in one dashboard**.
  - **that requires little user customization and minimal start-up training**.
  - In addition, it **should easily integrate with other existing applications such as CRM**.
- **3. Technical support**
  - One need to look for a web analytics provider **that will provide reliable support during the process of setting up and even after**. Here are some of the questions one need to ask when it comes to support:
    - (i) On what hours or days is **live support accessible?**
    - (ii) What **support channels are available** (email, telephone, live chat)?
    - (iii) Is there any **documentation provided in their language**?
    - (iv) Does the **vendor provide any kind of product Training**?
- **4. Price and ongoing costs**
  - Take time to **compare the costs and identify one which is within budget.**
  - In addition, one need to **find out if there are any ongoing costs after purchase**.
  - For instance, **some solutions require add-ons which come at an extra cost**.
  - Therefore, **be sure to get a full breakdown of all the costs before signing**.
- **5. Vendor credibility**
  - One can get such information from their **website's 'About Page' and by reading news items online**.
  - In addition, **visit relevant forums to find out what past customers are saying** about the vendor.
  - If **most people are not happy** would be **advisable to look elsewhere.**
- **6. Test, test, test**
  - **Don't buy** any web analytics software **without testing it first**.
  - Most **vendors offer a free trial version**

5. Explain factors for Measuring the success of a website

- **Conversion rate :** Conversion rate is the number of customers that complete a sale after visiting your site and viewing a product.
- **Gross margin :** Gross margin is the actual profit you earn on top of the cost of goods sold
- **Average order value :** The average order value (AOV) is the monetary value of an average customer order on your site.
- **Cost per acquisition :** how much it costs to gain a new customer.
- **Cart abandonment rate** : percent of customers that add an item to cart and then abandon the purchase.
- **Checkout abandonment rate :** percent of customers that initiate checkout and then abandon purchase.
- **Device type :** How customers access your store is important for giving them an ideal experience.
- **Site speed** : Website speeds and checkout load times are important for maintaining and engaging customers
- **Customer Engagement :** often measured using **reactions, shares, and subscriptions.**
- **Bounce rate :** The bounce rate is the percentage of customers that abruptly leave your site after visiting only one page.
- **Customer survey results and feedback :**
- **Customer product reviews on your site and online**
- **Customer service calls, chats, and emails :**
- **Average ticket resolution time**
- **Blog views and shares**

6. Explain 3.0 and Semantic Web.

- Web 3.0 was **originally called the Semantic Web by World Wide Web inventor** Tim Berners-Lee
- was **aimed at being a more autonomous, intelligent, and open internet.**
- data will be interconnected in a decentralized way,
- Furthermore, users and machines will be able to interact with data.
- But for this to happen, programs need to understand information **both conceptually and contextually**.
- With this in mind, the two cornerstones of Web 3.0 are semantic web and artificial intelligence (AI).
- **Semantic Web :** The semantic web **improves web technologies in order to generate, share and connect content through search and analysis** based on the ability to understand the meaning of words, rather than on keywords or numbers.

7. Explain Characteristics of Semantic Web

- Machine-interpretable, structured, interlinked, open access data repositories.
- Globally edited adaptive information resources.
- Unique web resource identifiers for every bit of information

8. Explain Components of Semantic Web

The Semantic Web is based on several standards and tools:

- **XML** provides a **basic format for structured documents**, though it has no particular semantics.
- **XML Schema** is a **language that provides data typing and allows document structure to be constrained** to allow predictable computation.
- **Resource Description Framework (RDF)** is a **simple three-part data model (subject, predicate, object) for referring to objects or resources and how they relate to one another**. An RDF-based model is frequently **represented using XML**.
- **Uniform (or universal) resource identifiers (URI)** are **short character strings that identify resources on the Web**: documents, images, downloadable files, services, electronic mailboxes and the like.
- **RDF Schema (RDF-S) describes the properties, classes and hierarchies of RDF resources**.
- **Web Ontology Language (OWL)** is **used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms**.
    - This **representation of terms and their interrelationships is called an ontology**.
    - OWL has **more facilities for expressing meaning than XML, RDF and RDF-S**,
    - its **ability to represent machine-interpretable content**.

9. Explain Semantic Web Stack

- Hypertext Web technologies
    - **Internationalized Resource Identifier (IRI)** generalization of URI, provides means for uniquely identifying semantic web resources.
    - **Unicode** serves to represent and manipulate text in many languages.
    - **XML**
    - **XML Namespaces** provides a way to use markups from more sources.
- Standardized Semantic Web technologies
    - RDF
    - RDF Schema(RDFS)
    - OWL
    - SPARQL
        - RDF query language
        - it can be used to query any RDF-based data
    - RIF
        - rule interchange format.
        - It is important, for example, to allow describing relations that cannot be directly described using description logic used in OWL.

10. Discuss N-Triples and Turtle

- N Triples:
  - The N-Triples is a **textual expression of RDF graph**.
  - A **line in N-Triples represents one triple of subject/predicate/object**.
  - These **may be separated by white space**.
  - This sequence is **terminated by a '.' and a new line** (optional at the end of a document).
  - An N-Triples document **contains no parsing directives**.
- Turtle:
  - is an extension of N-Triples.
  - Turtle is a **file format or syntax** that is **used to express the data in an RDF data model**.
  - All RDF written in Turtle should be usable inside the query language part of the SPARQL.
  - A Turtle document **contains a sequence of blank lines, directives and statements, thereby generating the triple**.
  - A Unicode **character string that is encoded in UTF-8 is called as a Turtle document**.
  - A Turtle document **allows writing down an RDF graph in a compact textual form**.

11. Discuss Ontology

- An ontology **defines all of the elements involved in a business ecosystem and organizes them by their relationship to each other**.
- The benefits of building an ontology are:
  - Everyone agrees on a common set of terms used to describe things
  - Different systems – databases and applications – can communicate with each other without having to directly connect to each other.

