University of Mumbai

Web X.0

(Course Code : ITC602)

Semester VI-Information Technology

Strictly as per the New Syllabus (REV-2019 'C' Scheme) of Mumbai University w.e.f. academic year 2021-2022

Dr. Nilesh M. Patil

Ph.D. (Computer Engineering)

Department of Information Technology

Fr. Conceicao Rodrigues College of Engineering

Bandra (West), Mumbai

Ms. Shraddha Subhash More

Assistant Professor,

Department of Information Technology

St. John College of Engineering and Management,

Palghar

TECH-NEO PUBLICATIONS

Where Authors Inspire Innovation

A Sachin Shah Venture

M6-104

Web X.0

(Course Code : ITC602)

(MU - Semester 6/ Information Technology – For New Syll. 2021-2022)

Authors : Dr. Nilesh M. Patil, Ms. Shraddha More

First Edition for New Syllabus : January 2022

Tech-Neo ID : M6-104

Published by

Mrs. Nayana Shah, Mr. Sachin S. Shah

Managing Director, B. E (Industrial Electronics)

An Alumunus of IIM Ahmedabad

&Mr. Rahul S. Shah

Permanent Address

Tech-Neo Publications LLP

Sr. No. 38/1, Behind Pari Company, Khedekar Industrial Estate, Narhe, Maharashtra, Pune-411041.

Email :info@techneobooks.com

Website :www.techneobooks.com

Printed at : Image Offset (Mr. Rahul Shah)

Dugane Ind. Area, Survey No. 28/25, Dhayari Near Pari Company, Pune - 411041. Maharashtra State, India.

E-mail : rahulshahimage@gmail.com

About Managing Director....
- Mr. Sachin Shah

Over 25 years of experience in Academic Publishing...

With over two and a half decades of experience in bringing out more than 1200 titles in Engineering, Polytechnic, Pharmacy, Computer Sciences and Information Technology,

Sachin Shah is a name synonymous with quality and innovative content.

A driven Educationalist...

1. A B.E. in Industrial Electronics (1992 Batch) from Bharati Vidyapeeth's College of Engineering, affiliated to University of Pune.
2. An Alumnus of IIM Ahmedabad.
3. A Co-Author of a bestselling book on "Engineering Mathematics" for Polytechnic Students of Maharashtra State.
4. Sachin has for over a decade, been working as a Consultant for Higher Education in USA and several other countries.

With path-breaking career...

A publishing career that started with handwritten cyclostyled notes back in 1992.

Sachin Shah has to his credit setting up and expansion of one of the leading companies in higher education publishing.

An experienced professional and an expert...

An energetic, creative & resourceful professional Sachin Shah's extensive experience of closely working with the best & the most eminent authors of Publishing Industry, ensures high standards of quality in contents.

This ability has helped students to attain better understanding and in-depth knowledge of the subject.

A visionary...

A gregarious person, **SACHINSHAH** is a thought leader who has been simplifying the methods of learning and bridging the gap between the best authors in the publishing industry and the student community for decades.

Preface

It delights us to write this book on **"Web X.0"** for the students of Mumbai University. This book has been strictly written as per the prescribed curriculum.

Every chapter of the book corresponds to the respective module mentioned in the syllabus. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.

We are thankful to **Mr. Sachin Shah, Managing Director of Tech Neo Publications**for his encouragement and support.

We are also thankful to the staff of Tech-Neo Publications for their timely efforts in the making of this book. We, together have taken every possible care to eliminate errors in the book. If they still exist, kindly let us know.

We are also thankful to our family, friends, colleagues, and students.

- Dr. Nilesh M. Patil

Ms. Shraddha Subhash More

Syllabus…

Mumbai University

B. E. (Information Technology)

Course Code	Course Name	Teaching Scheme(Contact Hours)			Credits Assigned			
		Theory	Practical	Tutorial	Theory	Practical/ Oral	Tutorial	Total
ITC602	Web X.0	03	- -	--	03	- -	- -	03

Course Code	Course Name	Examination Scheme							
		Theory					Term Work	Pract /Oral	Total
		Internal Assessment			End Sem Exam	Exam Duration (in Hrs)			
		Test1	Test2	Avg.					
ITC602	Web X.0	20	20	20	80	3	- -	- -	100

Course Objectives:

Sr. No.	Course Objectives
The course aims:	
1.	To understand the digital evolution of web technology.
2.	To learn Type Script and understand how to use it in web application.
3.	To empower the use of AngularJS to create web applications that depend on the Model-View-Controller Architecture.
4.	To gain expertise in a leading document-oriented NoSQL database, designed for speed, scalability, and developer agility using MongoDB.
5.	To build web applications quickly and with less code using Flask framework.
6.	To gain knowledge of Rich Internet Application Technologies.

Course Outcomes :

Sr. No.	Course Outcomes	Cognitive levels ofattainment as perBloom's Taxonomy
On successful completion, of course, learner/student will be able to:		
1.	Understand the basic concepts related to web analytics and semantic web.	**L1, L2**
2.	Understand how TypeScript can help you eliminate bugs in your code and enable you to scale your code.	**L1, L2**
3.	Understand AngularJS framework and build dynamic, responsive single-page web applications.	**L2, L3**
4.	Apply MongoDB for frontend and backend connectivity using REST API.	**L1, L2, L3**
5.	Apply Flask web development framework to build web applications with less code.	**L1, L2, L3**
6.	Develop Rich Internet Application using proper choice of Framework.	**L1, L2, L3, L4**

Prerequisite: Object Oriented Programming, Python Programming, HTML and CSS.

Sr. No.	Module	Detailed Content	Hours	CO Mapping
0	**Prerequisite**	HTML/HTML5 (Tags, Attributes and their properties),CSS/CSS3 (Types and Properties), Basics of Java Script, Python Programming.	02	- -
I	**Introduction to WebX.0**	**Evolution of WebX.0; Web Analytics 2.0:** Introduction to Web Analytics, Web Analytics 2.0, Clickstream Analysis, Strategy to choose your web analytics tool, Measuring the success of a website; **Web3.0 and Semantic Web:** Characteristics of Semantic Web, Components of Semantic Web, Semantic Web Stack, N-Triples and Turtle, Ontology, RDF and SPARQL. **Self-learning Topics :** Semantic Web Vs AI, SPARQL Vs SQL. **(Refer Chapter 1)**	04	CO1
II	**Type Script**	Overview, TypeScript Internal Architecture, TypeScript Environment Setup, TypeScript Types, variables and operators, Decision Making and loops, TypeScript Functions, TypeScript Classes and Objects, TypeScript Modules **Self-learning Topics :** Javascript Vs TypeScript**.(Refer Chapter 2)**	06	CO2
III	**Introduction to AngularJS**	Overview of AngularJS, Need of AngularJS in real web sites, AngularJS modules, AngularJS built-in directives, AngularJS custom directives, AngularJS expressions, Angular JS Data Binding, AngularJS filters, AngularJS controllers, AngularJS scope, AngularJS dependency injection, Angular JS Services, Form Validation, Routing using ng-Route, ng-Repeat, ng-style, ng-view, Built-in Helper Functions, Using Angular JS with Typescript. **Self-learning Topics:** MVC model, DOM model, Javascript functions and Error Handling. **(Refer Chapter 3)**	08	CO3
IV	**MongoDB and Building REST API using MongoDB**	**MongoDB :** Understanding MongoDB, MongoDB Data Types, Administering User Accounts, Configuring Access Control, Adding the MongoDB Driver to Node.js, Connecting to MongoDB from Node.js, Accessing and Manipulating Databases, Manipulating MongoDB Documents from Node.js, Accessing MongoDB from Node.js, Using Mongoose for Structured Schema and Validation. **REST API :** Examining the rules of REST APIs, Evaluating API patterns, Handling typical CRUD functions (create, read, update, delete), Using Express and Mongoose to interact with MongoDB, Testing API endpoints **Self-learning Topics :** MongoDB vs SQL DB **(Refer Chapter 4)**	08	CO4
V	**Flask**	Introduction, Flask Environment Setup, App Routing, URL Building, Flask HTTP Methods, Flask Request Object, Flask cookies, File Uploading in Flask. **(Refer Chapter 5)**	06	CO5
VI	**Rich Internet Application**	**AJAX :** Introduction and Working **Developing RIA using AJAX Techniques :** CSS, HTML, DOM, XML HTTP Request, JavaScript, PHP, AJAX as REST Client **Introduction to Open Source Frameworks and CMS for RIA :** Django, Drupal, Joomla. **Self-learning Topics :** Applications of AJAX in Blogs, Wikis and RSS Feeds. **(Refer Chapter 6)**	05	CO6

Assessment :

Internal Assessment (IA) for 20 marks :

IA will consist of Two Compulsory Internal Assessment Tests. Approximately 40% to 50% ofsyllabus content must be covered in First IA Test and remaining 40% to 50% of syllabus content must be covered in second IA Test.

Question paper format

- Question Paper will comprise of a total of six questions each carrying 20 marksQ.1 will be compulsory and should cover maximum contents of the syllabus.
- Remaining questions will be mixed in nature (part (a) and part (b) of each question must be from different modules. For example, if Q.2 has part (a) from Module 3 then part (b) must be from any other Module randomly selected from all the modules)
- A total of four questions need to be answered

Index

❑❑❑

MODULE 1

CHAPTER 1

INTRODUCTION TO WEB X.0

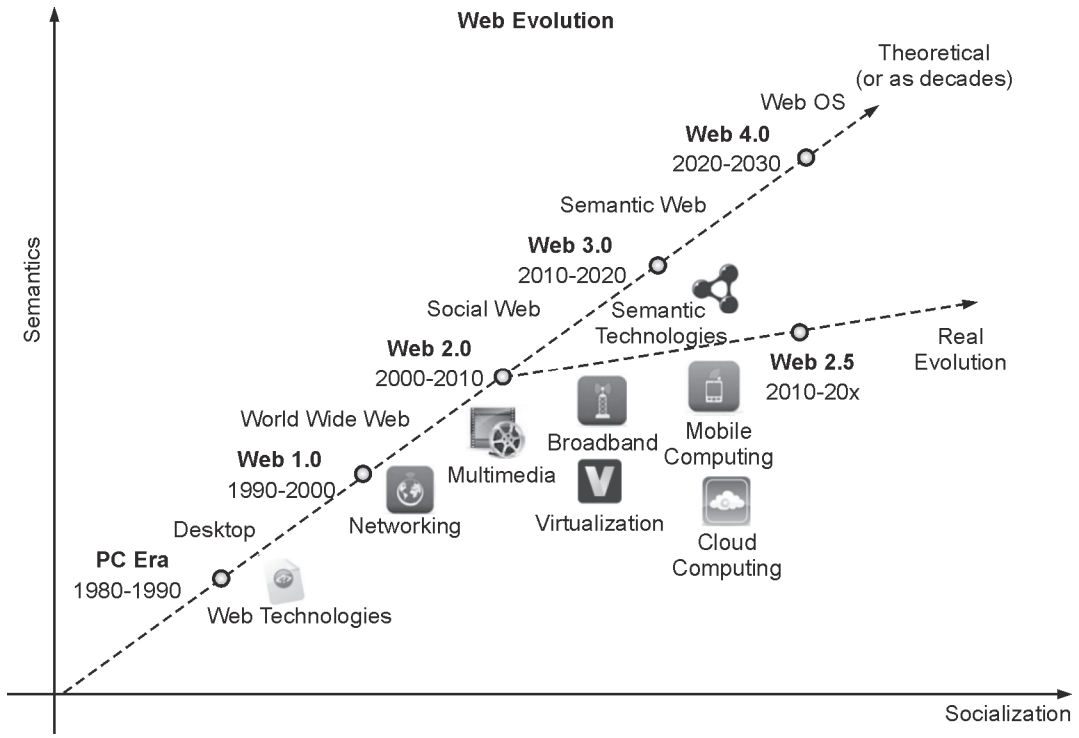
University Prescribed Syllabus w.e.f Academic Year 2021-2022

Evolution of WebX.0; Web Analytics 2.0: Introduction to Web Analytics, Web Analytics 2.0, Clickstream Analysis, Strategy to choose your web analytics tool, Measuring the success of a website; Web3.0 and Semantic Web: Characteristics of Semantic Web, Components of Semantic Web, Semantic Web Stack, N-Triples and Turtle, Ontology, RDF and SPARQL.

Self-learning Topics : Semantic Web Vs AI, SPARQL Vs SQL.

1.1 EVOLUTION OF WEBX.0

World Wide Web is the primary tool used by billions of people to share, read, and write information to interact with other people via internet. World Wide Web made much progress since its advent. In this section, we will be sharing a brief idea of the evolution happened in web from web 1.0 to 2.0 and to web 3.0.



(1A1)**Fig. 1.1.1 : Evolution of WebX.0**

1.1.1 Web1.0

- Web 1.0 refers to the first stage of the World Wide Web evolution.
- Earlier, there were only a few content creators in Web 1.0 with a huge majority of users who were consumers of content.
- Personal web pages were common, consisting mainly of static pages hosted on ISP-run web servers, or on free web hosting services.
- In Web 1.0, advertisements on websites while surfing the internet were banned.
- Also, in Web 1.0, Ofoto was an online digital photography website, on which users could store, share, view, and print digital pictures.
- Web 1.0 is a content delivery network (CDN) that enables the showcase of the piece of information on the websites. It can be used as a personal website.
- It costs the user as per pages viewed. It has directories that enable users to retrieve a particular piece of information.
- Four design essentials of a Web 1.0 site include:
  1. Static pages.
  2. Content is served from the server's file system.
  3. Pages built using Server Side Includes or Common Gateway Interface (CGI).
  4. Frames and Tables are used to position and align the elements on a page.

1.1.2 Web2.0

- Web 2.0 refers to worldwide websites which highlight user-generated content, usability, and interoperability for end users.
- Web 2.0 is also called the participative social web.

- It does not refer to a modification to any technical specification, but to modify the way Web pages are designed and used.
- The transition is beneficial but it does not seem that when the changes occur.
- Interaction and collaboration with each other are allowed by Web 2.0 in a social media dialogue as the creator of user-generated content in a virtual community.
- The web browser technologies are used in Web 2.0 development and it includes AJAX and JavaScript frameworks.

☞ **Five major features of Web 2.0**

1. Free sorting of information, permits users to retrieve and classify the information collectively.
2. Dynamic content that is responsive to user input.
3. Information flows between the site owner and site users by means of evaluation & online commenting.
4. Developed APIs to allow self-usage, such as by a software application.
5. Web access leads to concern different, from the traditional Internet user base to a wider variety of users.

1.1.3 Web3.0

- It refers to the evolution of web utilization and interaction which includes altering the Web into a database.
- It enabled the up-gradation of the back-end of the web, after a long time of focus on the front-end (Web 2.0 has mainly been about AJAX, tagging, and another front-end user-experience innovation).
- Web 3.0 is a term that is used to describe many evolutions of web usage and interaction among several paths. In this, data isn't owned but instead shared, where services show different views for the same web / the same data.
- The Semantic Web (3.0) promises to establish "the world's information" in a more reasonable way than Google can ever attain with their existing engine schema. This is particularly true from the perspective of machine conception as opposed to human understanding.
- The Semantic Web necessitates the use of a declarative ontological language like OWL to produce domain-specific ontologies that machines can use to reason about information and make new conclusions, not simply match keywords.

Below are 5 main features that can help us define Web 3.0 :

1. Semantic Web

The succeeding evolution of the Web involves the Semantic Web. The semantic web improves web technologies in demand to create, share and connect content through search and analysis based on the capability to comprehend the meaning of words, rather than on keywords or numbers.

2. Artificial Intelligence

Combining this capability with natural language processing, in Web 3.0, computers can distinguish information like `humans in order to provide faster and more relevant results. They become more intelligent to fulfill the requirements of users.

3. 3D Graphics

The three-dimensional design is being used widely in websites and services in Web 3.0. Museum guides, computer games, e-commerce, geospatial contexts, etc. are all examples that use 3D graphics.

4. Connectivity

With Web 3.0, information is more connected thanks to semantic metadata. As a result, the user experience evolves to another level of connectivity that leverages all the available information.

5. Ubiquity

Content is accessible by multiple applications, every device is connected to the web, the services can be used everywhere.

1.1.4 Comparison between Web 1.0, Web 2.0 and Web 3.0

	Web 1.0	Web 2.0	Web 3.0
Definition (According to Tim Berners-Lee)	Read-only	Read-write	Read-write-execute
Technologies associated with the era	• File and Web Servers • Content and Enterprise Portals • Search Engines (AltaVista, Yahoo!) • E-mail (Yahoo!, Hotmail) • P2P File Sharing (Napster, BitTorrent) • Publish and Subscribe Technologies	• Ajax and JavaScript frameworks • Adobe Flex • Enterprise Java, Microsoft.NET Framework (Server side) • Blogs • Wikis • Instant Messaging	• Semantic Searching • Knowledge Bases • Ontologies • Personal Intelligent Digital Assistants
Precedence Order	First Stage	Second Stage	Third Stage
Type of Web	Simply Web	Social Web	Semantic Web
No. of users	Millions	Billions	Trillons
Basic concept	Connect information	Connect people	Connect knowledge
Associated websites	CNN	Flikr, YouTube, Blogger	Google Maps, My Yahoo!
Years	1990-2000	2000-2010	2010-2020
Features	• Hyper linking and bookmarking on pages. • No communication between server and user. • Websites were Static. • It allowed only browsing of content.	• Better interaction. • Includes functions like Video streaming, Online documents, etc. • Introduction of web applications. • Everything becomes online and stores on servers.	• Smart, web based applications and functionalities. • An amalgamation of Web technology and Knowledge Representation (KR).

1.1.5 Web4.0

- Web 4.0 can be considered as an Ultra-Intelligent Electronic Agent, Symbiotic web and Ubiquitous web.
- Interaction between humans and machines was motive behind the symbiotic web. It is powerful as human brain.
- Progress in the development of telecommunication, advancement on nanotechnology in the world and controlled interfaces are using web 4.0.
- In simple words, machines would be clever on reading the contents of the web, and react in the form of executing and deciding what to execute first to load the websites fast with superior quality and performance and build more commanding interfaces.
- Web 4.0 will be read write concurrency web. It ensures global transparency governance, distribution, participation, collaboration in to key communities such as industry, political, social and other communities.
- Web OS will be such as a middleware in which it will start functioning like an operating system. Web OS will

be parallel to the human brain and implies a massive web of highly intelligent interaction.

1.1.6 Web 5.0

- Web 5.0 is still an underground idea in progress and there is no exact definition of how it would be.
- Web 5.0 can be considered as Symbiotic web, decentralized i.e. it is not possible to have a Personal Server (PS) for any personal data or information stored on the net, and people tries to get interconnected via Smart Communicator (SC), like Smart phones,
- Tablets or Personal Robots which is represented as its own avatar inside the SC, will be able to surf alone in the 3D Virtual world of the Symbiotic.
- The Symbiotic servers will be able to use a part of "memory and calculation power" of each interconnected SC, in order to calculate the billions and billions needed data to build the 3D world, and to feed its Artificial Intelligence surf alone.
- Currently the Web is "emotionally" neutral: do not feel the user perceives. The company Emotive Systems has created neuro-technology through headphones that allow users to interact with content that meets their emotions or change in real time facial expression an "avatar".

1.2 WEB ANALYTICS 2.0

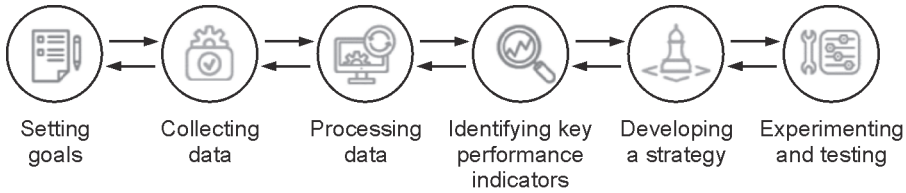
- If you're a developer who runs their own site, or you have a blog hosted on a blogging platform, it's likely that you'll be well aware of web analytics.
- And while awareness of this extremely useful tool is good, many people don't quite know how to fully harness the power of web analytics.
- The first thing that springs to mind when starting a website is how many visitors you will get. And the next step involves how these visitors might be converted into paying customers. At the end of the day, you want to understand the people who are coming to your site, and anticipate their needs and desires.
- Web analytics is a powerful tool for any business with a website or an online presence. By monitoring how prospective customers and visitors interact with your online resources, you can move on to tailoring these experiences with the aim of increasing your sales, clicks, and conversions.
- On the surface, it can seem like a really daunting world to negotiate and explore. However, there are many different resources out there that can help you make sense of web analytics.

1.2.1 Introduction to Web Analytics

- Web analytics is the process of analyzing the behavior of visitors to a website. This involves tracking, reviewing and reporting data to measure web activity, including the use of a website and its components, such as webpages, images and videos.
- Data collected through web analytics may include traffic sources, referring sites, page views, paths taken and conversion rates. The compiled data often forms a part of customer relationship management analytics (CRM analytics) to facilitate and streamline better business decisions.
- Web analytics enables a business to retain customers, attract more visitors and increase the dollar volume each customer spends.
- Analytics can help in the following ways :
  1. Determine the likelihood that a given customer will repurchase a product after purchasing it in the past.
  2. Personalize the site to customers who visit it repeatedly.
  3. Monitor the amount of money individual customers or specific groups of customers spend.
  4. Observe the geographic regions from which the most and the least customers visit the site and purchase specific products.
  5. Predict which products customers are most and least likely to buy in the future.
- The objective of web analytics is to serve as a business metric for promoting specific products to the customers who are most likely to buy them and to determine which products a specific customer is most likely to purchase. This can help improve the ratio of revenue to marketing costs.
- In addition to these features, web analytics may track the clickthrough and drilldown behavior of customers within a website, determine the sites from which customers most often arrive, and communicate with browsers to track and analyze online behavior. The results of web analytics are provided in the form of tables, charts and graphs.

1.2.2 Web Analytics Process

The web analytics process involves the following steps as depicted in Fig. 1.2.1.



(1A2)**Fig. 1.2.1 : Web Analytics Process**

1. **Setting goals :** The first step in the web analytics process is for businesses to determine goals and the end results they are trying to achieve. These goals can include increased sales, customer satisfaction and brand awareness. Business goals can be both quantitative and qualitative.
2. **Collecting data :** The second step in web analytics is the collection and storage of data. Businesses can collect data directly from a website or web analytics tool, such as Google Analytics. The data mainly comes from Hypertext Transfer Protocol requests including data at the network and application levels and can be combined with external data to interpret web usage. For example, a user's Internet Protocol address is typically associated with many factors, including geographic location and clickthrough rates.
3. **Processing data ;** The next stage of the web analytics funnel involves businesses processing the collected data into actionable information.
4. **Identifying key performance indicators (KPIs) :** In web analytics, a KPI is a quantifiable measure to monitor and analyze user behavior on a website. Examples include bounce rates, unique users, user sessions and on-site search queries.
5. **Developing a strategy :** This stage involves implementing insights to formulate strategies that align with an organization's goals. For example, search queries conducted on-site can help an organization develop a content strategy based on what users are searching for on its website.
6. **Experimenting and testing :** Businesses need to experiment with different strategies in order to find the one that yields the best results. For example, A/B testing is a simple strategy to help learn how an audience responds to different content. The process involves creating two or more versions of content and then displaying it to different audience segments to reveal which version of the content performs better.

1.2.3 Types of Web Analytics

The two main categories of web analytics are off-site web analytics and on-site web analytics.

1. Off-site web analytics

- The term off-site web analytics refers to the practice of monitoring visitor activity outside of an organization's website to measure potential audience.
- Off-site web analytics provides an industrywide analysis that gives insight into how a business is performing in comparison to competitors.
- It refers to the type of analytics that focuses on data collected from across the web, such as social media, search engines and forums.

2. On-site web analytics

- On-site web analytics refers to a narrower focus that 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 and can include details on site engagement, such as what content is most popular.
- Two technological approaches to on-site web analytics include log file analysis and page tagging.
- **Log file analysis**, also known as log management, is the process of analyzing data gathered from log files to monitor, troubleshoot and report on the performance of a website. Log files hold records of virtually every action taken on a network server, such as a web server, email server, database server or file server.
- **Page tagging** is the process of adding snippets of code into a website's Hyper Text Markup Language code using a tag management system to track website visitors and their interactions across the website. These

snippets of code are called tags. When businesses add these tags to a website, they can be used to track any number of metrics, such as the number of pages viewed, the number of unique visitors and the number of specific products viewed.

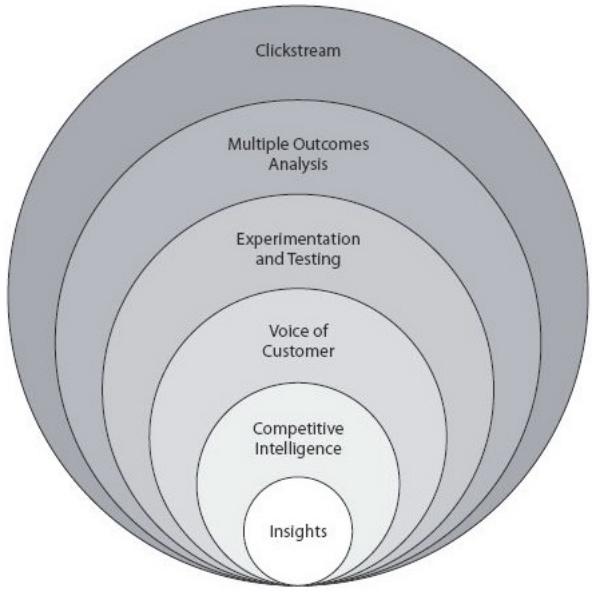
1.2.4 Web Analytics Tools

- Web analytics tools report important statistics on a website, such as where visitors came from, how long they stayed, how they found the site and their online activity while on the site.
- In addition to web analytics, these tools are commonly used for product analytics, social media analytics and marketing analytics.
- Some examples of web analytics tools include the following :

1. **Google Analytics :** Google Analytics is a web analytics platform that monitors website traffic, behaviours and conversions. The platform tracks page views, unique visitors, bounce rates, referral Uniform Resource Locators, average time on-site, page abandonment, new vs. returning visitors and demographic data.
2. **Optimizely :** Optimizely is a customer experience and A/B testing platform that helps businesses test and optimize their online experiences and marketing efforts, including conversion rate optimization.
3. **Kissmetrics :** Kissmetrics is a customer analytics platform that gathers website data and presents it in an easy-to-read format. The platform also serves as a customer intelligence tool, as it enables businesses to dive deeper into customer behavior and use this information to enhance their website and marketing campaigns.
4. **Crazy Egg :** Crazy Egg is a tool that tracks where customers click on a page. This information can help organizations understand how visitors interact with content and why they leave the site. The tool tracks visitors, heatmaps and user session recordings.

1.2.5 Web Analytics 2.0 Model

Fig. 1.2.2 illustrates the model for web analytics 2.0.



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

Let's elaborate each of the terms shown in Fig. 1.2.2.

- **Clickstream :** The term "clickstream" refers to the collection, storage, processing, and analysis of click-level data on your website. Webtrends, Google Analytics, and other Clickstream products provide click-level data. Clickstream data is also foundational; it aids in the 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.
- **Multiple Outcome Analysis :** Measuring Outcomes entails linking customer behaviour to the company's objectives. The most effective application of web analytics will be to link outcomes to earnings and bonuses for your report recipients. Only three types of outcomes are attempted to be delivered by a website: Boost revenue; cut costs; and improve customer satisfaction and loyalty. Regardless of whether your website is for ecommerce, tech support, social media, or just general propaganda, everything you do on it must deliver on these three Outcomes.
- **Experimentation and Testing :** We can improve the services on web by running live experiments on the websites. Businesses can initially post several ideas on their websites to collect feedback and responses from their clients. 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 :** The information gathered by the web analytics tool is limited to the pages that visitors visited, the content they accessed, the products they purchased, the services they subscribed to, and so on. If a user leaves a page, though, you must determine what kind of information he or she was looking for. In this scenario, the customer's voice is important. Customers' feedback can be gathered using an online survey.
- **Competitive Intelligence :** Competitive intelligence means analysing competitors by assessing their strengths and weaknesses. You must also analyse the business strategies of your competitors for long-term business growth. 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.

1.2.6 Clickstream Analysis

- Clickstream analytics is the 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 :** 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

1.2.7 Terms Associated with Clickstream Analysis

There are various metrics used to define the clickstream. These metrics are as follows :

- **Visits :** It means the number of times a site is visited, no matter the number of unique visitors that make up those sessions.
- **Visitor :** Also called a unique visitor; is an individual visiting a website during a period of time.
- **Unique Visitor :** Refers to the number of distinct individuals who request pages from a website during a specific period, no matter how many times they visit.
- **Returning Visitor :** Refers to a visitor who can be identified with multiple visits, through cookies or authentication.
- **Pageviews per visit :** Refers to the average number of pageviews per visit over a given time period.
- **Pageviews :** Refers to the instance of an Internet user visiting a particular page on a site. A pageview is recorded whenever a full page of your website is viewed or refreshed.
- **Time on Page :** It represents the time spent on each page of a website. It is calculated by dividing the total time that visitors spend on a website to the total number of visits.
- **Time on Site :** It is calculated by taking the timestamp of the final page of the visit (the Exit Page), and subtracting the timestamp of the first page.
- **Bounce Rate :** Also called an exit rate. It refers to the percentage of visitors who leave a website without visiting more than one page of the website. The bounce rate for a website must be as minimum as possible. High bounce rate means that the content of the website is not relevant for visitors.
- **Conversion Rate :** It is the total number of visitors to a website who turn into the customers or subscribers.
- **Engagement :** The ability of a website to draw the attention of a user or visitor is engagement. The content of the website must be unique and attractive such that the visitor gets what he/she is looking for from the website. Engagement includes the frequency of visits, user action, pages visited per session and average session duration.

1.2.8 Strategy to Choose Your Web Analytics Tool

- Web analytics is very crucial for the success of any business that has a website.
- It provides important data can be used to evaluate the progress of a business and thus make appropriate improvements.
- However, this requires choosing the right web analytics software for your business.
- With the numerous solutions in the market, choosing one can be a very daunting task.

Here are some of the things one need to consider when choosing a 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.
- Some of the metrics one might want to measure include website traffic, online sales, SEO ranking, sales lead quality, customer engagement, subscriber growth, inbound links and time spent on site.
- 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. This way, one will be able to analyze their results at a glance.
- Look for a web analytics solution 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

- Different web analytic solutions some with different prices. 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

- It would be advisable to work with a web analytics vendor that has been in business for several years and is sound financially.
- 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 with their services, then it 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 which one can use to try out the solution. Does the solution have an appealing touch and feel? Is it intuitive enough? Does it provide all the metrics that business requires? Consider such questions before making a decision.

1.2.9 Measuring the Success of a Website

1.2.9(A) Measuring the Success of Ecommerce Website

There are an increasing number of e-commerce businesses today with numerous products and services online. With the rise in number there is also a rise in competition. In order to stay ahead of the competition, data is important. Initially you will focus on building your website in order to launch your business. After the launch, you need to start focusing on analysing your e-commerce metrics.

Keeping a track of these key metrics will help your business thrive. With the help of these metrics you can analyse the performance of your business. Measuring the right e-commerce metrics can help you make informed business decisions. It can enable you to see which areas need work and optimisation.

Here are some key e-commerce metrics that will help you measure the success of your business.

- **Conversion rate :** Conversion rate is the number of customers that complete a sale after visiting your site and viewing a product. Conversion rate is closely tied to overall revenue metrics. It represents the actual sales based on customers viewing your products.
- **Gross margin :** Gross margin is the actual profit you earn on top of the cost of goods sold (COGS). This is essentially your profit on the product after sale, factoring how much you spent on the inventory yourself. Knowing how much you earn on each sale is extremely important to ensure you are growing and scaling properly. This is a core metric to compare to other benchmarks to gain insight into how sustainable your growth is.
- **Average order value :** The average order value (AOV) is the monetary value of an average customer order on your site. You will also want to track the average abandoned order value (AAOV), which is the average value of an order that's abandoned during either the checkout or cart stages of purchase. Track AOV as a whole, and then segment by device type, platforms, and traffic sources. Identifying the sources of your highest AOV customers will help you run more ROI-positive marketing campaigns.
- **Cost per acquisition :** The cost per acquisition is how much it costs to gain a new customer. This will include advertising costs, email campaigns, discounts offered, and anything else it took to actually make the sale to a customer. This gives you an idea of how much effort and money it takes to actually get a paying customer.
- **Cart abandonment rate :** The cart abandonment rate is the percent of customers that add an item to cart and then abandon the purchase. The number of items added to the cart, the value of items added to cart, and how long they shop are all important as well. This metric gives you insight into how many customers are interested in products but do not proceed to buy.
- **Checkout abandonment rate :** The checkout abandonment rate is the percent of customers that initiate checkout and then abandon purchase. This is a step further than cart abandonment, which means you've expended more effort to get them to that stage. It's important to capitalize at checkout, as customers are closest to making a purchase at this point. This gives you specific information about incomplete transactions after customers are poised to buy. You will want to analyze and develop strategies to decrease this rate over time, as it can have a major impact on your revenue.
- **Customer lifetime value :** Customer Lifetime Value (CLV) is the amount of revenue on average that you earn per person throughout their entire lifetime as a customer for your company. It essentially breaks down how much a customer is worth to you on average, and makes a distinction between a customer who makes a $200 purchase now, and one who makes ten $100 purchases over the next five years - who is actually more valuable to you. You need this to be greater than acquisition costs for you to be earning revenue.
- **Revenue on advertising spent :** Divide revenue by the cost of advertising to get a ratio of the average revenue returned based on advertising costs. This value is how much you earn on each advertising dollar spent.
- **Device type :** How customers access your store is important for giving them an ideal experience. Segment your ecommerce analytics by device type to gain further insight into which devices are being used more frequently, and tailor your site's UX to each device type.

- **Site speed :** Website speeds and checkout load times are important for maintaining and engaging customers on your ecommerce site. The performance, efficiency, and speed of your ecommerce platform is core to your shoppers' experience.
- **Customer Engagement :** The level of engagement your customers have with your service, often measured using reactions, shares, and subscriptions. The more active and interested customers are, the more likely they are to purchase, share, and interact with your content.
- **Bounce rate :** The bounce rate is the percentage of customers that abruptly leave your site after visiting only one page. This usually indicates significant UX issues, such as page load times, appearance, and navigation delays that cause users to seek another site. This can also be a sign of poor marketing targeting, since the customers brought in by your campaigns aren't finding value in the site or products for sale.
- **Customer survey results and feedback :** Use customer surveys regularly to get consistent feedback. Run surveys to get feedback after site upgrades, major changes to the site, or whenever new features are introduced.
- **Customer product reviews on your site and online :** Reviews on a product page can mean the difference between a purchase and an abandoned cart. Featuring product reviews on your own site is the preferred option, since shoppers are more likely to trust the opinion of fellow customers than marketing text. However, third-party review sites will also work in a pinch.
- **Customer service calls, chats, and emails :** Collect data on how many customer service communications are initiated. Look at this value as a whole and segment it by each unique channel of communication, including calls, chats, and emails.
- **Average ticket resolution time :** Also known as a service-level agreement (SLA) time. Track both the average SLA, and the average time from ticket creation to resolution (open to close). Treat these times as sacred: the longer it takes for shopper problems to be solved, the less likely they are to become a repeat customer.
- **Traffic source :** Identify the main channel by which shoppers find you to help improve your marketing and advertising efforts. Sources include paid, organic search, referrals, direct, and others.
- **Traffic volume :** Site traffic is a measure of activity on your website, and is measured in visitors (users) and times visiting the site (sessions). Site traffic can be studied as a whole, or further broken down into more specific metrics, like page views per visit, time on site, bounce rate, and new versus returning customers.
- **Social media engagement (followers, reactions, and shares) :** Social media interaction, including reactions, shares, and subscriptions indicate the level of engagement your customers have with your site. They also provide a direct line of communication, allowing customers to give feedback on what they like and dislike about your product.
- **Valid email collection rate :** The number of valid emails you are able to capture is helpful to understand how many customers are interested in your service and willing to provide a method of communication. You can further break this down by device type and the stage in the process they give you their email (registration, checkout, etc.).
- **Blog views and shares :** If your ecommerce site has a blog attached, the view and share rate of the blog is a good signal of the usefulness of your content. While virality can have value, building a consistent, loyal readership is a better way to build sustainable growth and a reliable revenue stream.

1.2.9(B) Measuring the success of Non-Ecommerce Website

- Normally, the web analytics knowledge base mainly concentrates in the KPIs. (Key Performance indicators) for ecommerce sites.
- It is mainly concerned with the orders, conversion rates, transactions and revenue for e-commerce sites.
- Non-ecommerce sites are generally outlooked in the more prominent world of ecommerce websites. For all the non-ecommerce website owners, here are some KPIs that are concerned with non-ecommerce websites.

- Before going into detail, let us consider the definition of none-commerce websites. They are either lead generation sites or pure content sites, where visitors come for the sake of consuming content and they leave without any business transactions.
- It can be a technical support site, a blog, a site full of white papers, a social networking site or a brand website.

Now, let us have a look into the major KPIs that satisfies the executive managers of a non-ecommerce website.

1. Visitor Loyalty

- It is defined as the frequency of visits on website for a specific time period. One will get the data for the average visits per visitor.
- For increasing the visitor loyalty for one's site, set a target of a particular number of visitors that one should achieve over a specific time period and analyze the trends of the progress towards the goal set by website owner.
- Compare performance over the period and ensure that they are making progress in achieving the predefined percentage of web traffic.
- It will help to know about the characteristics of visitors. One should update their website frequently depending upon web analytics for increased web traffic.

2. Recency

- It is the number of repeated visits on the website. The greater the recency of visits, the better for the site.
- If owner have set the target of the recency of visits KPIs to be one day ago, it means that he/she want visitors to visit their site every day to enjoy the so-called valuable content on the site.

3. The length of visit

- It refers to the quality of visits during a reporting period depending on how long each visit lasts.
- It is the most common and the most prominent KPIs used in none e-commerce sites.
- If by analyzing the web analytics report, one gets a clear picture of the distribution of this KPI, then it is easier to proceed further.
- Give importance to the visitor satisfaction and they will definitely stay on website for a long period and even until eternity. Identify own goals and measure this KPI for success.

4. The depth of visit

- It is mainly concerned with the number of pages consumed by the visitor in the reporting time period.
- It is important to note that most of the websites are now web pages and not a single page. It is more powerful than the average page views per visitor.
- It helps us to have a broad picture of our customer experience. Here also it helps to improve our goals and measure success.

1.3 WEB 3.0 AND SEMANTIC WEB

- Web 3.0 was originally called the Semantic Web by World Wide Web inventor Tim Berners-Lee, and was aimed at being a more autonomous, intelligent, and open internet.
- The Web 3.0 definition can be expanded as follows: data will be interconnected in a decentralized way, which would be a huge leap forward to our current generation of the internet (Web 2.0), where data is mostly stored in centralized repositories.
- 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).

1.3.1 Characteristics of Web 3.0

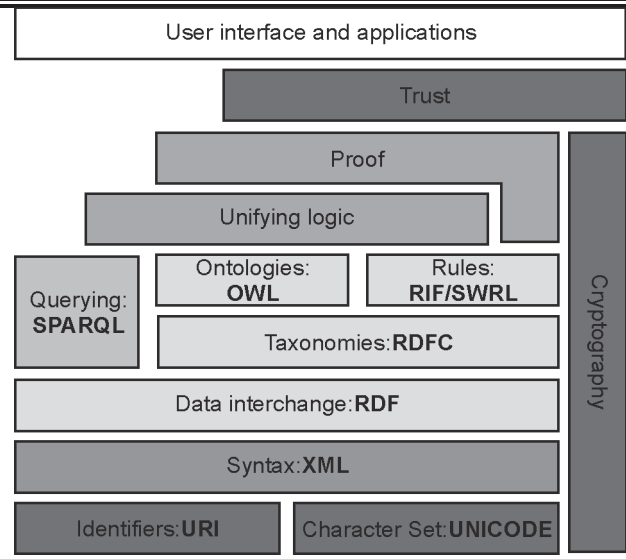
Below are 5 main features that can help us define Web 3.0 :

1. **Semantic Web :** The next evolution of the Web involves the 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.
2. **Artificial Intelligence :** Combining this capability with natural language processing, in Web 3.0, computers can understand information like humans in order to provide faster and more relevant results. They become more intelligent to satisfy the needs of users.

3. **3D Graphics :** The three dimensional design is being used extensively in websites and services in Web 3.0. Museum guides, computer games, ecommerce, geospatial contexts, etc. are all examples that use 3D graphics.
4. **Connectivity :** With Web 3.0, information is more connected thanks to semantic metadata. As a result, the user experience evolves to another level of connectivity that leverages all the available information.
5. **Ubiquity :** Content is accessible by multiple applications, every device is connected to the web, the services can be used everywhere.

1.3.2 Semantic Web Stack and its Components

- The **Semantic Web Stack**, also known as **Semantic Web Cake** or **Semantic Web Layer Cake**, illustrates the architecture of the Semantic Web.
- By encouraging the inclusion of semantic content in web pages, the Semantic Web aims at converting the current web, dominated by unstructured and semi-structured documents into a "web of data".
- The Semantic Web Stack is an illustration of the hierarchy of languages, where each layer exploits and uses capabilities of the layers below. It shows how technologies that are standardized for Semantic Web are organized to make the Semantic Web possible. It also shows how Semantic Web is an extension (not replacement) of classical hypertext web. The illustration was created by Tim Berners-Lee.
- As shown in Fig. 1.3.1 of the Semantic Web Stack, the following languages or technologies are used to create Semantic Web. The technologies from the bottom of the stack up to OWL are currently standardized and accepted to build Semantic Web applications. It is still not clear how the top of the stack is going to be implemented. All layers of the stack need to be implemented to achieve full visions of the Semantic Web.



(1A4)**Fig. 1.3.1 : Semantic Web Stack**

1. Hypertext Web technologies

- The bottom layers contain technologies that are well known from hypertext web and that without change provide basis for the semantic web.
- **Internationalized Resource Identifier (IRI),** generalization of URI, provides means for uniquely identifying semantic web resources. Semantic Web needs unique identification to allow provable manipulation with resources in the top layers.
- **Unicode** serves to represent and manipulate text in many languages. Semantic Web should also help to bridge documents in different human languages, so it should be able to represent them.
- **XML** is a markup language that enables creation of documents composed of semi-structured data. Semantic web gives meaning (semantics) to semi-structured data.
- **XML Namespaces** provides a way to use markups from more sources. Semantic Web is about connecting data together, and so it is needed to refer more sources in one document.

2. Standardized Semantic Web technologies

- Middle layers contain technologies standardized by W3C to enable building semantic web applications.
- **Resource Description Framework (RDF)** is a framework for creating statements in a form of so-called triples. It enables to represent information about resources in the form of graph - the semantic web is sometimes called Giant Global Graph.

- **RDF Schema (RDFS)** provides basic vocabulary for RDF. Using RDFS it is for example possible to create hierarchies of classes and properties.
- **Web Ontology Language (OWL)** extends RDFS by adding more advanced constructs to describe semantics of RDF statements. It allows stating additional constraints, such as for example cardinality, restrictions of values, or characteristics of properties such as transitivity. It is based on description logic and so brings reasoning power to the semantic web.
- **SPARQL** is a RDF query language it can be used to query any RDF-based data (i.e., including statements involving RDFS and OWL). Querying language is necessary to retrieve information for semantic web applications.
- **RIF** is a rule interchange format. It is important, for example, to allow describing relations that cannot be directly described using description logic used in OWL.