12. Discuss RDF

- RDF is a **standard for data interchange that is used for representing highly interconnected data**.
- Each RDF statement is a **three-part structure consisting of resources** where every resource is identified by a URI.
- Representing data in RDF **allows information to be easily identified, disambiguated and interconnected** by AI systems.
- RDF stands for **Resource Description Framework** and is a standard for **describing web resources and data interchange, developed and standardized with the World Wide Web Consortium (W3C)**.
- **RDF is the easiest, most powerful and expressive standard designed by now**.
- **Features of RDF**
    - independent parties can exchange and use it.
    - RDF is designed to be **read and understood by computers**.
    - RDF is **written in XM**L.
    - RDF is a **part of the W3C's Semantic Web Activity**.
    - RDF **is a W3C Recommendation**.

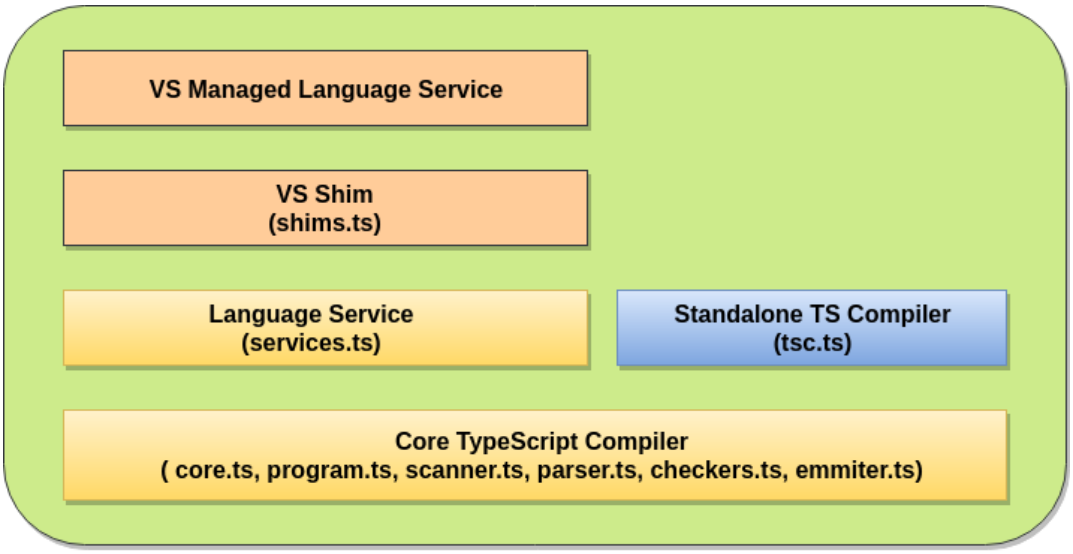
13. Discuss SPARQL

- It stands for Simple Protocol and RDF Query Language.
- SPARQL accesses a web of linked data.
- Syntax:
    - SELECT <variable list>
    - WHERE [<graph pattern>]
    - The variables in the variable list are bound by the graph pattern.
- ability to federate queries across different repositories.
- uses the keyword OPTIONAL instead of LEFT OUTER JOIN.
- SPARQL services vary in whether or not they have a pre-determined RDF database.

Module 2: Type Script

1. Explain TypeScript Internal Architecture

- The TypeScript language is internally divided into **three main layers**. Each of these layers is divided into sublayers or components.



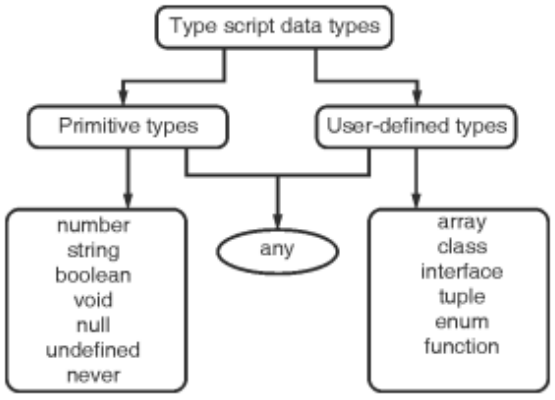
Components of TypeScript

- In the following diagram, we can see the three layers and each of their internal components. These layers are:
  - Language
  - The TypeScript Compiler
  - The TypeScript Language Services
- **Language:** It features the TypeScript language elements. It comprises elements like **syntax, keywords, and type annotations**.
- **The TypeScript Compiler:** The TypeScript compiler (TSC) **transform the TypeScript program equivalent to its JavaScript code**. It also **performs the parsing, and type checking** of our TypeScript code to JavaScript code.
- **The TypeScript Language Services:** The language service **provides information which helps editors and other tools to give better assistance features such as automated refactoring and IntelliSense**. It exposes an additional layer around the core-compiler pipeline. It supports some standard typical editor operations like code **formatting and outlining, colorization, statement completion, signature help, etc**.

2. Discuss TypeScript Types with suitable examples

- The TypeScript language supports different types of values.
- It provides data types for the JavaScript to transform it into a strongly typed programing language.
- JavaScript doesn't support data types, but with the help of TypeScript, we can use the data types feature in JavaScript.
- Following is the syntax of defining a type or data type to the variables in typescript.
    - [Keyword] [Variable Name]: [Data Type] = [Value];
    - [Keyword] [Variable Name]: [Data Type];
- **[Keyword] :** It's a keyword of the variable either var, let or const to define the scope and usage of variable.
- **[Variable Name]** : It's a name of the variable to hold the values in our application.
- **[Data Type] :** It's a type of data the variable can hold such as number, string, boolean, etc.
- **[Value]** : Assigning a required value to the variable.

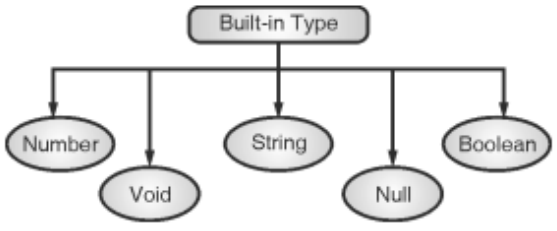
We can classify the TypeScript data type as following:



(1B3)**Fig. 2.5.1 : TypeScript Types**

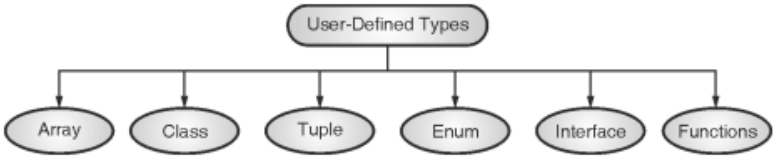
2.5.1 Built-in or Primitive Type

The TypeScript has five built-in data types, which are given below.



(1B4)**Fig. 2.5.2 : Built-in Type**

User-Defined Types



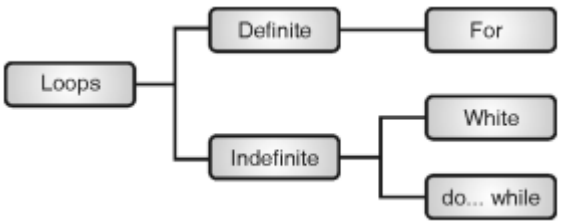
(1B5)**Fig. 2.5.3 : User-Defined Types**

3. Explain variables and operators in TS

- **Variables**:
  - A variable is the storage location, which is used to store value/information to be referenced and used by programs.
  - It acts as a container for value in code and must be declared before the use.
  - We can declare a variable by using the var keyword.
  - In TypeScript, the variable follows the same naming rule as of JavaScript variable declaration.
  - These rules are:
    - The variable name **must be an alphabet or numeric digits**.
    - The variable name **cannot start with digits**.
    - The variable name **cannot contain spaces and special character, except the underscore(_) and the dollar($)** sign.
  - The let keyword is similar to var keyword in some respects, and **const is a let which prevents re-assignment to a variable**.
  - var keyword is used to declare a variable
  - Variables declared in TypeScript with the var keyword have function scope.
- **Operators**:
  - An Operator is a **symbol which operates on a value** or data.
  - It r**epresents a specific action** on working with data.
  - The data on which operators operates is called operand.
  - It can be used with one or more than one values to produce a single value.
  - All of the standard JavaScript operators are available with the TypeScript program.
  - In TypeScript, an operator can be classified into the following ways.
    - 1. Arithmetic operators (+,-,/,*,%,++,--)
    - 2. Comparison (Relational) operators (==,===,!=,!==,>,>=,<,<=)
    - 3. Logical operators (&&,||,!)
    - 4. Bitwise operators (&,|,^xor,~not, >>,<<,>>> bitwise right shift with 0)
    - 5. Assignment operators (=,+=,-=,*=,/=,%=)
    - 6. Ternary/conditional operator(condition**?**expr1**:**expr2)
    - 7. Concatenation operator (+)
    - 8. Type Operator
      - **In** to check for the existence of a property on an object.
      - **Delete** used to delete the properties from the objects.
      - **Instanceof** to check if the object is of a specified type or not.
      - **Typeof** It returns the data type of the operand.

4. Explain decision Making and loops with suitable examples.

- Decision Making
  - if statement
    - var num:number = 5
    - if (num> 0) {
    - console.log("number is positive")
    - }
  - if...else statement
    - var num:number = 12;
    - if (num % 2==0) {
    - console.log("Even");
    - } else {
    - console.log("Odd");
    - }
  - else...if and nested if statements
    - var num:number = 2
    - if(num> 0) {
    - console.log(num+" is positive")
    - } else if(num< 0) {
    - console.log(num+" is negative")
    - } else {
    - console.log(num+" is neither positive nor negative")
    - }
  - switch statement
    - var grade:string = "A";
    - switch(grade) {
    - case "A": {
    - console.log("Excellent");
    - break;
    - }
    - case "B": {
    - console.log("Good");
    - break;
    - }
    - default: {
    - console.log("Invalid choice");
    - break;
    - }
    - }
- Loops:



5. Explain TypeScript Functions

- Functions are the **fundamental building block** of any applications in JavaScript.
- It **makes the code readable, maintainable, and reusable**.
- We can use it to build up layers of abstraction, mimicking classes, information hiding, and modules.
- In TypeScript, however, we have the concept of classes, namespaces, and modules, but functions still are an integral part in describing how to do things.
- TypeScript also allows adding new capabilities to the standard JavaScript functions to make the code easier to work.
- These are the main advantages of functions.
    - **Code reusability** :
    - **Less coding** :
    - **Easy to debug** :
- A function **without a name is known as an anonymous function**.
- These type of functions are **dynamically declared at runtime**.
- It is **defined as an expressio**n.
- We can store it in a variable, so it does not need function names.
- We can **invoke it by using the variable name**, which contains function.

6. Explain TypeScript Classes and Objects

- In ECMAScript 6, object-oriented class based approach was introduced.
- TypeScript introduced classes to avail the benefit of object-oriented techniques like encapsulation and abstraction.
- The class in TypeScript is compiled to plain JavaScript functions by the TypeScript compiler to work across platforms and browsers.
- A class definition can contain the following properties:
    - 1. Fields : It is a variable declared in a class.
    - 2. Methods : It represents an action for the object.
    - 3. Constructors : It is responsible for initializing the object in memory.
    - 4. Nested class and interface : It means a class can contain another class.
- **Syntax**: class <class_name>{
  field;
  Method;
  }
- **Example**: class Student {
  studCode: number;
  studName: string;
  constructor(code: number, name: string) {
  this.studName = name;
  this.studCode = code;
  }
  getGrade() : string { return "A+" ; } }

- A class creates an object by using the new keyword followed by the class name.
- The new keyword allocates memory for object creation at runtime.
- All objects get memory in heap memory area.
- **Syntax**
  - let object_name = new class_name(parameter)
- **new keyword** : It is used for instantiating the object in memory.
- The right side of the expression invokes the constructor, which can pass values.
- **Example**
  - Let obj = new Student();

TypeScript supports three **access modifiers** - public, private, and protected.