3. Unrealized Semantic Web technologies

- Top layers contain technologies that are not yet standardized or contain just ideas that should be implemented in order to realize Semantic Web.
- **Cryptography** is important to ensure and verify that semantic web statements are coming from trusted source. This can be achieved by appropriate digital signature of RDF statements.
- **Trust** to derived statements will be supported by (a) verifying that the premises come from trusted source and by (b) relying on formal logic during deriving new information.
- **User interface** is the final layer that will enable humans to use semantic web applications.

1.3.3 Resource Description Framework (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).
- While there are many conventional tools for dealing with data and more specifically for dealing with the relationships between data, RDF is the easiest, most powerful and expressive standard designed by now.

1.3.3(A) Features

- RDF stands for Resource Description Framework.
- RDF is a framework for describing resources on the web.
- RDF provides a model for data, and a syntax so that independent parties can exchange and use it.
- RDF is designed to be read and understood by computers.
- RDF is not designed for being displayed to people.
- RDF is written in XML.
- RDF is a part of the W3C's Semantic Web Activity.
- RDF is a W3C Recommendation.

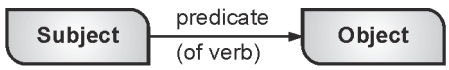
1.3.3(B) RDF Rules

- RDF uses Web identifiers (URIs) to identify resources.
- RDF describes resources with properties and property values.
- A **Resource** is anything that can have a URI, such as "http://www.w3schools.com/RDF"
- A **Property** is a Resource that has a name, such as "author" or "homepage"
- A **Property value** is the value of a Property, such as "Nilesh Patil" or "http://www.w3schools.com"
- The following RDF document could describe the resource "http://www.w3schools.com/RDF":

```
<?xml version="1.0"?>
<RDF>
<Description about="http://www.w3schools.com/RDF">
<author>Nilesh Patil</author>
<homepage>http://www.w3schools.com</homepage>
</Description>
</RDF>
```

1.3.3(C) RDF Triples

- The way RDF connects data pieces together is via triples (three positional statements).
- In plain English, an RDF statement states facts, relationships and data by linking resources of different kinds.
- With the help of an RDF statement, just about anything can be expressed by a uniform structure, consisting of three linked data pieces as shown in Fig. 1.3.2.

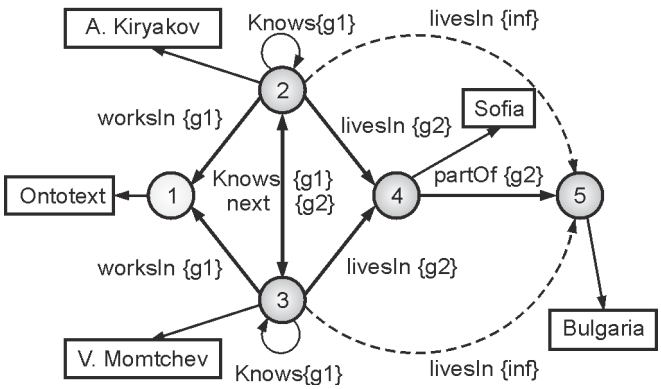


(1A5)**Fig. 1.3.2 : RDF Triple**

- Let's consider the following statement: "Apoptosis is a type I programmed cell death". Translated into RDF statements, the example fact (statement) would state the information in the following way : "Apoptosis is a type I programmed cell death" = > "Apoptosis of neutrophils" is the subject in two separate statements: 1. <apoptosis> is_a <type 1 programmed cell death> and 2. <apoptosis> type <biological process>.

1.3.3(D)The RDF Knowledge Graph

- Being a powerful and expressive framework for representing data, RDF is used for building knowledge graphs - richly interlinked, interoperable and flexible information structures.
- The Fig. 1.3.3 demonstrates the expressivity of RDF.



(1A6)**Fig. 1.3.3 :RDF: Directed Labeled Cyclic Multigraph with Labels, Types, Logic and Semantics**

- The nodes in an RDF knowledge graph could be either resources, represented by a unique resource identifier (URI, e.g., the well-known URLs), literals (e.g., the same as in XML) or auxiliary blank nodes.
- The types of the edges are called predicates, (e.g., partOf or knows). Named graphs or contexts (e.g., g1 and g2 above) can be used to manage components in the graph, (e.g., by provenance).
- Each edge in the graph represents a fact and can be seen as a quadruple <subject, predicate, object, context>.
- Classes, predicates and named graphs are all defined as URIs. This way they can appear as nodes in the graph, get their descriptions, i.e. instance data and schema can be managed and accessed in an uniform model.
- The nodes in the Fig. 1.3.3 are numbered for better readability; those must have URIs.
- Knowledge graphs, represented in RDF, provide the best framework for data integration, unification, linking and reuse, because they combine :

1. **Expressivity :** The standards in the Semantic Web stack **RDF(S)** and **OWL** allow for a fluent representation of various types of data and content: data schema, taxonomies and vocabularies, all sorts of metadata, reference and master data. The RDF* extension makes it easy to model provenance and other structured metadata.
2. **Formal semantics :** All standards in the Semantic Web stack come with well specified semantics, which allow humans and computers to interpret schema, ontologies and data in unambiguous manner.
3. **Performance :** All the specifications have been thought out, and proven in practice, to allow for efficient management of graphs of billions of facts and properties.
4. **Interoperability :** There is a range of specifications for data serialization, access (SPARQL Protocol for end-points), management (SPARQL Graph Store) and federation. The use of globally unique identifiers facilitates data integration and publishing.
5. **Standardization :** All the above is standardized through the W3C community process, to make sure that the requirements of different actors are satisfied - all the way from logicians to enterprise data management professionals and system operations teams.

1.3.3(E) RDF Example

Consider the following CD list.

Title	Artist	Country	Company	Price	Year
Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1985
Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988

An RDF document

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">
<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
<cd:artist>Bob Dylan</cd:artist>
<cd:country>USA</cd:country>
<cd:company>Columbia</cd:company>
<cd:price>10.90</cd:price>
<cd:year>1985</cd:year>
</rdf:Description>
<rdf:Description
rdf:about="http://www.recshop.fake/cd/Hide your heart">
<cd:artist>Bonnie Tyler</cd:artist>
<cd:country>UK</cd:country>
<cd:company>CBS Records</cd:company>
<cd:price>9.90</cd:price>
<cd:year>1988</cd:year>
</rdf:Description>
</rdf:RDF>
```

- The first line of the RDF document is the XML declaration. The XML declaration is followed by the root element of RDF documents: <rdf:RDF>.
- The xmlns:rdf namespace, specifies that elements with the rdf prefix are from the namespace "http://www.w3.org/1999/02/22-rdf-syntax-ns#".
- The xmlns:cd namespace, specifies that elements with the cd prefix are from the namespace "http://www.recshop.fake/cd#".
- The <rdf:Description> element contains the description of the resource identified by the rdf:about attribute.
- The elements: <cd:artist>, <cd:country>, <cd:company>, etc. are properties of the resource.

1.3.4 Web Ontology Language (OWL)

- Web Ontology Language (often stylized as OWL) is a Semantic Web language that is designed to process and integrate information over the web, making sense of it in a manner similar to human reasoning.
- Web Ontology Languages are built upon a standard of the World Wide Web Consortium called Resource Description Framework (RDF).
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.
- OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users.
  1. **OWL Lite** supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies.
  2. **OWL DL** supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computed) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL.
  3. **OWL Full** is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or

OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

1.3.4(A) OWL Syntax

In this section we describe the syntax of OWL. We illustrate step-by-step how to build an ontology using OWL. We also explain how to define the header of ontology, its classes, properties and relationships. After reading this section the reader should be able to recognize an ontology written in OWL and identify some of its components.

Header

The first element in an OWL document is an rdf:RDF element which specifies a set of XML namespace's declarations that provide a means to unambiguously interpret identifiers and make the rest of the ontology presentation much more readable. For example,

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">

    <!-- OWL Header Example -->
    <owl:Ontologyrdf:about="http://www.plantspecies.com/p
lants">
    <dc:title>The plantspecies.com Example Book
Ontology</dc:title>
    <dc:description>An example ontology written for the
plantspecies.com RDFS & OWL introduction
tutorial</dc:description>
    </owl:Ontology>

    <!-- Remainder Of Document Omitted For Brevity... -->

</rdf:RDF>
```

A namespace is composed by: reserved XML attribute *xmlns*, a prefix that identify the namespace and the value.

OWL Classes, Subclasses & Individuals

- The primary purpose of your ontology is to classify things in terms of semantics, or meaning.
- In OWL, this is achieved through the use of *classes* and *subclasses*, instances of which in OWL are called *individuals*. The individuals that are members of a given OWL class are called its *class extension*.
- A *class* in OWL is a classification of individuals into groups which share common characteristics. If an individual is a member of a class, it tells a machine reader that it falls under the semantic classification given by the OWL class.
- Here is our example OWL ontology again, this time with some added classes and subclasses. We define three plant classes: the flowering plants class and shrubs class. which are both subclasses of the planttype class.
- The planttype class is the highest level class of all plant types.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:plants="http://www.plantspecies.com/plants#">

<!-- OWL Header Omitted For Brevity -->

<!-- OWL Class Definition - Plant Type -->
<owl:Classrdf:about="http://www.plantspecies.com/plants#p
lanttype">

    <rdfs:label>The plant type</rdfs:label>
    <rdfs:comment>The class of all plant
types.</rdfs:comment>

</owl:Class>

<!-- OWL Subclass Definition - Flower -->
<owl:Classrdf:about="http://www.plantspecies.com/plants#fl
owers">

<!-- Flowers is a subclassification of planttype -->
<rdfs:subClassOfrdf:resource="http://www.plantspecies.com/
plants#planttype"/>

<rdfs:label>Flowering plants</rdfs:label>
```

```
<rdfs:comment>Flowering plants, also known as angiosperms.</rdfs:comment>

</owl:Class>

<!-- OWL Subclass Definition - Shrub -->
<owl:Classrdf:about="http://www.plantspecies.com/plants#shrubs">
<!-- Shrubs is a subclassification of planttype -->
<rdfs:subClassOfrdf:resource="http://www.plantspecies.com/plants#planttype"/>

<rdfs:label>Shrubbery</rdfs:label>
<rdfs:comment>Shrubs, a type of plant which branches from the base.</rdfs:comment>

</owl:Class>

<!-- Individual (Instance) Example RDF Statement -->
<rdf:Descriptionrdf:about="http://www.plantspecies.com/plants#magnolia">

<!-- Magnolia is a type (instance) of the flowers classification -->
<rdf:typerdf:resource="http://www.plantspecies.com/plants#flowers"/>

</rdf:Description>

</rdf:RDF>
```

OWL Properties

Individuals in OWL are related by properties. There are two types of property in OWL :

- **Object properties** (owl:ObjectProperty) relates individuals (instances) of two OWL classes.
- **Datatype properties** (owl:DatatypeProperty) relates individuals (instances) of OWL classes to literal values.

Let's first add a *data type* property (one which links an instance to a literal value) and add the name of the *species family* the Magnolia is part of.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:plants="http://www.plantspecies.com/plants#">

<!-- OWL Header Omitted For Brevity -->

<!-- OWL Classes Omitted For Brevity -->

<!-- Define the family property -->
<owl:DatatypePropertyrdf:about="http://www.plantspecies.com/plants#family"/>

<rdf:Descriptionrdf:about="http://www.plantspecies.com/plants#magnolia">

<!-- Magnolia is a type (instance) of the flowers class -->
<rdf:typerdf:resource="http://www.plantspecies.com/plants#flowers"/>

<!-- The magnolia is part of the 'Magnoliaceae' family -->
<plants:family>Magnoliaceae</plants:family>

</rdf:Description>

</rdf:RDF>
```

Finally, let's add an *object* property (one which links an instance to another instance). Let's say we're running a shop, and we want to link this plant (Magnolia) to another plant which we know as the shop owner is equally as popular. Let's add a property called "similarlyPopularTo":

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:plants="http://www.plantspecies.com/plants#">

<!-- OWL Header Omitted For Brevity -->

<!-- OWL Classes Omitted For Brevity -->

<!-- Define the family property -->
<owl:DatatypePropertyrdf:about="http://www.plantspecies.com/plants#family"/>

<!-- Define the similarlyPopularTo property -->
```

```
<owl:ObjectProperty
rdf:about="http://www.plantspecies.com/plants#similarlyPop
ularTo"/>

<!-- Define the Orchid class instance -->
<rdf:Descriptionrdf:about="http://www.plantspecies.com/pla
nts#orchid">

<!-- Orchid is an individual (instance) of the flowers class --
>
<rdf:typerdf:resource="http://www.plantspecies.com/plants#
flowers"/>

<!-- The orchid is part of the 'Orchidaceae' family -->
<plants:family>Orchidaceae</plants:family>

<!-- The orchid is similarly popular to the magnolia -->
<plants:similarlyPopularTordf:resource="http://www.plantsp
ecies.com/plants#magnolia"/>

</rdf:Description>

<!-- Define the Magnolia class instance -->
<rdf:Descriptionrdf:about="http://www.plantspecies.com/pla
nts#magnolia">

<!-- Magnolia is an individual (instance) of the flowers class -
->
<rdf:typerdf:resource="http://www.plantspecies.com/plants#
flowers"/>

<!-- The magnolia is part of the 'Magnoliaceae' family -->
<plants:family>Magnoliaceae</plants:family>

<!-- The magnolia is similarly popular to the orchid -->
<plants:similarlyPopularTordf:resource="http://www.plantsp
ecies.com/plants#orchid"/>
</rdf:Description>
</rdf:RDF>
```

To illustrate, we have defined a new individual (instance) of the flowers class representing the Orchid with the URI *http://www.linkeddatatools.com/plants#orchid*.

1.3.5 SPARQL

- SPARQL (Simple Protocol and RDF Query Language) is the SQL like query language we use to shape and return linked data from a triple store.
- SPARQL queries contain triple patterns, much like the data itself, which utilise the relationships to quickly navigate any linked data.
- This language is common for all linked data, so queries can traverse across multiple RDF databases at once.

Syntax

```
SELECT ?subject ?predicate ?object
WHERE {
?subject ?predicate ?object .
}
```

- This query selects all triples matching the pattern: ? subject ? predicate ? object which is all triples. The ? indicates a variable so ? example is a variable called "example". These variables match every possible entity, predicate or literal that fit the patterns in the query.
- The example below shows a SPARQL query to find the title of a book from the given data graph. The query consists of two parts: the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph pattern to match against the data graph. The basic graph pattern in this example consists of a single triple pattern with a single variable (?title) in the object position.
- **Data**

```
<http://example.org/book/book1><http://purl.org/dc/element
s/1.1/title> "SPARQL Tutorial".
```

- **Query**

```
SELECT ?title
WHERE
{
<http://example.org/book/book1><http://purl.org/dc/element
s/1.1/title> ?title .
}
```

- **Result**

title
"SPARQL Tutorial"

1.3.6 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.
- The RDF graph represented by an N-Triples document contains exactly each triple matching the N-Triples triple production.

Example

```
<http://one.example/subject1><http://one.example/predicate1><http://one.example/object1> . # comments here
# or on a line by themselves
_:subject1 <http://an.example/predicate1> "object1" .
_:subject2 <http://an.example/predicate2> "object2" .
```

N-Triples Language

There are four significant N-Triples language :

1. **Simple Triples :** The simplest triple statement is a sequence of (subject, predicate, object) terms, separated by whitespace and terminated by '.' after each triple.

   Example

```
<http://example.org/#spiderman><http://www.perceive.net/schemas/relationship/enemyOf><http://example.org/#green-goblin> .
```

2. **IRIs** : An ***IRI*** (Internationalized Resource Identifier) within an RDF graph is a Unicode string [UNICODE] that conforms to the syntax defined in RFC 3987. IRIs in the RDF abstract syntax must be absolute, and may contain a fragment identifier. IRIs are the generalization of URIs. IRIs are enclosed in '<' and '>' and may contain numeric escape sequences.

   Example

```
<http://example.org/#green-goblin>.
```

3. **Literals** : Literals are used to identify values such as strings, numbers, dates. Literals have a lexical form followed by a language tag, a datatype IRI, or neither. The representation of the lexical form consists of
   - an initial delimiter " (U+0022),
   - a sequence of permitted characters or numeric escape sequence or string escape sequence, and
   - a final delimiter.
4. **RDF Blank Nodes** : RDF blank nodes in N-Triples are expressed as _: followed by a blank node label which is a series of name characters. The characters in the label are liberalized as follows:
   - The characters _ and [0-9] may appear anywhere in a blank node label.
   - The character . may appear anywhere except the first or last character.
   - The characters -, U+00B7, U+0300 to U+036F and U+203F to U+2040 are permitted anywhere except the first character.

1.3.7 Turtle

- Turtle, the Terse RDF Triple Language, 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.
- An RDF graph is made up of triples consisting of a subject, predicate and object.
- Comments may be given after a '#' that is not part of another lexical token and continue to the end of the line.
- It has several implementations and as a mostly subset of Notation 3 (N3), is usable in systems that support N3.
- All RDF written in Turtle should be usable inside the query language part of the SPARQL.
- SPARQL uses Turtle/N3 style syntax for the Triple patterns and for RDF triples in the CONSTRUCT clause.
- This allows using RDF written in Turtle to allow forming "queries by example", using the data to make an initial query which can then be edited to use variables where bindings are wanted.
- 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.

Example

```
# this is a comment in a Turtle document
<http://techneobooks.com/path/#fragment>
<http://techneobooks.com/path/#fragment>
```

1.4 SELF-LEARNING TOPICS

☞ Semantic Web Vs AI

- AI or expert systems technologies use elaborate reasoning models to answer complex questions automatically. These systems often include machine-learning algorithms that can improve the system's decision-making capabilities over time.
- Semantic web technologies allow people to locate information by concept instead of by keyword or key phrase. With semantic search, people can easily distinguish between searching for John F. Kennedy, the airport, and John F. Kennedy, the president.
- The Semantic Web, is providing new techniques that can be used to help create 'intelligent agents' that can allow users to find the answers to their queries more precisely.
- Suppose, for example, you want to know the average size of an elephant. You could go online and type 'what is the average size of an elephant' into a search engine.
- Unfortunately, most search engines will not tell you the answer. Rather, they will identify many documents that might include the information that you are seeking. Instead, you are likely to find articles about the average weight of an elephant, some general articles about elephants, and maybe an article about the average size of an elephant's foot.
- Clearly, the search engine does not really understand what you want to know.
- If, on the other hand, you have an Apple iPhone running the Siri™ application, which was introduced in 2011, you can ask it the same question, and you will see a screen telling you the average length of an elephant '(18 to 25) feet' and a number of other relevant facts about elephants.
- Siri, it seems, figured out what you meant in your question and produced a single relevant, and (one hopes) correct answer."
- In artificial intelligence, an intelligent agent (IA) is an autonomous entity which observes through sensors and acts upon an environment using actuators (i.e., it is an agent) and directs its activity towards achieving goals (i.e., it is rational).
- Intelligent agents may also learn or use knowledge to achieve their goals. They may be very simple or very complex: a reflex machine such as a thermostat is an intelligent agent.
- Intelligent agents are often described schematically as an abstract functional system similar to a computer program.
- For this reason, intelligent agents are sometimes called abstract intelligent agents (AIA) to distinguish them from their real world implementations as computer systems, biological systems, or organizations.
- Some definitions of intelligent agents emphasize their autonomy, and so prefer the term autonomous intelligent agents. Still others consider goal-directed behavior as the essence of intelligence and so prefer a term borrowed from economics, 'rational agent'.

☞ SPARQL Vs SQL

Sr. No.	SPARQL	SQL
1.	It stands for Simple Protocol and RDF Query Language.	It stands for Structured Query Language.
2.	SPARQL accesses a web of linked data.	SQL accesses tables in relational databases.
3.	**Syntax:** SELECT <variable list> WHERE [<graph pattern>] The variables in the variable list are bound by the graph pattern.	**Syntax:** SELECT <attribute list> FROM <table name> WHERE <test expression> The test expression capture both the rows and columns relevant to a particular query.

Sr. No.	SPARQL	SQL
4.	The main feature of the SPARQL which will impress SQL users is the ability to federate queries across different repositories.	SQL has no standard system for query federation.
5.	SPARQL uses the keyword OPTIONAL instead of LEFT OUTER JOIN.	SQL uses the keyword LEFT OUTER JOIN instead of OPTIONAL.
6.	To select data with values not matching some specified value, SPARQL uses the NOT EXISTS filter.	To select data with values not matching some specified value, SQL uses NOT EQUAL.

7.	Example:
	SELECT ?name ?city
	WHERE {
	?who<Person#fname> ?name ;
	<Person#addr> ?adr .
	?adr<Address#city> ?city ;
	<Address#state> "MA"
	}
		Example:
		SELECT Person.fname, Address.city
		FROM Person, Address
		WHERE Person.addr=Address.ID
		AND Address.state="MA"

8.	SPARQL services vary in whether or not they have a pre-determined RDF database.	SQL queries operate over a given database.

Descriptive Questions

Q. 1 Explain the features that define Web 3.0.

Q. 2 Compare and contrast Web 1.0, Web 2.0 and Web 3.0.

Q. 3 Explain web analytics. Give some of the examples of a web analytics tool.

Q. 4 Discuss the steps to choose a web analytics tool.

Q. 5 Explain clickstream analysis in brief.

Q. 6 Discuss the procedure to measure the success rate of a website.

Q. 7 Define Semantic Web. Explain the components of Semantic Web Stack.

Q. 8 Explain RDF in detail.

Q. 9 Explain Web Ontology Language (OWL) in brief.

Q. 10 Write short note on:

(a) N-Triples (b) Turtle

Chapter Ends…

❑❑❑

MODULE 2

CHAPTER 2

Typescript

University Prescribed Syllabus w.e.f Academic Year 2021-2022

Overview, TypeScript Internal Architecture, TypeScript Environment Setup, TypeScript Types, variables and operators, Decision Making and loops, TypeScript Functions, TypeScript Classes and Objects, TypeScript Modules

Self-learning Topics : Javascript Vs TypeScript.

2.1 OVERVIEW OF TYPESCRIPT

- TypeScript is an open-source, object-oriented language developed and maintained by Microsoft, licensed under Apache 2 license.
- TypeScript extends JavaScript by adding data types, classes, and other object-oriented features with type-checking.
- It is a typed superset of JavaScript that compiles to plain JavaScript.
- Official website: https://www.typescriptlang.org

2.1.1 Why TypeScript?

- JavaScript is a dynamic programming language with no type system.
- JavaScript provides primitive types like string, number, object, etc., but it doesn't check assigned values. JavaScript variables are declared using the **var** keyword, and it can point to any value. JavaScript doesn't support classes and other object-oriented features. So, without the type system, it is not easy to use JavaScript to build complex applications with large teams working on the same code.
- The type system increases the code quality, readability and makes it easy to maintain and refactor codebase. More importantly, errors can be caught at compile time rather than at runtime.
- Hence, the reason to use TypeScript is that it catches errors at compile-time, so that you can fix it before you run code. It supports object-oriented programming features like data types, classes, enums, etc., allowing JavaScript to be used at scale.
- TypeScript compiles into simple JavaScript. The TypeScript compiler is also implemented in TypeScript and can be used with any browser or JavaScript engines like Node.js. TypeScript needs an ECMAScript 3 or higher compatible environment to compile. This is a condition met by all browsers and JavaScript engines today.
- Some of the most popular JavaScript frameworks like Angular.js and WinJS are written in TypeScript.

2.1.2 How to use TypeScript?

- TypeScript code is written in a file with **.ts** extension and then compiled into JavaScript using the TypeScript compiler.
- A TypeScript file can be written in any code editor.
- A TypeScript compiler needs to be installed on your platform.
- Once installed, the command **tsc<filename>.ts** compiles the TypeScript code into a plain JavaScript file.
- JavaScript files can then be included in the HTML and run on any browser.

(1B1)**Fig. 2.1.1 : Compile TypeScript to JavaScript**

2.1.3 TypeScript Features

1. **Cross-Platform :** TypeScript runs on any platform that JavaScript runs on. The TypeScript compiler can be installed on any Operating System such as Windows, macOS, and Linux.
2. **Object-Oriented Language :** TypeScript provides powerful features such as Classes, Interfaces, and Modules. You can write pure object-oriented code for client-side as well as server-side development.
3. **Static type-checking :** TypeScript uses static typing. This is done using type annotations. It helps type checking at compile time. Thus, you can find errors while typing the code without running your script each time. Additionally, using the type inference mechanism, if a variable is declared without a type, it will be inferred based on its value.
4. **Optional Static Typing :** TypeScript static typing is optional, if you prefer to use JavaScript's dynamic typing.
5. **DOM Manipulation :** Like JavaScript, TypeScript can be used to manipulate the DOM.
6. **ES 6 Features :** TypeScript includes most features of planned ECMAScript 2015 (ES 6, 7) such as class, interface, Arrow functions etc.

Module 2

2.1.4 TypeScript Advantages

1. TypeScript is an open-source language with continuous development and maintenance by Microsoft.
2. TypeScript runs on any browser or JavaScript engine.
3. TypeScript is similar to JavaScript and uses the same syntax and semantics. All of TypeScript's code finally gets converted into JavaScript. This allows a quicker learning curve for front-end developers currently coding in JavaScript.
4. TypeScript is also closer in syntax to backend languages like Java and Scala. This helps backend developers write front-end code faster.
5. TypeScript code can be called from an existing JavaScript code. TypeScript also works with existing JavaScript frameworks and libraries without any issues.
6. The TypeScript Definition file, with **.d.ts** extension, provides support for existing JavaScript libraries like JQuery, D3.js, etc. So, TypeScript code can add JavaScript libraries using type definitions to avail the benefits of type-checking, code auto-completion, and documentation in existing dynamically-typed JavaScript libraries.
7. TypeScript has support for the latest JavaScript features from ECMAScript 2015. It includes features from ES6 and ES7 that can run in ES5-level JavaScript engines like Node.js. This offers a massive advantage of using features from future JavaScript versions in current JavaScript engines.
8. TypeScript has easy integration with task runner tools like Grunt and Gulp to automate the workflow.

2.1.5 TypeScript Vs 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);

2.2 TYPESCRIPT INTERNAL ARCHITECTURE

TypeScript has the following three fundamentals components.

- **Language :** It features the TypeScript language elements**.** It consists of syntax, keywords, and type annotations.
- **The TypeScript Compiler** : The TypeScript compiler (tsc) changes the instructions written in TypeScript to its JavaScript equivalent. Browser doesn't support the execution of TypeScript code directly. So the program written in TypeScript must be re-written in JavaScript equivalent code which supports the execution of code in the browser directly. To perform this, TypeScript comes with TypeScript compiler named "tsc." The current version of TypeScript compiler supports ES6, by default. It compiles the source code in any module like ES6, SystemJS, AMD, etc.
- We can install the TypeScript compiler by locally, globally, or both with any **npm** package. Once installation completes, we can compile the TypeScript file by running "tsc" command on the command line.

Example

```
$ tschelloworld.ts  //
```

It compiles the TS file helloworld into the helloworld.js file.

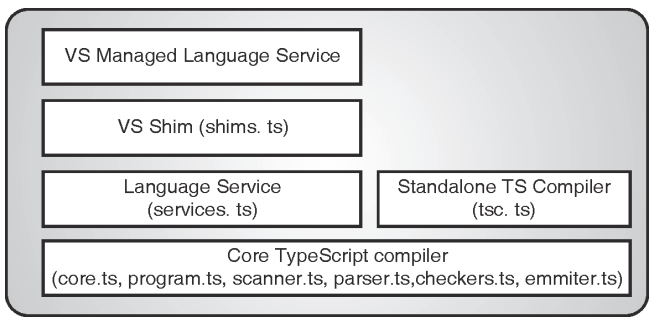
☞ Compiler Configuration

The TypeScript compiler configuration is given in **tsconfig.json** file and looks like the following:

```
{
 "compilerOptions": {
   "declaration": true,
   "emitDecoratorMetadata": false,
   "experimentalDecorators": false,
   "module": "none",
   "moduleResolution": "node",
   "noFallthroughCasesInSwitch": false,
   "noImplicitAny": false,
   "noImplicitReturns": false,
   "removeComments": false,
   "sourceMap": false,
   "strictNullChecks": false,
   "target": "es3"
 },
 "compileOnSave": true
}
```

☞ Declaration file

- When we compile the TypeScript source code, it gives an option to generate a **declaration file** with the extension **.d.ts**. This file works as an interface to the components in the compiled JavaScript. If a file has an extension **.d.ts**, then each root level definition must have the **declare** keyword prefixed to it. It makes clear that there will be no code emitted by TypeScript, which ensures that the declared item will exist at runtime. The declaration file provides IntelliSense for JavaScript libraries like jQuery.
- **The TypeScript Language Service –** The "Language Service" reveals an extra layer around the core compiler pipeline that are editor-like applications. The language service assists the common set of typical editor operations like statement completion, signature help, code formatting and outlining, colorization, etc.



(1B2)**Fig. 2.2.1 : TypeScript Internal Architecture**

Module 2

2.3 TYPESCRIPT ENVIRONMENT SETUP

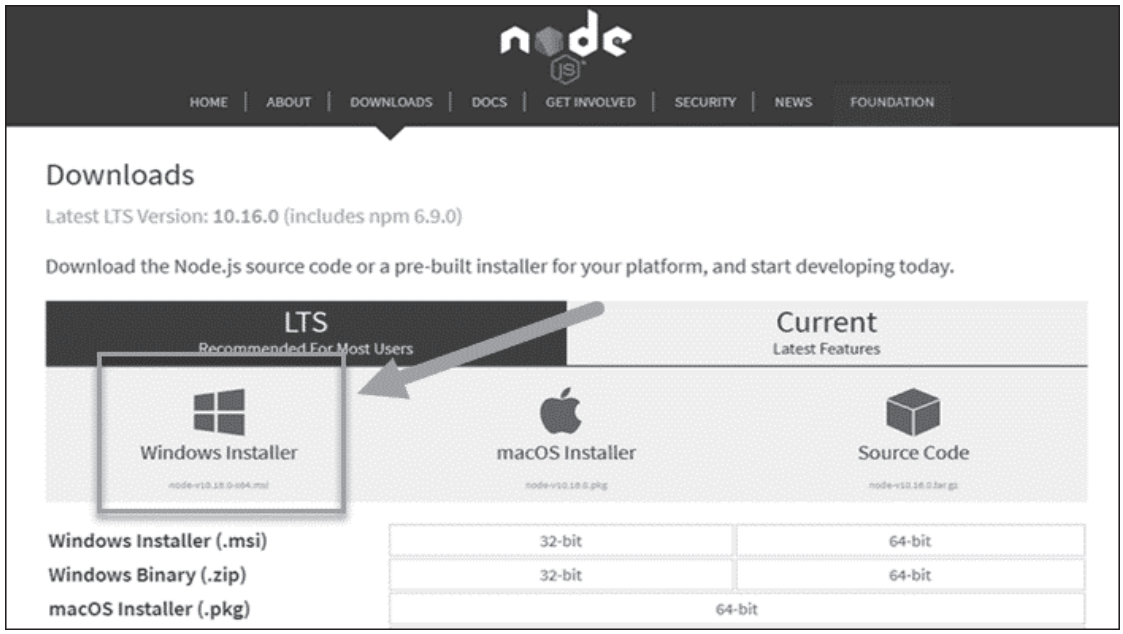
There are two ways to install TypeScript:

1. Install TypeScript using Node.js Package Manager (npm).
2. Install the TypeScript plug-in in your IDE (Integrated Development Environment).

2.3.1 Install TypeScript using Node.js Package Manager (npm) on Windows

Node.js is an open source, cross-platform runtime environment for server-side JavaScript. Node.js is required to run JavaScript without a browser support. It uses Google V8 JavaScript engine to execute code. You may download Node.js source code or a pre-built installer for your platform. Node is available here - **https://nodejs.org/en/download**

- **Step 1 : Download and run the .msi installer for Node.**



Note : There are other versions available. If you have an older system, you may need the 32-bit version. You can also use the top link to switch from the stable LTS version to the current version. If you are new to Node.js or don't need a specific version, choose LTS.

- **Step 2 : Install Node.js and NPM from Browser**

1. Once the installer finishes downloading, launch it. Open the **downloads** link in your browser and click the file. Or, browse to the location where you have saved the file and double-click it to launch.
2. The system will ask if you want to run the software - click **Run**.
3. You will be welcomed to the Node.js Setup Wizard - click **Next**.
4. On the next screen, review the license agreement. Click **Next** if you agree to the terms and install the software.
5. The installer will prompt you for the installation location. Leave the default location, unless you have a specific need to install it somewhere else – then click **Next**.

6. The wizard will let you select components to include or remove from the installation. Again, unless you have a specific need, accept the defaults by clicking **Next**.
7. Finally, click the **Install** button to run the installer. When it finishes, click **Finish**.

- **Step 3 : Verify Installation**

  To verify if the installation was successful, enter the command *node –v* in the terminal window.

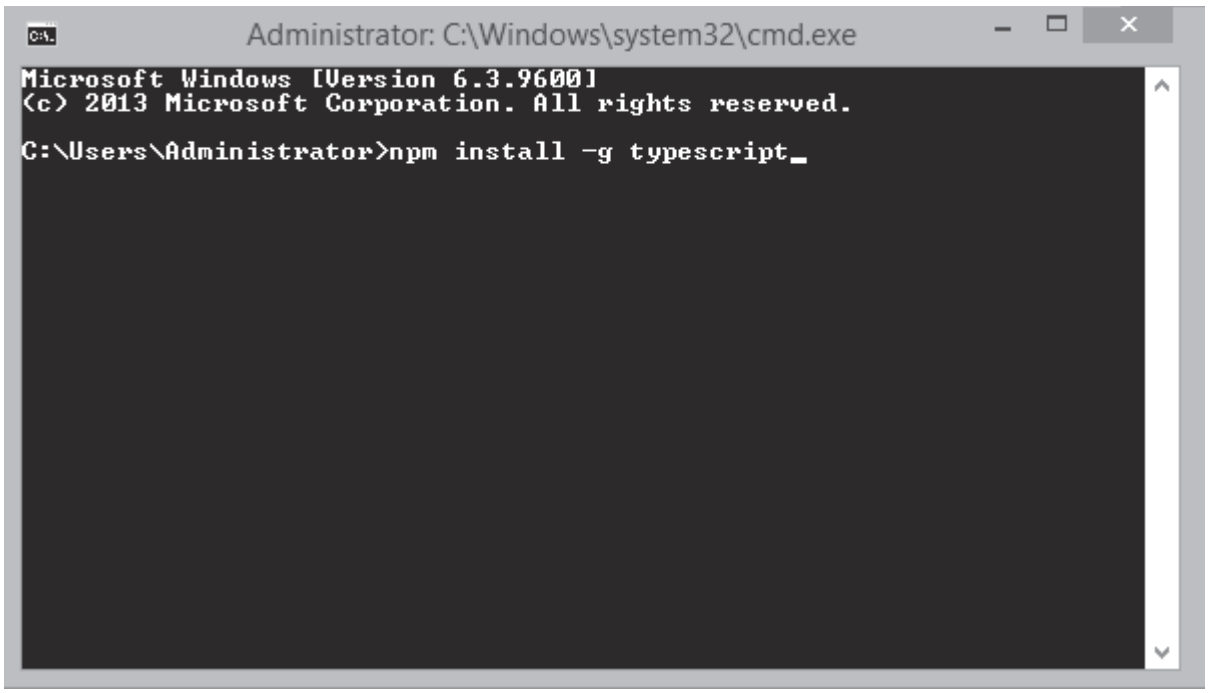
```
C:\Users>node -v
v4.2.3

C:\Users>_
```


  Also to check the npm version, enter the command *npm –v* in the terminal window.

- **Step 4 : TypeScript Installation**

  Type the following command in the terminal window to install TypeScript.

npm install -g typescript



  To verify the installation was successful, enter the command **tsc -v** in the Terminal Window**.**

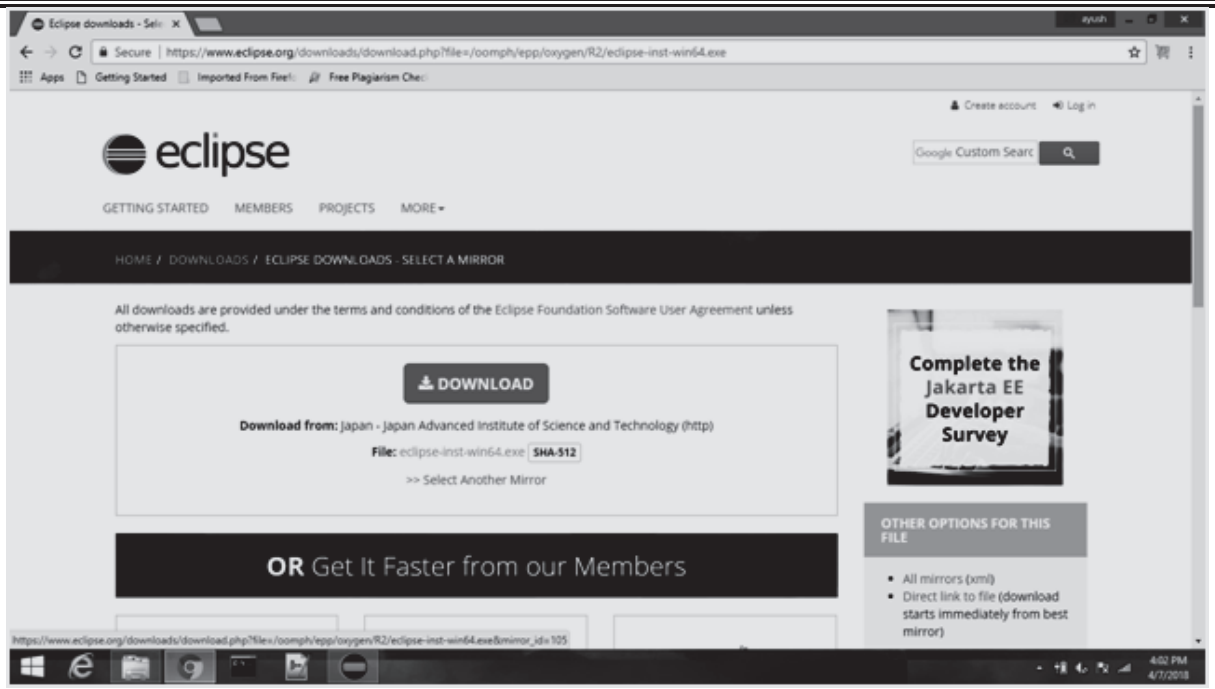
2.3.2 Install TypeScript plug-in in your IDE

Install IDE like Eclipse, Visual Studio, WebStorm, Atom, Sublime Text, etc. Here, we install Eclipse.

- **Step 1: Download Eclipse IDE**

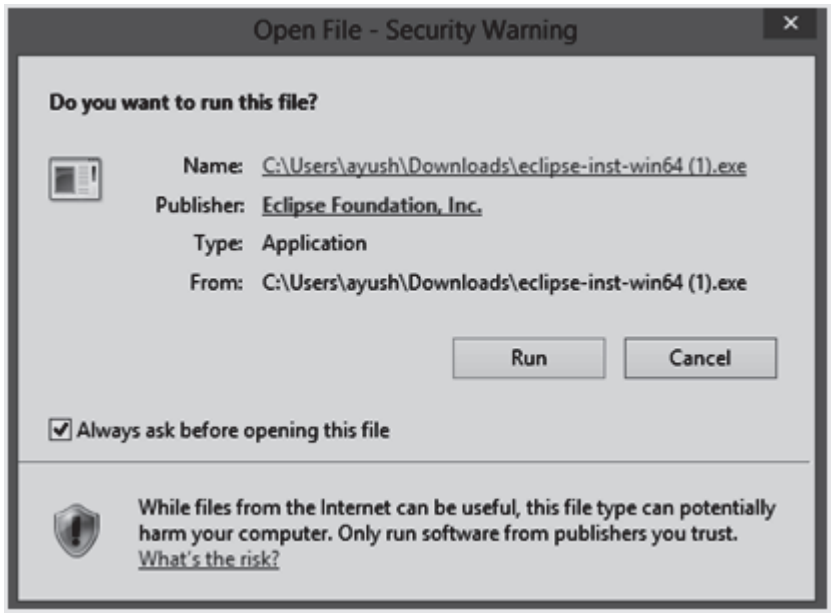
  Eclipse IDE can be downloaded from https://www.eclipse.org/downloads/

  You can download the latest version of eclipse i.e. eclipse oxygen from that page.



Step 2: Install Eclipse

Double click on the exe file which has just been downloaded. The screen will look like following. Click Run to proceed the installation.

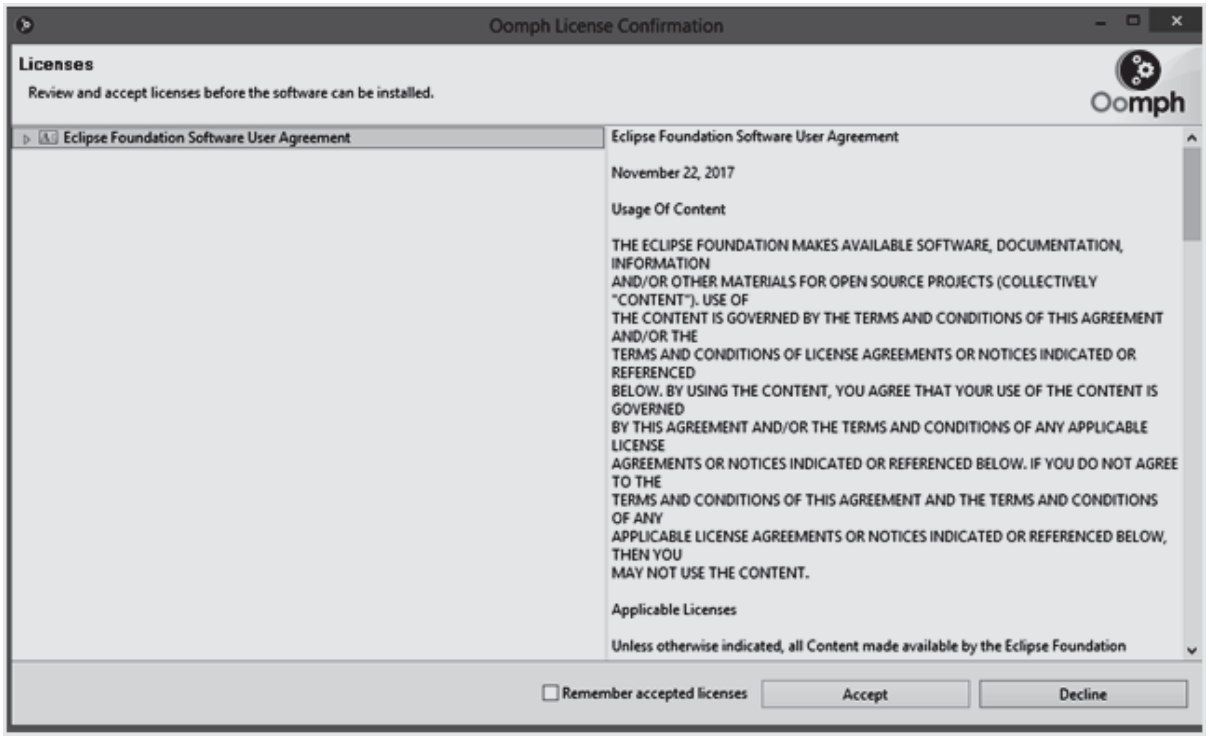


Choose the software suit which you want to install. In our case, we have chosen Eclipse IDE for Javascript and Web Developers which is recommended in our case.

Module 2

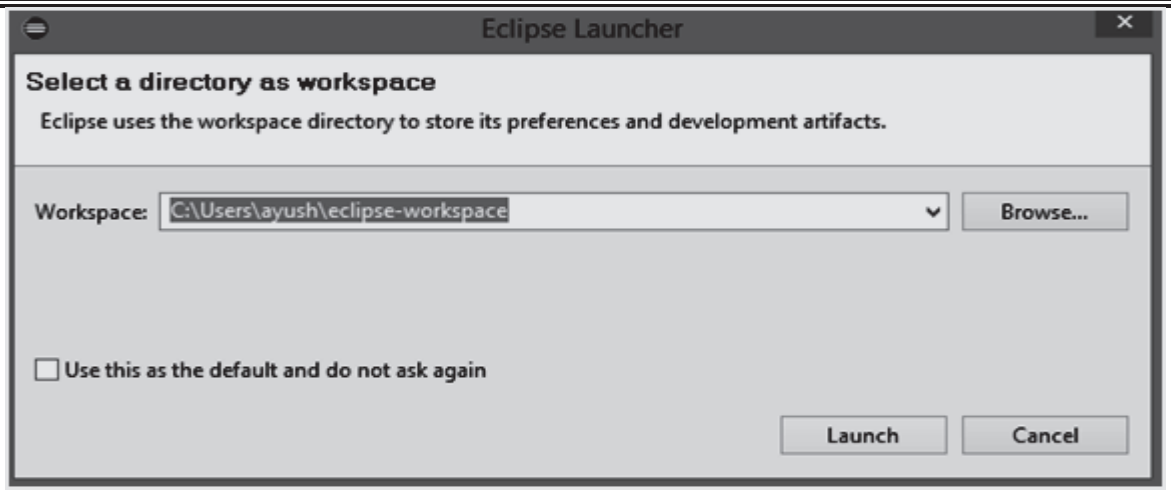
Now, the Setup is ready to install Eclipse in the directory shown in the image. However, we can select any destination folder present on our system. Just click install when you done with the directory selection.

The set up will ask us to accept the Eclipse Foundation Software Agreement. Just click **Accept** to continue.



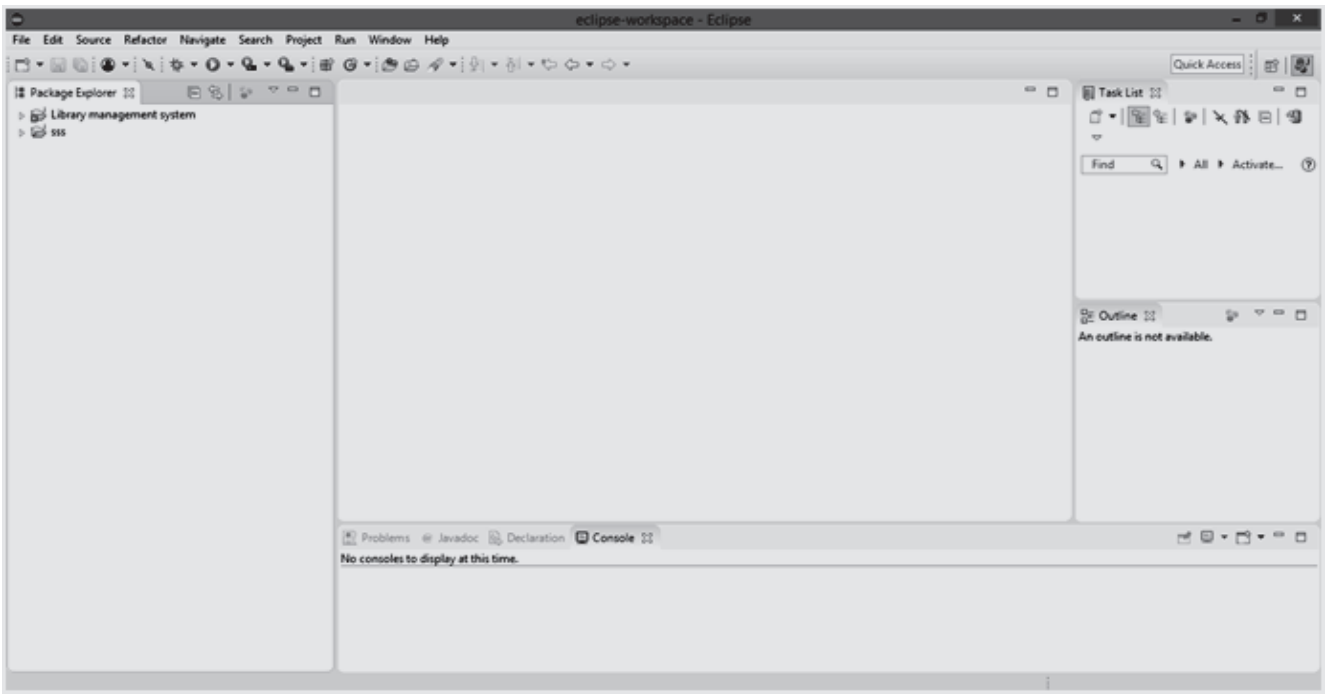
Now, we will have to wait for the time the Eclipse will be installing on our system. Once the installation will be done, the following screen will appear. Just click the **LAUNCH** button to launch eclipse.

Browse the workspace directory, just click the **Launch** button once you done with the process.



Module 2

We have got the Eclipse IDE opened on our system. However, the screen will appear like following. Now, we are all set to configure Eclipse in order to run the TypeScript application.



Go to Help → Eclipse Marketplace → Select TypeScript IDE and press Install.

Now create a new project as follows:

File → New → TypeScript → TypeScript Project → Click Next.

You will see in the left pane of the window : TypeScript Resources (version) directory with package.json and tsconfig.json files.

Right click on tsconfig.json → Run As → Compile TypeScript

2.4 TYPESCRIPT FIRST PROGRAM

In this section, we are going to learn how we can write a program in TypeScript, how to compile it, and how to run it. Let us write a program in the text editor, save it, compile it, run it, and display the output to the console. To do this, we need to perform the following steps.

- **Step 1 :** Open the Text Editor and write the following code.

```
var message: string = “Hello World”;
console.log(message);
```

- **Step 2 :** Save the above file as "**.ts**" extension.
- **Step 3 :** Compile the TypeScript code. To compile the source code, open the command prompt, and then go to the file directory location where we saved the above file. For example, if we save the file on the desktop, go to the terminal window and type: **cd Desktop/ folder_name**. Now, type the following command **tscfilename.ts** for compilation and press Enter.

It will generate JavaScript file with **".js"** extension at the same location where the TypeScript source file exists. The below ".js" file is the output of TypeScript (.ts) file.

```
var message = “Hello World”;
console.log(message);
```

> **NOTE :** If we directly run "**.ts**" file on the web browser, it will throw an error message. But after the compilation of "**.ts**" file, we will get a "**.js**" file, which can be executed on any browser.

- **Step 4 :** Now, to run the above JavaScript file, type the following command in the terminal window: **node filename.js** and press Enter. It gives us the final output as:

```
Hello World
```

2.5 TYPESCRIPT TYPES

- 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.
- TypeScript plays an important role when the object-oriented programmer wants to use the type feature in any scripting language or object-oriented programming language.
- The Type System checks the validity of the given values before the program uses them. It ensures that the code behaves as expected.
- TypeScript provides data types as an optional Type System.

☞ TypeScript Data Types Syntax

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];
```

If you observe the above syntax, we added data type after the variable name to tell the compiler that the type of data the variable can hold or the type of variable that belongs to.

- **[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.

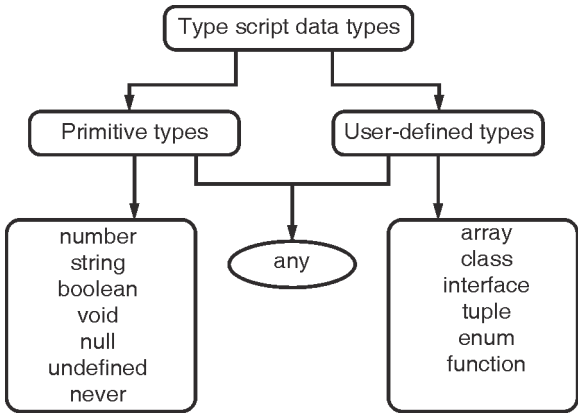
☞ TypeScript Data Types Example

Following is the example of defining the variables with required data types in typescript.

Code	Output

letuname: string = "Nilesh Patil"; // string
letulocation: string = "Mumbai"; // string
let age: number = 38; // number
letisCompleted: boolean = true; // boolean
console.log("Name:" + uname);
console.log("Location:" + ulocation);
console.log("Age:" + age);
console.log("Is Completed:" + is Completed);
	Name:Nilesh Patil
	Location:Mumbai
	Age:38
	Is Completed:true

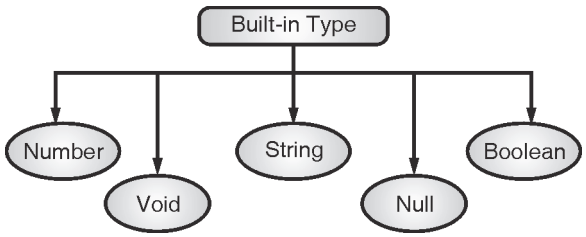
We can classify the TypeScript data type as following:



(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**

(1) Number

- Like JavaScript, all the numbers in TypeScript are stored as floating-point values.
- These numeric values are treated like a number data type.
- The numeric data type can be used to represent both integers and fractions.
- TypeScript also supports Binary (Base 2), Octal(Base 8), Decimal(Base 10), and Hexadecimal(Base 16) literals.

Syntax

```
let identifier: number = value;
```

Example

Code	Output

let first: number = 14.0; // number
let second: number = 0x73CF; // hexadecimal
let third: number = 0o547 ; // octal
let fourth: number = 0b111001; // binary
console.log(first);
console.log(second);
console.log(third);
console.log(fourth);
	14
	29647
	359
	57

Module 2

☞ TypeScript Number Type Methods

The following are the different types of methods available for number type in TypeScript.

Method	Description
toExponential()	It will return the number in exponential notation string format.
toFixed()	It will return the number in fixed-point notation string format.
toLocaleString()	It will convert the number to a string by using the current or specified locale.
toPrecision()	It will return the number in either exponential or fixed-point notation with a specified number of digits in string format.
toString()	It will return the string representation of number in the specified base.
valueOf()	It will return the primitive value of a specified object.

Example

```
let num1: number = 446;
// toExponential()
console.log('Exponential Method')
console.log(num1.toExponential()); // 4.46e+2
console.log(num1.toExponential(1)); // 4.5e+2
console.log(num1.toExponential(2)); // 4.46e+2

// toString()
```

```
console.log('String Method')
console.log(num1.toString()); // 446
console.log(num1.toString(2)); // 110111110
console.log(num1.toString(4)); // 12332


// valueOf()
console.log('ValueOf Method')
console.log(num1.valueOf()); // 446
console.log(typeof num1); // number

let num2: number = 78.65123;
// toFixed()
console.log('Fixed Method')
console.log(num2.toFixed()); // 79
console.log(num2.toFixed(1)); // 78.7
console.log(num2.toFixed(2)); // 78.65

let num3: number = 20456.543;
// toLocaleString()
console.log('LocaleString Method')
console.log(num3.toLocaleString()); // 20,456.543 - US
console.log(num3.toLocaleString('de-DE')); // 20.456,543 - German

let num4: number = 46.6521;
// toPrecision()
console.log('Precision Method')
console.log(num4.toPrecision()); // 46.6521
console.log(num4.toPrecision(1)); // 5e+1
console.log(num4.toPrecision(2)); // 47
console.log(num4.toPrecision(3)); // 46.7
```

(2) String

- We use the string data type to represents the text in TypeScript. String type work with textual data.
- We include string literals in our scripts by enclosing them in single or double quotation marks.
- It also represents a sequence of Unicode characters. It embeds the expressions in the form of $ {expr}.

Syntax

```
let identifier: string = " ";
            or
let identifier: string = ' ';
```

Example

Code

let empName: string = "Sachin";
let empDept: string = "Techneo";
 	
// Before-ES6
let output1: string = empName + " works in the " + empDept + " department.";
 	
// After-ES6
let output2: string = `${empName} works in the ${empDept} department.`;
 	
console.log(output1);
console.log(output2);

Output
Sachin works in the Techneo department. Sachin works in the Techneo department.

(3) Boolean

- The string and numeric data types can have an unlimited number of different values, whereas the boolean data type can have only two values. They are "true" and "false."
- A boolean value is a truth value which specifies whether the condition is true or not.

Syntax

```
let identifier: boolean = boolean value;
```

Example

Code

letisPresent:boolean = true;
letisAbsent:boolean = false;
console.log(isPresent);
console.log(isAbsent);

Output
true false

(4) Void

- A void is a return type of the functions which do not return any type of value. It is used where no data type is available.

- A variable of type void is not useful because we can only assign undefined or null to them.
- An undefined data type denotes uninitialized variable, whereas null represents a variable whose value is undefined.

Example

```
let tempNum: void = undefined;
tempNum = null;
tempNum = 123; //Error: Type 'number' is not assignable to
                    type 'void'.
```

(6) Null

- Null represents a variable whose value is undefined.
- Much like the void, it is not extremely useful on its own.
- The Null accepts the only one value, which is null.
- The Null keyword is used to define the Null type in TypeScript, but it is not useful because we can only assign a null value to it.

Example

Code	Output

let num: number = null;
let bool: boolean = null;
let str: string = null;
console.log(num)
console.log(bool)
console.log(str)
	null
	null
	null

(7) Undefined

- The Undefined primitive type denotes all uninitialized variables in TypeScript and JavaScript.
- It has only one value, which is undefined.
- The undefined keyword defines the undefined type in TypeScript, but it is not useful because we can only assign an undefined value to it.

Example

Code

let num: number = undefined;
let bool: boolean = undefined;
let str: string = undefined;

(8) Any

- It is the "super type" of all data type in TypeScript.
- It is used to represents any JavaScript value.
- It allows us to opt-in and opt-out of type-checking during compilation.
- If a variable cannot be represented in any of the basic data types, then it can be declared using "**Any**" data type.
- Any type is useful when we do not know about the type of value (which might come from an API or 3rd party library), and we want to skip the type-checking on compile time.

Syntax

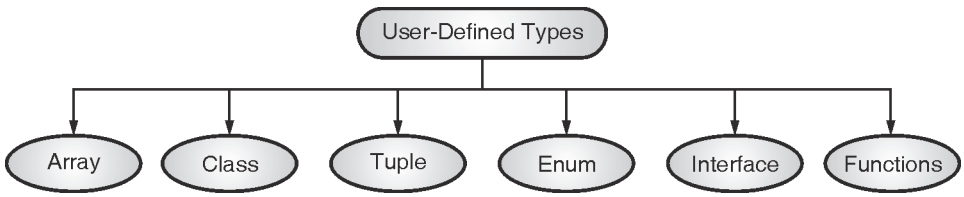
```
let identifier: any = value;
```

Example

Example 1	*Example 2*

let val: any = 'Hello';
val = 234; // OK
val = true; // OK
	function ProcessData(x, y) {
	  return x + y;
	}
	var result;
	result = ProcessData("Hello ", "Any!"); //Hello Any!
	result = ProcessData(2, 3); //5
	console.log(result);

2.5.2 User-Defined Types



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

(1) Array

- An array is a collection of elements of the same data type.
- Like JavaScript, TypeScript also allows us to work with arrays of values.
- An array can be written in two ways.

1. Use the type of the elements followed by [] to denote an array of that element type.

Example

```
let fruits: string[] = ['Apple', 'Orange', 'Banana'];
console.log(fruits[0]);
console.log(fruits[1]);
```

2. Using a generic array type, Array<elementType>.

Example

```
let fruits: Array<string> = ['Apple', 'Orange', 'Banana'];
console.log(fruits[0]);
console.log(fruits[1]);
```

(2) Class

- Classes are used to create reusable components and acts as a template for creating objects.
- It is a logical entity which store variables and functions to perform operations.
- TypeScript gets support for classes from ES6.
- It is different from the interface which has an implementation inside it, whereas an interface does not have any implementation inside it.

Example

Code	Output

class Employee {
empCode: number;
empName: string;
 	
constructor(code: number, name: string) {
this.empName = name;
this.empCode = code;
}
}
let emp = new Employee(100,"Steve");
console.log(emp)
	Employee: {
	"empName": "Steve",
	"empCode": 100
	}

(3) Tuple

- The Tuple is a data type which includes two sets of values of different data types.
- It allows us to express an array where the type of a fixed number of elements is known, but they are not the same.
- For example, if we want to represent a value as a pair of a number and a string, then it can be written as:

Code	Output

// Declare a tuple
let a: [string, number];
// Initialize it
a = ["hi", 10]; // OK
console.log(a)
	["hi", 10]

(4) Interface

- An Interface is a structure which acts as a contract in our application.
- It defines the syntax for classes to follow, means a class which implements an interface is bound to implement all its members.
- It cannot be instantiated but can be referenced by the class which implements it.
- The TypeScript compiler uses interface for type-checking that is also known as "duck typing" or "structural subtyping."

Example

Code	Output

interface IPerson {
firstName:string,
lastName:string,
sayHi: ()=>string
}
 	
varcustomer:IPerson = {
firstName:"Tom",
lastName:"Hanks",
sayHi: ():string =>{return "Hi there"}
}
 	
console.log("Customer Object ")
console.log(customer.firstName)
console.log(customer.lastName)
console.log(customer.sayHi())
	Customer Object
	Tom
	Hanks
	Hi there

(5) Enums

- Enums define a set of named constant.
- TypeScript provides both string-based and numeric-based enums.
- By default, enums begin numbering their elements starting from 0, but we can also change this by manually setting the value to one of its elements.
- TypeScript gets support for enums from ES6.

Example

Code	Output

enumPrintMedia {
Newspaper,
Newsletter,
Magazine,
Book
}
console.log(PrintMedia.Newspaper);
console.log(PrintMedia.Magazine);
	0
	2

enumPrintMedia {
Newspaper = “Indian Express”,
Newsletter = “Ascent”,
Magazine = “India Today”,
Book = “WebX.0”
}
console.log(PrintMedia.Newspaper);
console.log(PrintMedia.Magazine);
	Indian Express
	India Today

(6) Function

1. A function is the logical blocks of code to organize the program.
2. Like JavaScript, TypeScript can also be used to create functions either as a **named function** or as an **anonymous function**.
3. Functions ensure that our program is readable, maintainable, and reusable.
4. A function declaration has a function's name, return type, and parameters.

Example

Code	Output

function disp_details(id:number,name:string,mail_id?:string) {
console.log("ID:", id);
console.log("Name",name);
 	
if(mail_id!=undefined)
console.log("Email Id",mail_id);
}
disp_details(123,"Sachin");
disp_details(111,"Altab","altab@technebooks.com");
	Name Sachin
	ID: 111
	Name Altab
	Email Id
	altab@techneobooks.com

Module 2

2.6 TYPESCRIPT 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:
  1. The variable name must be an **alphabet** or **numeric digits**.
  2. The variable name cannot start with digits.
  3. The variable name cannot contain **spaces** and **special character**, except the **underscore**(_) and the **dollar**($) sign.
- In ES6, we can define variables using **let** and **const** keyword. These variables have similar syntax for variable declaration and initialization but differ in scope and usage. In TypeScript, there is always recommended to define a variable using **let** keyword because it provides the **type safety**.
- The **let** keyword is similar to **var** keyword in some respects, and **const** is a let which prevents re-assignment to a variable.

2.6.1 Variable Declaration

We can declare a variable in one of the four ways:

1. Declare type and value in a single statement

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

2. Declare type without value. Then the variable will be set to undefined.

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

3. Declare its value without type. Then the variable will be set to any.

```
var [identifier] = value;
```

4. Declare without value and type. Then the variable will be set to any and initialized with undefined.

```
var [identifier];
```

Let's understand all the three variable keywords one by one.

var keyword

- Generally, **var** keyword is used to declare a variable in JavaScript.

```
var x = 10;
```

- We can also declare a variable inside the function:

```
function a() {
varmsg = " Welcome to WebX.0 !! ";
    return msg;
}
a();
```

- We can also access a variable of one function with the other function:

```
function a() {
var x = 10;
   return function b() {
var y = x+5;
        return y;
   }
}
var  b = a();
b();      //returns '15'
```

- **Scoping rules**

  Variables declared in TypeScript with the var keyword have function scope. This variable has global scope in the function where they are declared. It can also be accessed by any function which shares the same scope.

Example

```
function f()
{
   var X = 2; //Available globally inside f()
   if(true)
   {
      var Y = 5; //Available globally inside f()
      console.log(X); //Output 2
      console.log(Y); //Output 5
   }
   console.log(X); //Output 2
   console.log(Y); //Output 5
}
f();
console.log(X); //Returns undefined because value cannot access from outside function
console.log(Y); //Returns undefined because value cannot access from outside function
```

let keyword

- The let keyword is similar to the var keyword.
- The var declaration has some problems in solving programs, so ES6 introduced let keyword to declare a variable in TypeSript and JavaScript.
- The let keyword has some restriction in scoping in comparison of the var keyword.
- The let keyword can enhance our code readability and decreases the chance of programming error.
- The let statement are written as same syntax as the var statement:

```
var declaration: var a = 10;
let declaration: let a = 10;
```

- The key difference between var and let is not in the syntax, but it differs in the semantics. The variable declared with the let keyword are scoped to the nearest enclosing block which can be smaller than a function block.

- **Block Scoping**

  When the variable is declared using the let keyword, it uses block scoping or lexical scoping. Unlike variable declared using var keyword whose scopes leak out to their containing function, a block-scoped variable cannot be visible outside of its containing block.

```
function f(input: boolean) {
   let x = 100;
   if (input) {
      // "x" exists here
      let y = x + 1;
      return y;
   }
// Error: Cannot find name 'y'.
 return y;
}
```

Here, we have two local variables x and y. Scope of x is limited to the body of the function f() while the scope of y is limited to the containing if statement's block.

☞ Re-declaration and Shadowing

- With the var declaration, it did not matter how many times we declared variables. We just got only one. In the below example, all declarations of x refer to the same x, which is perfectly valid. But there is some bug, which can be found by the let declaration.

Example without let keyword	Example with let keyword:

function f(a) {
  var a;
  var a;
  if (true) {
  var a;
  }
}
	let a = 10;
	let a = 20;
	// it gives error: can't re-declare 'a' in the same scope

- **Shadowing** is the act of introducing a new name in a more nested scope. It declares an identifier which has already been declared in an outer scope. This is not incorrect, but there might be confusion.
- It will make the outer identifier unavailable inside the loop where the loop variable is shadowing it. It can introduce bugs on its own in the event of accidental Shadowing, as well as it also helps in preventing the application from certain bugs.

Example

```
varcurrencySymbol = "$";
function showMoney(amount) {
varcurrencySymbol = "€";
document.write(currencySymbol + amount);
}
showMoney("100");
```

- The above example has a global variable name that shares the same name as the inner method. The inner variable is used only in that function, and all other functions will use the global variable declaration.
- The Shadowing is usually avoided in writing of clearer code. While in some scenarios, if there may be fitting to take advantage of it, we should use it with the best judgment.

☞ Hoisting

Module 2

1. Hoisting of var

Hoisting is a mechanism of JavaScript. In hoisting, variables and function declarations are moved to the top of their enclosing scope before code execution. We can understand it with the following example.

Example

```
function get(x){
  console.log(a);  //printing x variable. Value is undefined
  //declared variable after console hoisted to the top at run time
var a = x;
  //again printing x variable. Value is 3.
  console.log(a);
}
get(4);
```

Output:

```
Undefined
4
```

2. Hoisting of let

A variable declared with **let** keyword is not hoisted. If we try to use a let variable before it is declared, then it will result in a **ReferenceError**.

Example

```
{
  //program doesn't know about variable b so it will give me an error.
  console.log(b); // ReferenceError: b is not defined
  let b = 3;
}
```

3. const declarations

- The const declaration is used to declare permanent value, which cannot be changed later.
- It has a fixed value.

- The const declaration follows the same scoping rules as let declaration, but we cannot re-assign any new value to it.

Example

```
function constTest(){
const VAR = 10;
console.log("Value is: " +VAR);
}
constTest();
```

Output

```
Value is: 100
```

- What will happen when we try to re-assign the const variable?

If we try to re-assign the existing const variable in a code, the code will throw an error. So, we cannot re-assign any new value to an existing const variable.

Example

```
function constTest(){
  const VAR = 100;
  console.log("Output: " +VAR);  // Output: 100
  const VAR = 100;
  console.log("Output: " +VAR);  //Uncaught TypeError:
Assignment to constant variable
}
constTest();
```

Output

```
SyntaxError: Identifier 'VAR' has already been declared.
```

2.6.2 var Vs let keyword

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;
		}

2.7 Typescript Operators

- An Operator is a symbol which operates on a value or data.
- It represents 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.
- Example

  10 + 20 = 30;

  In the above example, the values '10', '20' and '30' are known as an operand, whereas '+' and '=' are known as operators.
- In TypeScript, an operator can be classified into the following ways.
  1. Arithmetic operators
  2. Comparison (Relational) operators
  3. Logical operators
  4. Bitwise operators
  5. Assignment operators
  6. Ternary/conditional operator
  7. Concatenation operator
  8. Type Operator

2.7.1 Arithmetic Operators

Arithmetic operators take numeric values as their operands, performs an action, and then returns a single numeric value.

The most common arithmetic operators are addition(+), subtraction(–), multiplication(*), and division(/).

Operator	Operator_Name	Description	Example

+	Addition	It returns an addition of the values.	let a = 10;
			let b = 20;
			let c = a + b;
			console.log(c);
			//Output30

-	Subtraction	It returns the difference of the values.	let a = 20;
			let b = 10;
			let c = a - b;
			console.log(c);
			//Output10

*	Multiplication	It returns the product of the values.	let a = 30;
			let b = 20;
			let c = a * b;
			console.log(c);
			//Output600

/	Division	It performs the division operation, and returns the quotient.	let a = 100;
			let b = 20;
			let c = a / b;
			console.log(c);
			//Output5

Operator	Operator_Name	Description	Example

%	Modulus	It performs the division operation and returns the remainder.	let a = 75;
			let b = 20;
			let c = a % b;
			console.log(c);
			//Output15

++	Increment	It is used to increments the value of the variable by one.	let a = 15;
			a++;
			console.log(a);
			//Output16

--	Decrement	It is used to decrements the value of the variable by one.	let a = 15;
			a--;
			console.log(a);
			//Output14

2.7.2 Comparison (Relational) Operators

The comparison operators are used to compares the two operands. These operators return a Boolean value true or false. The important comparison operators are given below.

Operator	Operator_Name	Description	Example

==	Is equal to	It checks whether the values of the two operands are equal or not.	let a = 10;
			let b = 20;
			console.log(a==b); //false
			console.log(a==10); //true
			console.log(10=='10'); //true

===	Identical(equal and of the same type)	It checks whether the type and values of the two operands are equal or not.	let a = 10;
			let b = 20;
			console.log(a===b); //false
			console.log(a===10); //true
			console.log(10==='10'); //false

!=	Not equal to	It checks whether the values of the two operands are equal or not.	let a = 10;
			let b = 20;
			console.log(a!=b); //true
			console.log(a!=10); //false
			console.log(10!='10'); //false

!==	Not identical	It checks whether the type and values of the two operands are equal or not.	let a = 10;
			let b = 20;
			console.log(a!==b); //true
			console.log(a!==10); /false
			console.log(10!=='10'); //true

>	Greater than	It checks whether the value of the left operands is greater than the value of the right operand or not.	let a = 30;
			let b = 20;
			console.log(a>b); //true
			console.log(a>30); //false
			console.log(20> 20'); //false

Operator	Operator_Name	Description	Example

>=	Greater than or equal to	It checks whether the value of the left operands is greater than or equal to the value of the right operand or not.	let a = 20;
			let b = 20;
			console.log(a>=b); //true
			console.log(a>=30); //false
			console.log(20>='20'); //true

<	Less than	It checks whether the value of the left operands is less than the value of the right operand or not.	let a = 10;
			let b = 20;
			console.log(a<b); //true
			console.log(a<10); //false
			console.log(10<'10'); //false

<=	Less than or equal to	It checks whether the value of the left operands is less than or equal to the value of the right operand or not.	let a = 10;
			let b = 20;
			console.log(a<=b); //true
			console.log(a<=10); //true
			console.log(10<='10'); //true

2.7.3 Logical Operators

Logical operators are used for combining two or more condition into a single expression and return the Boolean result true or false. The Logical operators are given below.

Operator	Operator_Name	Description	Example

&&	Logical AND	It returns true if both the operands(expression) are true, otherwise returns false.	let a = false;
			let b = true;
			console.log(a&&b); /false
			console.log(b&&true); //true
			console.log(b&&10); //10 which is also 'true'
			console.log(a&&'10'); //false

||	Logical OR	It returns true if any of the operands(expression) are true, otherwise returns false.	let a = false;
			let b = true;
			console.log(a||b); //true
			console.log(b||true); //true
			console.log(b||10); //true
			console.log(a||'10'); //'10' which is also 'true'

!	Logical NOT	It returns the inverse result of an operand(expression).	let a = 20;
			let b = 30;
			console.log(!true); //false
			console.log(!false); //true
			console.log(!a); //false
			console.log(!b); /false
			console.log(!null); //true

2.7.4 Bitwise Operators

The bitwise operators perform the bitwise operations on operands. The bitwise operators are as follows.

Operator	Operator_Name	Description	Example

&	Bitwise AND	It returns the result of a Boolean AND operation on each bit of its integer arguments.	let a = 2;
			let b = 3;
			let c = a & b;
			console.log(c);
			//Output2

|	Bitwise OR	It returns the result of a Boolean OR operation on each bit of its integer arguments.	let a = 2;
			let b = 3;
			let c = a | b;
			console.log(c); // Output 3

^	Bitwise XOR	It returns the result of a Boolean Exclusive OR operation on each bit of its integer arguments.	let a = 2;
			let b = 3;
			let c = a ^ b;
			console.log(c); //Output1

~	Bitwise NOT	It inverts each bit in the operands.	let a = 2;
			let c = ~ a;
			console.log(c); //Output-3

>>	Bitwise Right Shift	The left operand's value is moved to the right by the number of bits specified in the right operand.	let a = 2;
			let b = 3;
			let c = a >> b;
			console.log(c); //Output0

<<	Bitwise Left Shift	The left operand's value is moved to the left by the number of bits specified in the right operand. New bits are filled with zeroes on the right side.	let a = 2;
			let b = 3;
			let c = a << b;
			console.log(c); //Output16

>>>	Bitwise Right Shift with Zero	The left operand's value is moved to the right by the number of bits specified in the right operand and zeroes are added on the left side.	let a = 3;
			let b = 4;
			let c = a >>> b;
			console.log(c); //Output0

2.7.5 Assignment Operators

Assignment operators are used to assign a value to the variable. The left side of the assignment operator is called a variable, and the right side of the assignment operator is called a value. The data-type of the variable and value must be the same otherwise the compiler will throw an error. The assignment operators are as follows.

Operator	Operator_Name	Description	Example

=	Assign	It assigns values from right side to left side operand.	let a = 10;
			let b = 5;
			console.log("a=b:" +a); //
			Output 10

+=	Add and assign	It adds the left operand with the right operand and assigns the result to the left side operand.	let a = 10;
			let b = 5;
			let c = a += b;
			console.log(c); //Output15

Operator	Operator_Name	Description	Example

-=	Subtract and assign	It subtracts the right operand from the left operand and assigns the result to the left side operand.	let a = 10;
			let b = 5;
			let c = a -= b;
			console.log(c); //Output 5

*=	Multiply and assign	It multiplies the left operand with the right operand and assigns the result to the left side operand.	let a = 10;
			let b = 5;
			let c = a *= b;
			console.log(c); //Output 50

/=	Divide and assign	It divides the left operand with the right operand and assigns the result to the left side operand.	let a = 10;
			let b = 5;
			let c = a /= b;
			console.log(c); //Output2

%=	Modulus and assign	It divides the left operand with the right operand and assigns the result to the left side operand.	let a = 16;
			let b = 5;
			let c = a %= b;
			console.log(c); //Output1

2.7.6 Ternary/ Conditional Operator

The conditional operator takes three operands and returns a Boolean value based on the condition, whether it is true or false. Its working is similar to an if-else statement. The conditional operator has right-to-left associativity. The syntax of a conditional operator is given below.

```
expression ? expression-1 : expression-2;
```

- **Expression :** It refers to the conditional expression.
- **expression-1 :** If the condition is true, expression-1 will be returned.
- **expression-2 :** If the condition is false, expression-2 will be returned.

Example

```
let num = 10;
let result = (num > 0) ? "True":"False"
console.log(result);
```

Output

```
True
```

2.7.7 Concatenation Operator

- The concatenation (+) operator is an operator which is used to append the two string.
- In concatenation operation, we cannot add a space between the strings.
- We can concatenate multiple strings in a single statement.
- The following example helps us to understand the concatenation operator in TypeScript.

Example

```
let message = "Welcome to " + "WebX.0";
console.log("Result of String Operator: " +message);
```

Output

```
Result of String Operator: Welcome to WebX.0
```

2.7.8 Type Operators

- These are a collection of operators available which can assist you when working with objects in TypeScript.
- Operators such as typeof, instanceof, in, and delete are the examples of Type operator.
- The detail explanation of these operators is given below.

Operator_Name	Description	Example
In	It is used to check for the existence of a property on an object.	let Bike = {make: 'Honda', model: 'CLIQ', year: 2018}; console.log('make' in Bike); // Output:true

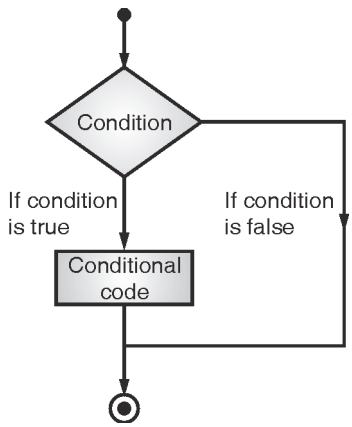
delete	It is used to delete the properties from the objects.	let Bike = { Company1: 'Honda',
		Company2: 'Hero',
		Company3: 'Royal Enfield'
		};
		delete Bike.Company1;
		console.log(Bike);
		//Output:{ Company2: 'Hero', Company3: 'Royal Enfield' }

typeof	It returns the data type of the operand.	let message = "Welcome to " + "Techneo";
		console.log(typeof message);
		//Output:String

instanceof	It is used to check if the object is of a specified type or not.	let arr = [1, 2, 3];
		console.log(arrinstanceof Array); // true
		console.log(arrinstanceof String); // false

2.8 DECISION MAKING IN TYPESCRIPT

- Decision-making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- Fig. 2.8.1, Shows the general form of a typical decision-making structure found in most of the programming languages:



(1B6)**Fig. 2.8.1**

- A decision-making construct evaluates a condition before the instructions are executed.

Decision-making constructs in TypeScript are classified as follows:

2.8.1 if statement

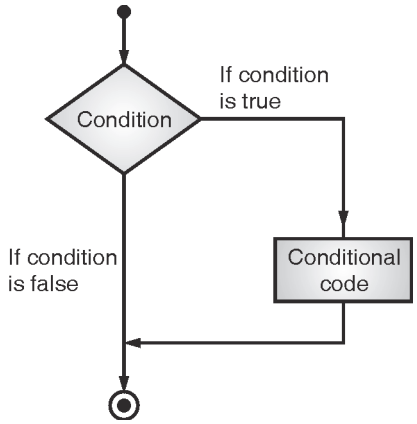
- An 'if' statement consists of a Boolean expression followed by one or more statements.
- The 'if' construct evaluates a condition before a block of code is executed.

Syntax

```
if(boolean_expression) {
  // statement(s) will execute if the boolean expression is true
}
```

- If the Boolean expression evaluates to true, then the block of code inside the if statement will be executed.
- If the Boolean expression evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Flowchart



(1B7)**Fig. 2.8.2 : Flowchart**

Example

```
varnum:number = 5
if (num> 0) {
console.log("number is positive")
}
```

Output

```
number is positive
```

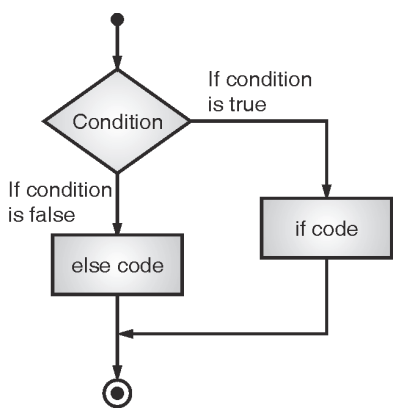
2.8.2 if...else statement

An 'if' statement can be followed by an optional 'else' statement, which executes when the Boolean expression is false.

Syntax

```
if(boolean_expression) {
  // statement(s) will execute if the boolean expression is true
} else {
  // statement(s) will execute if the boolean expression is false
}
```

Flowchart



(1B8)**Fig. 2.8.3**

- The **if** block guards the conditional expression. The block associated with the **if** statement is executed if the Boolean expression evaluates to true.
- The **if** block may be followed by an optional **else** statement. The instruction block associated with the else block is executed if the expression evaluates to false.

Example

```
varnum:number = 12;
if (num % 2==0) {
  console.log("Even");
} else {
  console.log("Odd");
}
```

Output

```
Even
```

2.8.3 else...if and nested if statements

- You can use one 'if' or 'else if' statement inside another 'if' or 'else if' statement(s).
- The **else…if** ladder is useful to test multiple conditions.

Syntax

```
if (boolean_expression1) {
  //statements if the expression1 evaluates to true
} else if (boolean_expression2) {
  //statements if the expression2 evaluates to true
} else {
  //statements if both expression1 and expression2 result to
false
}
```

- When using **if…else…if** and **else** statements, there are a few points to keep in mind.
  1. An **if** can have zero or one **else's** and it must come after any **else..if**'s.
  2. An **if** can have zero to many **else..if**'s and they must come before the **else**.
  3. Once an **else..if** succeeds, none of the remaining **else..if**'s or **else**'s will be tested.

Example

The snippet displays whether the value is positive, negative or zero.

Module 2

```
varnum: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")
}
```

Output

```
2 is positive
```

2.8.4 switch statement

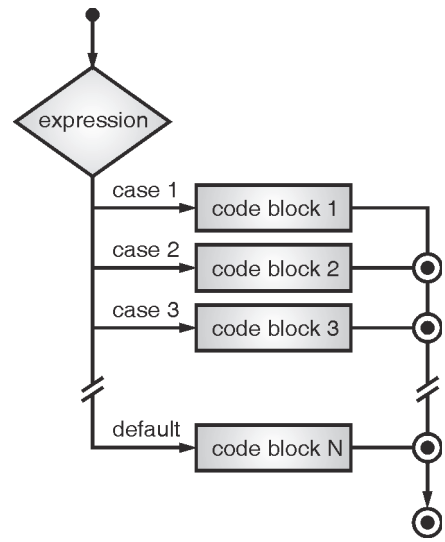
- A 'switch' statement allows a variable to be tested against a list of values.
- The **switch** statement evaluates an expression, matches the expression's value to a case clause, and executes statements associated with that case.

Syntax

```
switch(variable_expression) {
  case constant_expr1: {
    //statements;
    break;
  }
  case constant_expr2: {
    //statements;
    break;
  }
  default: {
    //statements;
    break;
  }
}
```

- The value of the variable_expression is tested against all cases in the switch. If the variable matches one of the cases, the corresponding code block is executed. If no case expression matches the matches the value of the variable_expression, the code within the default block is associated.
- The following rules apply to a switch statement:
  1. There can be any number of case statements within a switch.
  2. The case statements can include only constants. It cannot be a variable or an expression.
  3. The data type of the variable_expression and the constant expression must match.
  4. Unless you put a break after each block of code, execution flows into the next block.
  5. The case expression must be unique.
  6. The default block is optional.

Flowchart



(1B9)**Fig. 2.8.4**

Example

The example verifies the value of the variable grade against the set of constants (A, B, C, D, and E) and executes the corresponding blocks. If the value in the variable doesn't match any of the constants mentioned above, the default block will be executed.

```
vargrade:string = "A";
switch(grade) {
  case "A": {
    console.log("Excellent");
    break;
  }
  case "B": {
    console.log("Good");
    break;
  }
  case "C": {
    console.log("Fair");
    break;
  }
  case "D": {
```

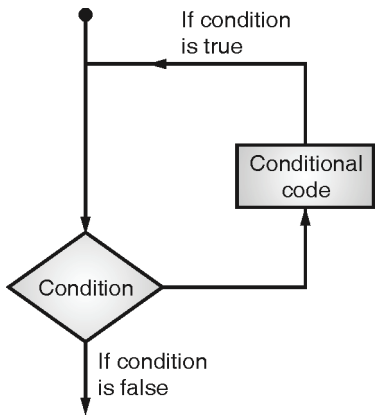
```
    console.log("Poor");
    break;
  }
  default: {
console.log("Invalid choice");
    break;
  }
}
```

Output

```
Excellent
```

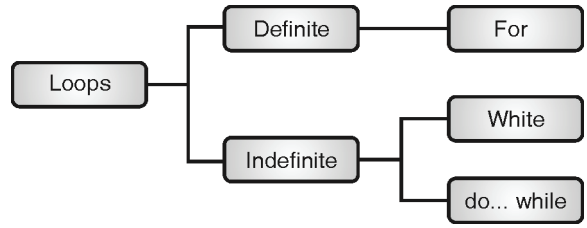
2.9 LOOPS IN TYPESCRIPT

- You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times.
- Fig. 2.9.1 shows the general form of a loop statement in most of the programming languages.



(1B10)**Fig. 2.9.1**

- TypeScript provides different types of loops to handle looping requirements.
- The Fig. 2.9.2 illustrates the classification of loops:



(1B11)**Fig. 2.9.2**

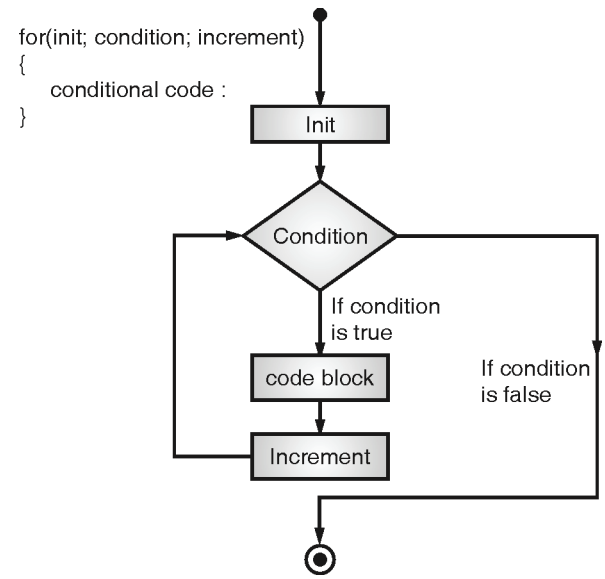
2.9.1 Definite Loop

- A loop whose number of iterations are definite/fixed is termed as a **definite loop**. The *for loop* is an implementation of a definite loop.
- The **for** loop executes the code block for a specified number of times. It can be used to iterate over a fixed set of values, such as an array.
- The syntax of the **for** loop is as below

```
for (initial_count_value; termination-condition; step) {
  //statements
}
```

- The loop uses a count variable to keep track of the iterations. The loop initializes the iteration by setting the value of *count* to its initial value. It executes the code block; each time the value of *count* satisfies the termination_condtion. The *step* changes the value of *count* after every iteration.