- **Public** - **By default - Public** members **are accessible everywhere without restrictions** even from the multiple level sub-classes without any compile errors.
- **Private** - A private member **cannot be accessed outside of its containing class**. Private members can be accessed only within the class and **even their subclasses won't be allowed to use** their private properties and attributes.
- **Protected** - A protected member **cannot be accessed outside of its containing class.** Protected members **can be accessed only within the class and by the instance of its sub/child class**.
- **Readonly** - using the **readonly keyword**. The Readonly properties **must be initialized at their declaration or in the constructor**. By using the readonly modifiers, we provide the read-only access modifier to mark a **class property as immutable**

7. Explain TypeScript Modules

- JavaScript has a concept of modules from ECMAScript 2015. TypeScript shares this concept of a module.
- A module is a way to create a group of related variables, functions, classes, and interfaces, etc.
- It **executes in the local scope, not in the global scope**.
- We can **create a module by using the export** keyword and can **use in other modules by using the import** keyword.
- At runtime, **the module loader is responsible for locating and executing** all dependencies of a module before executing it.
- We can divide the module into two categories:
  - 1. Internal Module:
    - were in the earlier version of Typescript.
    - **used for logical grouping** of the classes, interfaces, functions, variables into a single unit
    - internal modules are **obsolete**.
  - 2. External Module
    - External modules are also known as a **module**.
    - **used to specify the load dependencies** between the multiple external js files.
    - If the application has only one js file, the external module is not relevant.

8. Explain Inheritance in typescript.

- provides the ability of a program to create a new class from an existing class.
- acquires the **properties and behaviors of a class from another class**.
- The class whose members are inherited is called the **base class**
- the class that inherits those members is called the **derived/child/subclass.**
- TypeScript **supports only single inheritance and multilevel inheritance.**
- It **doesn't support multiple and hybrid inheritance**.
- We can use inheritance for **Method Overriding**
- **Single Inheritance:**

```
class Shape {
        Area:number
        constructor(area:number)
        {
                this.Area = area
        }
}
class Square extends Shape {
        display():void
        {
                console.log("Area of the square: "+this.Area)
        }
}
varobj = new Square(25);
obj.display()
```

Multilevel Inheritance:

```
class Animal {
        eat():void {
        console.log("Eating")
        }
}
class Dog extends Animal {
        bark():void {
        console.log("Barking")
        }
}
class BabyDog extends Dog{
        weep():void {
        console.log("Weeping")
        }
}
let obj = new BabyDog();
obj.eat();
obj.bark(); obj.weep();
```

9. Explain function overloading with suitable examples.

- TypeScript provides the concept of function overloading.
- You can have multiple functions with the **same name but different parameter types and return type**.
- However, the **number of parameters should be the same**.
- Function overloading with a **different number of parameters and different types along with the same function name is not supported**.
- **Advantages of function overloading:**
  - 1. It **saves the memory space**
  - 2. It **provides code reusability**
  - 3. It **increases the readability** of the program.
  - 4. **Code maintenance is easy**.
- **Example:**

```
function add(a:string, b:string):string;
function add(a:number, b:number): number;
function add(a: any, b:any): any {
        return a + b;
}
console.log(add("Hello ", "Steve")); // returns "Hello Steve"
console.log(add(10, 20)); // returns 30
```

10. Explain features in TS.

- **Cross-Platform :** TypeScript **runs on any platform** that JavaScript runs on.
- **Object-Oriented Language** : TypeScript provides powerful features such as **Classes, Interfaces, and Modules.**
- **Static type-checking :** TypeScript uses static typing. **This is done using type annotations**. **It helps type checking at compile time.**
- **Optional Static Typing :** TypeScript static typing is optional, if you prefer to use JavaScript's dynamic typing.
- **DOM Manipulation :** TypeScript can be used to manipulate the DOM.
- **ES 6 Features**: TypeScript includes most features of planned ECMAScript 2015 (ES 6, 7) such as **class, interface, Arrow functions** etc.

11. Typescript v/s Javascript.

Sr. No.	JavaScript	TypeScript
1.	It doesn't support strongly typed or static typing.	It supports strongly typed or static typing feature.
2.	Netscape developed it in 1995.	Anders Hejlsberg developed it in 2012.
3.	JavaScript source file is in ".js" extension.	TypeScript source file is in ".ts" extension.
4.	It is directly run on the browser.	It is not directly run on the browser.
5.	It is just a scripting language.	It supports object-oriented programming concept like classes, interfaces, inheritance, generics, etc.
6.	It doesn't support optional parameters.	It supports optional parameters.
7.	It is interpreted language that's why it highlighted the errors at runtime.	It compiles the code and highlighted errors during the development time.
8.	JavaScript doesn't support modules.	TypeScript gives support for modules.
9.	In this, number, string are the objects.	In this, number, string are the interface.
10.	JavaScript doesn't support generics.	TypeScript supports generics.

11.	Example:
	<script>
	function addNumbers(a, b) {
	return a + b;
	}
	var sum = addNumbers(15, 25);
	document.write('Sum of the numbers is: ' + sum);
	</script>
		Example:
		function addNumbers(a, b) {
		return a + b;
		}
		var sum = addNumbers(15, 25);
		console.log('Sum of the numbers is: ' + sum);

12. Var v/s Let

Sr. No.	Var	let
1.	The var keyword was introduced with JavaScript.	The let keyword was added in ES6 (ES 2015) version of JavaScript.
2.	It has global scope.	It is limited to block scope.
3.	It can be declared globally and can be accessed globally.	It can be declared globally but cannot be accessed globally.

4.	Variable declared with var keyword can be re-declared and updated in the same scope.
	Example:
	function varGreeter(){
	var a = 10;
	var a = 20; //a is replaced
	console.log(a);
	}
	varGreeter();
		Variable declared with let keyword can be updated but not re-declared.
		Example:
		function varGreeter(){
		let a = 10;
		let a = 20; //SyntaxError:
		//Identifier 'a' has already been declared
		console.log(a);
		}
		varGreeter();

5.	It is hoisted.
	Example:
	{
	console.log(c); // undefined.
	//Due to hoisting
	var c = 2;
	}
		It is not hoisted.
		Example:
		{
		console.log(b); // ReferenceError:
		//b is not defined
		let b = 3;
		}

13. Null v/s Undefined.

Null	Undefined
Null is the intentional absence of a value (null is explicit)	Undefined is the unintentional absence of a value (undefined is implicit)
Null must be assigned to a variable	The default value of any unassigned variable is undefined.
The typeof null is an object. (and not type null)	Typeof undefined is undefined type
You can empty a variable by setting it to null	You can Undefine a variable by setting it to Undefined
null is always falsy	undefined is always falsy
null is equal to undefined when compared with == (equality check)	
null is not equal to undefined when compared with === (strict equality check)	
When we convert null to a number it becomes zero	when we convert undefined to number it becomes NaN
You can represent undefined as a JSON (JavaScript Object Notation)	null is a valid value in JSON.