Flowchart



(1B12)**Fig. 2.9.3**

Example

```
varnum:number = 5;
vari:number;
var factorial = 1;

for(i = num;i>=1;i--) {
  factorial *= i;
}
console.log(factorial)
```

Output

```
120
```

The for...in loop

- Another variation of the *for* loop is the *for... in* loop. The *for… in* loop can be used to iterate over a set of values as in the case of an array or a tuple.
- The for...in loop is used to iterate through a list or collection of values. The data type of *val* here should be string or any.
- The syntax of the **for..in** loop is as given below

```
for (varval in list) {
  //statements
}
```

Example

```
varj:any;
varn:any = "123"

for(j in n) {
  console.log(n[j])
}
```

Output

```
1
2
3
```

2.9.2 Indefinite Loop

- An indefinite loop is used when the number of iterations in a loop is indeterminate or unknown.
- Indefinite loops can be implemented using while and do…while loop.

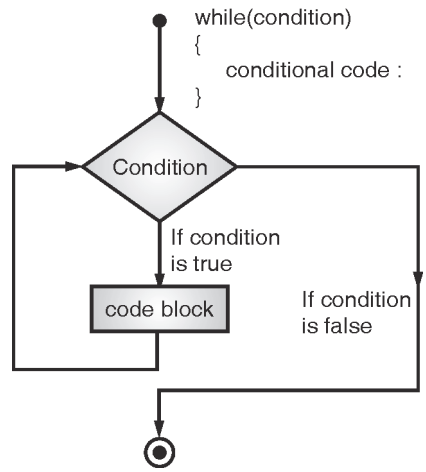
2.9.2(A)while loop

- The **while** loop executes the instructions each time the condition specified evaluates to true.
- In other words, the loop evaluates the condition before the block of code is executed.

Syntax

```
while(condition) {
  // statements if the condition is true
}
```

Flowchart



(1B13)**Fig. 2.9.4**

Example

The below code snippet uses a **while** loop to calculate the factorial of the value in the variable num.

```
varnum:number = 5;
varfactorial:number = 1;

while(num>=1) {
  factorial = factorial * num;
num--;
}
console.log("The factorial  is "+factorial);
```

Output

```
The factorial is 120
```

2.9.2(B)do...while loop

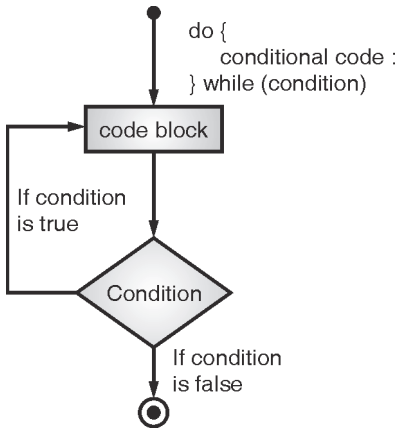
- The do…while loop is similar to the while loop except that the do...while loop doesn't evaluate the condition for the first time the loop executes. However, the condition is evaluated for the subsequent iterations.

- In other words, the code block will be executed at least once in a do…while loop.

Syntax

```
do {
  //statements
} while(condition)
```

Flowchart



(1B14)**Fig. 2.9.5**

Example

```
varn:number = 4;
do {
  console.log(n);
  n--;
} while(n>=0);
```

Output

```
4
3
2
1
0
```

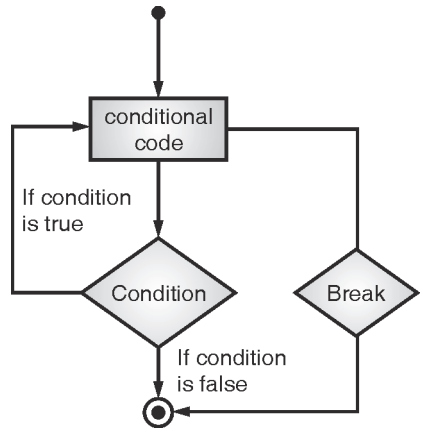
2.9.3 break statement

- The **break** statement is used to take the control out of a construct.
- Using **break** in a loop causes the program to exit the loop.

Syntax

```
Break
```

Flowchart



(1B15)**Fig. 2.9.6**

Example

```
vari:number = 1
while(i<=10) {
  if (i % 5 == 0) {
    console.log ("The first multiple of 5 between 1 and 10 is :
"+i)
    break    //exit the loop if the first multiple is found
  }
i++
}  //outputs 5 and exits the loop
```

Output

```
The first multiple of 5 between 1 and 10 is : 5
```

2.9.4 continue statement

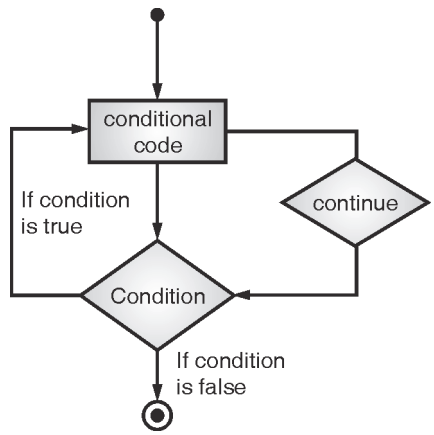
- The **continue** statement skips the subsequent statements in the current iteration and takes the control back to the beginning of the loop.
- Unlike the break statement, the continue doesn't exit the loop.
- It terminates the current iteration and starts the subsequent iteration.

Module 2

Syntax

```
continue
```

Flowchart



(1B16)**Fig. 2.9.7**

Example

The below example displays the number of odd values between 0 and 20. The loop exits the current iteration if the number is even. This is achieved using the **continue** statement.

```
varnum:number = 0
varcount:number = 0;

for(num=0;num<=20;num++) {
  if (num % 2==0) {
    continue
  }
  count++
}
console.log (" The count of odd values between 0 and 20 is:
"+count)
```

Output

```
The count of odd values between 0 and 20 is: 10
```

2.9.5 infinite loop

- An infinite loop is a loop that runs endlessly.
- The **for** loop and the **while** loop can be used to make an endless loop.
- **Syntax:** Infinite Loop using **for** loop

```
for(;;) {
  //statements
}
```

- **Example**: Infinite Loop using **for** loop

```
for(;;) {
console.log("This is an endless loop")
}
```

- **Syntax:** Infinite Loop using **while** loop

```
while(true) {
  //statements
}
```

- **Example**: Infinite Loop using **while** loop

```
While(true){
console.log("This is an endless loop")
}
```

2.10 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.

2.10.1 Advantages of Functions

These are the main advantages of functions.

1. **Code reusability** : We can call a function several times without writing the same block of code again. The code reusability saves time and reduces the program size.
2. **Less coding** : Functions makes our program compact. So, we don't need to write many lines of code each time to perform a common task.
3. **Easy to debug** : It makes the programmer easy to locate and isolate faulty information.

2.10.2 Function Aspects

There are three aspects of a function.

- **Function declaration:** A function declaration tells the compiler about the function name, function parameters, and return type.

- The syntax of the function declaration is:

```
function functionName( [arg1, arg2, ...argN] );
```

- **Function definition :** It contains the actual statements which are going to executes. It specifies what and how a specific task would be done. The syntax of the function definition is:

```
function functionName( [arg1, arg2, ...argN] ){
    //code to be executed
}
```

- **Function call :** We can call a function from anywhere in the program. The parameter/argument cannot differ in function calling and a function declaration. We must pass the same number of functions as it is declared in the function declaration. The syntax of the function call is:

```
FunctionName();
```

2.10.3 Function Creation

We can create a function in two ways. These are:

1. Named Function
2. Anonymous Function

2.10.3(A) Named Functions

When we declare and call a function by its given name, then this type of function is known as a **named** function.

Syntax

```
functionName( [arguments] ) { }
```

Example

```
//Function Definition
function display() {
console.log("Hello WebX.0!");
}
//Function Call
display();
```

Output

```
Hello WebX.0!
```

2.10.3(B) Anonymous Function

- 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 expression.
- We can store it in a variable, so it does not need function names.
- Like standard function, it also accepts inputs and returns outputs.
- We can invoke it by using the variable name, which contains function.

Syntax

```
let res = function( [arguments] ) { }
```

Example

```
// Anonymous function
let myAdd = function (x: number, y: number) : number {
   return x + y;
};
// Anonymous function call
console.log(myAdd(2,3));
```

Output

```
5
```

2.10.4 Function Parameter

- Parameters are the values or arguments that passed to a function.
- In TypeScript, the compiler accepts the same number and type of arguments as defined in the function signature.
- If the compiler does not match the same parameter as in the function signature, then it will give the compilation error.
- Function parameter can be categories into the following:
  1. Optional Parameter
  2. Default Parameter
  3. Rest Parameter

2.10.4(A) Optional Parameter

- In JavaScript, we can call a function without passing any arguments. Hence, in a JavaScript function, the parameter is optional, and when we do this, each parameter value is undefined.
- Unlike JavaScript, the TypeScript compiler will throw an error if we try to invoke a function without

Module 2

providing the exact number and types of parameters as declared in its function signature.

- To overcome this problem, we can use optional parameters by using the question mark sign ('?'). It means that the parameters which may or may not receive a value can be appended with a "?" sign to mark them as optional.
- In below example, **e_mail_id** is marked as an optional parameter.

Syntax

```
function function_name(parameter1[:type], parameter2[:type], parameter3 ? [:type]) { }
```

Example

```
function
showDetails(id:number,name:string,e_mail_id?:string) {
console.log("ID:", id, " Name:",name);
  if(e_mail_id!=undefined)
  console.log("Email-Id:",e_mail_id);
}
showDetails(101,"Nilesh Patil");
showDetails(105,"Sachin","sachin@techneobooks.com");
```

Output

```
ID: 101  Name: Nilesh Patil
ID: 105  Name: Sachin
Email-Id: sachin@techneobooks.com
```

2.10.4(B) Default Parameter

- TypeScript provides an option to set default values to the function parameters.
- If the user does not pass a value to an argument, TypeScript initializes the default value for the parameter.
- The behavior of the default parameter is the same as an optional parameter.
- For the default parameter, if a value is not passed in a function call, then the default parameter must follow the required parameters in the function signature.
- However, if a function signature has a default parameter before a required parameter, we can still call a function which marks the default parameter is passed as an undefined value.

Note : We cannot make the parameter optional and default at the same time.

Syntax

```
function function_name(parameter1[:type], parameter2[:type] = default_value) { }
```

Example

```
functiondisplayName(name: string, greeting: string = "Hello") : string {
   return greeting + ' ' + name + '!';
}
console.log(displayName('TechNeo'));   //Returns "Hello TechNeo!"
console.log(displayName('TechNeo', 'Hi'));   //Returns "Hi TechNeo!".
console.log(displayName('Sachin'));   //Returns "Hello Sachin!"
```

Output

```
Hello TechNeo!
Hi TechNeo!
Hello Sachin!
```

2.10.4(C) Rest Parameter

- The rest parameter is used to pass **zero or more** values to a function.
- We can declare it by prefixing the **three "dot"** characters ('...') before the parameter.
- It allows the functions to have a different number of arguments without using the arguments object.
- The TypeScript compiler will create an array of arguments with the rest parameter so that all array methods can work with the rest parameter.
- The rest parameter is useful, where we have an undetermined number of parameters.
- Rules to follow in rest parameter :
  1. Only one rest parameter is allowed in a function.
  2. It must be an array type.
  3. It must be the last parameter in a parameter list.

Syntax

```
function function_name(parameter1[:type], parameter2[:type], ...parameter[:type]) { }
```

Example

```
function sum(a: number, ...b: number[]): number {
  let result = a;
  for (vari = 0; i<b.length; i++) {
  result += b[i];
  }
  return result;
}
let result1 = sum(3, 5);
let result2 = sum(3, 5, 7, 9);
console.log(result1 +"\n" + result2);
```

Output

```
8
24
```

2.10.5 TypeScript Arrow Function (Lambda Function)

- ES6 version of TypeScript provides an arrow function which is the shorthand syntax for defining the anonymous function, i.e., for function expressions.
- It omits the function keyword.
- We can call it fat arrow (because -> is a thin arrow and => is a "fat" arrow).
- It is also called a Lambda function.
- The arrow function has lexical scoping of "this" keyword.
- The motivation for arrow function is:
  1. When we don't need to keep typing function.
  2. It lexically captures the meaning of this keyword.
  3. It lexically captures the meaning of arguments.

Syntax

We can split the syntax of an Arrow function into three parts :

- Parameters: A function may or may not have parameters.
- The arrow notation/lambda notation (=>).
- Statements: It represents the function's instruction set.

```
(parameter1, parameter2, ..., parameterN) => expression;
```

- If we use the **fat arrow (=>)** notation, there is no need to use the **function** keyword. Parameters are passed in the brackets (), and the function expression is enclosed within the curly brackets {}.

- **Arrow function with Parameter**

  The following program is an example of arrow function with parameters.

```
let sum = (x: number, y: number): number => {
   return x + y;
}
console.log(sum(10, 20)); //returns 30
```

- In the above example, **sum** is an arrow function. **(x:number, y:number)** denotes the parameter types, **:number** specifies the return type. The fat arrow => separates the function parameters and the function body. The right side of **=>** can contain one or more code statements.

- **Arrow function without a parameter**

  The following program is an example of arrow function without parameters.

```
let Print = () =>console.log("Hello TypeScript");
Print(); //Output: Hello TypeScript
```

- In the arrow function, if the function body consists of only one statement, then there is no need of the curly brackets and the return keyword. We can understand it from the below example.

```
let sum = (x: number, y: number) => x + y;
console.log(sum(3, 4)); //returns 7
```

- **Arrow function in a class**

  We can include the arrow function as a property in a class. The following example helps to understand it more clearly.

```
class Employee {
   empCode: number;
   empName: string;

   constructor(code: number, name: string) {
      this.empName = name;
      this.empCode = code;
   }

display = () =>console.log("empCode: "+ this.empCode
+'\nempName: ' + this.empName)
}
let emp = new Employee(1, 'Sachin');
emp.display();
```

Module 2

Output

```
empCode:1
empName: Sachin
```

2.10.6 TypeScript Function Overloading

- 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 is also known as method overloading.
- **The Function/Method overloading is allowed when**:
  1. The function name is the same
  2. The number of parameters is different in each overloaded function.
  3. The number of parameters is the same, and their type is different.
  4. The all overloads function must have the same return type.
- Suppose we have to perform **multiplication** of the numbers, which has a different number of parameters. We write the **two** methods such as mul_a(number, number) for **two parameters**, and mul_b(number, number, number) for **three parameters**. Now, it may be difficult for us as well as other programmers to understand the behavior of the method because its name **differs**. In that case, we need to use function overloading, which increases the readability of the program.

Advantages of function overloading

1. It saves the memory space so that program execution becomes fast.
2. It provides code reusability, which saves time and efforts.
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
```

In the above example :

- The first **two** lines are the function overload **declaration**. It has two overloads:
  1. A Function which accepts a **string** parameter.
  2. A Function which accepts a **number** parameter.
- The third line is the **function definition**, where the data type of the parameters is set to **any**.
- The last two statements **invoke** the overloaded function.

Function overloading in a class

The following example helps to understand the use of method overloading in a class.

```
class A
{
   public foo(s: string): number;
   public foo(n: number): string;
   public foo(arg: any): any
   {
      if (typeof(arg) === 'number')
         return arg.toString();
      if (typeof(arg) === 'string')
         return arg.length;
   }
}
let obj = new A();
console.log("Result: " +obj.foo(101));
console.log("Length of String: "+ obj.foo("TechNeoBooks"));
```

Output

```
Result: 101
Length of String: 12
```

Note : Function overloading with a different number of parameters and different types along with the same function name is not supported.

2.11 TYPESCRIPT CLASSES AND OBJECTS

- In object-oriented programming languages like Java and C#, classes are the fundamental entities used to create reusable components.
- Functionalities are passed down to classes and objects are created from classes. However, until ECMAScript 6 (also known as ECMAScript 2015), this was not the case with JavaScript.
- JavaScript has been primarily a functional programming language where inheritance is prototype-based.
- Functions are used to build reusable components.
- 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 to declare a class

A class keyword is used to declare a class in TypeScript. We can create a class with the following 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+" ;
   }
}
```

2.11.1 Creating an Object of a Class

- 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.
- We can create an object as below.

Syntax

```
let object_name = new class_name(parameter)
```

1. **new keyword :** It is used for instantiating the object in memory.
2. The right side of the expression invokes the constructor, which can pass values.

Example

```
//Creating an object or instance
let obj = new Student();
```

2.11.2 Object Initialization

- Object initialization means storing of data into the object.
- There are three ways to initialize an object. These are:

1. By Reference Variable

Example

```
/Creating an object or instance
let obj = new Student();
//Initializing an object by reference variable
obj.id = 101;
obj.name = "Sachin Shah";
```

2. By Method

A method is similar to a function used to expose the behavior of an object.

A method provides code reusability and code optimization.

Example

```
//Defining a Student class.
class Student {
Sid:number;
Sname:string;

   //creating method or function
display():void {
console.log("Student ID is: "+this.Sid)
console.log("Student Name is: "+this.Sname)
   }
}

//Creating an object or instance
letstd = new Student();
std.Sid = 101;
std.Sname = "Sachin Shah";
std.display();
```

Output

```
Student ID is: 101
Student Name is: Sachin Shah
```

3. By Constructor

A constructor is used to **initialize** an object. In TypeScript, the constructor method is always defined with the name "**constructor**." In the constructor, we can access the member of a class by using **this** keyword.

Note : It is not necessary to always have a constructor in the class.

Example

```
//Defining a Student class.
classStudentData {
   //defining fields
   id: number;
name:string;

   //defining constructor
constructor(id: number, name:string) {
      this.id = id;
      this.name = name;
  }
  //creating method or function
display():void {
console.log("Function displays Student ID as: "+this.id)
console.log("Function displays Student Name as:
"+this.name)
   }
}

//Creating an object or instance
let stud = new StudentData(101, "Sachin")

//access the field
console.log("Reading attribute value of Student as: "
+stud.id,)
console.log("Reading attribute value of Student as: "
+stud.name)

//access the method or function
stud.display()
```

Output

```
Reading attribute value of Student as: 101
Reading attribute value of Student as: Sachin
Function displays Student ID as: 101
Function displays Student Name as: Sachin
```

2.11.3 Data Hiding

- It is a technique which is used to hide the internal object details.
- A class can control the visibility of its data members from the members of the other classes. This capability is termed as encapsulation or data-hiding.
- OOPs uses the concept of access modifier to implement the encapsulation.
- The access modifier defines the visibility of class data member outside its defining class.
- TypeScript supports the three types of access modifier. These are: Public, Private and Protected.

1. Public Access Modifier

- In TypeScript by default, all the members (properties and methods) of a class are public.
- So, there is no need to prefix members with this keyword.

- We can access this data member anywhere without any restriction.

Example

```
class Student {
   public studCode: number;
studName: string;
}
let stud = new Student();
stud.studCode = 101;
stud.studName = "Sachin Shah";
console.log(stud.studCode+ " "+stud.studName);
```

Output

```
101 Sachin Shah
```

- In the above example, **studCode** is public, and **studName** is declared without a modifier, so TypeScript treats them as **public** by default. Since data members are public, they can be accessed outside of the class using an object of the class.

2. Private Access Modifier

- The private access modifier cannot be accessible outside of its containing class.
- It ensures that the class members are visible only to that class in which it is containing.

Example

```
class Student {
public studCode: number;
private studName: string;
constructor(code: number, name: string){
this.studCode = code;
this.studName = name;
}
public display() {
return (`My unique code: ${this.studCode}, my name:
${this.studName}.`);
}
}

let student: Student = new Student(1, "Sachin Shah");
console.log(student.display());
```

Output

```
My unique code: 1, my name: Sachin Shah.
```

In the above example, **studCode** is private, and **studName** is declared without a modifier, so TypeScript treats it as public by default. If we access the private member outside of the class, it will give a compile error.

3. Protected Access Modifier

- A Protected access modifier can be accessed only within the class and its subclass.
- We cannot access it from the outside of a class in which it is containing.

Module 2

Example

```
class Student {
   public studCode: number;
   protected studName: string;
constructor(code: number, name: string){
this.studCode = code;
this.studName = name;
      }
}
class Person extends Student {
   private department: string;

constructor(code: number, name: string, department: string) {
super(code, name);
this.department = department;
   }
   public getProfile() {
      return (`My unique code: ${this.studCode}, my name:
${this.studName} and I am in ${this.department} Branch.`);
   }
}
let sachinShah: Person = new Person(1, "SachinShah", "CS");
console.log(sachinShah.getProfile());
```

Output

```
My unique code: 1, my name: SachinShah and I am in CS
Branch.
```

In the above example, we can't use the name from outside of **Student** class. We can still use it from within an instance method of Person class because **Person** class derives from Student class.

4. Readonly Modifier

- We can make the properties of the class, type, or interface readonly by using the readonly modifier.
- This modifier needs to be initialized at their declaration time or in the constructor.
- We can also access readonly member from the outside of a class, but its value cannot be changed.

Example

```
class Company {
readonly city: string = "Pune";
readonly name: string;

constructor(contName: string) {
 this.name = contName;
 }
showDetails() {
 console.log(this.name + " : " + this.city);
 }
}
let comp = new Company("Techneo Publications");
comp.showDetails(); // Techneo Publications : Pune
comp.name = "TCS"; //Error, name can be initialized only
within constructor
```

Output

```
TechneoPublications : Pune
```

2.12 TYPESCRIPT INHERITANCE

- Inheritance is an aspect of OOPs languages, which provides the ability of a program to create a new class from an existing class.
- It is a mechanism which acquires the **properties** and **behaviors** of a class from another class.
- The class whose members are inherited is called the **base class**, and the class that inherits those members is called the **derived/child/subclass**.
- In child class, we can override or modify the behaviors of its parent class.
- The TypeScript uses class inheritance through the **extends** keyword.
- TypeScript supports only **single** inheritance and **multilevel** inheritance. It doesn't support multiple and hybrid inheritance.

Syntax

We can declare a class inheritance as below.

```
class sub_class_name extends super_class_name
{
    // methods and fields
}
```

- We can use inheritance for Method Overriding (so runtime polymorphism can be achieved) and code reusability.

Example

```
class Car {
Color:string
  constructor(color:string) {
this.Color = color
  }
}
class Audi extends Car {
   Price: number
constructor(color: string, price: number) {
      super(color);
this.Price = price;
   }
display():void {
console.log("Color of Audi car: " + this.Color);
console.log("Price of Audi car: " + this.Price);
   }
}
let obj = new Audi(" Black", 8500000 );
obj.display();
```

Output

```
Color of Audi car: Black
Price of Audi car: 8500000
```

In the above example, the **Audi class** extends the **Car class** by using the **extends** keyword. It means the Audi class can include all the members of the Car class. The constructor of the Audi class initializes its own members as well as the parent class's properties by using a special keyword '**super**.' The super keyword is used to call the parent constructor and its values.

2.12.1 Single Inheritance

- Single inheritance can inherit properties and behavior from at most **one parent class**.
- It allows a derived/subclass to inherit the properties and behavior of a base class that enable the **code reusability** as well as we can add new features to the existing code.
- The single inheritance makes the code less repetitive.

Example:

```
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()
```

Output

```
Area of the square: 25
```

2.12.2 Multilevel Inheritance

- When a derived class is derived from another derived class, then this type of inheritance is known as **multilevel inheritance**.
- Thus, a multilevel inheritance has more than one parent class.
- It is similar to the **relation** between Grandfather, Father, and Child.

Example:

```
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();
```

Output

```
Eating
Barking
Weeping
```

2.13 TYPESCRIPT INTERFACES

- An Interface is a structure which acts as a **contract** in our application.
- It defines the syntax for classes to follow, means a class which implements an interface is bound to implement all its members.
- We cannot instantiate the interface, but it can be referenced by the class object that implements it.
- The TypeScript compiler uses interface for **type-checking** (also known as "duck typing" or "structural subtyping") whether the object has a specific structure or not.
- The interface contains only the **declaration** of the **methods** and **fields**, but not the **implementation**.
- We cannot use it to build anything. It is inherited by a class, and the class which implements interface defines all members of the interface.
- When the Typescript compiler compiles it into JavaScript, then the interface will be disappeared from the JavaScript file. Thus, its purpose is to help in the development stage only.

2.13.1 Interface Declaration

- We can declare an interface as below.

```
interface interface_name {
      // variables' declaration
      // methods' declaration
}
```

Module 2

- An **interface** is a keyword which is used to declare a TypeScript Interface.
- An **interface_name** is the name of the interface.
- An interface body contains variables and methods declarations.

Example

In the below example, we have created an interface OS with properties name and language of string type. Next, we have defined a function, with one argument, which is the type of interface OS.

```
interface OS {
   name: String;
   language: String;
}
let OperatingSystem = (type: OS): void => {
 console.log('Android ' + type.name + ' has ' + type.language
+ ' language.');
};
let Eclair = {name: 'E', language: 'Java'}
OperatingSystem(Eclair);
```

Output

```
Android E has Java language.
```

2.13.2 Use of Interface

We can use the interface for the following things:

- Validating the specific structure of properties
- Objects passed as parameters
- Objects returned from functions.

2.13.3 Interface Inheritance

- We can inherit the interface from the other interfaces. In other words, Typescript allows an interface to be inherited from zero or more **base types**.
- The base type can be a **class** or **interface**.
- We can use the "**extends**" keyword to implement inheritance among interfaces.

Syntax

```
child_interface extends parent interface{
}
```

Example

```
interface Person {
name:string
age:number
}
interface Employee extends Person {
gender:string
empCode:number
}
letempObject = <Employee>{};
empObject.name = "Nilesh"
empObject.age = 38
empObject.gender = "Male"
empObject.empCode = 10570
console.log("Name: "+empObject.name);
console.log("Employee Code: "+empObject.empCode);
console.log("Employee Gender: "+empObject.gender);
console.log("Employee Age: "+empObject.age);
```

Output

```
Name: Nilesh
Employee Code: 10570
Employee Gender: Male
Employee Age: 38
```

- Let us take the example, which helps us to understand the **multiple interface** inheritance.

Example

```
interface Person {
name:string
}
interfacePersonDetail {
age:number
gender:string
}
interface Employee extends Person, PersonDetail {
empCode:number
}
letemppObject = <Employee>{};
emppObject.name = "Nilesh"
emppObject.age = 38
emppObject.gender = "Male"
emppObject.empCode = 10570
console.log("Name: "+emppObject.name);
console.log("Employee Code: "+emppObject.empCode);
console.log("Employee Gender: "+emppObject.gender);
console.log("Employee Age: "+emppObject.age);
```

Output

```
Name: Nilesh
Employee Code: 10570
Employee Gender: Male
Employee Age: 38
```

2.13.4 Array Type Interface

We can also use interfaces to describe the array type. The following example helps us to understand the Array Type Interface.

Example

In the below example, we have declared **nameArray** that returns **string** and **ageArray** that returns **number**. The type of index in the array is always **number** so that we can retrieve array elements with the use of its **index position** in the array.

```
// Array which return string
interface nameArray {
   [index:number]:string
}
// use of the interface
let myNames: nameArray;
myNames = ['Nilesh', 'Shraddha', 'Sachin'];
console.log("My Name is: " +myNames[1]);

// Array which return number
interface ageArray {
   [index:number]:number
}
var myAges: ageArray;
myAges=[38, 32,48];
console.log("My age is: " +myAges[1]);
```

Output:

```
My Name is: Shraddha
My age is: 32
```

2.13.5 Interface in a Class

TypeScript also allows us to use the interface in a class. A class implements the interface by using the **implements** keyword. We can understand it with the below example.

Example

In the above example, we have declared **Person** interface with **firstName**, **lastName** as property and **FullName** and **GetAge** as **method/function**. The **Employee** class implements this interface by using the **implements** keyword. After implementing an interface, we must declare the properties and methods in the class. If we do not implement those properties and methods, it throws a **compile-time** error. We have also declared a constructor in the class. So when we instantiate the class, we need to pass the necessary parameters otherwise it throws an error at compile time.

Module 2

```
// defining interface for class
interface Person {
firstName: string;
lastName: string;
   age: number;
FullName();
GetAge();
}
// implementing the interface
class Employee implements Person {
firstName: string;
lastName: string;
age:number;
FullName() {
      return this.firstName + ' ' + this.lastName;
   }
GetAge() {
      return this.age;
   }
constructor(firstN: string, lastN: string, getAge: number) {
this.firstName = firstN;
this.lastName = lastN;
this.age = getAge;
   }
}
// using the class that implements interface
let myEmployee = new Employee('Sachin', 'Shah', 48);
let fullName = myEmployee.FullName();
let Age = myEmployee.GetAge();
console.log("Name of Person: " +fullName + '\nAge: ' +
Age);
```

Output

```
Name of Person: Sachin Shah
Age: 48
```

2.13.6 Interface Vs Inheritance

SN	Interface	Inheritance
1.	An Interface is a structure which acts as a contract in our application. It defines the required functions, and the class is responsible for implementing it to meet that contract.	Inheritance is object-oriented programming that allows similar objects to inherit the functionality and data from each other.
2.	In an interface, we can only declare properties and methods.	In inheritance, we can use a superclass to declare and defines variables and methods.
3.	An interface type objects cannot declare any new methods or variables.	In this, we can declare and define its own variables and methods of a subclass that inherits a superclass.
4.	Interface enforces the variables and methods which have to be present in an object.	A subclass extends the capability of a superclass to suit the "custom" needs.
5.	Interface are classes that contain body-less structure (abstract or virtual functions). So, we have to derive the interface and then implement all of the functions in the subclass.	Inheritance is the process where one subclass acquires the properties of its superclass.

2.14 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**. In other words, the variables, functions, classes, and interfaces declared in a module cannot be accessible outside the module directly.
- We can create a module by using the **export** keyword and can use in other modules by using the **import** keyword.
- Modules import another module by using a **module loader**.
- At runtime, the module loader is responsible for locating and executing all dependencies of a module before executing it.
- The most common modules loaders which are used in JavaScript are the **Common JS** module loader for **Node.js** and **require.js** for Web applications.
- We can divide the module into **two** categories:

  1. Internal Module 2. External Module

2.14.1 Internal Module

- Internal modules were in the **earlier version** of Typescript.
- It was used for **logical grouping** of the classes, interfaces, functions, variables into a single unit and can be exported in another module.
- The modules are named as a **namespace** in the latest version of TypeScript. So today, internal modules are **obsolete**. But they are still supported by using namespace over internal modules.
- Internal Module Syntax in Earlier Version:

```
module Sum {
  export function add(a, b) {
console.log("Sum: " +(a+b));
  }
}
```

- Internal Module Syntax from ECMAScript 2015:

```
namespace Sum {
  export function add(a, b) {
console.log("Sum: " +(a+b));
  }
}
```

2.14.2 External Module

- External modules are also known as a **module**.
- When the applications consisting of hundreds of files, then it is almost impossible to handle these files without a modular approach.

- External Module is 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.
- ECMAScript 2015(ES6) module system treats every file as a module.

2.14.3 Module Declaration

We can declare a module by using the **export** keyword. The syntax for the module declaration is given below.

```
//FileName :EmployeeInterface.ts
export interface Employee {
  //code declarations
}
```

We can use the declare module in other files by using an **import** keyword, which looks like below. The **file/module** name is specified without an **extension**.

```
import { class/interface name } from 'module_name';
```

Example : Let us understand the module with the following example.

1. Module Creation

```
//FileName: addition.ts
export class Addition{
   constructor(private x?: number, private y?: number){
   }
   Sum(){
      console.log("SUM: " +(this.x + this.y));
   }
}
```

2. Accessing the module in another file by using the **import** keyword.

```
//FileName: app.ts
import {Addition} from './addition';
let addObject = new Addition(10, 20);
addObject.Sum();
```

3. Compiling and Execution of Modules

   Open the **terminal** and go to the location where you stored your **project**. Now, type the following command in the terminal window.

```
$ tsc --module commonjs app.ts
$ node ./app.js
```

4. **Output**:

   SUM: 30

2.14.4 Importing Multiple Modules in a Single File

We can define multiple modules in a single file. The syntax for multiple module declaration is given below.

```
import { export_name1, export_name2 } from 'module_name';
```

Example : Let us understands the module with the following example.

1. Module Creation

```
//FileName: Modules.ts
export class Addition{
   constructor(private x?: number, private y?: number){
   }
   Sum(){
      console.log("SUM: " +(this.x + this.y));
   }
}
export class Subtraction{
   constructor(private a?: number, private b?: number){
   }
   Subtract(){
      console.log("SUBTRACTION: " +(this.a - this.b));
   }
}
```

2. Accessing the module in another file by using the **import** keyword.

```
//FileName: app.ts
import {Addition, Subtraction} from './Modules';
let addObject = new Addition(10, 20);
let subObject = new Subtraction(20,10);
addObject.Sum();
subObject.Subtract();
```

3. Compiling and Execution of Modules

   Open the **terminal** and go to the location where you stored your **project**. Now, type the following command in the terminal window.

```
$ tsc --module commonjs app.ts
$ node ./app.js
```

4. Output:

   SUM: 30

   SUBTRACTION: 10

Module 2

Descriptive Questions

Q. 1 State the features of TypeScript. Also state the advantages of TypeScript.

Q. 2 Explain the internal architecture of TypeScript in detail.

Q. 3 Discuss with example different types in TypeScript.

Q. 4 How to declare variables in TypeScript? Differentiate between let and var.

Q. 5 Explain with suitable example

a. Arithmetic operators

b. Comparison (Relational) operators

c. Logical operators

d. Bitwise operators

e. Assignment operators

f. Ternary/conditional operator

g. Concatenation operator

Q. 6 Discuss definite and indefinite loops with suitable example in TypeScript.

Q. 7 Explain TypeScript Functions in detail.

Q. 8 Explain different types of access modifiers in TypeScript.

Q. 9 Explain TypeScript Interfaces with suitable example.

Q. 10 Explain TypeScript Modules with suitable example.

Chapter Ends…

❑❑❑

MODULE 3

CHAPTER 3

Introduction to AngularJS

University Prescribed Syllabus w.e.f Academic Year 2021-2022

Overview of AngularJS, Need of AngularJS in real web sites, AngularJS modules, AngularJS built-in directives, AngularJS custom directives, AngularJS expressions, Angular JS Data Binding, AngularJS filters, AngularJS controllers, AngularJS scope, AngularJS dependency injection, Angular JS Services, Form Validation, Routing using ng-Route, ng-Repeat, ng-style, ng-view, Built-in Helper Functions, Using Angular JS with Typescript

Self-learning Topics: MVC model, DOM model, Javascript functions and Error Handling

3.1 OVERVIEW OF ANGULARJS

- AngularJS is a client side JavaScript MVC framework to develop a dynamic web application.
- AngularJS was originally started as a project in Google but now, it is open source framework.
- AngularJS is entirely based on HTML and JavaScript, so there is no need to learn another syntax or language.
- AngularJS is also called just "Angular".
- AngularJS changes static HTML to dynamic HTML. It extends the ability of HTML by adding built-in attributes and components and also provides an ability to create custom attributes using simple JavaScript.
- AngularJS can be used to create Single Page Applications.
- **AngularJS website** - https://angularjs.org

3.1.1 Features of AngularJS

(a) General Features

The general features of AngularJS are as follows :

- AngularJS is an 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 Angular JS 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.
- Overall, AngularJS is a framework to build large scale, high-performance, and easy-to-maintain web applications.

(b) Core Features

The core features of AngularJS are as follows:

(1) Data-binding : It is the automatic synchronization of data between model and view components.

(2) Scope : These are objects that refer to the model. They act as a glue between controller and view.

(3) Controller : These are JavaScript functions bound to a particular scope.

(4) Services : AngularJS comes with several built-in services such as $http to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.

(5) Filters : These select a subset of items from an array and returns a new array.

(6) Directives : Directives are markers on DOM elements such as elements, attributes, css, and more. These can be used to create custom HTML tags that serve as new, custom widgets. Angular JS has built-in directives such as ng-Bind, ng-Model, etc.

(7) 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*.

(8) Routing : It is concept of switching views.

(9) Model View Whatever : MVW is a design pattern for dividing an application into different parts called Model, View, and Controller, each with distinct responsibilities. Angular JS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-View-Model). The Angular JS team refers it humorously as Model View Whatever.

(10) 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.

(11) Dependency Injection : AngularJS has a built-in dependency injection subsystem that helps the developer to create, understand, and test the applications easily.

3.1.2 Advantages of AngularJS

The advantages of AngularJS are :

(1) It provides the capability to create Single Page Application in a very clean and maintainable way.

(2) It provides data binding capability to HTML. Thus, it gives user a rich and responsive experience.

(3) AngularJS code is unit testable.

Module 3

(4) AngularJS uses dependency injection and make use of separation of concerns.

(5) AngularJS provides reusable components.

(6) With AngularJS, the developers can achieve more functionality with short code.

(7) In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.

(8) AngularJS applications can run on all major browsers and smart phones, including Android and iOS based phones/tablets.

3.1.3 Disadvantages of AngularJS

Though AngularJS comes with a lot of merits, here are some points of concern :

(1) Not Secure : Being JavaScript only framework, application written in Angular JS are not safe. Server side authentication and authorization is must to keep an application secure.

(2) Not degradable : If the user of your application disables JavaScript, then nothing would be visible, except the basic page.

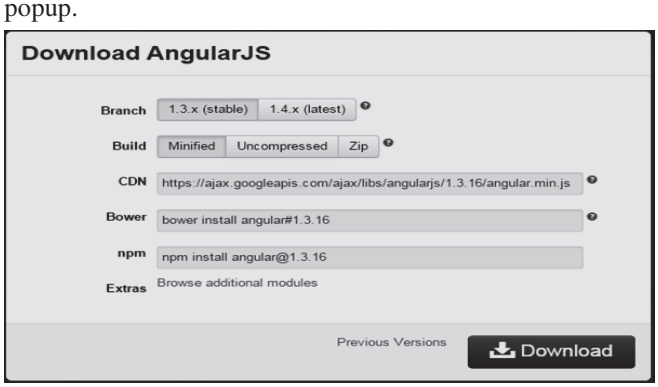
3.1.4 AngularJS Environment Setup

We need the following tools to setup a development environment for AngularJS :

(1)	AngularJS Library	(2)	Editor/IDE
(3)	Web server	(4)	Browser

(1) AngularJS Library

To download AngularJS library, go to angularjs.org -> click download button, which will open the following popup.



Download AngularJS Library

- Select the required version from the popup and click on download button in the popup.
- CDN: You can include AngularJS library from CDN url -

  https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js

(2) Editor

AngularJS is eventually HTML and JavaScript code. So, you can install any good editor/IDE as per your choice.

The following editors are recommended :

(1) Sublime Text (2) Aptana Studio 3

(3) Ultra Edit (4) Eclipse

(5) Visual Studio

Online Editor

You can also use the following online editors for learning purpose.

- plnkr.co
- jsbin.com

(3) Web server

Use any web server such as IIS, apache etc., locally for development purpose.

(4) Browser

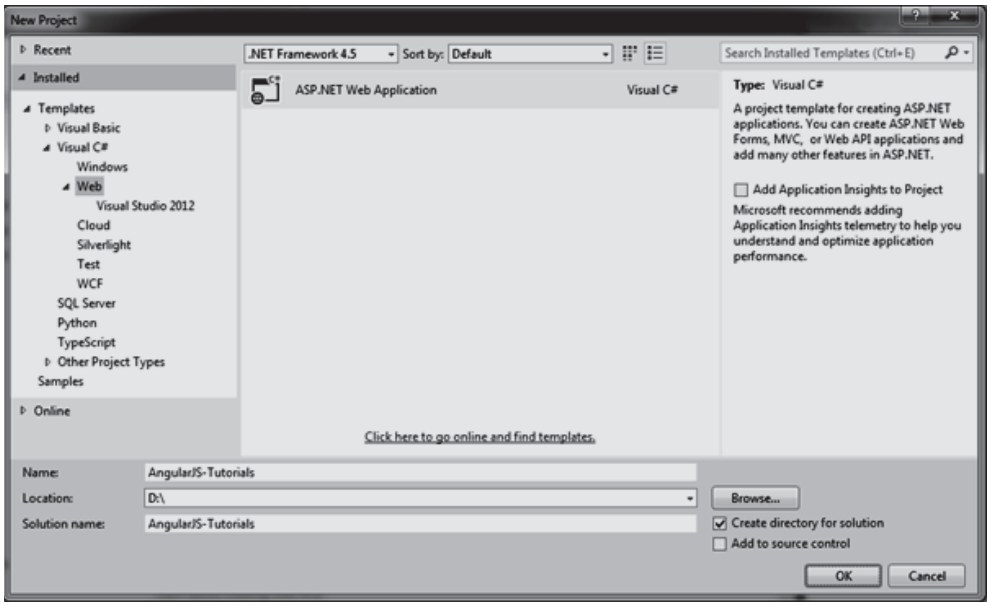
You can install any browser of your choice as AngularJS supports cross-browser compatibility. However, it is recommended to use Google Chrome while developing an application.

Angular Seed

- Use Angular seed project to quickly get started on AngularJS application. The Angular-seed is an application skeleton for a typical AngularJS web application. You can use it to quickly bootstrap your angular webapp projects and development environment for your project.
- Download angular-seed from **GitHub**
- Let's setup Angular project in Visual Studio 2013 for web.

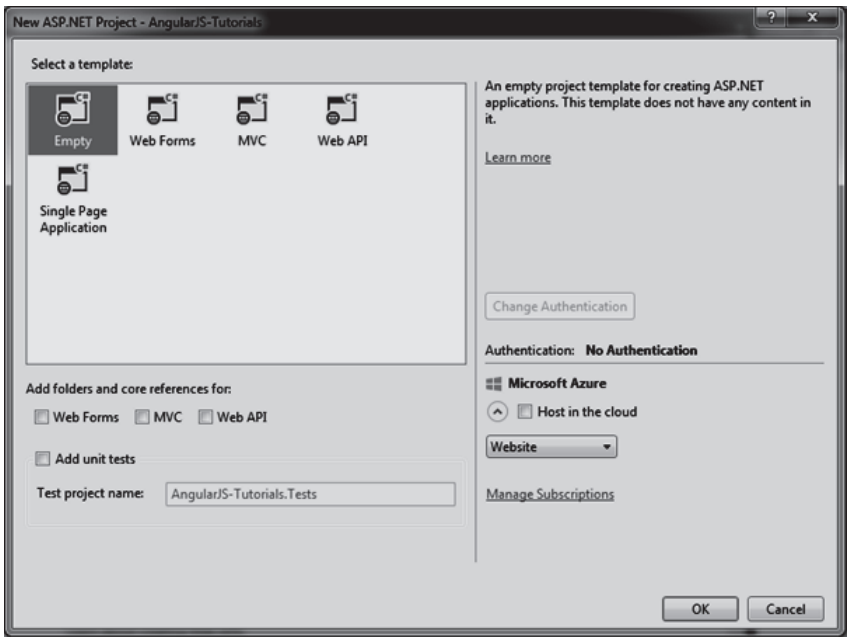
Setup AngularJS Project in Visual Studio

- You can create AngularJS application in any version of Visual Studio. Here, we will use Visual Studio 2013 for web.
- First, create new project by clicking on New Project link on start page. This will open New Project dialog box, as shown below.



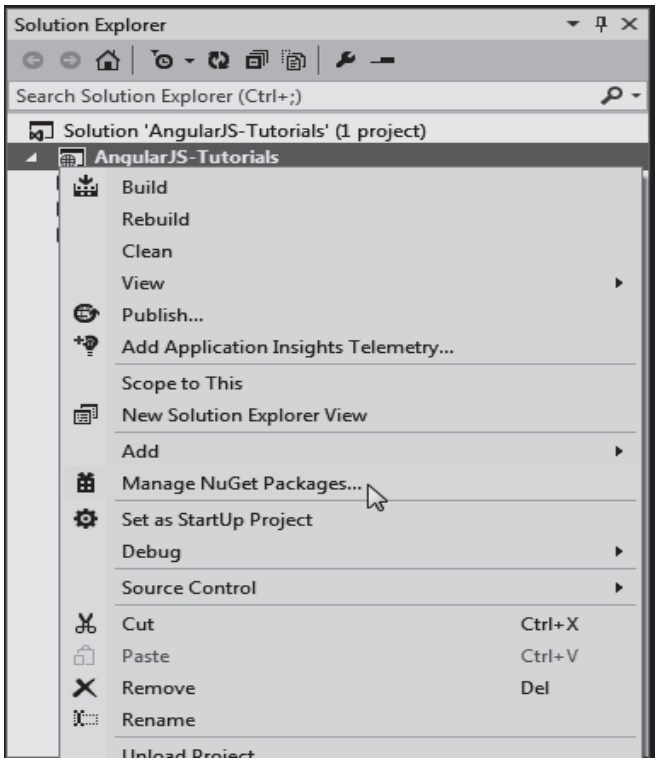
AngularJS in Visual Studio

- Select Web in the left pane and ASP.NET Web Application in the middle pane and then click OK.
- In the New ASP.NET Project dialog box, select Empty template and then click OK.

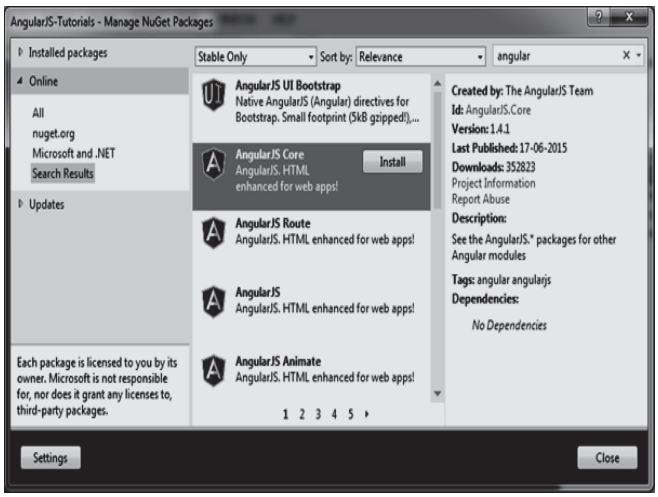


- This will create an empty website project in Visual Studio.
- Now, install AngularJS library from NuGet package manager. Right click on the project in Solution Explorer and select Manage NuGet Packages.

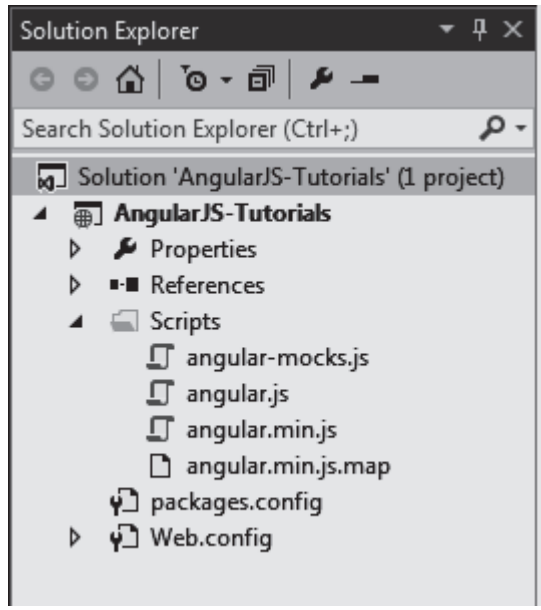
Module 3



- Search for "angular" in the Manage NuGet Packages dialog box and install AngularJS Core.



- This will add AngularJS files into Scripts folder such as angular.js, angular.min.js, and angular-mocks.js, as follows:



- Now, you can start writing AngularJS web application.

3.2 NEED OF ANGULARJS IN REAL WEBSITES

The most valuable reasons of AngularJS usefulness lie in their concepts and benefits that they provide. However, documentation and wide community also give a point to Angular usage. The major reasons for using AngularJs in real websites include :

(1) Major community : It opens a large scale of options both to developers and clients. Developers may easily find various info and solutions. Clients have wider choices of development teams.

(2) Readable code : While code is more understandable to readers it is a way easier to maintain. Moreover, clients don't bind to developers in this case and have minimum problems if change programmers that maintain it.

(3) Google as creation: Someone may say it isn't a direct benefit of AngularJS. However, it is heavily supported.

(4) Customizable : Developers don't require to use all libraries and even may change them. As a result, the size of the framework may be heavily reduced.

(5) Flexible : Filters allow distilling data before it gets into view. So, it helps to deal with pagination, formatting and reversing a text.

(6) Fast for small : While any large and complex web app takes much time to develop, AngularJS allows developing small apps really fast. In addition, it also may be used to develop large projects.

(7) Charge dependency : Dependency injection will help you manage back-end. In the end, web app becomes faster and more stable.

(8) Pre-made solutions: AngularJS contains a bunch of different solutions ready to use in the app. UI-router, routing modules and much more.

(9) Easy testing : AngularJS allows to write and test module code separately from the rest of an app. As a result, only necessary parts of apps and chosen services will be tested.

3.3 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.

Example 1 : In the below example, the angular.module() method creates an application module, where the first parameter is a module name which is same as specified by ng-app directive. The second parameter is an array of other dependent modules []. In the above example we are passing an empty array because there is no dependency.

> **Note** : The angular.module() method returns specified module object if no dependency is specified. Therefore, specify an empty array even if the current module is not dependent on other module.

- Now, you can add other modules in the myApp module.
- The below example also demonstrates creating controller module in myApp module. We have created a controller named "myController" using myApp.controller() method. Here, myApp is an object of a module, and controller() method creates a controller inside "myApp" module. So, "myController" will not become a global function.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myApp">
<h1>AngularJS Module Demo: </h1>
<div ng-controller="myController">
    {{message}}
</div>

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

    myApp.controller("myController", function ($scope) {
      $scope.message = "Hello Angular World!";
    });
</script>
</body>
</html>
```

Output :

```
AngularJS Module Demo:
Hello Angular World!
```

In the above example, we created application module and controller in the same HTML file. However, we can create separate JavaScript files for each module as shown below.

```
<!DOCTYPE html>
<html >
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myApp">
<div ng-controller="myController">
    {{message}}
</div>
<script src="app.js" ></script>
<script src="myController.js" ></script>
</body>
</html>
```

Module 3

File Name: app.js

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

File Name: myController.js

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

Example 2 : Here, we use application module using ng-app directive, and controller using ngcontroller directive. We import the mainApp.js and studentController.js in the main HTML page. In **mainApp.js** we declare an application **mainApp** module using angular.module function and pass an empty array to it. This array generally contains dependent modules. In studentController.js, we declare a controller **studentController** module using mainApp.controller function.

File Name: testAngularJS.html

```
<html>
<head>
<title>Angular JS Modules</title>
<script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular
.min.js"></script>
<script src =
"/angularjs/src/module/mainApp.js"></script>
<script src =
"/angularjs/src/module/studentController.js"></script>

<style>
      table, th , td {
        border: 1px solid grey;
        border-collapse: collapse;
        padding: 5px;
      }
      table tr:nth-child(odd) {
        background-color: #f2f2f2;
      }
      table tr:nth-child(even) {
        background-color: #ffffff;
      }
</style>
</head>

<body>
<h2>AngularJS Sample Application</h2>
<div ng-app = "mainApp" ng-controller =
"studentController">

<table border = "0">
<tr>
<td>Enter first name:</td>
<td><input type = "text" ng-model =
"student.firstName"></td>
</tr>
<tr>
<td>Enter last name: </td>
<td><input type = "text" ng-model =
"student.lastName"></td>
</tr>
<tr>
<td>Name: </td>
<td>{{student.fullName()}}</td>
</tr>
<tr>
<td>Subject:</td>

<td>
<table>
<tr>
<th>Name</th>
<th>Marks</th>
</tr>
<tr ng-repeat = "subject in student.subjects">
<td>{{ subject.name }}</td>
<td>{{ subject.marks }}</td>
</tr>
</table>
</td>
</tr>
</table>
</div>

</body>
</html>
```

File Name: mainApp.js

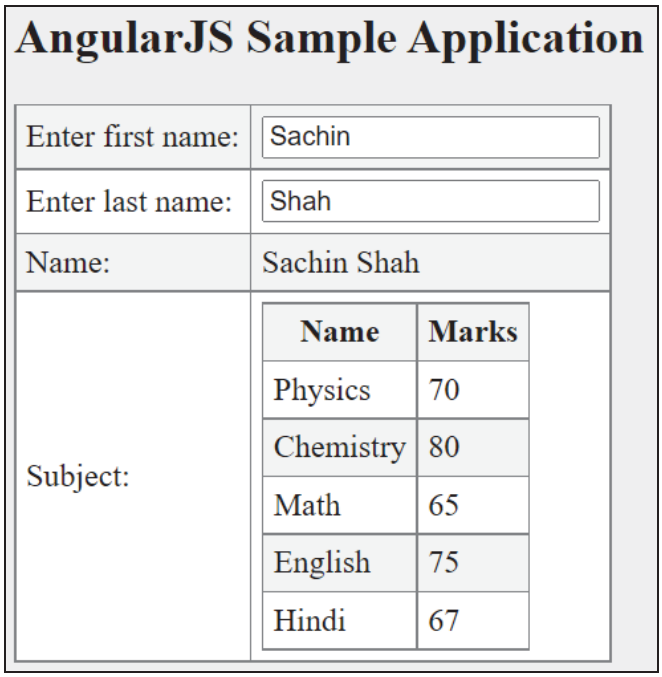
```
var mainApp = angular.module("mainApp", []);
```

File Name: studentController.js

```
mainApp.controller("studentController", function($scope) {
  $scope.student = {
    firstName: "Sachin",
```

```
    lastName: "Shah",
    fees:500,

subjects:[
      {name:'Physics',marks:70},
      {name:'Chemistry',marks:80},
      {name:'Math',marks:65},
      {name:'English',marks:75},
      {name:'Hindi',marks:67}
    ],
    fullName: function() {
      var studentObject;
      studentObject = $scope.student;
      return studentObject.firstName + " " +
studentObject.lastName;
    }
  };
});
```

Output :



3.4 ANGULAR JS BUILT-IN DIRECTIVES

- 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.
- Most of the directives in AngularJS are starting with **ng-** where ng stands for Angular. AngularJS includes various built-in directives.
- In addition to this, you can create custom directives for your application.
- The Table 3.4.1, lists the important built-in AngularJS directives.

Table 3.4.1

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.

3.4.1 ng-app

- The ng-app directive is a starting point of AngularJS Application. It initializes the AngularJS framework automatically.
- AngularJS framework will first check for ng-app directive in a HTML document after the entire document is loaded and if ng-app is found, it bootstraps itself and compiles the HTML template.

Module 3

- Typically, ng-app directives should be placed at the root of an HTML document e.g. <html> or <body> tag, so that it can control the entire DOM hierarchy. However, you can place it in any DOM element.
- The AngularJS framework will only process the DOM elements and its child elements where the ng-app directive is applied.
- **Example** : In the below example, ng-app directive is placed in the div element whose id is "myDiv". Therefore, AngularJS will only compile myDiv and its child elements. It will not compile the parent or sibling elements of myDiv.

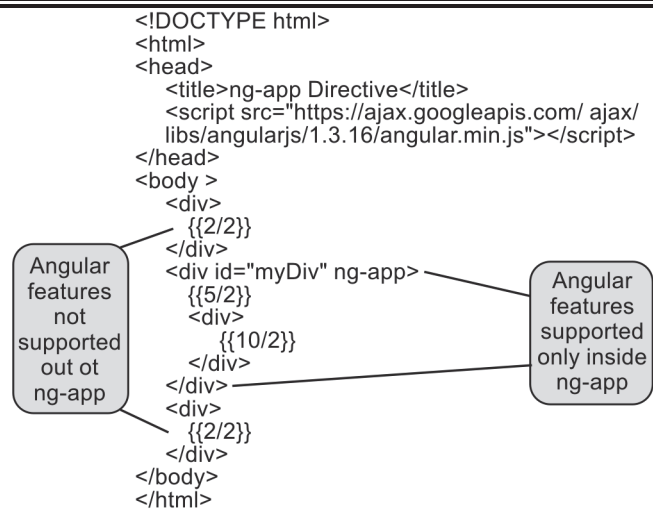
```
<!DOCTYPE html>
<html>
<head>
<title>ng-app Directive</title>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body >
<div>
    {{2/2}}
</div>
<div id="myDiv" ng-app>
    {{5/2}}
<div>
        {{10/2}}
</div>
</div>
<div>
    {{2/2}}
</div>
</body>
</html>
```

Output :

```
{{2/2}}
2.5
5
{{2/2}}
```

The Fig. 3.4.1 illustrates the above example.

Note that multiple ng-app directives are **NOT** allowed in a single HTML document.



(1C1)**Fig. 3.4.1 : Bootstrapping of ng-app**

ng-app with Module name

The ng-app directive can also specify an application module name. This application module separates different parts of your application such as controllers, services, filters etc.

Example : In the above example, we have specified a module name using ng-app = 'myAngularApp' in the <body> tag, and then we have created 'myAngularApp' module using angular.module() function inside <script>.

```
<!DOCTYPE html>
<html>
<head>
<title>ng-app Directive</title>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myAngularApp">
<div>
    {{2/2}}
</div>
<div>
    {{5/2}}
<div>
        {{10/2}}
</div>
</div>
<script>
    var app = angular.module('myAngularApp', []);
```

```
</script>
</body>
</html>
```

Output :

```
1
2.5
5
```

3.4.2 ng-init

- The ng-init directive can be used to initialize variables in AngularJS application.
- The following example demonstrates ng-init directive that initializes variable of string, number, array, and object.

```
<!DOCTYPE html>
<html >
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body >
<div ng-app ng-init="greet='Hello World!'; amount= 100; myArr = [100, 200]; person = { firstName:'Sachin', lastName :'Shah'}">
    {{greet}}          <br />
    {{amount}}  <br />
    {{myArr[1]}}    <br />
    {{person.firstName}}  <br />
    {{person.lastname}}
</div>
</body>
</html>
```

Output :

```
Hello World!
100
200
Sachin
```

- In the above example, we initialized variables of string, number, array and object. These variables can be used anywhere in the DOM element hierarchy where it is declared e.g. variables in the above example cannot be used out of element.

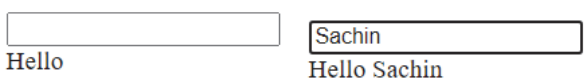
3.4.3 ng-model

- The ng-model directive is used for two-way data binding in AngularJS.
- It binds <input>, <select> or <textarea> elements to a specified property on the $scope object. So, the value of the element will be the value of a property and vice-versa.
- Variables initialized in ng-init are different from the properties defined using ng-model directive.
- The variables initialized in ng-init are not attached to $scope object whereas ng-model properties are attached to $scope object.

Example

```
<!DOCTYPE html>
<html >
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app>
<input type="text" ng-model="name" />
<div>
    Hello {{name}}
</div>
</body>
</html>
```

Output



3.4.4 ng-bind

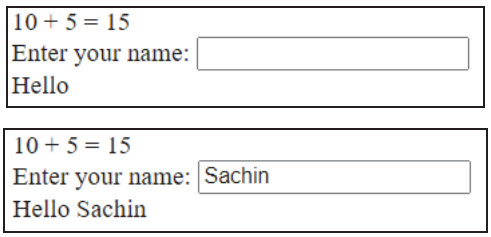
- The ng-bind directive binds the model property declared via $scope or ng-model directive or the result of an expression to the HTML element.
- It also updates an element if the value of an expression changes.

Module 3

Example

```
<!DOCTYPE html>
<html >
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<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>
</body>
</html>
```

Output :



- In the above example, ng-bind directive binds a result of an expression "10 + 5" to the . The same way, it binds a value of a model property "name" to the . The value of "name" property will be the value entered in a textbox.

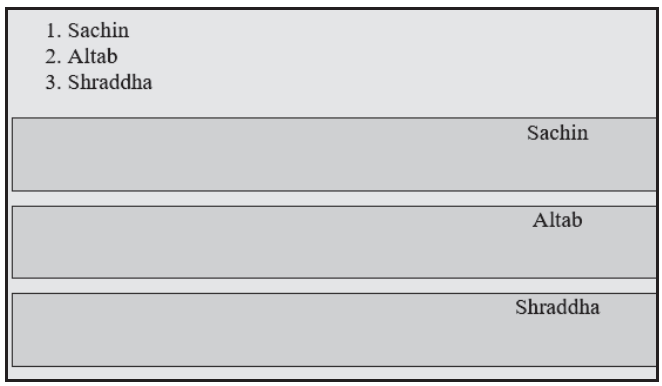
3.4.5 ng-repeat

- The ng-repeat directive repeats HTML once per each item in the specified array collection.

Example :

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
<style>
    div {
        border: 1px solid green;
        width: 100%;
        height: 50px;
        display: block;
        margin-bottom: 10px;
        text-align:center;
        background-color:yellow;
    }
</style>
</head>
<body ng-app="" ng-
init="students=['Sachin','Altab','Shraddha']">
<ol>
<li ng-repeat="name in students">
        {{name}}
</li>
</ol>
<div ng-repeat="name in students">
    {{name}}
</div>
</body>
</html>
```

Output :



- In the above example, ng-repeat is used with students array. It creates <li> element for each item in the students array. Using the same way, it repeats the element.

3.4.6 ng-if

- The ng-if directive creates or removes an HTML element based on the Boolean value returned from the specified expression.
- If an expression returns true, then it recreates an element otherwise removes an element from the HTML document.

3.4.7 ng-readonly

- The ng-readonly directive makes an HTML element read-only, based on the Boolean value returned from the specified expression.
- If an expression returns true, then the element will become read-only, otherwise not.

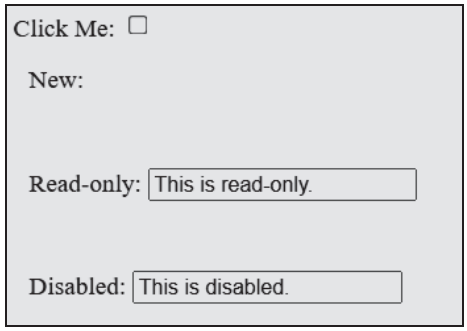
3.4.8 ng-disabled

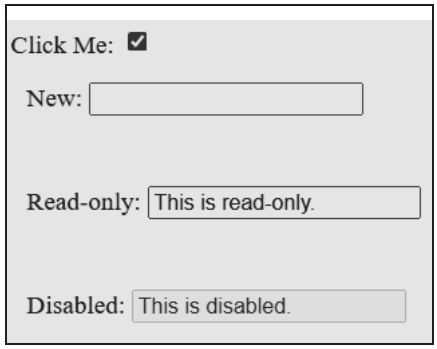
- The ng-disabled directive disables an HTML element, based on the Boolean value returned from the specified expression.
- If an expression returns true, the element will be disabled, otherwise not.

The following example demonstrates **ng-if, ng-readonly, and ng-disabled** directives.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
<style>
    div {
        width: 100%;
        height: 50px;
        display: block;
        margin: 15px 0 0 10px;
    }
</style>
</head>
<body ng-app ng-init="checked=true" >
  Click Me: <input type="checkbox" ng-model="checked"
/><br />
<div>
    New: <input ng-if="checked" type="text" />
</div>
<div>
    Read-only: <input ng-readonly="checked" type="text"
value="This is read-only." />
</div>
<div>
    Disabled: <input ng-disabled="checked" type="text"
value="This is disabled." />
</div>
</body>
</html>
```

Output





3.4.9 ng-click

- The "ng-click directive" in AngularJS is used to apply custom behavior when an element in HTML is clicked.
- This directive is normally used for buttons because that is the most commonplace for adding events that respond to clicks performed by the user. It is also used to popup alerts when a button is clicked.

```
<!DOCTYPE html>
<html>
<head>
<title>Event Registration</title>
</head>
<body ng-app="">

<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
<h1> AngularJS ng-click Directive Example</h1>

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

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

</body>
</html>
```

Module 3

Output

From the above output,

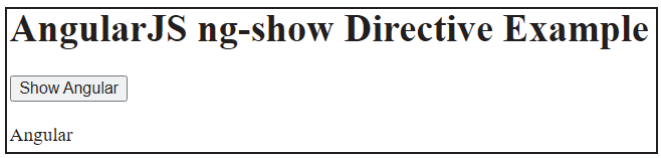
- We can see that the button "Increment" is displayed and the value of the count variable is initially zero.
- When you click on the Increment button, the value of the count is incremented accordingly.

3.4.10 ng-show

The **ng-Show directive** in AngularJS is used to show or hide a given specific HTML element based on the expression provided to the ng-show attribute. In the background, the HTML element is shown or hidden by removing or adding the .ng-hide CSS class onto the element. It is a predefined CSS class which sets the display to none.

```
<!DOCTYPE html>
<html>
<head>
<title>Event Registration</title>
</head>
<body ng-app="">
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
<h1> AngularJS ng-show Directive Example</h1>
<div ng-app="DemoApp" ng-controller="DemoController">
<input type="button" value="Show Angular" ng-click="ShowHide()"/>
<br><br><div ng-show = "IsVisible">Angular</div>
</div>
<script type="text/javascript">
   var app = angular.module('DemoApp',[]);
app.controller('DemoController',function($scope){
      $scope.IsVisible = false;
      $scope.ShowHide = function(){
$scope.IsVisible = true;
      }
      });
</script>
</body>
</html>
```

Output



- We are attaching the ng-click event directive to the button element. Over here we are referencing a function called "ShowHide" which is defined in our controller - DemoController.
- We are attaching the ng-show attribute to a div tag which contains the text Angular. This is the tag which we are going to show/hide based on the ng-show attribute.
- In the controller, we are attaching the "IsVisible" member variable to the scope object. This attribute will be passed to the ng-show angular attribute to control the visibility of the div control.
- We are initially setting this to false so that when the page is first displayed the div tag will be hidden.**Note**: When the attributes ng-show is set to true, the subsequent control which in our case is the div tag will be shown to the user. When the ng-show attribute is set to false the control will be hidden from the user.
- We are adding code to the ShowHide function which will set the IsVisible member variable to true so that the AngularJS show hide div on click tag can be shown to the user.

From the output,

- We can initially see that the div tag which has the text "AngularJS" is not shown and this is because the isVisible scope object is initially set to false which is then subsequently passed to the ng-show directive of the div tag.
- When we click on the "Show AngularJS" button, it changes the isVisible member variable to become true and hence the text "Angular" becomes visible to the user.

3.4.11 ng-hide

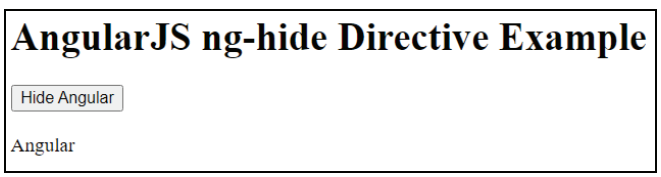
- The **ng-hide directive** in AngularJS is a function using which an element will be hidden if the expression is TRUE. If the Expression is FALSE, the element will be shown. In the background, the element is shown or

hidden by removing or adding the .ng-hide CSS class onto the element.

- On the other hand, with ng-hide, the element is hidden if the expression is true and it will be shown if it is false.

```
<!DOCTYPE html>
<html>
<head>
<title>Event Registration</title>
</head>
<body ng-app="">
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
<h1> AngularJS ng-hide Directive Example</h1>
<div ng-app="DemoApp" ng-controller="DemoController">
<input type="button" value="Hide Angular" ng-
click="ShowHide()"/>
<br><br><div ng-hide="IsVisible">Angular</div>
</div>
<script type="text/javascript">
   var app = angular.module('DemoApp',[]);
app.controller('DemoController',function($scope){
      $scope.IsVisible = false;
      $scope.ShowHide = function(){
      $scope.IsVisible = $scope.IsVisible = true;
      }
      });
</script>
</body>
</html>
```

Output



- We are attaching the ng-click event directive to the button element. Over here we are referencing a function called ShowHide which is defined in our controller - DemoController.
- We are attaching the ng-hide attribute to a div tag which contains the text Angular. This is the tag, which we will use to show hide in Angular based on the ng-show attribute.
- In the controller, we are attaching the IsVisible member variable to the scope object. This attribute will be passed to the ng-show angular attribute to control the visibility of the div control. We are initially setting this to false so that when the page is first displayed the div tag will be hidden.
- We are adding code to the ShowHide function which will set the IsVisible member variable to true so that the div tag can be shown to the user.

From the output,

- We can initially see that the div tag which has the text "AngularJs" is initially shown because the property value of false is sent to the ng-hide directive.
- When we click on the "Hide Angular" button the property value of true will sent to the ng-hide directive. Hence the output shows that in which the word "angular" is hidden.

Directive Syntax

- AngularJS directives can be applied to DOM elements in many ways. It is not mandatory to use ng- syntax only.
- Directive can start with x- or data-, for example ng-model directive can be written as data-ng-model or x-ng-model.
- Also, the - in the directive can be replaced with : or _ or both. For example, ng-model can be written as ng_model or ng:model. It can also be a mix with data- or x-.

⏭ 3.5 ANGULARJS CUSTOM DIRECTIVES

- Custom directives are used in AngularJS 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 application during bootstrap finds the matching elements and do one time activity using its compile() method of the custom directive then process the element using link() method of the custom directive based on the scope of the directive.

- AngularJS provides support to create custom directives for following type of elements.
  - **(1) Element directives :** Directive activates when a matching element is encountered.
  - **(2) Attribute :** Directive activates when a matching attribute is encountered.
  - **(3) CSS :** Directive activates when a matching css style is encountered.
  - **(4) Comment :** Directive 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. Here we are using name attribute's value as scope's child.

Example

```
<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>
```

Output

AngularJS Sample Application

Student: **Nilesh Patil** , Roll No: **1**

Student: **Sachin Shah** , Roll No: **2**

Parameter	Details
Scope	Property to set the scope of the directive. It can be set as false, true or as an isolate scope: { @, =, <, & }.
scope: falsy	Directive uses parent scope. No scope created for directive.
scope: true	Directive inherits parent scope prototypically as a new child scope. If there are multiple directives on the same element requesting a new scope, then they will share one new scope.
scope: { @ }	One way binding of a directive scope property to a DOM attribute value. As the attribute value bound in the parent, it will change in the directive scope.
scope: { = }	Bi-directional attribute binding that changes the attribute in the parent if the directive attribute changes and vice-versa.
scope: { < }	One way binding of a directive scope property and a DOM attribute expression. The expression is evaluated in the parent. This watches the identity of the parent value so changes to an object property in the parent won't be reflected in the directive. Changes to an object property in a directive will be reflected in the parent, since both reference the same object
scope: { & }	Allows the directive to pass data to an expression to be evaluated in the parent.
compile: function	This function is used to perform DOM transformation on the directive template before the link function runs. It accepts tElement (the directive template) and tAttr (list of attributes declared on the directive). It does not have access to the scope. It may return a function that will be registered as a post-link function or it may return an object with pre and post properties with will be registered as the pre-link and post-link functions.
link: function/object	The link property can be configured as a function or object. It can receive the following arguments: scope (directive scope), iElement (DOM element where directive is applied), iAttrs (collection of DOM element attributes), controller (array of controllers required by directive), transcludeFn. It is mainly used to for setting up DOM listeners, watching model properties for changes, and updating the DOM. It executes after the template is cloned. It is configured independently if there is no compile function.
pre-link function	Link function that executes before any child link functions. By default, child directive link functions execute before parent directive link functions and the pre-link function enables the parent to link first. One use case is if the child requires data from the parent.
post-link function	Link function that executives after child elements are linked to parent. It is commonly used for attaching event handlers and accessing child directives, but data required by the child directive should not be set here because the child directive will have already been linked.
restrict: string	Defines how to call the directive from within the DOM. Possible values (Assuming our directive name is demoDirective):E- Element name (<demo-directive></demo-directive>),A-Attribute (<div demo-directive>) C- Matching class ( ),M- By comment (<!-- directive: demo-directive -->). The restrictproperty

Parameter	Details
	can also support multiple options, for example -restrict: "AC" will restrict the directive to *Attribute* OR*Class*. If omitted, the default value is "EA"(Element or Attribute).
require: 'demoDirective'	Locate demoDirective's controller on the current element and inject its controller as the fourth argument to the linking function. Throw an error if not found.
require: '?demoDirective'	Attempt to locate the demoDirective's controller or pass null to the link fn if not found.
require: '^demoDirective'	Locate the demoDirective's controller by searching the element and its parents. Throw an error if not found.
require: '^^demoDirective'	Locate the demoDirective's controller by searching the element's parents. Throw an error if not found.
require: '?^demoDirective'	Attempt to locate the demoDirective's controller by searching the element and its parents or pass null to the link fn if not found.
require: '?^^demoDirective'	Attempt to locate the demoDirective's controller by searching the element's parents, or pass null to the link fn if not found.

3.6 ANGULARJS EXPRESSIONS

- AngularJS expression is like JavaScript expression surrounded with braces - {{ expression }}.
- AngularJS evaluates the specified expression and binds the result data to HTML.
- AngularJS expression can contain literals, operators and variables like JavaScript expression.
- 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>
```

Output

```
AngularJS Expression Demo :
2 + 2 = 4
2 - 2 = 0
2 * 2 = 4
2 / 2 = 1
```

- AngularJS expression is like JavaScript code expression except for the following differences :
  (1) AngularJS expression cannot contain conditions, loops, exceptions or regular expressions e.g. if-else, ternary, for loop, while loop etc.
  (2) AngularJS expression cannot declare functions.
  (3) AngularJS expression cannot contain comma or void.
  (4) AngularJS expression cannot contain return keyword.
- AngularJS expression contains literals of any data type.

Example

```
<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>
    {{"Hello World"}}<br />
    {{100}}<br />
    {{true}}<br />
    {{10.2}}
</div>
</body>
</html>
```

Output

```
AngularJS Expression Demo:
Hello World
100
true
10.2
```

- AngularJS expression can contain arithmetic operators which will produce the result based on the type of operands, similar to JavaScript :

```
<!DOCTYPE html>
<html >
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body >
<div ng-app>
    {{"Hello" + " World"}}<br />
    {{50 + 100 }}<br />
    {{true + false}}<br />
    {{10.0 + 10.2}}<br />
</div>
</body>
</html>
```

Output

```
Hello World
150
1
20.2
```

- AngularJS expression can contain variables declared via ng-init directive. The ng-init directive is used to declare AngularJS application variables of any data type.

```
<!DOCTYPE html>
<html >
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body >
<div ng-app ng-init="greet='Hello World!'; amount= 10000;rateOfInterest = 10.5; duration=10;  myArr = [100, 200]; person = { firstName:'Sachin', lastName :'Shah'}">
{{ (amount * rateOfInterest * duration)/100 }}<br />
    {{myArr[1]}} <br />
    {{person.firstName + " " + person.lastName}}
</div>
</body>
</html>
```

Output

```
10500
200
Sachin Shah
```

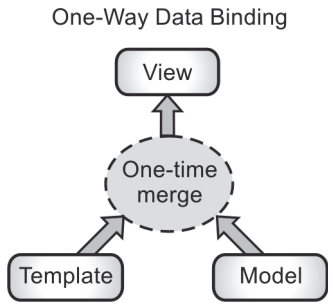
Module 3

3.7 ANGULARJS 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.

3.7.1 One-Way Data Binding

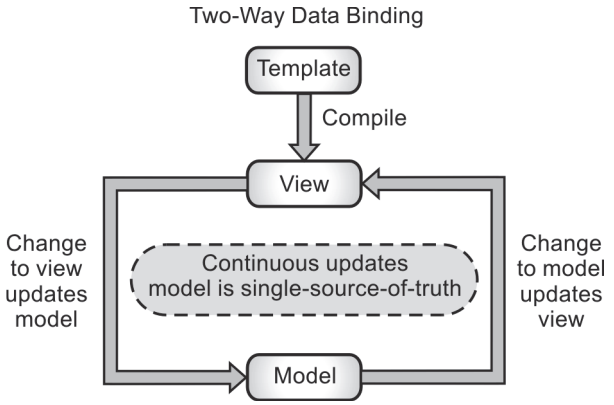
- The one-way data binding is an approach where a value is taken from the data model and inserted into an HTML element.
- There is no way to update model from view.
- It is used in classical template systems.
- These systems bind data in only one direction.



(1C2)**Fig. 3.7.1 : One-Way Data Binding**

3.7.2 Two-Way Data Binding

- Data-binding in Angular apps is the automatic synchronization of data between the model and view components.
- Data binding lets you treat the model as the single-source-of-truth in your application. The view is a projection of the model at all times. If the model is changed, the view reflects the change and vice versa.



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

Example

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
<body>
<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>
</body>
</html>
```

Output

Input something in the input box:

Name:

You wrote:

Input something in the input box:

Name: Ajeet

You wrote: Ajeet

- In the above example, the {{ firstName }} expression is an AngularJS data binding expression. Data binding in AngularJS binds AngularJS expressions with AngularJS data.
- {{ firstName }} is bound with ng-model="firstName".

Let's take another example where two text fields are bound together with two ng-model directives :

```
<!DOCTYPE html>
<html>
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
<body>
<div data-ng-app="" data-ng-init="quantity=1;price=20">
<h2>Cost Calculator</h2>
Quantity: <input type="number" ng-model="quantity">
Price: <input type="number" ng-model="price">
<p><b>Total in rupees:</b> {{quantity * price}}</p>
</div>
</body>
</html>
```

Output

Cost Calculator

Quantity: 1 Price: 20

Total in rupees: 20

3.8 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. The Table 3.8.1, lists important filters.

Table 3.8.1

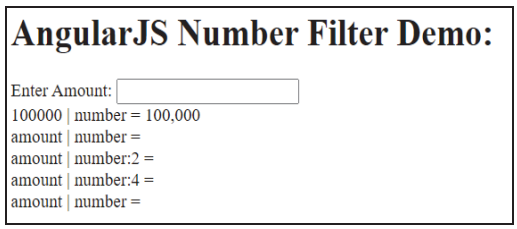
Filter Name	Description
Number	Formats a numeric data as text with comma and fraction.
Currency	Formats numeric data into specified currency format and fraction.
Date	Formats date to string in specified format.
Uppercase	Converts string to upper case.
Lowercase	Converts string to lower case.
Filter	Filters 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.

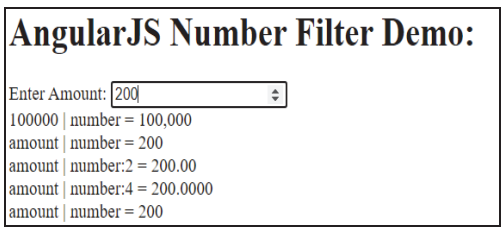
3.8.1 Number Filter

- A number filter formats numeric data as text with comma and specified fraction size.
- **Syntax**: {{ number_expression | number:fractionSize}}
- If a specified expression does not return a valid number, then number filter displays an empty string.
- The following example demonstrates how to use number filter with number expression or a model property.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app >
<h1>AngularJS Number Filter Demo: </h1>
   Enter Amount: <input type="number" ng-model="amount" /><br />

   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





3.8.2 Currency Filter

- The currency filter formats a number value as a currency.
- When no currency symbol is provided, default symbol for current locale is used.
- **Syntax**:{{ expression | currency : 'currency_symbol' : 'fraction'}}
- In the below example, we have applied currency filter to person.salary, which is a numeric property. It can be displayed with different currency symbols and fractions.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myApp" >
<h1>AngularJS currency Filter Demo: </h1>
<div ng-controller="myController">
      Default currency: {{person.salary | currency}} <br />
      Custom currency identifier: {{person.salary | currency:'Rs.'}} <br />
      No Fraction: {{person.salary | currency:'Rs.':0}} <br />
```

Module 3

```
    Fraction 2: <span ng-bind="person.salary|
currency:'GBP':2"></span>
</div>
<script>
    var myApp = angular.module('myApp', []);

    myApp.controller("myController", function ($scope) {
      $scope.person = { firstName: 'James', lastName:
'Bond', salary: 100000}
    });
</script>
</body>
</html>
```

Output

```
AngularJS currency Filter Demo:
Default currency: $100,000.00
Custom currency identifier: Rs.100,000.00
No Fraction: Rs.100,000
Fraction 2: GBP100,000.00
```

3.8.3 Date Filter

- Formats date to string based on the specified format.
- **Syntax**: {{ date_expression | date : 'format'}}

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/an
gular.min.js"></script>
</head>
<body ng-app >
<h1>AngularJS Date Filter Demo: </h1>
<div ng-init="person.DOB = 231408764592">
    Default date: {{person.DOB| date}} <br />
    Short date: {{person.DOB| date:'short'}} <br />
    Long date: {{person.DOB | date:'longDate'}} <br />
    Year: {{person.DOB | date:'yyyy'}} <br />
</div>
</body>
</html>
```

Output

```
AngularJS Date Filter Demo:
Default date: May 2, 1977
Short date: 5/2/77 1:42 PM
Long date: May 2, 1977
Year: 1977
```

3.8.4 Uppercase/lowercase Filter

- The uppercase filter converts the string to upper case and lowercase filter converts the string to lower case.
- **Syntax** : {{expression | lowercase}} or {{expression | uppercase}}

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/an
gular.min.js"></script>
</head>
<body ng-app >
<h1>AngularJS Uppercase/Lowercase Filter Demo: </h1>
<div ng-
init="person.firstName='Sachin';person.lastName='Shah'">
    Lower case: {{person.firstName + ' ' + person.lastName
| lowercase}} <br />
    Upper case: {{person.firstName + ' ' + person.lastName
| uppercase}}
</div>
</body>
</html>
```

Output

```
AngularJS Uppercase/Lowercase Filter Demo:
Lower case: sachin shah
Upper case: SACHIN SHAH
```

3.8.5 Filter

- Filter selects items from an array based on the specified criteria and returns a new array.
- **Syntax** : {{ expression | filter : filter_criteria }}
- In the below example, searchCriteria contains a text entered in the input box, which will be used to filter items of an array myArr using filter:searchCriteria expression.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myApp" >
<h1>AngularJS Filter Demo: </h1>
<div ng-controller="myController">
      Enter Name to search: <input type="text" ng-model="searchCriteria" /><br />
      Result: {{myArr | filter:searchCriteria}}
</div>
<script>
      var myApp = angular.module('myApp', []);

      myApp.controller("myController", function ($scope) {
         $scope.myArr = ['Sachin', 'Altab', 'Nilesh', 'Shraddha', 'Anagha', 'Ram']
      });
</script>
</body>
</html>
```

Output :

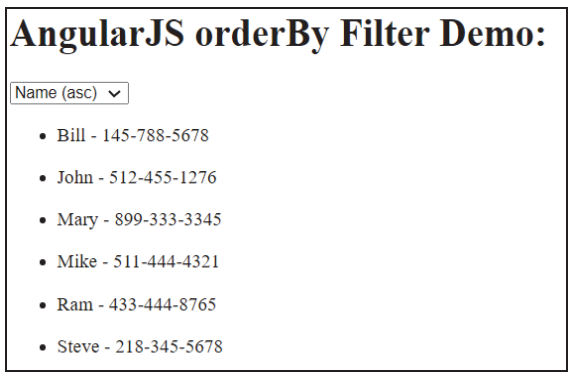
3.8.6 orderBy Filter

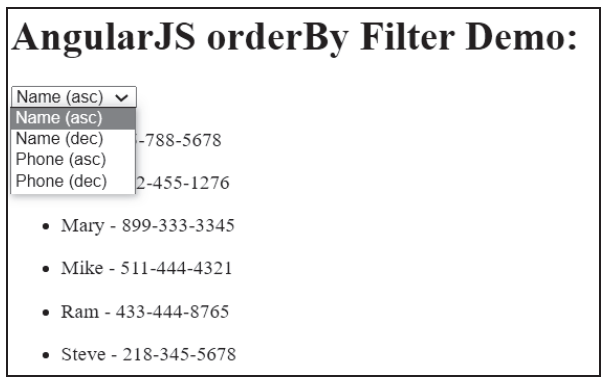
- The orderBy filter sorts an array based on specified expression predicate.
- **Syntax**:{{ expression | orderBy : predicate_expression : reverse}}
- The above example displays a list of person names and phone numbers in a particular order specified using orderBy:SortOrder filter. SortOrder is a model property and will be set to the selected value in the dropdown.
- Therefore, based on the value of SortOrder, ng-repeat directive will display the data.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myApp" >
<h1>AngularJS orderBy Filter Demo: </h1>
<div ng-controller="myController">
<select ng-model="SortOrder">
<option value="+name">Name (asc)</option>
<option value="-name">Name (dec)</option>
<option value="+phone">Phone (asc)</option>
<option value="-phone">Phone (dec)</option>
</select>
<ul ng-repeat="person in persons | orderBy:SortOrder">
<li>{{person.name}} - {{person.phone}}</li>
</ul>
</div>
<script>
      var myApp = angular.module('myApp', []);

      myApp.controller("myController", function ($scope) {

         $scope.persons = [{ name: 'John', phone: '512-455-1276' },
{ name: 'Mary', phone: '899-333-3345' },
{ name: 'Mike', phone: '511-444-4321' },
{ name: 'Bill', phone: '145-788-5678' },
{ name: 'Ram', phone: '433-444-8765' },
{ name: 'Steve', phone: '218-345-5678' }]

         $scope.SortOrder = '+name';
      });
</script>
</body>
</html>
```

Module 3

Output





3.9 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>
```

- Here, we declare a controller named *studentController*, using the ng-controller directive. We define it as follows:

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

- The studentController is defined as a JavaScript object with $scope as an argument.
- The $scope refers to application which uses the studentController object.
- The $scope.student is a property of studentController object.
- The firstName and the lastName are two properties of $scope.student object. We pass the default values to them.
- The property fullName is the function of $scope.student object, which returns the combined name.
- In the fullName function, we get the student object and then return the combined name.
- As a note, we can also define the controller object in a separate JS file and refer that file in the HTML page.
- Now we can use studentController's student property using ng-model or using expressions as follows:

```
Enter first name: <input type = "text" ng-model =
"student.firstName"><br>
Enter last name: <input type = "text" ng-model =
"student.lastName"><br>
<br>
You are entering: {{student.fullName()}}
```

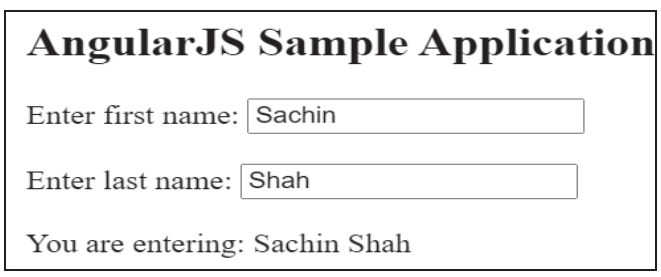
- We bound student.firstName and student.lastname to two input boxes.
- We bound student.fullName() to HTML.
- Now whenever you type anything in first name and last name input boxes, you can see the full name getting updated automatically.

The following example shows the use of controller :

```
<html>
<head>
<title>Angular JS Controller</title>
```

```
<script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular
.min.js"></script>
</head>
<body>
<h2>AngularJS Sample Application</h2>
<div ng-app = "mainApp" ng-controller =
"studentController">
      Enter first name: <input type = "text" ng-model =
"student.firstName"><br><br>
      Enter last name: <input type = "text" ng-model =
"student.lastName"><br>
<br>

      You are entering: {{student.fullName()}}
</div>
<script>
      var mainApp = angular.module("mainApp", []);
      mainApp.controller('studentController', function($scope)
{
        $scope.student = {
          firstName: "Sachin",
          lastName: "Shah",

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

Output



3.10 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.
- As we have seen in the controller section, we can attach properties and methods to the $scope object inside controller function.
- The view can display $scope data using an expression, ng-model, or ng-bind directive, as shown below.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/an
gular.min.js"></script>
</head>
<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>
</body>
</html>
```

Output

Module 3

- AngularJS creates and injects a different $scope object to each controller in an application.
- So, the data and methods attached to $scope inside one controller cannot be accessed in another controller.
- With the nested controller, child controller will inherit the parent controller's scope object.
- Therefore, child controller can access properties added in parent controller but parent controller cannot access properties added in child controller.

Note : The ng-model directive is used for two-way data binding. It transfers the data from controller to view and vice-versa. An expression and ng-bind directive transfers data from controller to view but not vice-versa.

3.10.1 $rootscope

- 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.
- The following example demonstrates the $rootScope and $scope object.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myNgApp">
<h1>AngularJS $rootScope Demo: </h1>
<div ng-controller="parentController">
    Controller Name: {{controllerName}} <br />
    Message: {{message}} <br />
<div style="margin:10px 0 10px 20px;" ng-controller="childController">
        Controller Name: {{controllerName}} <br />
        Message: {{message}} <br />
</div>
</div>
<div  ng-controller="siblingController">
    Controller Name: {{controllerName}} <br />
    Message: {{message}} <br />
</div>
<script>
    var ngApp = angular.module('myNgApp', []);
    ngApp.controller('parentController', function ($scope, $rootScope) {
        $scope.controllerName = "parentController";
        $rootScope.message = "Hello World!";
    });
    ngApp.controller('childController', function ($scope) {
        $scope.controllerName = "childController";
    });
    ngApp.controller('siblingController', function ($scope) {
    });
</script>
</body>
</html>
```

Output

- As per the above example, properties added in $rootScope are available in all the controllers.
- The $scope object contains various methods.
- The following table lists important methods of $scope object.