Module 3: Introduction to AngularJS

1. Discuss the Need of AngularJS in web sites.

- AngularJS is a **efficient framework that can create Rich Internet Applications (RIA)**.
- AngularJS **provides developers an options to write client side applications using JavaScript in a clean Model View Controller (MVC) way**.
- Applications written in AngularJS **are cross-browser compliant**. AngularJS **automatically handles JavaScript code suitable for each browser**.
- AngularJS is **open source, completely free, and used by thousands of developers around the world.** It is **licensed under the Apache license version 2.0**.

Core Features:

- **Data-binding** − It is the **automatic synchronization of data between model and view** components.
- **Scope** − These are **objects that refer to the model.** They act as a **glue between controller and view**.
- **Controller** − These are **JavaScript functions bound to a particular scope**.
- **Services** − AngularJS **comes with several built-in services such as $http**. These are **singleton objects which are instantiated only once** in app.
- **Filters** − These **select a subset of items from an array and returns a new array**.
- **Directives** − Directives are **markers on DOM elements.** Can be **used to create custom HTML tags that serve as new, custom widgets**. AngularJS **has built-in directives such as ngBind, ngModel**, etc.
- **Templates** − These are the **rendered view with information from the controller and model**. These **can be a single file** (such as index.html) **or multiple views in one page** using partials.
- **Routing** − It is concept of **switching views**.
- **Model View Whatever** − MVW is a **design pattern for dividing an application into different parts called Model, View, and Controller**. **AngularJS implement something closer to** MVVM (**Model-View-ViewModel**).
- **Deep Linking** − Deep linking **allows to encode the state of application in the URL** so that it can be bookmarked. The **application can then be restored from the URL to the same state**.
- **Dependency Injection** − AngularJS has a **built-in dependency injection subsystem** that **helps the developer to create, understand, and test the applications** easily.

2. Explain AngularJS modules.

- A module in AngularJS is a **container of the different parts of an application such as controller, service, filters, directives, factories** etc.
- It **supports separation of concern using modules**.
- Modules are **used to separate logic such as services, controllers, application etc. from the code** and maintain the code clean.
- We **define modules in separate js files**.
- A **module is used as a Main() method**.

File Name: app.js

```
var myApp = angular.module("myApp", []);
```

File Name: myController.js

```
myApp.controller("myController", function ($scope) {
   $scope.message = "Hello Angular World!";
});
```

3. Explain AngularJS built-in directives with example and syntax.

- Directives are **markers on a DOM element that tell AngularJS to attach a specified behavior to that DOM element** or even transform the DOM element and its children. In short, **it extends the HTML**.

Directive	Description
ng-app	Auto bootstrap AngularJS application.
ng-init	Initializes AngularJS variables
ng-model	Binds HTML control's value to a property on the $scope object.
ng-controller	Attaches the controller of MVC to the view.
ng-bind	Replaces the value of HTML control with the value of specified AngularJS expression.
ng-repeat	Repeats HTML template once per each item in the specified collection.
ng-show	Display HTML element based on the value of the specified expression.
ng-readonly	Makes HTML element read-only based on the value of the specified expression.
ng-disabled	Sets the disable attribute on the HTML element if specified expression evaluates to true.
ng-if	Removes or recreates HTML element based on an expression.
ng-click	Specifies custom behavior when an element is clicked.

```
<div id="myDiv" ng-app>
    {{5/2}}
<div>
```

```
<li ng-repeat="name in students">
        {{name}}
</li>
```

```
<div ng-app ng-init="greet='Hello World!'; amount= 100;
myArr = [100, 200]; person = { firstName:'Sachin', lastName
:'Shah'}">
```

```
<body ng-app>
<input type="text" ng-model="name" />
<div>
    Hello {{name}}
</div>
```

```
<body ng-app="">
<div>
    10 + 5 = <span ng-bind="10 + 5"></span><br />

    Enter your name: <input type="text" ng-model="name"
/><br />
    Hello <span ng-bind="name"></span>
```

```
<div>
    Disabled: <input ng-disabled="checked" type="text"
value="This is disabled." />
</div>
```

```
<button ng-click="count = count + 1" ng-init="count=0">
Increment
</button>

<div>The Current Count is {{count}}</div>
```

4. Explain AngularJS custom directives

- Custom directives are used **to extend the functionality of HTML**.
- Custom directives are **defined using "directive" function**.
- A custom directive **simply replaces the element for which it is activated**.
- AngularJS provides support to create **custom directives for following type of elements**.
    - (1) Element directives : activates when a **matching element is encountered**.
    - (2) Attribute : activates when a **matching attribute is encountered.**
    - (3) CSS : activates when a **matching css style is encountered**.
    - (4) Comment : activates when a **matching comment is encountered**.
- For understanding custom directive :
    - (1) Define custom html tags.
    - (2) Define custom directive to handle above custom html tags.
    - (3) Define controller to update the scope for directive.

```
<html>
<head>
<title>Angular JS Custom Directives</title>
</head>
<body>
<h2>AngularJS Sample Application</h2>
<div ng-app = "mainApp" ng-controller =
"StudentController">
<student name = "Nilesh"></student><br/>
<student name = "Sachin"></student>
</div>

<script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular
.min.js"></script>

<script>
      var mainApp = angular.module("mainApp", []);

      mainApp.directive('student', function() {
        var directive = {};
directive.restrict = 'E';
directive.template = "Student: <b>{{student.name}}</b>
, Roll No: <b>{{student.rollno}}</b>";
```

```
directive.scope = {
student : "=name"
        }

directive.compile = function(element, attributes) {
element.css("border", "1px solid #cccccc");

          var linkFunction = function($scope, element,
attributes) {
element.html("Student: <b>"+$scope.student.name
+"</b> , Roll No:
<b>"+$scope.student.rollno+"</b><br/>");
element.css("background-color", "gray");
          }
          return linkFunction;
        }

        return directive;
      });
      mainApp.controller('StudentController', function($scope)
{
        $scope.Nilesh = {};
        $scope.Nilesh.name = "Nilesh Patil";
        $scope.Nilesh.rollno  = 1;

        $scope.Sachin = {};
        $scope.Sachin.name = "Sachin Shah";
        $scope.Sachin.rollno  = 2;
      });
</script>

</body>
</html>
```