Method	Description
$new()	Creates new child scope.
$watch()	Register a callback to be executed whenever model property changes.
$watchGroup()	Register a callback to be executed whenever model properties changes. Here, specify an array of properties to be tracked.
$watchCollection()	Register a callback to be executed whenever model object or array property changes.
$digest()	Processes all of the watchers of the current scope and its children.

Method	Description
$destroy()	Removes the current scope (and all of its children) from the parent scope.
$eval()	Executes the expression on the current scope and returns the result.
$apply()	Executes an expression in angular outside the angular framework.
$on()	Register a callback for an event.
$emit()	Dispatches the specified event upwards till $rootScope.
$broadcast()	Dispatches the specified event downwards to all child scopes.

3.11 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 following core components which can be injected into each other as dependencies.

(1) Value (2) Factory
(3) Service (4) Provider
(5) Constant

3.11.1 Value

- In AngularJS, value is a simple object. It can be a number, string or JavaScript object.
- It is used to pass values in factories, services or controllers during run and 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.
- The first parameter specifies the name of the value, and the second parameter is the value itself. Factories, services and controllers can now reference these values by their name.

Injecting a value

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);
});
```

3.11.2 Factory

- Factory is a function that is used to return value. When a service or controller needs a value injected from the factory, it creates the value on demand. It normally uses a factory function to calculate and return the value.
- Let's take an example that defines a factory on a module, and a controller which gets the factory created value injected :

```
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);
});
```

Note : It is not the factory function that is injected, but the value produced by the factory function.

Module 3

3.11.3 Service

In AngularJS, service is a JavaScript object which contains a set of functions to perform certain tasks. Services are created by using service() function on a module and then injected into controllers.

```
//define a module
var mainApp = angular.module("mainApp", []);
...
//create a service which defines a method square to return squ
are 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, CalcSer
vice, defaultInput) {
$scope.number = defaultInput;
$scope.result = CalcService.square($scope.number);
$scope.square = function() {
$scope.result = CalcService.square($scope.number);
}
});
```

3.11.4 Provider

In AngularJS, provider is used internally to create services, factory etc. during config phase (phase during which AngularJS bootstraps itself). It is the most flexible form of factory you can create. Provider is a special factory method with a get() function which is used to return the value/service/factory.

```
//define a module
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;
    };
  });
});
```

3.11.5 Constants

We cannot inject values into the module.config() function. Instead constants are used to pass values at config phase.

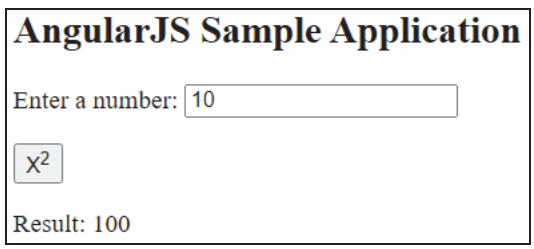
```
mainApp.constant("configParam", "constant value");
```

Let's take an example to deploy all above mentioned directives.

```
<!DOCTYPE html>
<html>
<head>
<title>AngularJS Dependency Injection</title>
</head>
<body>
<h2>AngularJS Sample Application</h2>

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

<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", 10);
```

```
        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>
</body>
</html>
```

Output



3.12 ANGULARJS 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.
- AngularJS also allows you to create custom service for your application.
- Most AngularJS services interact with the controller, model or custom directives. However, some services interact with view (UI) also for UI specific tasks.
- The Table 3.12.1, lists all the built-in AngularJS services.

Table 3.12.1

$anchorScroll	$exceptionHandler	$interval	$rootScope
$animate	$filter	$locale	$sceDelegate
$cacheFactory	$httpParamSerializer	$location	$sce
$templateCache	$httpParamSerializerJQLike	$log	$templateRequest
$compile	$http	$parse	$timeout
$controller	$httpBackend	$q	$window
$document	$interpolate	$rootElement	

- All the Angular services are **lazy instantiated** and **singleton**. It means AngularJS framework instantiates a service when an application component depends on it.
- Also, all the components share the same instance of a service.
- Let's see some of the important built-in services.

3.12.1 $http Service

- The $http service is one of the most common used services in AngularJS applications.
- The service makes a request to the server, and lets your application handle the response.
- $http is a service as an object. It includes following shortcut methods.

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.

Let's look at some of the important methods of $http.

Module 3

(a) $http.get()

- $http.get() method sends http GET request to the remote server and retrieves the data.
- **Signature**: HttpPromise $http.get(url)
- $http.get() method returns HttpPromise object, which includes various methods to process the response of http GET request.
- The following example demonstrates the use of $http service in a controller to send HTTP GET request.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app ="myApp">
<h1>AngularJS $http Demo: </h1>
<div>
<div ng-controller="myController">
        Response Data: {{data}} <br />
        Error: {{error}}
</div>
</div>
<script>
      var myApp = angular.module('myApp', []);
      myApp.controller("myController", function ($scope,
$http) {
         var onSuccess = function (data, status, headers,
config) {
            $scope.data = data;
         };
         var onError = function (data, status, headers, config) {
            $scope.error = status;
         }
         var promise = $http.get("/demo/getdata");
promise.success(onSuccess);
promise.error(onError);
    });
</script>
</body>
</html>
```

Output

- In the above example, 'myController' controller includes $http parameter, so that it can be used to send GET request. AngularJS automatically injects $scope parameter at runtime.
- The $http.get() method returns HttpPromise which includes methods like success() and error().
- The success() method registers a callback method which is called when a request completes successfully.
- The error() method registers a callback method which is called when a request fails and returns an error.

(b) $http.post()

- The $http.post() method sends Http POST request to the remote server to submit and retrieve the data.
- Signature: HttpPromise $http.post(url, dataToSubmit);
- The following example demonstrates $http.post() method.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myApp">
<h1>AngularJS $http.post() Demo: </h1>
<div ng-controller="myController">
      Response Data: {{data}} <br />
```

```
    Error: {{error}}
</div>
<script>
    var myApp = angular.module('myApp', []);
    myApp.controller("myController", function ($scope,
$http) {
        var onSuccess = function (data, status, headers,
config) {
            $scope.data = data;
        };
        var onError = function (data, status, headers, config) {
            $scope.error = status;
        }
        $http.post('/demo/submitData', { myData: 'Hello
World!' })
.success(onSuccess)
.error(onError);
    });
</script>
</body>
</html>
```

Output

(c) $http()

You can use construction function of $http service to perform http request, as shown below.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/an
gular.min.js"></script>
</head>
<body ng-app="myApp">
<h1>AngularJS $http Demo: </h1>
<div ng-controller="myController">
        Response Data: {{data}} <br />
            Error: {{error}}
</div>
<script>
    var myApp = angular.module('myApp', []);
     myApp.controller("myController", function ($scope,
$http) {
        var onSuccess = function (data, status, headers,
config) {
            $scope.data = data;
        };
        var onError = function (data, status, headers, config) {
            $scope.error = status;
        }
       var getReq = {
            method: 'GET',
            url: '/demo/getdata'
        };
       $http(getReq).success(onSuccess).error(onError);
       var postReq = {
            method: 'POST',
            url: '/demo/submitData',
            data: { myData: 'test data' }
        };
       $http(postReq).success(onSuccess).error(onError);
    });
</script>
</body>
</html>
```

Output

3.12.2 $log Service

- AngularJs includes logging service $log, which logs the messages to the browser's console.
- The $log service includes different methods to log the error, information, warning or debug information. It can be useful in debugging and auditing.

Module 3

- In the below example, controller function includes $log parameter which will be supplied by AngularJS framework.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myApp">
<h1>AngularJS $log Demo: </h1>
<div ng-controller="myController">
<p>Please check the browser console for the logging information.</p>
</div>
<script>
    var myApp = angular.module('myApp', []);
    myApp.controller("myController", function ($log) {
        $log.log('This is log.');
        $log.error('This is error.');
        $log.info('This is info.');
        $log.warn('This is warning.');
        $log.debug('This is debugging.');
    });
</script>
</body>
</html>
```

Output

3.12.3 $interval Service

- AngularJS includes $interval service which performs the same task as setInterval() method in JavaScript.
- The $interval is a wrapper for setInterval() method, so that it will be easy to override, remove or mocked for testing.
- The $interval service executes the specified function on every specified milliseconds duration.
- **Signature :** $interval(fn, delay, [count], [invokeApply], [Pass]);
- The following example demonstrates $interval service that displays a counter on each 1000 milliseconds.
- In the below example, $interval service calls increaseCounter() function on every 1000 milliseconds. The increaseCounter() function increases the $scope.counter property by 1. Thus, counter increases on every milliseconds.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myApp">
<h1>AngularJS $interval Demo: </h1>
<div ng-controller="myController">
    {{counter}}
</div>
<script>
    var myApp = angular.module('myApp', []);
    myApp.controller("myController", function ($scope,
$interval) {
        $scope.counter = 0;
        var increaseCounter = function () {
            $scope.counter = $scope.counter + 1;
        }
        $interval(increaseCounter, 1000);
    });
</script>
</body>
</html>
```

Output

3.12.4 $window Service

- AngularJs includes $window service which refers to the browser window object.
- In the JavaScript, window is a global object which includes many built-in methods like alert(), prompt() etc.

- The $window service is a wrapper around window object, so that it will be easy to override, remove or mocked for testing. It is recommended to use $window service in AngularJS instead of global window object directly.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app="myApp" ng-controller="myController">
<h1>AngularJS cancel $window Demo: </h1>
<button ng-click="DisplayAlert('Hello World!')">Display Alert</button>
<button ng-click="DisplayPrompt()">Display Prompt</button>
<script>
    var myApp = angular.module('myApp', []);

    myApp.controller("myController", function ($scope, $window) {

        $scope.DisplayAlert = function (message) {
            $window.alert(message);
        }

        $scope.DisplayPrompt = function () {
            var name = $window.prompt('Enter Your Name');

            $window.alert('Hello ' + name);
        }

    });
</script>
</body>
</html>
```

Output

3.13 FORM VALIDATION

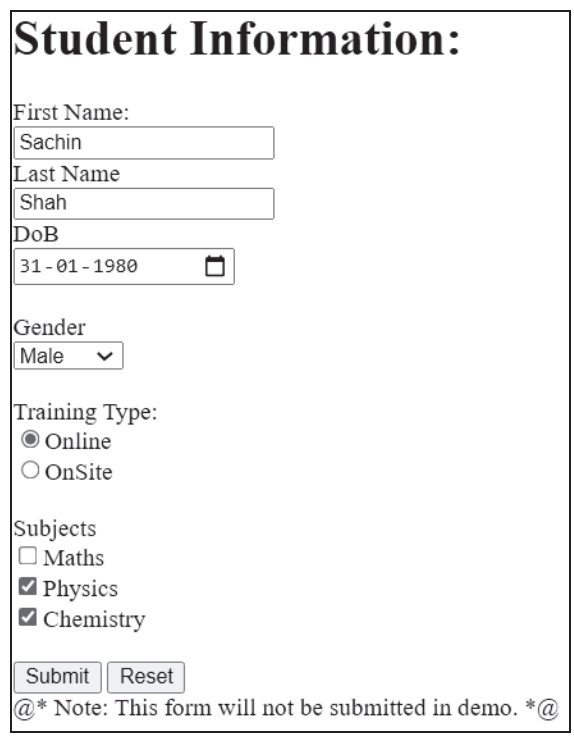
- The HTML form is a collection of input controls where user can enter the data. Here, you will learn how to display AngularJS form and submit the data.
- First, we will create following Student Information form with submit and reset functionality.


```
<!DOCTYPE html>
<html ng-app="studentApp">
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-controller="studentController">
<h1>Student Information:</h1>
<form ng-submit="submitStudnetForm()" >
<label for="firstName" >First Name: </label><br />
<input type="text" id="firstName" ng-model="student.firstName" /><br />

<label for="lastName">Last Name</label><br />
<input type="text" id="lastName" ng-model="student.lastName" /><br />
<label for="dob" >DoB</label><br />
```

Module 3

```
<input type="date" id="dob" ng-model="student.DoB"
/><br /><br />

<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><br /><br />
<span>Subjects</span><br />
<label><input type="checkbox" ng-model="student.maths"
/>Maths</label><br />
<label><input type="checkbox" ng-
model="student.physics" />Physics</label><br />
<label><input type="checkbox"  ng-
model="student.chemistry" />Chemistry</label><br
/><br />

<input type="submit" value="Submit" />
<input type="reset" ng-click="resetForm()" value="Reset"
/><br />
        @* Note: This form will not be submitted in demo.
*@
</form>
<script>
//1. create app module
    var studentApp = angular.module('studentApp', []);

//2. create controller
    studentApp.controller("studentController", function
($scope, $http) {

//3. attach originalStudent model object
        $scope.originalStudent = {
            firstName: 'Sachin',
            lastName: 'Shah',
            DoB: new Date('01/31/1980'),
            gender: 'male',
            trainingType: 'online',
            maths: false,
            physics: true,
            chemistry: true
        };

//4. copy originalStudent to student. student will be bind to a
form
        $scope.student =
angular.copy($scope.originalStudent);

 //5. create submitStudentForm() function. This will be called
when user submits the form
        $scope.submitStudnetForm = function () {

            // send $http request to save student
        };

//6. create resetForm() function. This will be called on Reset
button click.
        $scope.resetForm = function () {
            $scope.student =
angular.copy($scope.OriginalStudent);
        };
   });
</script>
</body>
</html>
```

Output



The following is a step by step explanation of the above example :

(1) Create an HTML page and wrap all the necessary input controlls into <form> tag.

(2) Create the AngularJS application module in the <script> tag.

(3) Create studentController in application module.

(4) Create originalStudent object and attach to the $scope with required properties. This will stay unchanged during entire life cycle.

(5) Create new student object and attach to the $scope and copy all the properties and values from originalStudent. This student object will be bound to the form using ng-model directive. Therefore, if user changes form values then the student object will also get changed.

(6) Create submitStudnetForm function which will get called when user submits the form using Submit button. Here, send http POST request to the remote server to submit the data using $http service.

(7) Create resetForm() function, which will reset the form values to the originalStudent values by copying it to student object.

(8) Apply ng-app, ng-controller directives.

(9) Apply ng-model directives to each HTML input element to bind appropriate properties of student object.

(10) Apply ng-submit directive to form which will call submitStudentForm() on the form submit event.

(11) Apply ng-click directive to reset button which will call resetForm() on the button click event.

- An AngularJS form can be submitted using either ng-submit or ng-click directive but not both.
- **Ng-submit :** Binds angular expression to onsubmit event when form does not include action attribute.
- **Ng-click :** Binds angular expression to onclick event.

> **Note** : The angular form can be submitted using ng-submit directive on the form tag or using ng-click directive on <input type="submit" /> element. Use either ng-submit or ng-click directive but not both to submit the form. The form will be submitted twice if both ng-submit and ng-click directives are used.

- Now, we will implement client-side validation in AngularJS form.
- AngularJS includes the following validation directives.

Directive	Description
ng-required	Sets required attribute on an input field.
ng-minlength	Sets minlength attribute on an input field.
ng-maxlength	Sets maxlength attribute on an input field. Setting the attribute to a negative or non-numeric value, allows view values of any length.
ng-pattern	Sets pattern validation error key if the ngModel value does not match the specified RegEx expression.

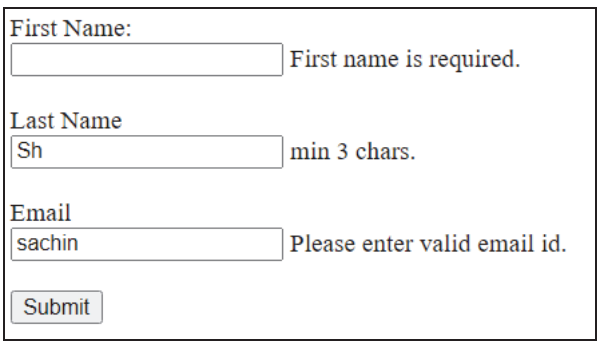
- Let's implement validation in the student form which contains First Name, Last Name and Email fields.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app >
<form name="studentForm" novalidate>
<label for="firstName">First Name: </label><br />
<input type="text" name="firstName" ng-model="student.firstName" ng-required="true" />
<span 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 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 ng-show="studentForm.email.$touched && studentForm.email.$error.email">Please enter valid email id.</span><br /><br />
```

Module 3

```
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Output



Let's understand the above example step by step :

(1) Apply novalidate attribute in <form> tag. The novalidate attribute will disable the browser's default validation.

(2) Set the name attribute in <form> and other elements, which will be used to obtain a reference of the elements.

(3) Now, set ng-required="true" on the input element of First Name. Also, set name attribute to the name of model property, "firstName" in this case.

(4) Create element to specify an error message with every input filed where the validation directive is applied.

(5) Set ng-show directives to element to an expression "studentForm.firstName.$touched && studentForm.firstName.$error.required". This expression will return true if a user tabbed out without entering FirstName.

(6) The same way set ng-minlength & ng-maxlength directives to last name. Also, set ng-show directive to "studentForm.lastName.$touched&& studentForm.lastName.$error.minlength" expression to element adjacent to LastName input field.

(7) Create another for maxlength validation message.

(8) Email will be validated automatically with input type=email. Also, create for email validation message.

We have applied an expression "studentForm.firstName.$touched&& studentForm.firstName.$error.required" to the , in the above example. $touched & $error are built-in properties which return the state of the specified input controls and form.

Let's learn about the state properties.

☞ State Properties

- Angular includes properties which return the state of form and input controls.
- The state of the form and control changes based on the user's interaction and validation errors.
- These built-in properties can be accessed using form name or input control name.
- To check the form status use formName.propertyName, and to check the state of input control, use formName.inputFieldName.propertyName.

The Table 3.13.1 lists the state properties.

Table 3.13.1

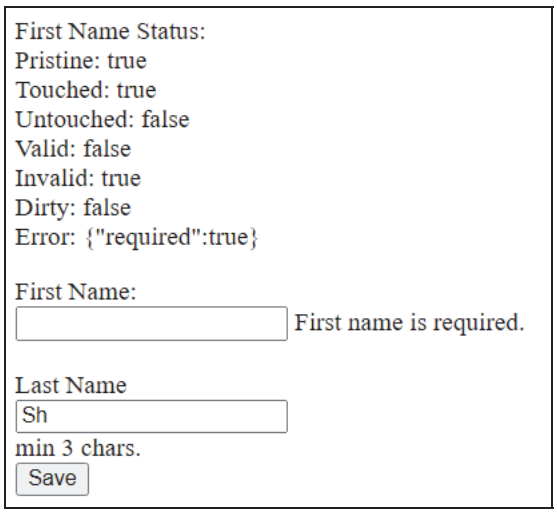
Property	Description
$error	$error object contains all the validation attributes applied to the specified element.
$pristine	Returns true if the user has not interacted with control yet else returns false.
$valid	Returns true if the model is valid
$invalid	Returns true if the model is invalid
$dirty	Returns true if user changed the value of model at least once
$touched	Returns true if the user has tabbed out from the control.
$untouched	Returns true if the user has not tabbed out from the control.

The following example demonstrates the state properties.

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
</head>
<body ng-app>
<form name="studentForm" novalidate>
<p>
```

```
        First Name Status: <br />
        Pristine: {{studentForm.firstName.$pristine}} <br
/>
        Touched: {{studentForm.firstName.$touched}}<br
/>
        Untouched:
{{studentForm.firstName.$untouched}}<br />
        Valid: {{studentForm.firstName.$valid}} <br />
        Invalid: {{studentForm.firstName.$invalid}} <br />
        Dirty: {{studentForm.firstName.$dirty}} <br />
        Error: {{studentForm.firstName.$error}} <br />

</p>
<label for="firstName">First Name: </label><br />
<input type="text" name="firstName" ng-
model="student.firstName" ng-required="true" />
<span 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" /><br />
<span ng-
show="studentForm.lastName.$error.minlength">min 3
chars.</span>
<span ng-
show="studentForm.lastName.$error.maxlength">Max 10
chars.</span><br />

<input type="submit" value="Save" />
</form>
</body>
</html>
```

Output



3.14 ANGULARJS ROUTING

- We can build Single Page Application (SPA) with AngularJS. It is a web app that loads a single HTML page and dynamically updates that page as the user interacts with the web app.
- AngularJS supports SPA using routing module **ngRoute**. This routing module acts based on the url. When a user requests a specific url, the routing engine captures that url and renders the view based on the defined routing rules.

3.14.1 ngRoute

- AngularJS ngRoute module provides routing, deep linking services and directives for angular applications. We have to download angular-route.js script that contains the ngRoute module from AngularJS official website to use the routing feature.
- You can also use the CDN in your application to include this file. In this section, We are going to use the Google CDN.

```
https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.
min.js
```

- If you are bundling this file into your application, then you can add it to your page with below code.

```
<script src="angular-route.js">
```

- If you want to include it from Google CDN, then use below code.

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16//a
ngular-route.min.js"></script>
```

- Then load the ngRoute module in your AngularJS application by adding it as a dependent module as shown below.

```
angular.module('appName', ['ngRoute']);
```

3.14.2 ngView

- ngView directive is used to display the HTML templates or views in the specified routes.
- Every time the current route changes, the included view changes with it according to the configuration of the $route service.

Module 3

3.14.3 $routeProvider

- $routeProvider is used to configure the routes. We use the ngRoute config() to configure the $routeProvider.
- The config() takes a function which takes the $routeProvider as parameter and the routing configuration goes inside the function. $routeProvider has a simple API, accepting either the when() or otherwise() method.

3.14.4 AngularJS Routing Syntax

The following syntax is used to configure the routes in AngularJS.

```
var app = angular.module("appName", ['ngRoute']);
app.config(function($routeProvider) {
    $routeProvider
        .when('/view1', {
            templateUrl: 'view1.html',
            controller: 'FirstController'
        })
        .when('/view2', {
            templateUrl: 'view2.html',
            controller: 'SecondController'
        })
        .otherwise({
            redirectTo: '/view1'
        });
});
```

- when() method takes a **path** and a **route** as parameters.
- **path** is a part of the URL after the # symbol.
- **route** contains two properties – templateUrl and controller.
- **templateUrl** property defines which HTML template AngularJS should load and display inside the div with the ngView directive.
- **controller** property defines which controllers should be used with the HTML template.
- When the application is loaded, **path** is matched against the part of the URL after the # symbol. If no route paths matches the given URL the browser will be redirected to the path specified in the otherwise() function.

3.14.5 AngularJS Routing Example

Now let's go through a simple example to understand the AngularJS rounting. At first, we will define a module, some routes, create controllers and create multiple views. Finally, we will create the shell page of our application to hold the multiple views.

(1) Create a module named mainApp and load ngRoute as a dependent module.

(2) Configure the routes using $routeProvider.

(3) We use two **paths** in the example, **/home and /viewStudents**.

(4) We use only a single controller in this example, StudentController

(5) **StudentController** is initialized with an array of **students** and a **message**. We will be showing the message in the home page and the students list in viewStudents page.

(6) Save this file as **main.js**

main.js

```
var mainApp = angular.module("mainApp", ['ngRoute']);
mainApp.config(function($routeProvider) {
    $routeProvider
        .when('/home', {
            templateUrl: 'home.html',
            controller: 'StudentController'
        })
        .when('/viewStudents', {
            templateUrl: 'viewStudents.html',
            controller: 'StudentController'
        })
        .otherwise({
            redirectTo: '/home'
        });
});
mainApp.controller('StudentController', function($scope) {
    $scope.students = [
        {name: 'Mark Waugh', city:'New York'},
        {name: 'Steve Jonathan', city:'London'},
        {name: 'John Marcus', city:'Paris'}
    ];
    $scope.message = "Click on the hyper link to view the students list.";
});
```

- For example, if the URL is like https://www.techneobooks.com/index.html#/home.
- The URL part after the # matches **/home**, it will load **home.html** page and if it matches **/viewStudents** then it will load **viewStudents.html** in to the shell page. If nothing matches, then it will g in otherwise condition and page will be redirected to **home.html**.
- Now we can create our views and save as **home.html** and **viewStudents.html** files.

home.html

```
<div class="container">
    <h2> Welcome </h2>
    <p>{{message}}</p>
    <a href="#/viewStudents"> View Students List</a>
</div>
```

- This is the default page of our application. In this view, we just print out the message, which we have already initialized in the **StudentController**.
- You can also see a link to the **viewStudents** page.

viewStudents.html

```
<div class="container">
    <h2> View Students </h2>
    Search:
    <br/>
        <input type="text" ng-model="name" />
    <br/>
    <ul>
        <li ng-repeat="student in students | filter:name">{{student.name}} , {{student.city}}</li>
    </ul>

    <a href="#/home"> Back</a>
</div>
```

- In the above view, you can see a list of students with a search option.
- Finally, follow below steps to complete our AngularJS routing example application.
    - ng-app auto-bootstraps our application **mainApp**
    - **ngView** directive is the placeholder of the views – **home.html** and **viewStudents.html**
    - Include **angular.min.js** and **angular-route.min.js**
    - Include **main.js** which we have created in the earlier steps.
    - Save the file as **index.html**

index.html

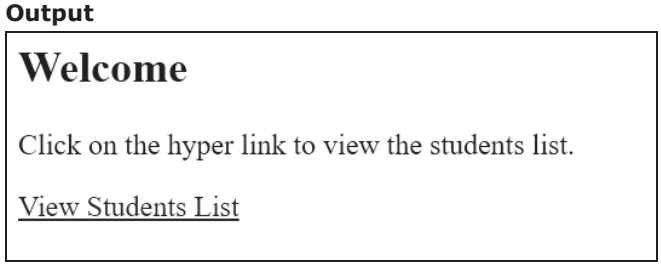
```
<!DOCTYPE html>
<html>
    <head lang="en">
    <meta charset="utf-8">
    <title>AngularJS Routing</title>

    </head>
    <body>

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

    <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"></script>
    <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16//angular-route.min.js"></script>
    <script type="text/javascript" src="main.js"></script>
    </body>
</html>
```

That's all for our AngularJS Routing example. Our application is ready to run.

Run your application

- Save all the files in the same directory.
- open index.html from your browser

Output



3.15 BUILT-IN HELPER FUNCTIONS

This section discusses about the various built-in helper functions in AngularJS.

3.15.1 angular.equals

- The angular.equals function compares and determines if 2 objects or values are equal.
- **Syntax** : angular.equals(value1, value2)

Module 3

- Angular.equals performs a deep comparison and returns true if and only if at least 1 of the following conditions is met.
  (1) If the objects or values pass the === comparison.
  (2) If both objects or values are of the same type, and all of their properties are also equal by using angular.equals.
  (3) Both values are equal to NaN.
  (4) Both values represent the same regular expression's result.
- This function is helpful when you need to deep compare objects or arrays by their values or results rather than just references.

Example

```
angular.equals(1, 1) // true
angular.equals(1, 2) // false
angular.equals({}, {}) // true, note that {}===={} is false
angular.equals({a: 1}, {a: 1}) // true
angular.equals({a: 1}, {a: 2}) // false
angular.equals(NaN, NaN) // true
```

3.15.2 angular.toJson

- The function angular.toJson will take an object and serialize it into a JSON formatted string.
- Unlike the native function JSON.stringify,
- This function will remove all properties beginning with $$ (as angular usually prefixes internal properties with $$).
- **Syntax**: angular.toJson(object)
- As data needs to be serialized before passing through a network, this function is useful to turn any data you wish to transmit into JSON.
- This function is also useful for debugging as it works similarly to a .toString method would act.

Example

```
angular.toJson({name: "barf", occupation: "mog",
$$somebizzareproperty: 42})
// "{"name":"barf","occupation":"mog"}"
angular.toJson(42)
// "42"
angular.toJson([1, "2", 3, "4"])
// "[1,"2",3,"4"]"
var fn = function(value) {return value}
angular.toJson(fn)
// undefined, functions have no representation in JSON
```

3.15.3 angular.copy

- The angular.copy function takes an object, array or a value and creates a deep copy of it.
- Syntax : angular.copy(object)

Example

Objects

```
let obj = {name: "vespa", occupation: "princess"};
let cpy = angular.copy(obj);
cpy.name = "yogurt"
// obj = {name: "vespa", occupation: "princess"}
// cpy = {name: "yogurt", occupation: "princess"}
```

Arrays

```
var w = [a, [b, [c, [d]]]];
var q = angular.copy(w);
// q = [a, [b, [c, [d]]]]
```

- In the above example **angular.equals(w, q)** will evaluate to true because **.equals** tests equality by value.
- However, w === q will evaluate to false because strict comparison between objects and arrays is done by reference.

3.15.4 angular.isString

- The function angular.isString returns true if the object or value given to it is of the type string.
- **Syntax** : angular.isString(value1)

Example

```
angular.isString("hello") // true
angular.isString([1, 2]) // false
angular.isString(42) // false
```

3.15.5 angular.isArray

- The angular.isArray function returns true if and only if the object or value passed to it is of the type Array.
- **Syntax** : angular.isArray(value)

Example

```
angular.isArray([]) // true
angular.isArray([2, 3]) // true
angular.isArray({}) // false
angular.isArray(17) // false
```

3.15.6 angular.merge

- The function angular.merge takes all the enumerable properties from the source object to deeply extend the destination object.
- The function returns a reference to the now extended destination object.
- **Syntax** : angular.merge(destination,source)

Example

```
angular.merge({}, {}) // {}
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.merge({a: 1}, {b: {c: {d: 2}}})
// {"a":1,"b":{"c":{"d":2}}}
```

3.15.7 angular.isDefined and angular.isUndefined

- The function angular.isDefined tests a value if it is defined.
- **Syntax** : angular.isDefined(someValue)

Example

```
angular.isDefined(42) // true
angular.isDefined([1, 2]) // true
angular.isDefined(undefined) // false
angular.isDefined(null) // true
```

- The function angular.isUndefined tests if a value is undefined (it is effectively the opposite of angular.isDefined).
- **Syntax** : angular.isUndefined(someValue)

Example

```
angular.isUndefined(42) // false
angular.isUndefined(undefined) // true
```

3.15.8 angular.isDate

- The angular.isDate function returns true if and only if the object passed to it is of the type Date.
- **Syntax** : angular.isDate(value)

Example

```
angular.isDate("lone star") // false
angular.isDate(new Date()) // true
```

3.15.9 angular.noop

- The angular.noop is a function that performs no operations, you pass angular.noop when you need to provide a function argument that will do nothing.
- **Syntax** : angular.noop
- A common use for angular.noop can be to provide an empty callback to a function that will otherwise throw an error when something else than a function is passed to it.

Example

```
$scope.onSomeChange = function(model, callback) {
  updateTheModel(model);
  if (angular.isFunction(callback)) {
callback();
  } else {
    throw new Error("error: callback is not a function!");
  }
};
$scope.onSomeChange(42, function() {console.log("hello
callback")});
// will update the model and print 'hello callback'
$scope.onSomeChange(42, angular.noop);
// will update the model
```

Additional Examples

```
angular.noop() // undefined
angular.isFunction(angular.noop) // true
```

3.15.10 angular.isElement

- The angular.isElement returns true if the argument passed to it is a DOM Element or a jQuery wrapped Element.
- **Syntax** : angular.isElement(element)
- This function is useful to type check if a passed argument is an element before being processed as such.

Example

```
angular.isElement(document.querySelector("body"))
// true
angular.isElement(document.querySelector("#some_id"))
```

Module 3

```
// false if "some_id" is not using as an id inside the selected DOM
angular.isElement("<div></div>")
// false
```

3.15.11 angular.isFunction

- The function angular.isFunction determines and returns true if and only if the value passed to is a reference to a function.
- The function returns a reference to the now extended destination object.
- **Syntax** : angular.isFunction(fn)

Example

```
var onClick = function(e) {return e};
angular.isFunction(onClick); // true
var someArray = ["pizza", "the", "hut"];
angular.isFunction(someArray ); // false
```

3.15.12 angular.identity

- The angular.identity function returns the first argument passed to it.
- **Syntax** : angular.identity(argument)
- This function is useful for functional programming; you can provide this function as a default in case an expected function was not passed.

Example

```
angular.identity(42) // 42
var mutate = function(fn, num) {
return angular.isFunction(fn) ? fn(num) :
angular.identity(num)
}
mutate(function(value)
{return value-7}, 42) // 35
mutate(null, 42) // 42
mutate("mount. rushmore", 42) // 42
```

3.15.13 angular.forEach

- The angular.for Each accepts an object and an iterator function.
- It then runs the iterator function over each enumerable property/value of the object.
- This function also works on arrays.
- .forEachfunction does not iterate over inherited properties (prototype properties); however the function will not attempt to process a null or an undefined value and will just return it.
- **Syntax**: angular.forEach(object, function(value, key) { // function});

Example

```
angular.forEach({"a": 12, "b": 34}, (value, key) =>
console.log("key: " + key + ", value: " + value))
// key: a, value: 12
// key: b, value: 34
angular.forEach([2, 4, 6, 8, 10], (value, key) =>
console.log(key))
// will print the array indices: 1, 2, 3, 4, 5
angular.forEach([2, 4, 6, 8, 10], (value, key) =>
console.log(value))
// will print the array values: 2, 4, 6, 7, 10
angular.forEach(undefined, (value, key) => console.log("key:
" + key + ", value: " + value))
// undefined
```

3.15.14 angular.isNumber

- The angular.isNumber function returns true if and only if the object or value passed to it is of the type Number, this includes +Infinity, -Infinity and NaN.
- **Syntax** : angular.isNumber(value)

Example

```
angular.isNumber("23") // false
angular.isNumber(23) // true
angular.isNumber(NaN) // true
angular.isNumber(Infinity) // true
This function will not cause a type coercion such as
"23" == 23 // true
```

3.15.15 angular.is Object

- The angular.isObject return true if and only if the argument passed to it is an object, this function will also return true for an Array and will return false for null even though typeof null is object.
- **Syntax** : angular.isObject(value)
- This function is useful for type checking when you need a defined object to process.

Example

```
angular.isObject({name: "skroob", job: "president"}) // true
angular.isObject(null) // false
angular.isObject([null]) // true
angular.isObject(new Date()) // true
angular.isObject(undefined) // false
```

3.15.16 angular.fromJson

- The function angular.fromJson will deserialize a valid JSON string and return an Object or an Array.
- **Syntax** : angular.fromJson(stringl object)
- Note that this function is not limited to only strings, it will output a representation of any object passed to it.

Example

```
angular.fromJson("{\"yogurt\": \"strawberries\"}")
// Object {yogurt: "strawberries"}
angular.fromJson('{jam: "raspberries"}')
// will throw an exception as the string is not a valid JSON
angular.fromJson(this)
// Window {external: Object, chrome: Object, _gaq: Y,
angular: Object, ng339: 3…} angular.fromJson([1, 2])
// [1, 2]
typeofangular.fromJson(new Date())
// "object"
```

3.16 USING ANGULARJS WITH TYPESCRIPT

- **TypeScript,** a free and open source programming language from Microsoft can be used to overcome these shortcomings.
- TypeScript is a typed superset of JavaScript which compiles to plain JavaScript. This is a strongly typed language and provides class-based OOPs like experience for designing JavaScript applications.
- TypeScript can improve refactoring and quality of angularjs applications. Another point to note in favor of using TypeScript for AngularJS apps is that the next version of Angular (Angular 2 and so on) uses TypeScript, so current AngularJS applications developed using TypeScript have a better chance of a smoother transition to Angular.

AngularJS with TypeScript - Technical Prerequisites

To implement the application in this following section, the following prerequisites must be satisfied

- JavaScript
- HTML
- Knowledge of AngularJS

The Implementation

- To implement our application, we will be using Visual Studio Code and Node.js.
- After installing Node.js and Visual Studio Code, we will need TypeScript using the instructions in the linkhttps://www.typescriptlang.org/. On Windows OS and with Visual Studio IDEs, we can download project templates for TypeScript from this link.

➪ **Step 1 :** Since we need to create a project workspace for our AngularJS application, create a new folder on your hard disk. Open VS Code. Using File > Open Folder, open the folder in the Visual Studio Code to create all files in the same folder workspace.

➪ **Step 2 :** To install TypeScript using Node Package Manager (npm), open the Node.js Command Prompt and navigate to the Folder created in the Step 1. Run the following command from the Command Prompt

```
npm install TypeScript -g
```

This will install TypeScript globally so that we can use its transpiler (compiler) to convert TypeScript to JavaScript.

➪ **Step 3 :** Using VSCode IDE, add a new folder in the project of name views, scripts and styles. As mentioned in Step 2, navigate to the scripts folder in the Command prompt and run following commands

```
npm install angular
npm install angular-route
```

- This will install the AngularJS framework and routing library for the current project.
- Navigate to the styles folder and run the following command

```
npm install bootstrap
```

Module 3

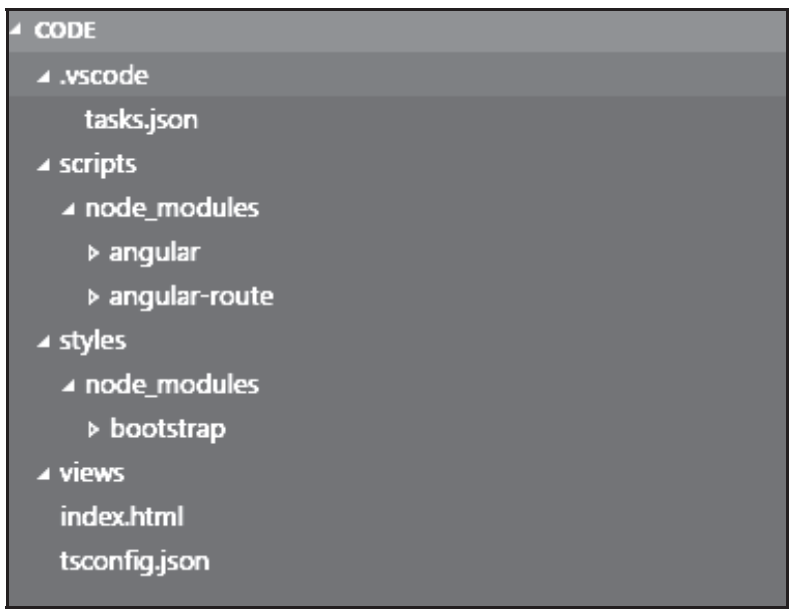
- In the project, add a new file with the name as **index.html**. This will be our view file to display UI. Since we need to define TypeScript project settings for the application, add a new file in the project of name tsconfig.json and add the following code in it

```
{
  "compilerOptions": {
    "target":"es5",
    "module":"amd",
    "sourceMap":true
  }
}
```

- We need to define a Task for the project so that it can invoke the TypeScript Transpiler to compile TypeScript into JavaScript. Press Ctrl+Shift+B in VSCode, this will show the **Configure Task Runner** option at the top of the IDE as shown in the following image.

```
// A task runner that calls the Typescript compiler (tsc) and
// Compiles a HelloWorld.ts program
{
    "version": "0.1.0",

    // The command is tsc. Assumes that tsc has been installed using npm install -g typescript
    "command": "tsc",

    // The command is a shell script
    "isShellCommand": true,

    // Show the output window only if unrecognized errors occur.
    "showOutput": "silent",

    // args is the HelloWorld program to compile.
    "args": ["HelloWorld.ts"],

    // use the standard tsc problem matcher to find compile problems
    // in the output.
    "problemMatcher": "$tsc"
}
```

- Replace the **"args"** value to blank. (Delete HelloWorld.ts).
- After Step 3, the project structure will be as shown below :

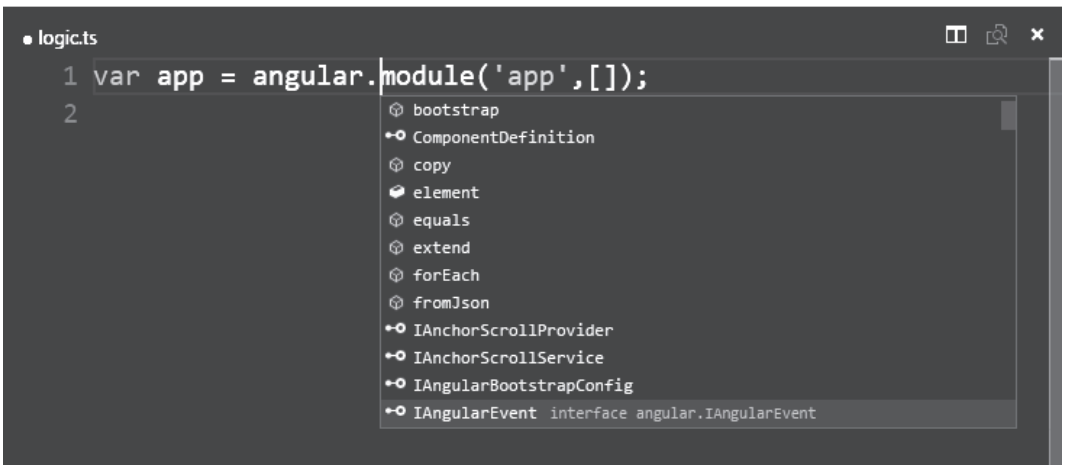


➪ **Step 4 :** In the project, we need intellisense for AngularJS. In VS Code we can implement it using Type Definition files (.d.ts). To install the required intellisense files run the following commands from the command prompt. (Note: In previous step we have navigated to the **scripts** folder, using cd.. navigate back to its parent.)

```
tsd install angular --resolve –save
```

This will install the angular.d.ts file. The --resolve switch will resolve all dependencies needed for angular.d.ts, in this case the dependency is jQuery.d.ts. The project will add **typings** folder containing angular.d.ts and jQuery.d.ts. It will also contain tsd.d.ts containing references for angular and jquery. The file tsd.json will be added to the project which defines required configurations for these files.

➪ **Step 5 :** In the project, add a new file of the name logic.ts. We will add necessary logic in this file. In VS Code, AngularJS intellisense will be displayed as shown here:



Module 3

➪ **Step 6 :** In the logic.ts file, we will add code for creating angular module and controller using TypeScript features line interfaces, classes, etc. The *interface* will define all methods and properties to be exposed to the view for databinding purpose. The angular Controller will be implemented using TypeScript class. This class will contain **properties, methods, constructor, etc.** The Constructor function of the controller class will be used to inject dependencies to the controller. Finally, we will register the TypeScript controller in our Angular module.

Adding Angular module

In the logic.ts, add the following line

```
var app = angular.module('app',[]);
```

- This is an Angular module, and has the same syntax as the JavaScript declaration of the angular module. It's the same because TypeScript is a superset of JavaScript.

Adding the interface

In the logic.ts add the following interface

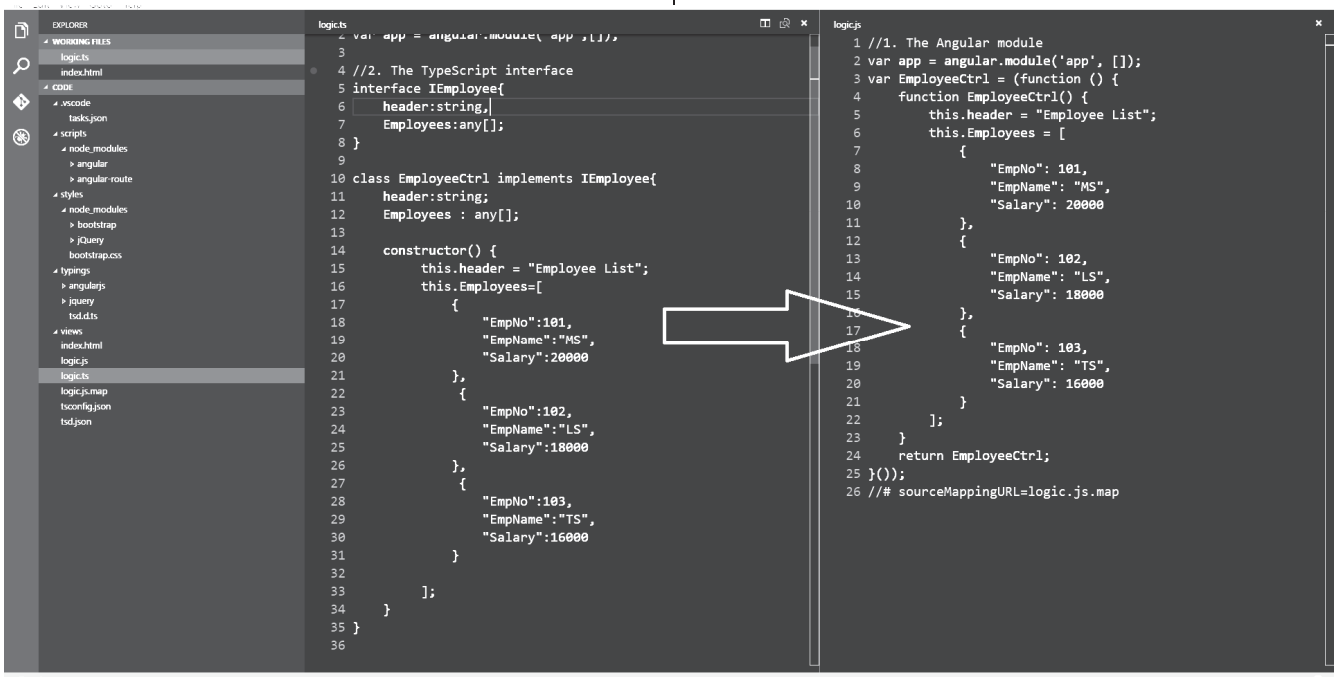
```
interface IEmployee{
   header:string,
   Employees:any[];
}
```

- In the TypeScript interface of name IEmployee, we have declared **header** property of type **string** and the **Employees** array of type **any.** The **any** type of TypeScript will allow us to store any type of data on Employees array varying from string to numeric or a JSON object.
- Adding the Controller class using TypeScript Class
- In logic.ts, add the following TypeScript code. This is the class which implements IEmployee interface and all its properties.

```
class EmployeeCtrl implements IEmployee{
   header:string;
   Employees : any[];

   constructor() {
      this.header = "Employee List";
      this.Employees=[
         {
            "EmpNo":101,
            "EmpName":"MS",
            "Salary":20000
         },
         {
            "EmpNo":102,
```

```
            "EmpName":"LS",
            "Salary":18000
        },
         {
            "EmpNo":103,
            "EmpName":"TS",
            "Salary":16000
        }

    ];
  }
}
```

- The class EmployeeCtrl implements IEmployee and implements all its properties. The Constructor function initializes the **header** and **Employees** array properties. Let's compile the TypeScript code.
- We have already added the **tasks.json** which is the Task Runner, this file contains **tsc** command for calling transpiler for TypeScript to generate JavaScript code. Press **Ctrl+Shift+B** to generate JavaScript.
- The following image shows the JavaScript code generated.



- As we know that we use function to create classes in JavaScript, on the same basis we have **EmployeeCtrl** function containing all the declaration of TypeScript class.
- The above code shows the **EmployeeCtrl** is in the global namespace in JavaScript, and this contains **EmployeeCtrl** function in it in IIFE (immediately invoked function expression).
- So here we need to be careful about the registration controller in the Angular module. To do so, we will add the registration at the bottom of the logic.ts file.
- Add the following line at the bottom of the logic.ts after the class code ends.

```
angular.module('app')
   .controller("EmployeeCtrl",EmployeeCtrl);
```

- Build the code using Ctrl+Shift+B, the logic.js file will be updated with EmployeeCtrl registered in the Angular Module as shown in the following image

```
    Employees : any[];

    constructor() {
        this.header = "Employee List";
        this.Employees=[
            {
                "EmpNo":101,
                "EmpName":"MS",
                "Salary":20000
            },
             {
                "EmpNo":102,
                "EmpName":"LS",
                "Salary":18000
            },
             {
                "EmpNo":103,
                "EmpName":"TS",
                "Salary":16000
            }

        ];
    }
}

angular.module('app')
    .controller("EmployeeCtrl",EmployeeCtrl);
```

```
1  //1. The Angular module
2  var app = angular.module('app', []);
3  var EmployeeCtrl = (function () {
4      function EmployeeCtrl() {
5          this.header = "Employee List";
6          this.Employees = [
7              {
8                  "EmpNo": 101,
9                  "EmpName": "MS",
10                 "Salary": 20000
11             },
12             {
13                 "EmpNo": 102,
14                 "EmpName": "LS",
15                 "Salary": 18000
16             },
17             {
18                 "EmpNo": 103,
19                 "EmpName": "TS",
20                 "Salary": 16000
21             }
22         ];
23     }
24     return EmployeeCtrl;
25 }());
26 angular.module('app')
27     .controller("EmployeeCtrl", EmployeeCtrl);
28 //# sourceMappingURL=logic.js.map
```


➪ **Step 6 :** To create a View, add the following markup in the **index.html**

```
<title></title>
<link rel=""stylesheet"" href=""/styles/bootstrap.css"">
<h1>{{empVM.header}}</h1>
<tableclass=""table" table-bordered="">
<thead>
<tr>
<td>EmpNo</td>
<td>EmpName</td>
<td>Salary</td>
</tr>
</thead>
<tbody>
<tr ng-repeat=""Emp" in="">
<td>{{Emp.EmpNo}}</td>
<td>{{Emp.EmpName}}</td>
<td>{{Emp.Salary}}</td>
</tr>
</tbody>
</table>
```

```
<a
href="/scripts/node_modules/angular/angular.min.js">/script
s/node_modules/angular/angular.min.js
</a>
 <a href="http://logic.js">http://logic.js</a>
```

The above html markup shows the Angular directive with databinding.

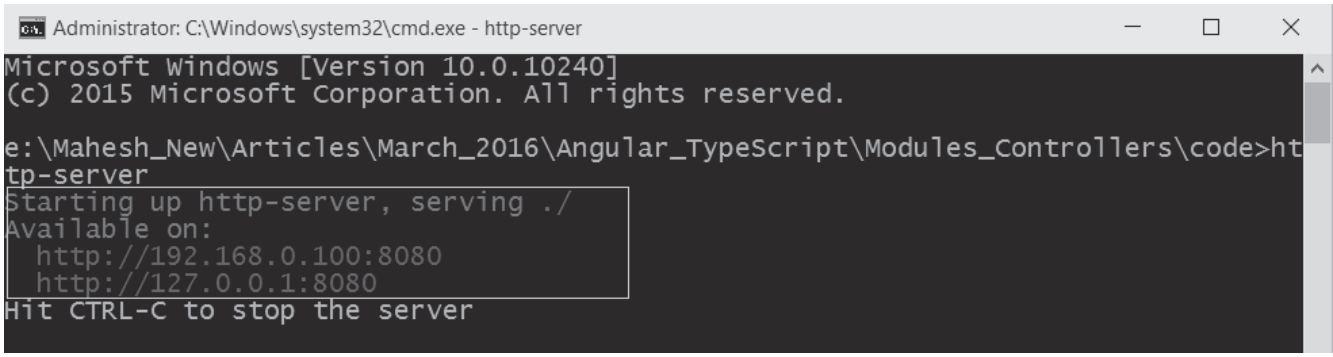
➪ **Step 7 :** To run the application using index.html file, we will install the *http server* node package manager. Right click on index.html and select option *Open in Command Prompt***.** This will open the command prompt. Run the following command from the Command prompt

```
npm  install -g http-server
```

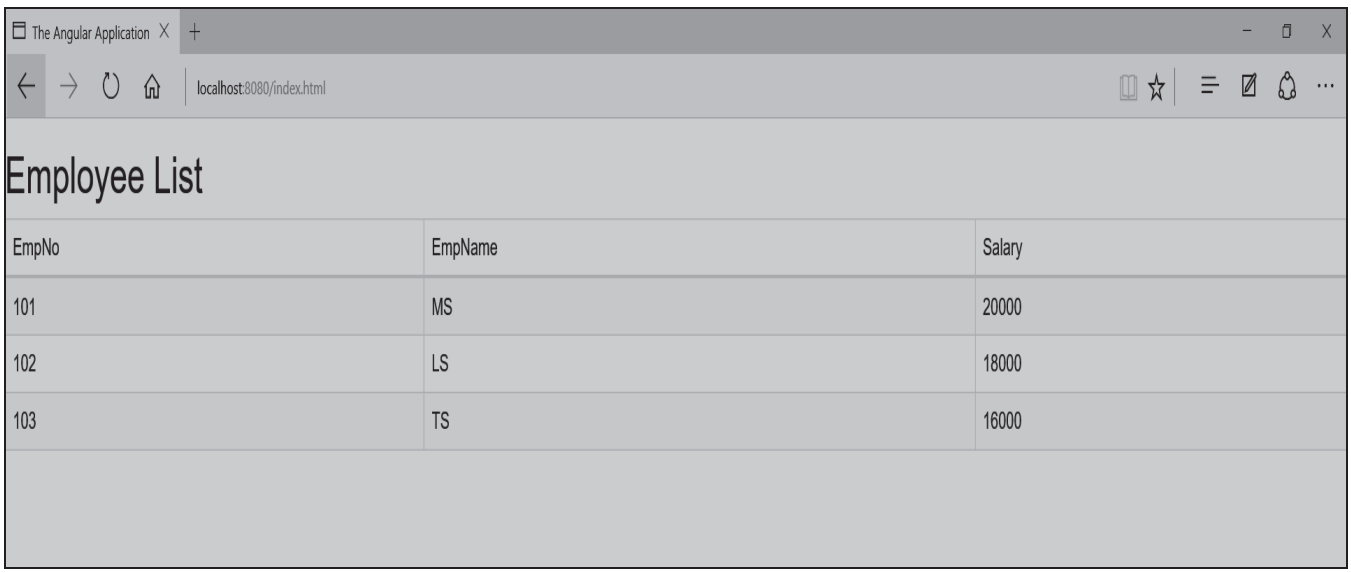
- This will install the http server. This is a simple zero configuration command line http server and does good in a development and testing environment.
- Run the following command from the same command prompt

```
http-server
```

- to start the http-server on port 8080 as shown in the following image



- Open the Browser and enter the following address
  http://localhost:8080/index.html
- This will show Employee Details as shown in the following image



➪ **Step 8 :** We will modify the IEmployee interface by adding the following method in it (highlighted)

```
interface IEmployee{
   header:string,
   Employees:any[];
   addEmployee():void;
}
```

We will change the Employee class by implementing the addEmploye() function as shown in the following code. (highlighted in bold)

```
//1. The Angular module
var app = angular.module('app',[]);
 class Employee{
   EmpNo:number;
   EmpName:string;
   Salary:number;
}
//2. The TypeScript interface
interface IEmployee{
   header:string,
   Employees:any[];
   addEmployee():void;
}

class EmployeeCtrl implements IEmployee{
   header:string;
   Employees : any[];
   Emp:Employee;

   constructor() {
      this.header = "Employee List";
      this.Employees=[
         {
            "EmpNo":101,
```

```
            "EmpName":"MS",
            "Salary":20000
          },
          {
            "EmpNo":102,
            "EmpName":"LS",
            "Salary":18000
          },
          {
            "EmpNo":103,
            "EmpName":"TS",
            "Salary":16000
          }
        ];
        this.Emp = new Employee();
        this.Emp.EmpNo = 0;
        this.Emp.EmpName = "";
        this.Emp.Salary = 0;
    }
    addEmployee(){
      this.Employees.push(this.Emp);
    }
}
angular.module('app')
    .controller("EmployeeCtrl",EmployeeCtrl);
```

The above code adds a new TypeScript class of name **Employee** with properties in it. The **EmployeeCtrl** class declares the **Emp** property of the type Employee. The constructor of this class instantiates the **Emp** set toits default values. The EmployeeCtrl class implements **addEmployee** method, this pushes Employee record in Employees array.

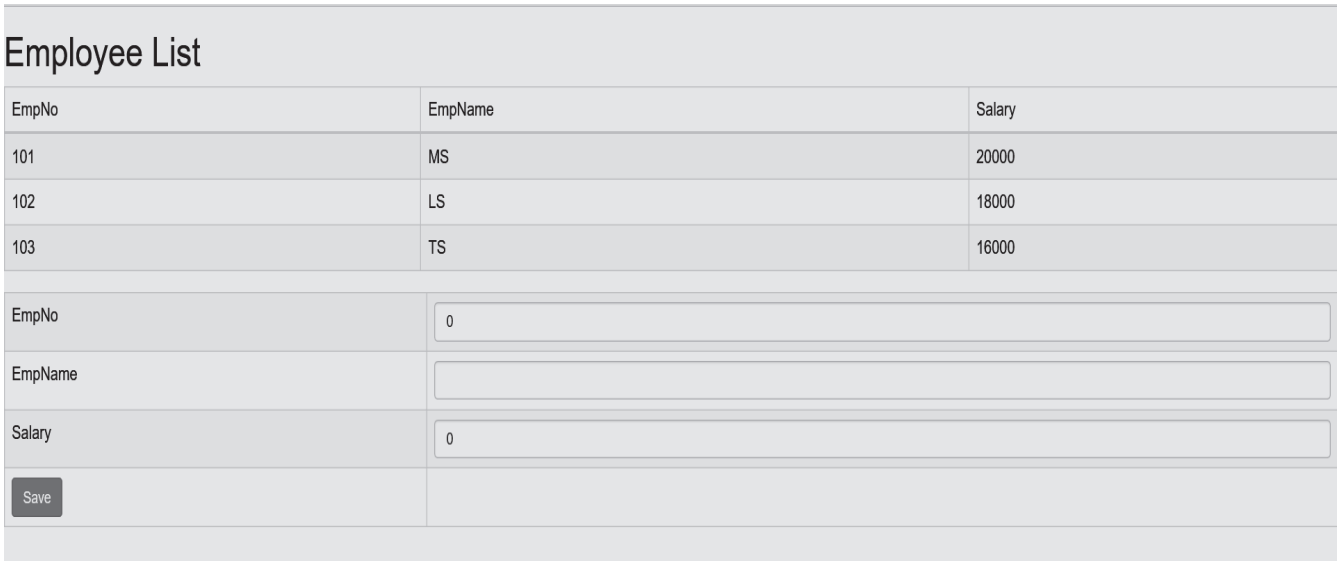
➪ **Step 9 :** Modify the index.html as shown in the following code (highlighted)

```
<title></title>
<link rel=""stylesheet"" href=""/styles/bootstrap.css"">
<h1>{{empVM.header}}</h1>
<table class=""table" table-bordered="">
<thead>
<tr>          <td>EmpNo</td>
<td>EmpName</td>
<td>Salary</td>              </tr>
</thead>
<tbody>
<tr ng-repeat=""Emp" in="">
<td>{{Emp.EmpNo}}</td>
<td>{{Emp.EmpName}}</td>
<td>{{Emp.Salary}}</td>
</tr>
</tbody>
</table>
 <table class=""table" table-bordered="">
<tbody>
<tr>
<td>EmpNo</td>
<td><input type=""text"" ng-
model=""empVM.Emp.EmpNo"" class="”form-control"">
</td>
</tr>
<tr>
<td>EmpName</td>
<td><input type=""text"" ng-
model=""empVM.Emp.EmpName"" class="”form-control"">
</td>
</tr>
<tr>
<td>Salary</td>
<td><input type=""text"" ng-
model=""empVM.Emp.Salary"" class="”form-control"">
</td>
</tr>
<tr>
<td><input type=""button"" value=""Save"" class=""btn"
ng-click=""empVM.addEmployee()"">
</td>
<td> </td>
</tr>
</tbody>
</table>
<a
href="/scripts/node_modules/angular/angular.min.js">/script
s/node_modules/angular/angular.min.js
</a>
<a href="http://logic.js">http://logic.js</a>
```

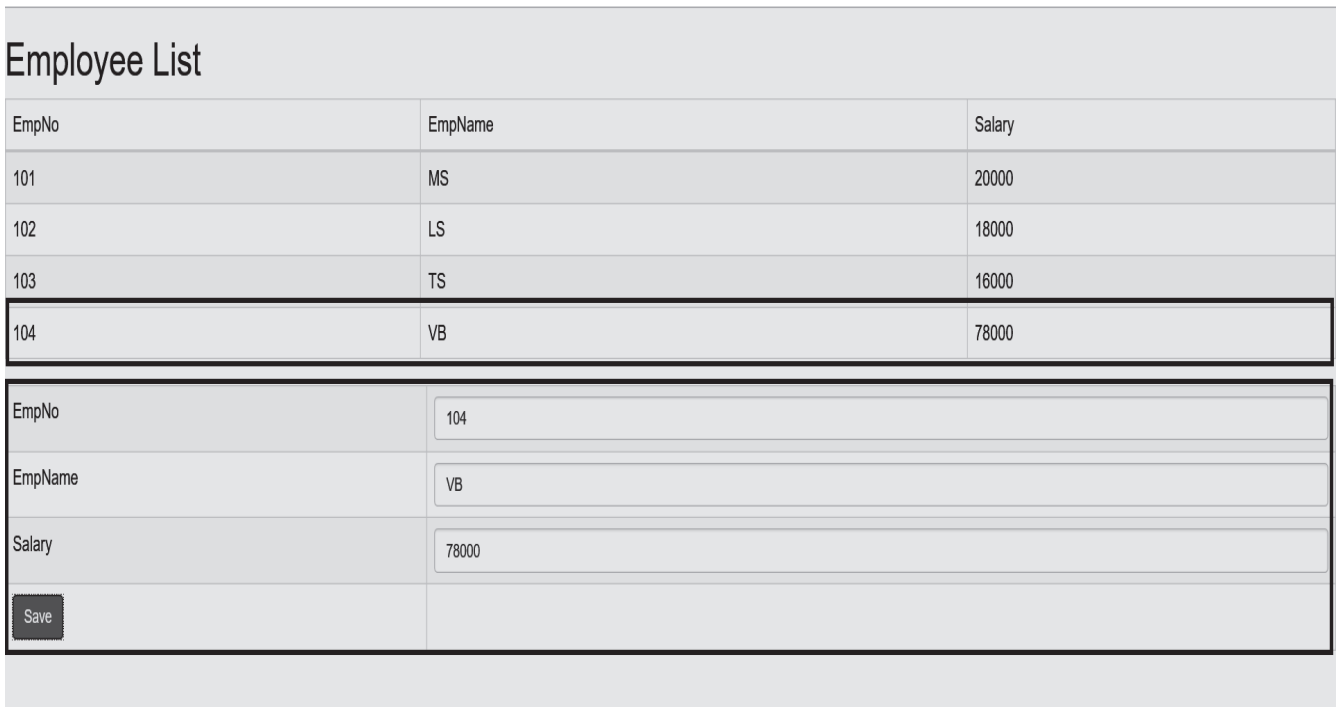
This modified code has text elements with data bound on them using **ng-model** directive.

Module 3

➪ **Step 10 :** Run the http server as explained in Step 7, and view the index.html in the browser, the page will be loaded as shown in the following image

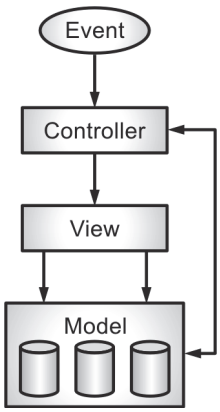


Add values in text boxes and click on **Save** button, the following result will be displayed



3.17 ANGULARJS MVC MODEL (SELF-LEARNING TOPIC)

- MVC stands for Model View Controller. It is a software design pattern for developing web applications.
- It is very popular because it isolates the application logic from the user interface layer and supports separation of concerns.



(1C4)**Fig. 3.17.1 : MVC Architecture**

The MVC pattern is made up of the following three parts:

(1) Model: It is responsible for managing application data. It responds to the requests from view and to the instructions from controller to update itself.

(2) View : It is responsible for displaying all data or only a portion of data to the users. It also specifies the data in a particular format triggered by the controller's decision to present the data. They are script-based template systems such as JSP, ASP, PHP and very easy to integrate with AJAX technology.

(3) Controller : It is responsible to control the relation between models and views. It responds to user input and performs interactions on the data model objects. The controller receives input, validates it, and then performs business operations that modify the state of the data model.

3.18 ANGULARJS HTML DOM (SELF-LEARNING TOPIC)

In AngularJS, some directives can be used to bind application data to attributes of HTML DOM elements.

These directives are:

Directive	Description
ng-disabled	It disables a given control.
ng-show	It shows a given control.
ng-hide	It hides a given control.
ng-click	It represents an AngularJS click event.

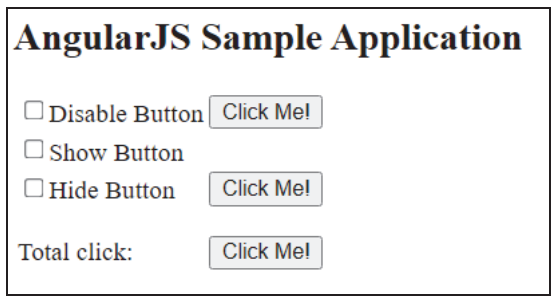
Example

```
<!DOCTYPE html>
<html>
<head>
<title>AngularJS HTML DOM</title>
</head>
<body>
<h2>AngularJS Sample Application</h2>
<div ng-app = "">
<table border = "0">
<tr>
<td><input type = "checkbox" ng-model =
"enableDisableButton">Disable Button</td>
<td><button ng-disabled = "enableDisableButton">Click
Me!</button></td>
</tr>
<tr>
<td><input type = "checkbox" ng-model =
"showHide1">Show Button</td>
<td><button ng-show = "showHide1">Click
Me!</button></td>
</tr>
<tr>
<td><input type = "checkbox" ng-model =
"showHide2">Hide Button</td>
```

Module 3

```
<td><button ng-hide = "showHide2">Click
Me!</button></td>
</tr>
<tr>
<td><p>Total click: {{ clickCounter }}</p></td>
<td><button ng-click = "clickCounter = clickCounter +
1">Click Me!</button></td>
</tr>
</table>
</div>
<script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular
.min.js"></script>
</body>
</html>
```

Output



Descriptive Questions

Q. 1 List the features of AngularJS. Also state its advantages and disadvantages.

Q. 2 State the need of AngularJS in real websites.

Q. 3 Explain modules in AngularJS with suitable example.

Q. 4 Explain the following directives in AngularJS with example.

(a) ng-app (b) ng-init

(c) ng-model (d) ng-bind

(e) ng-show

Q. 5 Illustrate the use of expressions in AngularJS with suitable example.

Q. 6 Discuss the following AngularJS Filters with suitable example.

(a) number (b) currency

(c) date (d) orderBy

Q. 7 Write short note on:

(a) AngularJS Controller

(b) AngularJS Scope

Q. 8 Explain in detail AngularJS Dependency Injection.

Q. 9 Explain AngularJS $http service in detail with its get() and post() methods.

Q. 10 Discuss various built-in helper functions in AngularJS.

Chapter Ends…

❑❑❑

MODULE 4

CHAPTER 4

MongoDB and Building REST API using MongoDB

University Prescribed Syllabus w.e.f Academic Year 2021-2022

MongoDB : Understanding MongoDB, MongoDB Data Types, Administering User Accounts, Configuring Access Control, Adding the MongoDB Driver to Node.js, Connecting to MongoDB from Node.js, Accessing and Manipulating Databases, Manipulating MongoDB Documents from Node.js, Accessing MongoDB from Node.js, Using Mongoose for Structured Schema and Validation.

REST API : Examining the rules of REST APIs, Evaluating API patterns, Handling typical CRUD functions (create, read, update, delete), Using Express and Mongoose to interact with MongoDB, Testing API endpoints

Self-learning Topics : MongoDB vs SQL DB

4.1 MONGODB

4.1.1 Understanding MongoDB

- MongoDB is an open source, document-oriented, NoSQL database that uses flexible documents instead of tables and rows to process and store various forms of data.
- MongoDB is a document database designed for ease of development and scaling.
- Dwight Merriman, Eliot Horowitz, and Kevin Ryan created MongoDB in 2007.
- Instead of using tables and rows as in the traditional relational databases, MongoDB makes use of collections and documents.
- Collections contain sets of documents and function which is the equivalent of relational database tables.
- Documents consist of key-value pairs which are the basic unit of data in MongoDB.
- It offers developers the flexibility to work with evolving data models.
- MongoDB's document-based architecture allows for embedded document arrays and the representation of complex hierarchical relationships in a single record.
- It is also schema free which means that the keys defined in the document are not fixed, as a result massive data migrations can be rolled out.
- Since MongoDB employs a dynamic schema design, users have unparalleled flexibility when creating data records, querying document collections through MongoDB aggregation and analyzing large amounts of information.

4.1.2 Why MongoDB?

The following are a few of the reasons why you should start using MongoDB.

- **Flexibility :** MongoDB's notion of documents that can contain sub-documents nested in complex hierarchies is really expressive and flexible.
- **Flexible query model :** A user can selectively index some part of a document or a query based on attribute values regular expressions or ranges.
- **Native aggregation :** Native aggregation allows its users to extract and transform data from MongoDB and either load them in a new format or export it from MongoDB to other data sources, it makes it extremely compatible.
- **Document-oriented** : Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.
- **Indexing** : Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
- **Replication :** MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.
- **Load balancing :** MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

4.1.3 Features of MongoDB

1. **Support Ad-hoc queries for optimized, real-time analytics**

   MongoDB supports field queries, range queries, and regular expression searches. Queries can return specific fields and also account for user-defined functions.

2. **Indexing appropriately for better query executions**

   MongoDB offers a broad range of indices and features with language-specific sort orders that support complex access patterns to datasets.

3. **Replication for better data availability and stability**

- In MongoDB, a primary server or node accepts all write operations and applies those same operations across secondary servers, replicating the data.
- If the primary server should ever experience a critical failure, any one of the secondary servers can be elected to become the new primary node.
- And if the former primary node comes back online, it does so as a secondary server for the new primary node.

4. **Sharding**

- Like replication via replication sets, sharding in MongoDB allows for much greater horizontal scalability.
- Horizontal scaling means that each shard in every

cluster houses a portion of the dataset in question, essentially functioning as a separate database.

- The collection of distributed server shards forms a single, comprehensive database much better suited to handling the needs of a popular, growing application with zero downtime.
- All operations in a sharding environment are handled through a lightweight process called mongos. Mongos can direct queries to the correct shard based on the shard key. Naturally, proper sharding also contributes significantly to better load balancing.

5. **Load balancing :** MongoDB supports large-scale load balancing via horizontal scaling features such as replication and sharding.
6. **Scalability:** The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database
7. **Aggregation Framework**
   MongoDB offers an extract transform and load framework that eliminates the need for complex data pipelines.
8. **Security features :** Both authentication and authorization are taken into account in MongoDB.
9. **JSON :** JSON is widely used across the web for front-end and API communication and as such it's easier when the database is also compatible with the same protocol.
10. **MapReduce :** MapReduce is an excellent tool to build data pipelines and MongoDB uses MapReduce readily.

4.1.4 Working of MongoDB

- The MongoDB environment provides you with a server on which you can launch MongoDB and create multiple databases. The data is stored in collections and documents due to the NoSQL database. As a result, the database, collection, and documents are linked as follows:

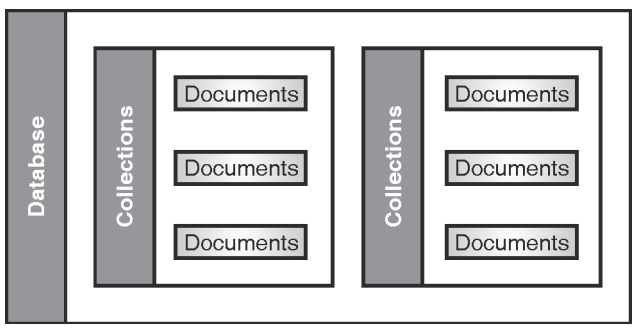


Fig. 4.1.1 : Working of MongoDB

- Collections in MongoDB database are similar to tables in MYSQL database. You are allowed to create multiple databases and multiple collections.
- Now inside of the collection we have documents. These documents contain the data we want to store in the MongoDB database and a single collection can contain multiple documents and you are schema-less means it is not necessary that one document is similar to another.
- The documents are created using the fields. Fields are key-value pairs in the documents, it is just like columns in the relation database. The value of the fields can be of any BSON data types like double, string, boolean, etc.
- The data stored in the MongoDB is in the format of BSON (Binary representation of JSON) documents. In the backend, the MongoDB server converts the JSON data into a binary form that is known as BSON and this BSON is stored and queried more efficiently.
- In MongoDB documents, you are allowed to store nested data. This nesting of data allows you to create complex relations between data and store them in the same document which makes the working and fetching of data extremely efficient as compared to SQL.

4.1.5 Applications of MongoDB

1. Internet of things
2. Mobile and Social Infrastructure
3. Real-time analysis
4. Personalization
5. Catalog management
6. Content management
7. Mainframe offloading
8. Enterprise Data Warehouse

Module 4

4.2 DOCUMENT DATABASE

- A record in MongoDB is a document, which is a data structure composed of field and value pairs.
- A document database is a NoSQL data store designed to store and query data in the form of JSON-like documents. Document databases store data as documents along with their metadata. The document is stored as a key/value pair, where the key is the document's unique identifier. Document databases, as opposed to relational databases, are faster to load, access, and parse.
- Document databases are designed to store large

documents in a key/value store that are easy to search and access. The entire document is read into a memory object that is easy to read and present.

☞ **Key characteristics of document databases**

1. Document DBMSs are NoSQL databases.
2. Document DBMSs use key/value to store and access documents data.
3. Document DBMSs have a flexible schema that can be different for each document. For example, one document can be an Author profile, while other document can be a blog.
4. Common examples of document DBMS include JSON, XML docs, Catalogs, serialized PDFs and Excel docs, Profile data, and serialized objects.

☞ **Example of Document Database**

Here is a document that stores a book data. As you can see from this document, it's a JSON document that has tags and values that defines a book including year published, book title, author, release date, publisher, and price.

```
[{
   "year": 2021,
   "title": "Internet Programming",
   "info": {
      "author": "Shraddha More",
      "release_date": "2021-05-22",
      "publisher": "Tech-Neo",
      "price": "230",
      "image_url": "IP.jpg"
   }
}, {
   "year": 2021,
   "title": "Python Programming",
   "info": {
      "author": "Sayali More",
      "release_date": "2021-09-13",
      "publisher": "Willey",
      "price": "250",
      "image_url": "Python.jpg"
   }
}]
```

☞ **The Advantages of using Documents are :**

1. Documents (i.e. objects) correspond to native data types in many programming languages.
2. Embedded documents and arrays reduce need for expensive joins.
3. Dynamic schema supports fluent polymorphism.

⏭ 4.3 DATATYPES IN MONGODB

The documents in MongoDB are stored in BSON, which is the binary encoded format of JSON, and we can make remote procedure calls in MongoDB using BSON. The BSON data format accepts a wide range of data types. The following are the MongoDB data types:

1. **Integer :** Integer is a data type that is used for storing a numerical value, i.e., integers as you can save in other programming languages. We can store integer data type in two forms 32-bit signed integer and 64-bit signed integer.

   Example: db.TestCollection.insert({"Integer example": 62})

2. **String :** String is one of the most frequently implemented data type for storing the data.BSON strings are of UTF-8.Drivers for each programming language convert from the string format of the language to UTF-8 while serializing and de-serializing BSON.

   Example: db.TestCollection.insert({"string data type" : "This is a sample message."})

3. **Object :** Object data type stores embedded documents. If a document contains another document in the form of the key-value pair then such type of document is known as an embedded document.

   Example:

```
varembeddedObject={"English" : 94, "ComputerSc." : 96,
"Maths" : 80,
"GeneralSc." : 85}
db.TestCollection.insert({"Object data type" : "This is Object",
"Marks" :embeddedObject})
```

4. **Double :** The double data type is used to store floating point values.

```
Example: db.TestCollection.insert({"double data type":
3.1415})
```

5. **Array :** The Array is the set of values. It can store the same or different data types values in it. In MongoDB, the array is created using square brackets([]).

   Example

```
var degrees = ["BCA", "BS", "MCA"]
db.TestCollection.insert({" Array Example" : " Here is an
example of array",
" Qualification" : degrees})
```

6. **Boolean :** The boolean data type is used to store either true or false.

```
Example: db.TestCollection.insert({"Nationality Indian":
true})
```

7. Date

- Date data type stores date. It is a 64-bit integer which represents the number of milliseconds. BSON data type generally supports UTC datetime and it is signed.
- There are various methods to return date, it can be returned either as a string or as a date object. Some method for the date are as follows:

  Date(): It returns the current date in string format.

  new Date() : Returns a date object. Uses the ISODate() wrapper.

  newISODate() : It also returns a date object. Uses the ISODate() wrapper.

  Example :

```
var date=new Date()
var date2=ISODate()
var month=date2.getMonth()
db.TestCollection.insert({"Date":date, "Date2":date2,
"Month":month}
```

8. Null : This MongoDB data types stores a null value in it.

```
Example: db.TestCollection.insert({" EmailID ": null})
```

9. Regular Expression : These MongoDB data types stores regular expressions in MongoDB. It maps directly to JavaScript RegExp.

Example

```
var regular=new RegExp("%Shraddha")
db.TestCollection.insert({Regular Expression: regular})
```

10. JavaScript : This datatype is used to store JavaScript code into the document without the scope.

```
Example: db.TestCollection.insert({Code: "function() {var x;
x=5}, scope:{}})
```

11. Symbol : This data type similar to the string data type. It is generally not supported by a mongo shell, but if the shell gets a symbol from the database, then it converts this type into a string type.

Example :

```
varsymb= "shr22#"
db.TestCollection.insert({Symbol: symb})
```

12. JavaScript with Scope : This MongoDB data type store JavaScript data with a scope.

```
Example: db.TestCollection.insert({Code: "function() {var x;
x=5}, scope:["Object"] })
```

13. Min & Max key : Min key compares the value of the lowest BSON element and Max key compares the value against the highest BSON element. Both are internal data types.

```
Example:db.TestCollection.insert([{a:
MinKey},{a:MaxKey}])
```

14. Timestamp : This data type is used to store a timestamp. It is useful when we modify our data to keep a record and the value of this data type is 64-bit. The value of the timestamp data type is always unique.

Example :

```
varts= new Timestamp()
db.TestCollection.insert({Stamp: ts})
```

15. Binary Data : This data type is used to store binary data.

```
Example: db.TestCollection.insert({binaryValue:
"110011001"})
```

16. Undefined : This data type stores the undefined values.

```
Example: db.TestCollection.insert({duration: undefined})
```

17. ObjectId

- Whenever we create a new document in the collection MongoDB automatically creates a unique object id for that document (if the document does not have it). There is an_id field in MongoDB for each document. The data which is stored in Id is of hexadecimal format and the length of the id is 12 bytes which consist:
- 4-bytes for Timestamp value, 5-bytes for Random values. i.e., 3-bytes for machine Id and 2-bytes for process Id and 3- bytes for Counter.
- **Example :** In the following example, when we insert a new document it creates a new unique object id for it.

```
> db.book.insertOne({Book:{name: "C in depth",writer:
"Aaksh"}
{    "acknowledged" : true,
     "insertedId" : ObjectId("601af71f6fd54aa34c9c6df9")
}
> db.bool.find().pretty()
> db.book.find().pretty()
{
     "_id" : ObjectId("601af71f6fd54aa34c9c6df9"),
     Book : {
          "name" : "C in dpeth" ,
          "writer" : "Aaksh"
     }
}
>
```

Module 4

You can also create your own id field, but make sure that the value of that id field must be unique.

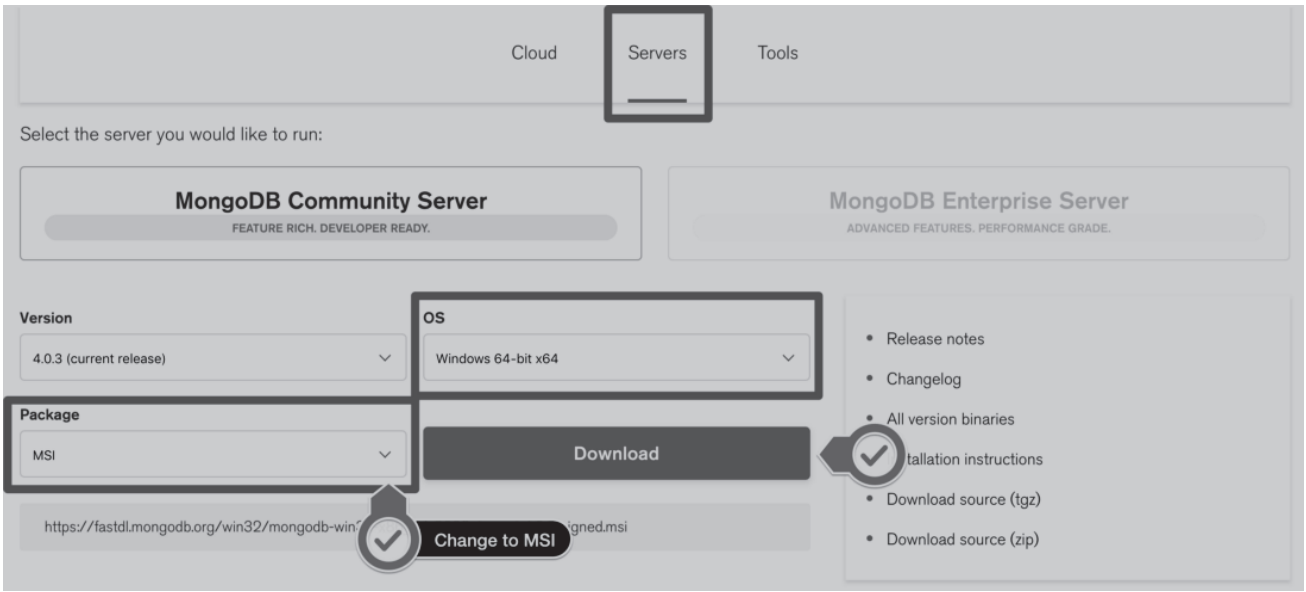
Example :

```
var id=ObjectId()
db.TestCollection.insert({_id:id})
```

4.4 INSTALLATION AND CONFIGURATION OF MONGODB

- **Step 1 :** Download the MongoDB MSI Installer Package.

  Navigate to the official MongoDB website (https://www.mongodb.com/) and download the current version of MongoDB. Make sure you select MSI as the package you want to download.

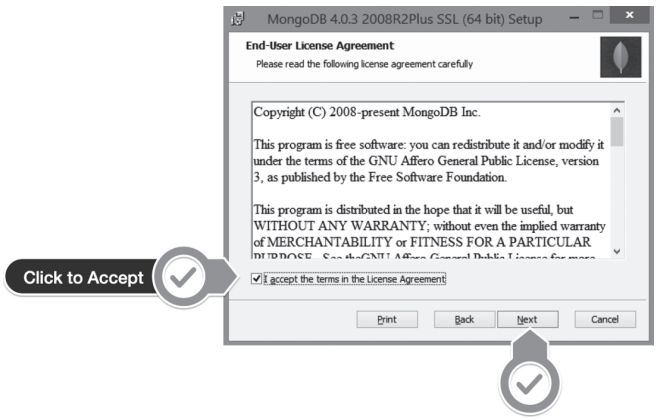


- **Step 2 :** Install MongoDB with the Installation Wizard.

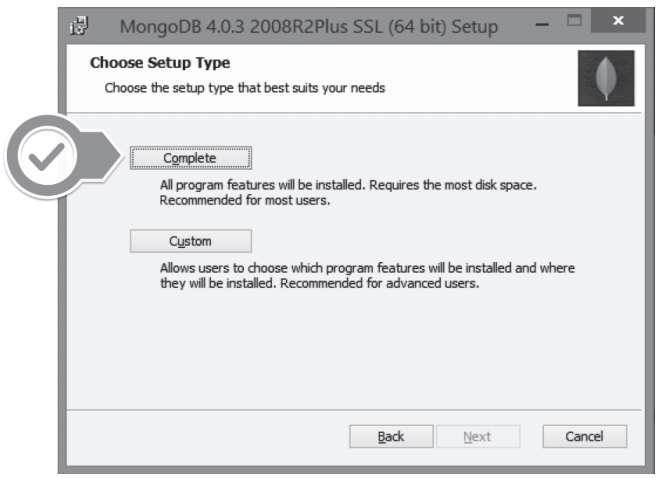
  (A) Make sure you are logged in as a user with Admin privileges. Then navigate to your downloads folder and double click on the .msi package you just downloaded. This will launch the installation wizard.

  (B) Click Next to start installation.

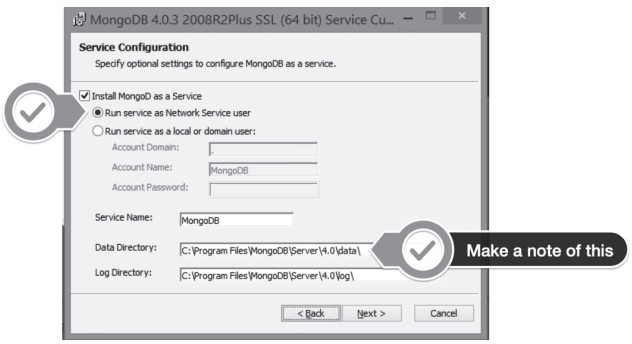
(C) Accept the licence agreement then click Next.



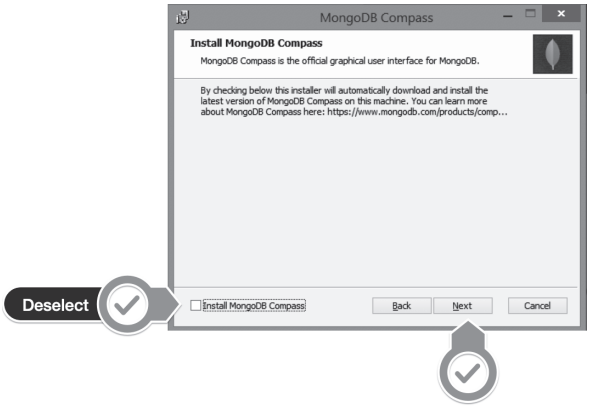
(D) Select the Complete setup.



(E) Select "Run service as Network Service user" and make a note of the data directory.

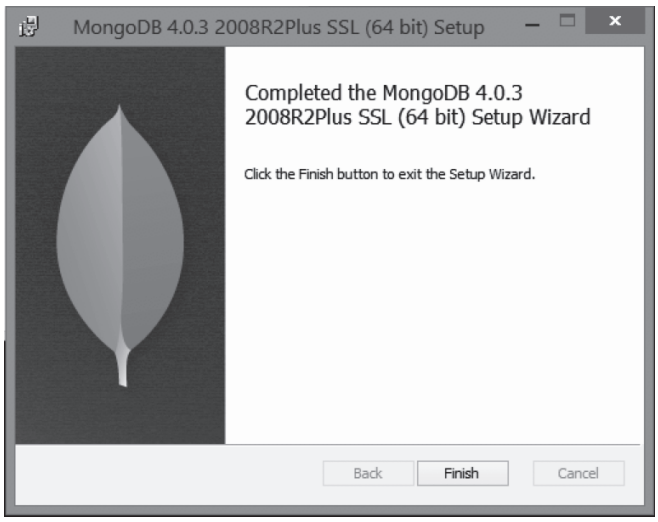


(F) We won't need Mongo Compass, so deselect it and click Next.



(G) Click Install to begin installation.

(H) Hit Finish to complete installation.



Module 4

- **Step 3:** Create the directory where MongoDB will store its files.
  - (A) Navigate to the **C Drive** on your computer using Explorer and create a new folder called **data** here.
  - (B) Inside the **data** folder you just created, create another folder called **db**.
- **Step 4:** Configuration through Mongo Shell
  - (A) Go to the local disk C and get into "Program Files". There you'll find a folder named "MongoDB".
  - (B) Open it and you'll find a folder named "bin" i.e. binaries folder. Copy the path, as given in the snippet path i.e. C:\Program Files\MongoDB\Server\4.0\bin.
  - (C) Open Settings and search "Path". The two options given below would pop up in front of you:
    1. Edit environment variable of your account
    2. Edit the system environment variable.
  - (D) Click on "Edit the system environment variable" and then click on "Environment Variables".
  - (E) In the Environment variable, you'll see the path as given in the snippet. Click "Path" and then press "Edit".
  - (F) In the Path given at your system, delete the previous ones and add new "The copied path" from binaries, and click "OK".
  - (G) Open Command prompts and type "mongod" to start the service.
  - (H) Open the command prompt and just type in mongo and press enter.