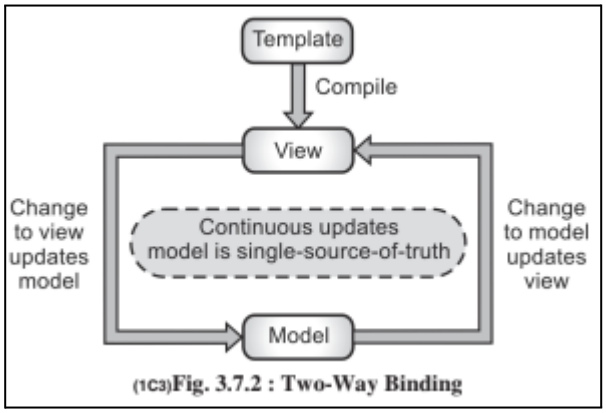
5. Discuss AngularJS expressions with syntax

- AngularJS expression is **surrounded with braces - {{ expression }}**.
- AngularJS **evaluates the specified expression and binds the result data to HTML**.
- AngularJS expression **can contain literals, operators and variables**.
- **cannot contain conditions, loops, exceptions or regular expressions**
- expression **cannot declare functions**.
- **cannot contain comma or void or return keyword**.
- **For example**, an expression {{2/2}} will produce the result 1 and will be bound to HTML.

```
<!DOCTYPE html>
<html >
<head>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body >
    <h1>AngularJS Expression Demo:</h1>
    <div ng-app>
        2 + 2 = {{2 + 2}} <br />
        2 - 2 = {{2 - 2}} <br />
        2 * 2 = {{2 * 2}} <br />
        2 / 2 = {{2 / 2}}
    </div>
</body>
</html>
```

6. Explain Angular JS Data Binding

- Data binding is a **very useful and powerful feature** used in software development technologies.
- It **acts as a bridge between the view and business logic** of the application.
- AngularJS **follows Two-Way data binding model**.
- In two way data binding, **any changes to the model are immediately reflected in the View** and **any changes in the View updates the model**.
- **Example**:



(1C3)**Fig. 3.7.2 : Two-Way Binding**

```
<div ng-app="" ng-init="firstName='Ajeet'">
<p>Input something in the input box:</p>
<p>Name: <input type="text" ng-model="firstName"></p>
<p>You wrote: {{ firstName }}</p>
</div>
```

Output

Input something in the input box:

Name:

You wrote:

Input something in the input box:

Name: Ajeet

You wrote: Ajeet

7. Explain AngularJS filters

- AngularJS Filters **allow us to format the data to display on UI without changing original format**.
- Filters can be used with an expression or directives **using pipe | sign**.
    - {{expression | filterName:parameter }}
- Angular **includes various filters to format data of different data types**.

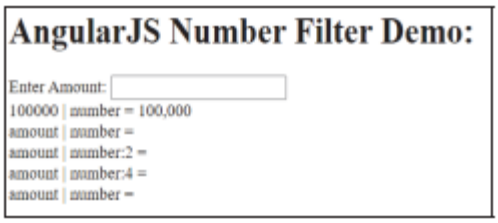
Filter Name	Description
Number	Formats a numeric data as text with comma and fraction.
Currency	Formats numeric data into specified currency format and fraction.
Date	Formats date to string in specified format.
Uppercase	Converts string to upper case.
Lowercase	Converts string to lower case.
Filter	Filters an array based on specified criteria and returns new array.
orderBy	Sorts an array based on specified predicate expression.
Json	Converts JavaScript object into JSON string
limitTo	Returns new array containing specified number of elements from an existing array.

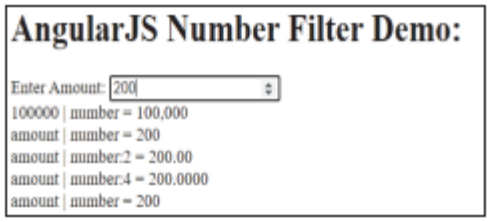
- **Example:**

```
100000 | number = {{100000 | number}} <br />
amount | number = {{amount | number}} <br />
amount | number:2 = {{amount | number:2}} <br />
amount | number:4 = {{amount | number:4}} <br />
```

```
  amount | number = <span ng-bind="amount |
number"></span>
</body>
</html>
```

Output





8. Explain AngularJS controllers

- AngularJS application **mainly relies on controllers to control the flow of data in the application**.
- A controller is **defined using ng-controller** directive.
- A controller is a JavaScript object that contains attributes/properties, and functions.
- **Each controller accepts $scope as a parameter**, which **refers to the application/module that the controller needs to handle**.

```
<div ng-app = "" ng-controller = "studentController">
  ...
</div>
```

```
<script>
        function studentController($scope) {
                $scope.student = {
                        firstName: "Sachin",
                        lastName: "Shah",

                        fullName: function() {
                                var studentObject;
                                studentObject = $scope.student;
                                return studentObject.firstName + " " +
                                studentObject.lastName;
                        }
                };
        }
</script>
```

9. Explain AngularJS scope

- The $scope in an AngularJS is a **built-in object, which contains application data and methods**.
- We **can create properties to a $scope object inside a controller** function **and assign a value or function to it**.
- The $scope is **glue between a controller and view (HTML)**.
- It **transfers data from the controller to view and vice-versa**.
- The **view can display $scope data using an expression, ng-model, or ng-bind directive**
- An **AngularJS application has a single $rootScope**. **All the other $scope objects are child objects**.
- The **properties and methods attached to $rootScope will be available to all the controllers**.

```
<body ng-app="myNgApp">
<h1>AngularJS $scope Demo: </h1>
<div ng-controller="myController">
    Message: <br />
    {{message}}<br />
<span ng-bind="message"></span><br />
<input type="text" ng-model="message" />
</div>
<script>
    var ngApp = angular.module('myNgApp', []);

    ngApp.controller('myController', function ($scope) {
      $scope.message = "Hello World!";
    });
</script>
```

10. Explain AngularJS dependency injection.

- Dependency Injection is a software design **in which components are given their dependencies instead of hard coding them** within the component.
- It **relieves a component from locating the dependency and makes dependencies configurable**.
- It also **helps in making components reusable, maintainable, and testable**.
- AngularJS provides a **supreme Dependency Injection mechanism**.
- It provides the following core components which can be **injected into each other as dependencies.**
    - **Value:** Value is a **simple JavaScript object**, which is **required to pass values to the controller during config phase**.

```
//define a module
var myModule = angular.module("myModule", []);
//create a value object and pass it a data.
myModule.value("numberValue", 100);
myModule.value("stringValue", "abc");
myModule.value("objectValue", { val1 : 123, val2 : "abc"} );
```

        - Here, values are defined **using the value() function** on the module.
    - **Factory**: Factory is a **function which is used to return value**. It **creates a value on demand whenever a service or a controller requires it.** It **generally uses a factory function to calculate and return the value**.

```
var myModule = angular.module("myModule", []);
myModule.factory("myFactory", function() {
   return "a value";
});
myModule.controller("MyController", function($scope,
myFactory) {
console.log(myFactory);
});
```

Injecting values into factory

To inject a value into AngularJS controller function, add a parameter with the same when the value is defined.

```
var myModule = angular.module("myModule", []);
myModule.value("numberValue", 100);
myModule.controller("MyController", function($scope,
numberValue) {
 console.log(numberValue);
});
```

- **Service**: Service is a **singleton JavaScript object containing a set of functions to perform certain tasks**. Service is **defined using service() function** and it is **then injected into the controllers**.

```
var mainApp = angular.module("mainApp", []);
...
//create a service which defines a method square to return square of a number.
mainApp.service('CalcService', function(MathService){
  this.square = function(a) {
    return MathService.multiply(a,a);
  }
});
//inject the service "CalcService" into the controller
mainApp.controller('CalcController', function($scope, CalcService, defaultInput) {
  $scope.number = defaultInput;
  $scope.result = CalcService.square($scope.number);
   $scope.square = function() {
    $scope.result = CalcService.square($scope.number);
  }
});
```

- **Provider**: Provider is **used by AngularJS internally to create services, factory, etc**. during the **config phas**e. The following script can be used to create MathService that we created earlier. Provider is a **special factory method with get() method** which is **used to return the value/service/factory**.

```
var mainApp = angular.module("mainApp", []);
//create a service using provider which defines a method square to return square of a number.
mainApp.config(function($provide) {
  $provide.provider('MathService', function() {
    this.$get = function() {
      var factory = {};

      factory.multiply = function(a, b) {
        return a * b;
      }
      return factory;
    };
  });
});
```

- **Constant**: Constants are **used to pass values at the config phase** considering the fact that **value cannot be used during the config phase**.

```
mainApp.constant("configParam", "constant value");
```

Example:

```
<div ng-app = "mainApp" ng-controller = "CalcController">
  <p>Enter a number: <input type = "number" ng-model = "number" /></p>
  <button ng-click = "square()">X<sup>2</sup></button>
  <p>Result: {{result}}</p>
</div>
  <script>
  var mainApp = angular.module("mainApp", []);

  mainApp.config(function($provide) {
    $provide.provider('MathService', function() {
      this.$get = function() {
        var factory = {};

        factory.multiply = function(a, b) {
          return a * b;
        }
        return factory;
      };
    });
  });

  mainApp.value("defaultInput", 5);

  mainApp.factory('MathService', function() {
    var factory = {};

    factory.multiply = function(a, b) {
      return a * b;
    }
    return factory;
  });
  mainApp.service('CalcService', function(MathService) {
    this.square = function(a) {
      return MathService.multiply(a,a);
    }
  });
mainApp.controller('CalcController', function($scope, CalcService, defaultInput) {
    $scope.number = defaultInput;
    $scope.result = CalcService.square($scope.number);
```

```
    $scope.square = function() {
      $scope.result = CalcService.square($scope.number);
    }
  });
</script>
```

11. Explain Angular JS Services

- AngularJS services are **JavaScript functions for specific tasks, which can be reused** throughout the application.
- AngularJS **built-in services starts with $**, same as other built-in objects.
- AngularJS **includes services for different purposes**. For example, $http service can be used to send an AJAX request to the remote server.

- $exceptionHandler
- $interval
- $rootScope
- $animate
- $filter
- $location $sce
- $http
- $timeout
- $log
- $controller
- $window
- $document

- All the Angular services are **lazy instantiated and singleton**.
- AngularJS framework **instantiates a service when an application component depends on it**.
- **$http**: one of the most common used. makes a request to the server, and lets your application handle the response. $http is a service as an object.

Method	Description
$http.get()	Perform Http GET request.
$http.head()	Perform Http HEAD request.
$http.post()	Perform Http POST request.
$http.put()	Perform Http PUT request.
$http.delete()	Perform Http DELETE request.
$http.jsonp()	Perform Http JSONP request.
$http.patch()	Perform Http PATCH request.

- **$log:** logs the messages to the browser's console.
- **$interval:** Executes the specified function on every specified millisecond's duration.
- **$filter** It is used to filter the array and object elements and return the filtered items.
- **$window:** Refers to the browser window object.

12. Design a registration / Feedback Form and perform Validation of fields.

```
<script>
        var app = angular.module('checkboxApp', []);
        app.controller('checkboxCtrl', function ($scope) {
                $scope.validationmsg = false;
                $scope.checkvalidation = function () {
                        var chkselct = $scope.chkselct;
                        if (chkselct == false || chkselct == undefined)
                                $scope.validationmsg = true;
                        else
                                $scope.validationmsg = false;
                }
        });
</script>

<body ng-app="checkboxApp" ng-controller="checkboxCtrl">
<h1>Student Information:</h1>
<form name="studentForm" novalidate>
        <label for="firstName">First Name</label><br />
        <input type="text" name="firstName" ng-model="student.firstName"
        ng-required="true" />
        <span style="color:red;" ng-show="studentForm.firstName.$touched &&
        studentForm.firstName.$error.required">First name is
        required.</span><br /><br />
        <label for="lastName">Last Name</label><br />
        <input type="text" name="lastName" ng-minlength="3" ng-maxlength="10" ng-
        model="student.lastName" />
        <span ng-show="studentForm.lastName.$touched &&
        studentForm.lastName.$error.minlength">min 3 chars.</span>
        <span style="color:red;" ng-show="studentForm.lastName.$touched &&
        studentForm.lastName.$error.maxlength">Max 10
        chars.</span><br /><br />
        <label for="dob">Email</label><br />
        <input type="email" id="email" ng-model="student.email" name="email" />
        <span style="color:red;" ng-show="studentForm.email.$touched &&
        studentForm.email.$error.email">Please enter valid email
        id.</span><br /><br />
        <label for="dob">DoB</label><br />
        <input type="date" id="dob" ng-model="student.DoB" /> <br /><br />
        <!-- <h1>{{student.DoB}}</h1> →
        <label for="gender">Gender</label> <br />
        <select id="gender" ng-model="student.gender">
        <option value="male">Male</option>
        <option value="female">Female</option>
```

```
</select><br /> <br />
<span>Training Type:</span><br />
<label><input value="online" type="radio" name="training" ng-
model="student.trainingType" />Online</label><br />
<label><input value="onsite" type="radio" name="training" ng-
model="student.trainingType" />OnSite</label>

<input type="submit" value="Submit" ng-click="checkvalidation();" />

</form>
```

13. Explain Routing using ng-Route

- Routing in AngularJS is **used when the user wants to navigate to different pages** in an application **but still wants it to be a single page** application.
- **Enable the user to create different URLs for different content in an application**.
- The **ngRoute module** helps **without reloading the entire applicatio**n.
- **$routeProvider** is **used to configure the routes**. **accepts either when() or otherwise() method**.
- The **ngRoute must be added as a dependency in the application modul**e

```
const app = angular.module("myApp", ["ngRoute"]);
```

- **Example:**

```
<script>
    var app = angular.module('ngRoutingDemo', ['ngRoute']);

    app.config(function ($routeProvider) {

      $routeProvider.when('/', {
        templateUrl: '/login.html',
        controller: 'loginController'
      }).when('/student', {
        template: '/student.html',
      }).when("/courses", {
          template : `<h1>Courses Offered</h1>
                        <ul>
                          <li>Machine Learning Foundation</li>
                        </ul>
       }).otherwise({
        redirectTo: "/"
      });
```

14. Explain ng-Repeat, ng-style, ng-view. With suitable example

- **ng-repeat:**

```
<body ng-app="app">
   <div ng-controller="controllerName">
      <ul>
         <li ng-repeat="student in students">{{student.name}}</li>
      </ul>
   </div>
<script>
   var app = angular.module("app", []);
   app.controller('controllerName', ['$scope', function ($scope) {
         $scope.students = [{ name: 'John' }, { name: 'Smith' }, { name:
'Allen' }, { name: ' Johnson' }, { name: 'Harris' }, { name: ' Williams' }, {
name: 'David' }];
   }]);
</script>
```

- **ng-style**

```
<body ng-app>
   <div>
      <button ng-click="style={color:'Red'}">Set Color</button>
                  <button   ng-click="style={backgroundColor:'yellow'}">Set
Background Color</button>
      <button ng-click="style={}">reset</button><br />
      <span ng-style="style">Sample text</span>
   </div>
</body>
```

- **ng-view:** The ng-view directive simply creates a place holder where a corresponding view can be placed based on the configuration.

```
<div ng-app = "mainApp">
         <div ng-view></div>
</div>
```

15. Discus Built-in Helper Functions

- **angular.isString:** returns true if the object or value given is of the type string
    - **Syntax:** angular.isString(value1)
    - **Examples**
        - angular.isString("hello") // true
        - angular.isString([1, 2]) // false
        - angular.isString(42) // false
- **angular.isArray**: Returns true if is of the type Array.
    - **Syntax:** angular.isArray(value)
    - **Examples**
        - angular.isArray([]) // true
        - angular.isArray([2, 3]) // true
        - angular.isArray(17) // false
- **angular.merge:** takes all the enumerable properties from the source object to deeply extend the destination object
    - **Syntax:** angular.isArray(value)
    - **Examples**
        - angular.merge({name: "king roland"}, {password: "12345"})
          // {name: "king roland", password: "12345"}
        - angular.merge({a: 1}, [4, 5, 6]) // {0: 4, 1: 5, 2: 6, a: 1}
- **angular.isDefined** tests a value if it is defined
    - **Syntax:** angular.isDefined(someValue)
    - **Examples**
        - angular.isDefined(42) // true
        - angular.isDefined(undefined) // false
- **angular.isUndefined** tests a value if it is defined
    - **Syntax:** angular.isUndefined(someValue)
    - **Examples**
        - angular.isUndefined(42) // false
        - angular.isUndefined(undefined) // true
- **angular.isDate** returns true if object passed to it is of the type Date.
    - **Syntax:** angular.isDate(value)
    - **Examples**
        - angular.isDate("lone star") // false
        - angular.isDate(new Date()) // true
- **angular.isNumber** returns true if Number, this **includes +Infinity, -Infinity and NaN**
    - **Syntax:** angular.isNumber(value)
    - **Examples**
        - angular.isNumber("23") // false
        - angular.isNumber(NaN) // true
        - angular.isNumber(Infinity) // true
- angular.isFunction, angular.toJson, angular.fromJson, isObject, forEach
    - angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(key))