```
Command Prompt - mongo
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\chinmayee.deshpande.BLRSIMPLILEARN>mongo
MongoDB shell version v4.4.4
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("ac702fee-69c0-41d5-b573-5e71b5046ad5") }
MongoDB server version: 4.4.4
---
The server generated these startup warnings when booting:
        2021-03-29T11:00:31.877+05:30: Access control is not enabled for the database. Read and write access to data
 configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

4.5 MONGODB CRUD OPERATIONS

CRUD operations create, read, update, and delete documents.

- **Step 1 :** Start Mongo Server and Mongo Shell

Now, first, start the mongo server using the following command.

mongod

Also, start the mongo shell using the following command.

mongo

- **Step 2 :** Create a database and collection.

Now, switch to an existing database or create a new **Mongodb** database using the following command.

use crud

Now, create a collection "operate" using the following command.

db.createCollection("operate")

MongoDB stores the documents in the collections. Collections are somewhat the same as tables in relational databases. If the collection does not exist, then MongoDB creates the collection for you when you first store data for that collection.

4.5.1 Create Operations

- Create or insert operations add new documents to a collection. If the collection does not exist, create operations also create the collection.
- You can insert a single document or multiple documents in a single operation.
- The MongoDB shell provides the following methods to insert documents into a collection:
  - To insert a single document, use **db.collection.insertOne()**.
  - To insert multiple documents, use **db.collection.insertMany()**.

Example

1. Insert a single document in MongoDB.

```
db.operate.insert({
enrollno: "110470116021",
name: "KrunalLathiya",
college: "VVP Engineering College",
course: {
courseName: "BE IT",
duration: "4 Years"
   },
address: {
city: "Rajkot",
state: "Gujarat",
country: "India"
   }
})
```

- It will insert a document in **MongoDB.** insertOne() returns a document that includes the newly inserted document's _id field value. You can check it by fetching that document using this query.

db.operate.find().pretty()

```
> db.operate.find().pretty()
{
        "_id" : ObjectId("5bd6d9fd19d7f970797108b3"),
        "enrollno" : "110470116021",
        "name" : "Krunal Lathiya",
        "college" : "VVP Engineering College",
        "course" : {
                "courseName" : "BE IT",
                "duration" : "4 Years"
        },
        "address" : {
                "city" : "Rajkot",
                "state" : "Gujarat",
                "country" : "India"
        }
}
```

2. Insert Multiple Documents in MongoDB

- If the documents do not specify an _id field, MongoDB adds the _id field with an ObjectId value to each document.

Module 4

```
db.operate.insertMany([{
enrollno: "1104701160 21",
name: "KrunalLathiya",
college: "VVP Engineering College",
course: {
courseName: "BE IT",
duration: "4 Years"
   },
address: {
city: "Rajkot",
state: "Gujarat",
country: "India"
   }
},{
enrollno: "1104701160 22",
name: "RushikeshVekariya",
college: "VVP Engineering College",
course: {
courseName: "BE IT",
duration: "4 Years"
   },
address: {
city: "Rajkot",
state: "Gujarat",
country: "India"
   }
}])
```

- insertMany() returns a document that includes the newly inserted documents' _id field values.

4.5.2 Read Operations

- Read operations retrieves the documents from the collection, i.e., queries the collection for the documents. Use the db.collection.find() method in the MongoDB Shell to query documents in a collection.
- To read all documents in the collection, pass an empty document as the query filter parameter to the find method. The query filter parameter determines the select criteria.
- **Example :** db.operate.find().pretty()

```
> db.operate.find().pretty()
{
        "_id" : ObjectId("5bd6d9fd19d7f970797108b3"),
        "enrollno" : "1104701160 21",
        "name" : "Krunal Lathiya",
        "college" : "VVP Engineering College",
        "course" : {
                "courseName" : "BE IT",
                "duration" : "4 Years"
        },
        "address" : {
                "city" : "Rajkot",
                "state" : "Gujarat",
                "country" : "India"
        }
}
{
        "_id" : ObjectId("5bd719cb19d7f970797108b4"),
        "enrollno" : "1104701160 21",
        "name" : "Krunal Lathiya",
        "college" : "VVP Engineering College",
        "course" : {
                "courseName" : "BE IT",
                "duration" : "4 Years"
        },
        "address" : {
                "city" : "Rajkot",
                "state" : "Gujarat",
                "country" : "India"
        }
}
{
        "_id" : ObjectId("5bd719cb19d7f970797108b5"),
        "enrollno" : "1104701160 22",
        "name" : "Rushikesh Vekariya",
        "college" : "VVP Engineering College",
        "course" : {
                "courseName" : "BE IT",
                "duration" : "4 Years"
        },
        "address" : {
                "city" : "Rajkot",
                "state" : "Gujarat",
                "country" : "India"
        }
}
```

- You can specify the query filters or criteria that identify the documents to return.

db.**operate.find().**limit(2)

Specify Equality Condition

To select documents which match an equality condition, specify the condition as a <field>:<value> pair in the query filter document.

Example

To return all movies where the title equals Titanic from the sample_mflix.movies collection:

```
usesample_mflix
db.movies.find( { "title": "Titanic" } )
```

☞ **Specify Conditions Using Query Operators**

Use query operators in a query filter document to perform more complex comparisons and evaluations. Query operators in a query filter document have the following form:

{ <field1>: { <operator1>: <value1> }, ... }

Example:

To return all movies where the title equals Titanic from the sample_mflix.movies collection:

```
usesample_mflix
db.movies.find( { "title": "Titanic" } )
```

☞ **Specify Conditions Using Query Operators**

- Use query operators in a query filter document to perform more complex comparisons and evaluations. Query operators in a query filter document have the following form:

  { <field1>: { <operator1>: <value1> }, ... }

Example

To return all movies from the sample_mflix.movies collection which are either rated PG or PG-13:

```
usesample_mflix
db.movies.find( { rated: { $in: [ "PG", "PG-13" ] } } )
```

☞ **Specify Logical Operators (AND / OR)**

- A compound query can specify conditions for more than one field in the collection's documents. Implicitly, a logical AND conjunction connects the clauses of a compound query so that the query selects the documents in the collection that match all the conditions.

Example

- To return movies which were released in Mexico and have an IMDB rating of at least 7:

```
usesample_mflix
db.movies.find( { countries: "Mexico", "imdb.rating": { $gte: 7 } } )
```

- Use the $or operator to specify a compound query that joins each clause with a logical OR conjunction so that the query selects the documents in the collection that match at least one condition.

Example

- To return movies from the sample_mflix.movies collection which were released in 2010 and either won at least 5 awards or have a genre of Drama:

```
usesample_mflix
db.movies.find( {
year: 2010,
   $or: [ { "awards.wins": { $gte: 5 } }, { genres: "Drama" } ]
} )
```

4.5.3 Update Operations

- Update operations modify existing documents in the collection.
- The MongoDB shell provides the following methods to update documents in a collection:
  - To update a single document, use **db.collection.updateOne().**
  - To update multiple documents, use **db.collection.updateMany().**
  - To replace a document, use **db.collection.replaceOne().**

Module 4

☞ **Update Operator Syntax**

- To update a document, MongoDB provides update operators, such as $set, to modify field values.
- To use the update operators, pass to the update methods an update document of the form:

```
{
<update operator>: { <field1>: <value1>, ... },
<update operator>: { <field2>: <value2>, ... },
 ...
}
```

- Some update operators, such as $set, create the field if the field does not exist.

☞ **Update a Single Document**

Use the db.collection.updateOne() method to update the first document that matches a specified filter.

Example

To update the first document in the sample_mflix.movies collection where title equals "Tag":

```
usesample_mflix
db.movies.updateOne( { title: "Tag" },
{
 $set: {
plot: "One month every year, five highly competitive friends
hit the ground running for a no-holds-barred game of tag"
 }
{ $currentDate: { lastUpdated: true } }
})
```

In the above example to update operation:

- Uses the **$set operator** to update the value of the plot field for the movie Tag.
- Uses the **$currentDate** operator to update the value of the lastUpdated field to the current date. If lastUpdated field does not exist, $currentDate will create the field.

☞ **Update Multiple Documents**

- Use the db.collection.updateMany() to update all documents that match a specified filter.
- To update all documents in the sample_airbnb.listings And Reviews collection to update where security_deposit is less than 100:

usesample_airbnb

```
db.listingsAndReviews.updateMany(
{ security_deposit: { $lt: 100 } },
 {
  $set: { security_deposit: 100, minimum_nights: 1 }
 }
)
```

- The update operation uses the $set operator to update the value of the security_deposit field to 100 and the value of the minimum_nights field to 1.

☞ **Replace a Document**

- To replace the entire content of a document except for the _id field, pass an entirely new document as the second argument to db.collection.replaceOne().
- When replacing a document, the replacement document must contain only field/value pairs. Do not include update operators expressions.
- The replacement document can have different fields from the original document. In the replacement document, you can omit the _id field since the _id field is immutable; however, if you do include the _id field, it must have the same value as the current value.

Example

- To replace the first document from the sample_analytics.accounts collection where account_id: 371138:

```
db.accounts.replaceOne(
{ account_id: 371138 },
{ account_id: 893421, limit: 5000, products: [ "Investment",
"Brokerage" ] }
)
```

- Run the following command to read your updated document:

```
db.accounts.findOne( { account_id: 893421 } )
```

4.5.4 Delete Operations

- The delete operations remove the documents from the collection.
- The MongoDB shell provides the following methods to delete documents from a collection:
  - To delete multiple documents, use **db.collection.deleteMany().**
  - To delete a single document, use **db.collection.deleteOne().**
  - **Delete All Documents**
- To delete all documents from a collection, pass an empty filter document {} to the db.collection.deleteMany() method.

Example:

- To delete all documents from the sample_mflix.movies collection:

usesample_mflix

db.movies.deleteMany({})

☞ **Delete All Documents that Match a Condition**

- You can specify criteria, or filters, that identify the documents to delete. The filters use the same syntax as read operations.
- To specify equality conditions, use <field>:<value> expressions in the query filter document.
- To delete all documents that match a deletion criteria, pass a filter parameter to the deleteMany() method.

Example

- To delete all documents from the sample_mflix.movies collection where the title equals "Titanic":

```
usesample_mflix
db.movies.deleteMany( { title: "Titanic" } )
```

☞ **Delete Only One Document that Matches a Condition**

- To delete at most a single document that matches a specified filter (even though multiple documents may match the specified filter) use the db.collection.deleteOne() method.

Example

To delete the first document from the sample_mflix.movies collection where the cast array contains "Brad Pitt":

```
usesample_mflix
db.movies.deleteOne( { cast: "Brad Pitt" }
```

4.6 CONNECT TO A MONGODB DATABASE USING NODE.JS

4.6.1 Access MongoDB in Node.js

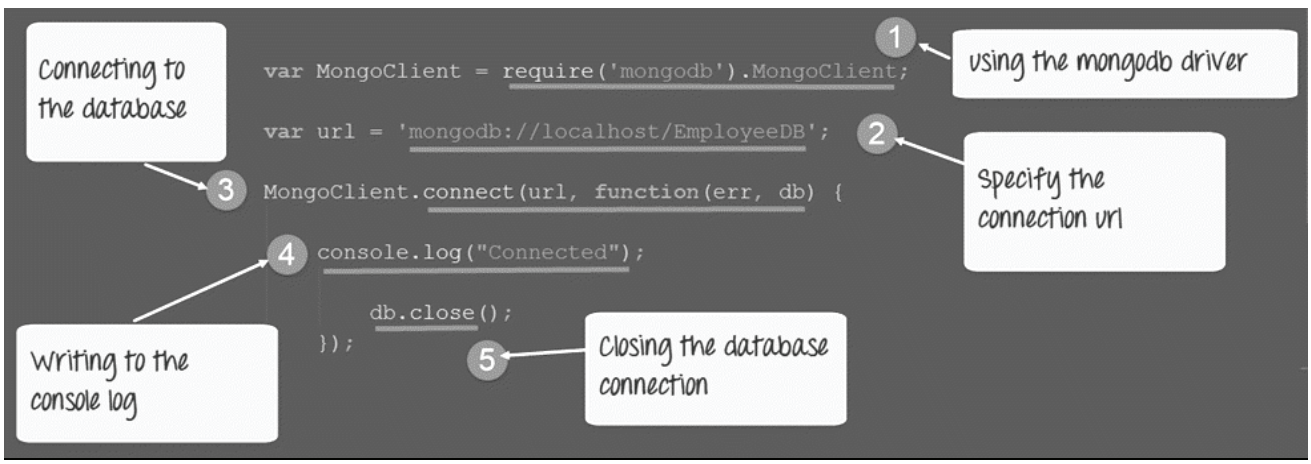
- In order to access MongoDB database, we need to install MongoDB drivers. The MongoDB driver allows you to easily interact with MongoDB databases from within Node.js applications. To download and install the official MongoDB driver, open the Command Terminal and execute the following command:

```
npm install mongodb
```

- Node.js can use this module to manipulate MongoDB databases:

```
var mongo = require('mongodb');
```

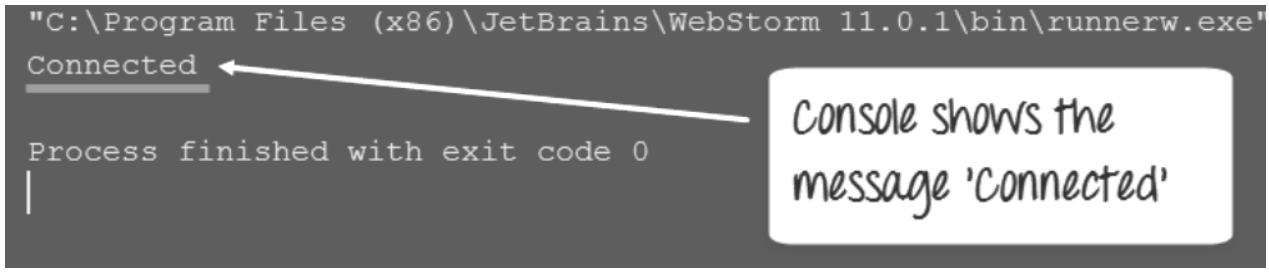
4.6.2 Creating and Closing a Connection to a MongoDB Database



1. The first step is to include the mongoose module, which is done through the require function. Once this module is in place, we can use the necessary functions available in this module to create connections to the database.
2. Next, we specify our connection string to the database. In the connect string, there are 3 key values which are passed.
   - The first is 'mongodb' which specifies that we are connecting to a mongoDB database.
   - The next is 'localhost' which means we are connecting to a database on the local machine.
   - The next is 'EmployeeDB' which is the name of the database defined in our MongoDB database.

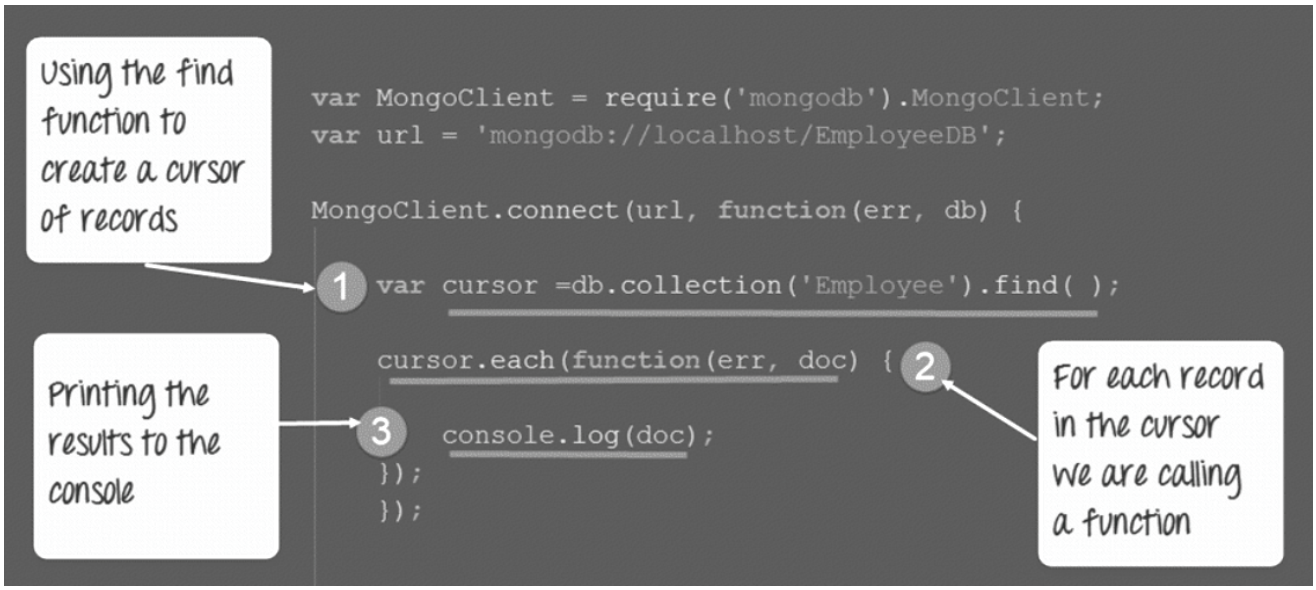
Module 4

3. The next step is to actually connect to our database. The connect function takes in our URL and has the facility to specify a callback function. It will be called when the connection is opened to the database. This gives us the opportunity to know if the database connection was successful or not.
4. In the function, we are writing the string "Connection established" to the console to indicate that a successful connection was created.
5. Finally, we are closing the connection using the db.close statement.
   - If the above code is executed properly, the string "Connected" will be written to the console as shown below.



4.6.3 Querying for Data in a MongoDB Database

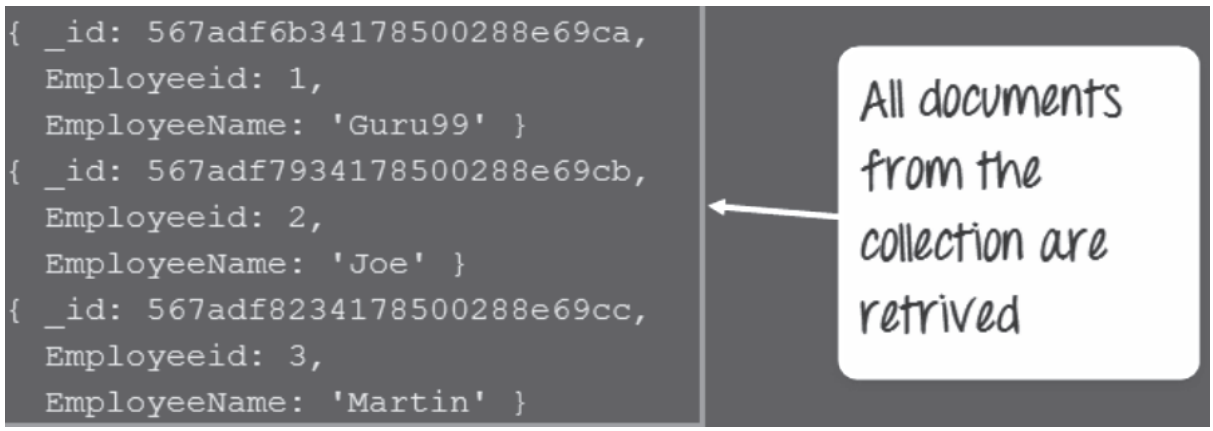
- Using the MongoDB driver we can also fetch data from the MongoDBdatabase. The below section will show how we can use the driver to fetch all of the documents from our Employee collection in our EmployeeDB database. This is the collection in our MongoDB database, which contains all the employee-related documents. Each document has an object id, Employee name, and employee id to define the values of the document.



1. In the first step, we are creating a cursor (A cursor is a pointer which is used to point to the various records fetched from a database. The cursor is then used to iterate through the different records in the database. Here we are defining a variable name called cursor which will be used to store the pointer to the records fetched from the database.) which points to the records which are fetched from the MongoDb collection. We also have the facility of specifying the collection 'Employee' from which to fetch the records. The find() function is used to specify that we want to retrieve all of the documents from the MongoDB collection.

2. We are now iterating through our cursor and for each document in the cursor we are going to execute a function.
3. Our function is simply going to print the contents of each document to the console.

Output

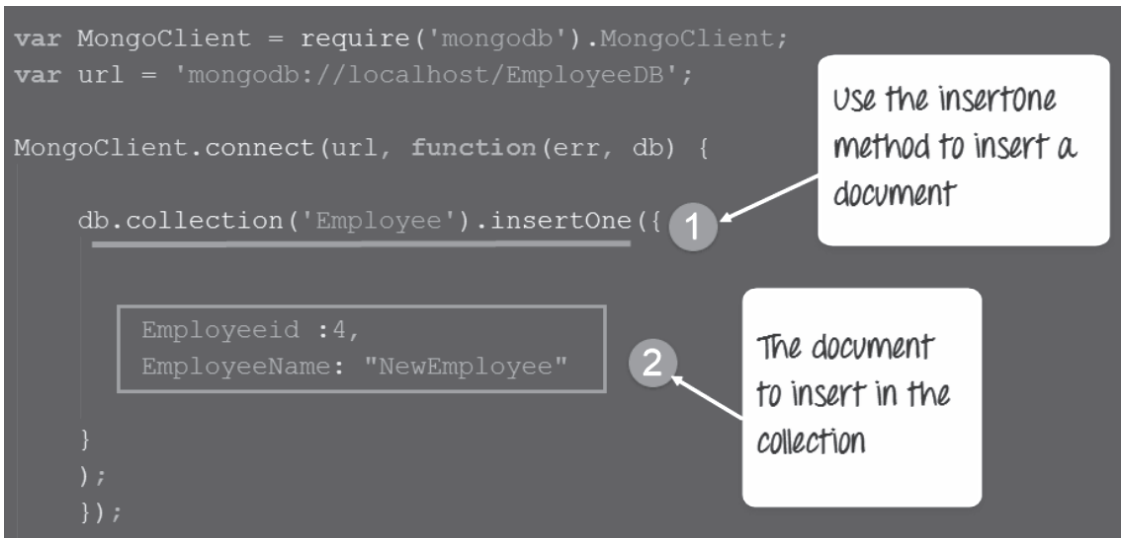


- It is also possible to fetch a particular record from a database. This can be done by specifying the search condition in the find() function. For example, suppose if you just wanted to fetch the record which has the employee name as Guru99, then this statement can be written as follows:

```
var cursor=db.collection('Employee').find({EmployeeName: "guru99"})
```

4.6.4 Inserting Documents in a Collection

- Documents can be inserted into a collection using the insertOne method provided by the MongoDB library.



1. Here we are using the insertOne method from the MongoDB library to insert a document into the Employee collection.
2. We are specifying the document details of what needs to be inserted into the Employee collection.

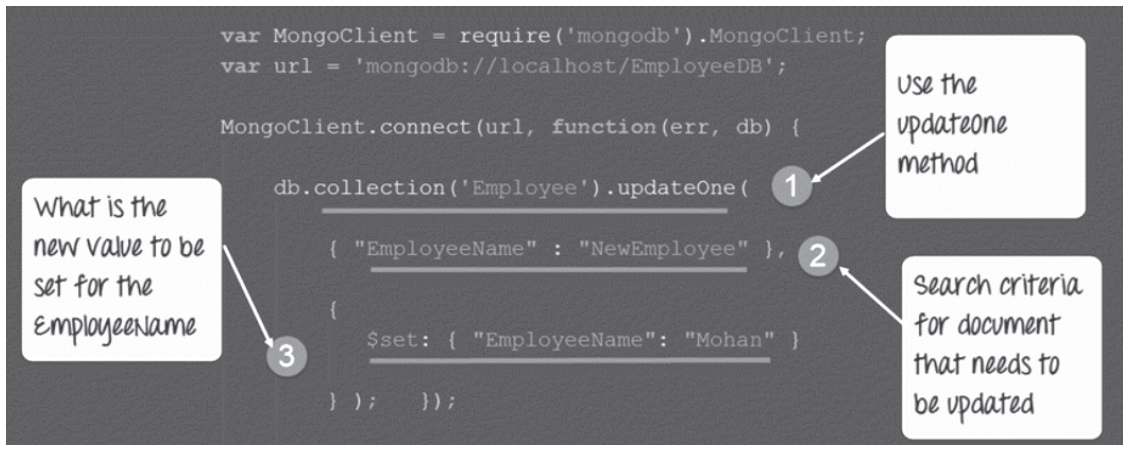
- If you now check the contents of your MongoDB database, you will find the record with Employeeid of 4 and EmployeeName of "NewEmployee" inserted into the Employee collection.

Module 4

- To check that the data has been properly inserted in the database, you need to execute the following commands in MongoDB
  - Use EmployeeDB
  - db.Employee.find({Employeeid :4 })
- The first statement ensures that you are connected to the EmployeeDb database. The second statement searches for the record which has the employee id of 4.

4.6.5 Updating Documents in a Collection

Documents can be updated in a collection using the updateOne method provided by the MongoDB library.



1. Here we are using the "updateOne" method from the MongoDB library, which is used to update a document in a mongoDB collection.
2. We are specifying the search criteria of which document needs to be updated. In our case, we want to find the document which has the EmployeeName of "NewEmployee."
3. We then want to set the value of the EmployeeName of the document from "NewEmployee" to "Mohan".

- If you now check the contents of your MongoDB database, you will find the record with Employeeid of 4 and EmployeeName of "Mohan" updated in the Employee collection.
- To check that the data has been properly updated in the database, you need to execute the following commands in MongoDB
  - Use EmployeeDB
  - db.Employee.find({Employeeid :4 })
- The first statement ensures that you are connected to the EmployeeDb database. The second statement searches for the record which has the employee id of 4.

4.6.6 Deleting Documents in a Collection

- Documents can be deleted in a collection using the "deleteOne" method provided by the MongoDB library.

```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

    db.collection('Employee').deleteOne(

        { "EmployeeName" : "Mohan" }

        );    });
```

1 Use the deleteOne method

2 Search criteria for which record needs to be deleted

1. Here we are using the "deleteOne" method from the MongoDB library, which is used to delete a document in a mongoDB collection.
2. We are specifying the search criteria of which document needs to be deleted. In our case, we want to find the document which has the EmployeeName of "Mohan" and delete this document.

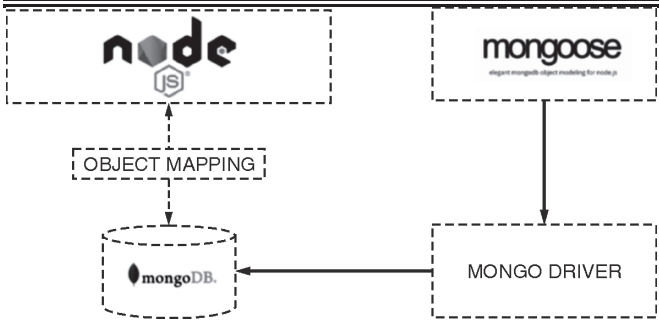
- If you now check the contents of your MongoDB database, you will find the record with Employeeid of 4 and EmployeeName of "Mohan" deleted from the Employee collection.
- To check that the data has been properly updated in the database, you need to execute the following commands in MongoDB
  - Use EmployeeDB
  - db.Employee.find()
- The first statement ensures that you are connected to the EmployeeDb database. The second statement searches and display all of the records in the employee collection. Here you can see if the record has been deleted or not.

4.7 USING MONGOOSE FOR STRUCTURED SCHEMA AND VALIDATION

4.7.1 Understanding Mongoose

- Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.
- Mongoose is an ODM library that wraps around the MongoDB Node.js driver. It provides a schema-based solution to model data stored in the MongoDB database.
- The main benefits of using Mongoose are:
  1. You can create a schema structure for you documents.
  2. Objects/documents in the model can be validated.
  3. Application data can by type casted into the object model.
  4. Business logic hooks can be applied using middleware.

Module 4



4.7.2 Mongoose Schemas and Models

Mongoose schemas

- A schema is fundamentally describing the data construct of a document. Mongoose uses schemas to model the data an application wishes to store and manipulate in MongoDB. This includes features such as type casting, validation, query building, and more. This schema defines the name of each item of data, and the type of data, whether it is a string, number, date, Boolean, and so on.
- The schema describes the attributes of the properties (aka fields) the application will manipulate. These attributes include such things as:
  - Data type (e.g. String, Number, etc.).
  - Whether or not it is required or optional.
  - Is it's value unique, meaning that the database is allowed to contain only one document with that value in that property.

Example

```
varuserSchema = new mongoose.Schema({
name: String,
email: String,
createdOn: Date,
verified: Boolean
});
```

Mongoose models

- A model is a compiled version of the schema. One instance of the model will map to one document in the database. More precisely, a model is a class that defines a document with the properties and behaviors as declared in our schema. All database operations performed on a document using Mongoose must reference a model.
- Creating a User instance based on the schema userSchema is a one line task:

```
var User = mongoose.model('User', userSchema);
```

Example for Schema and Model

// Mongoose schema and model definitions

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
```

// Create the schema for the Account database

```
constaccountSchema = new Schema({
account_no: { type: Number, required: true, unique: true },
owner_fname: { type: String, required: true, unique: false },
owner_mi: { type: String, required: true, unique: false },
owner_lname: { type: String, required: true, unique: false },
created_on: { type: Date, required: false, unique: false },
updated_on: { type: Date, required: false, unique: false },
});
```

// Create a model for the schema

```
const Account = mongoose.model('Account', accountSchema);
module.exports = Account;
```

4.7.3 Connecting to MongoDB with Mongoose

- **Step 1 :** Installing Mongoose on a Node.js environment
- Create and navigate to a new folder by running the following commands on a terminal.

  $ mkdirmongoose_tutorial

  $ cdmongoose_tutorial
- Then install Express and Mongoose by executing the following command on a terminal.

  $ npm install express mongoose –save
- **Step 2 :** Creating the connection
- Create a new file server.js to start our Express.js server.
- Load mongoose and express by adding the following code to server.js.

  server.js

```
const express = require("express");
const mongoose = require("mongoose");
const Router = require("./routes")
const app = express();
app.use(express.json());
```

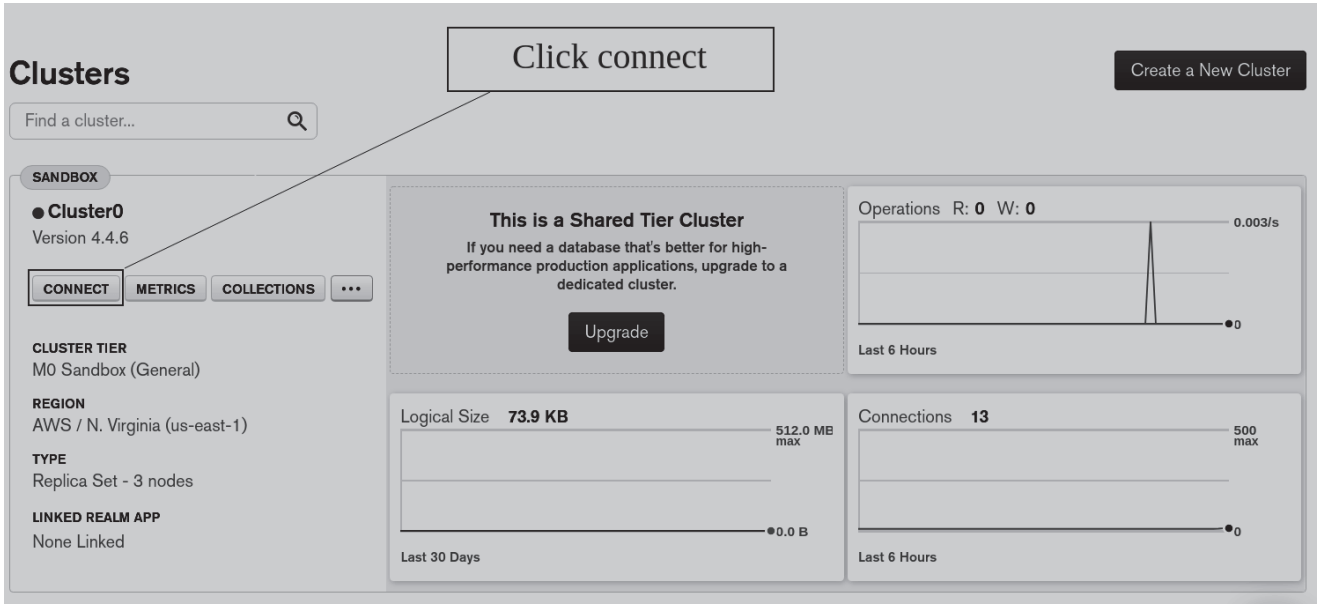
- Then connect to a local MongoDB instance using the mongoose.connect() function.

  server.js

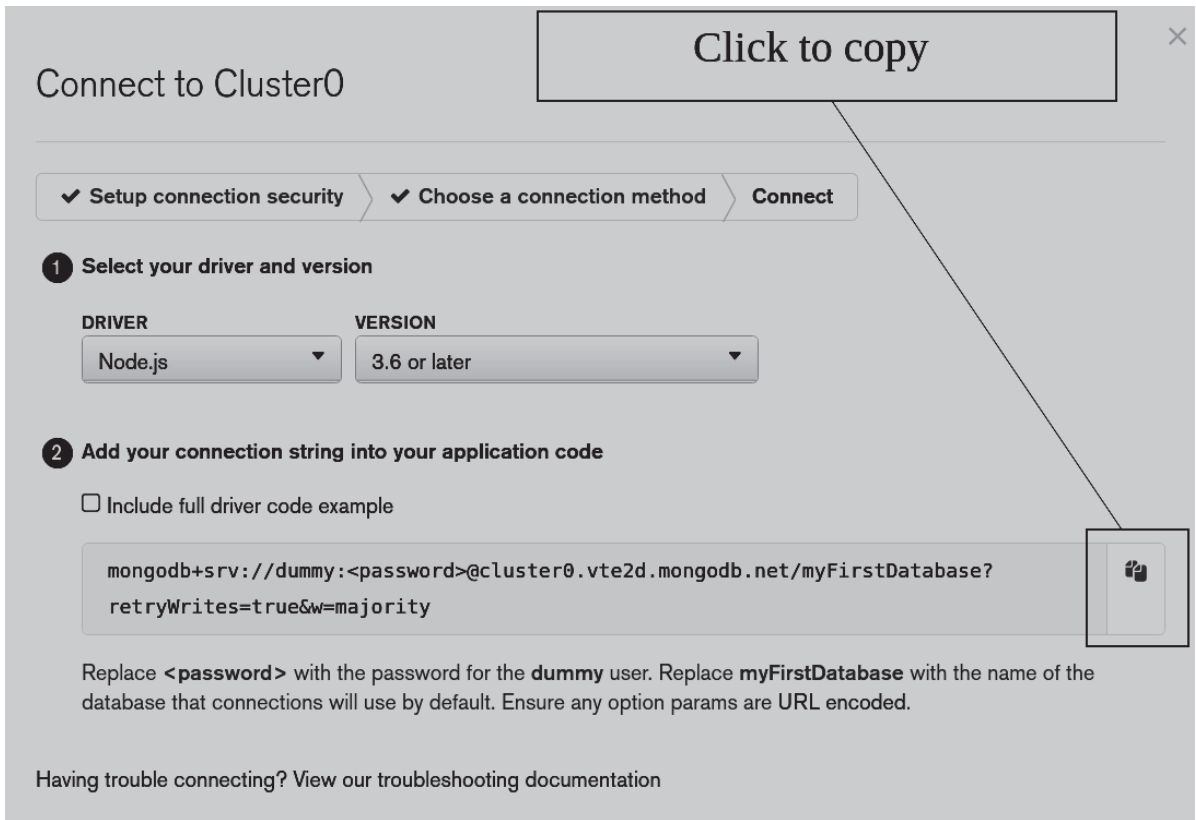
```
mongoose.connect('mongodb://localhost:27017/usersdb',
 {
useNewUrlParser: true,
useFindAndModify: false,
useUnifiedTopology: true
 }
);
```

To create a connection to MongoDB Atlas, follow the next steps.

1. Open your Cluster tab in MongoDB Atlas and click CONNECT.



2. Select Connect your application and choose Node.js for the driver.
3. Copy the connection string.



Module 4

With the connection at hand, create the following variables and replace their values using your actual credentials.

```
server.js
const username = "<mongodb username>";
const password = "<password>";
const cluster = "<cluster name>";
constdbname = "myFirstDatabase";

mongoose.connect(
  `mongodb+srv://${username}:${password}@${cluster}.mongodb.net/${dbname}?retryWrites=true&w=majority`,
  {
useNewUrlParser: true,
useFindAndModify: false,
useUnifiedTopology: true
  }
);
```

To make sure your connection was successful, add the following code right below your mongoose.connect().

```
server.js
// ...
constdb = mongoose.connection;
db.on("error", console.error.bind(console, "connection error: "));
db.once("open", function () {
console.log("Connected successfully");
});
```

Then, set the app to listen to port 3000.

```
server.js
// ...
app.use(Router);
app.listen(3000, () => {
console.log("Server is running at port 3000");
});
```

- **Step 3 :** Creating the schema

  Now let's define a collection schema for our application.

  Create another file models.js and add the following code.

```
models.js
const mongoose = require("mongoose");

constUserSchema = new mongoose.Schema({
name: {
type: String,
required: true,
  },
age: {
type: Number,
default: 0,
  },
```

```
});

const User = mongoose.model("User", UserSchema);
module.exports = User;
```

We create a schema UserSchema using the mongoose.Schema() method. The schema collects the name and age fields sent from the request.

We then export the schema using the last 2 lines.

- **Step 4 :** Creating the POST endpoint

  Create a new file routes.js. This file defines the endpoints for our app.

  Load express and the schema we created in Step 3 by adding the following code.

```
routes.js
const express = require("express");
constuserModel = require("./models");
const app = express();
```

Then create the POST endpoint by adding the following code.

```
routes.js
// ...
app.post("/add_user", async (request, response) => {
const user = new userModel(request.body);

try {
awaituser.save();
response.send(user);
   } catch (error) {
response.status(500).send(error);
   }
});
```

- We create a route /add_user to add a new user to the database. We parse the content to be saved to the database using the line const user = new userModel(request.body);.
- We then use a try/catch block to save the object to the database using the .save() method.

- **Step 5 :** Creating the GET endpoint

  Add the following lines of code to the routes.js file

  routes.js

```
// ...
app.get("/users", async (request, response) => {
const users = await userModel.find({});

try {
response.send(users);
  } catch (error) {
response.status(500).send(error);
  }
});
```

- We create a route /users to retrieve all the users saved using the /add_user route. We collect these users from the database using the .find() method. We then use a try/catch block to 'send' the users to this endpoint.

- Finally, export these endpoints by adding the line below.

routes.js

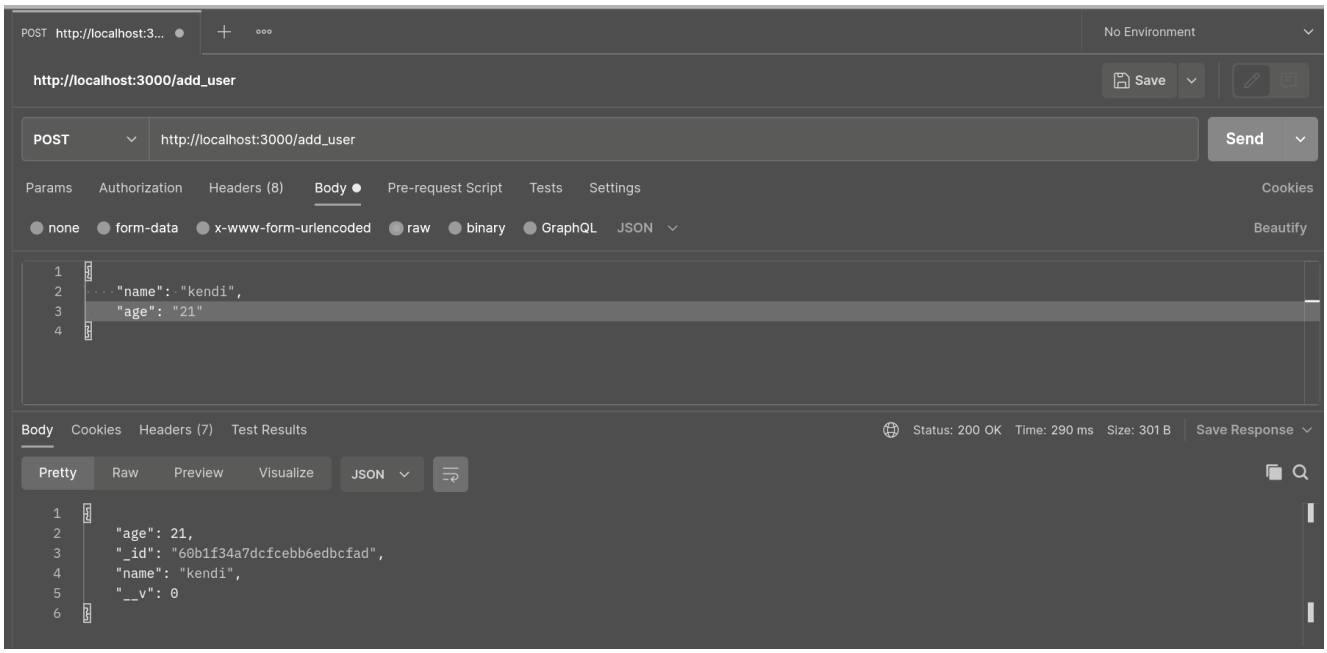
```
// ...
module.exports = app;
```

Serve the app by running the command below.

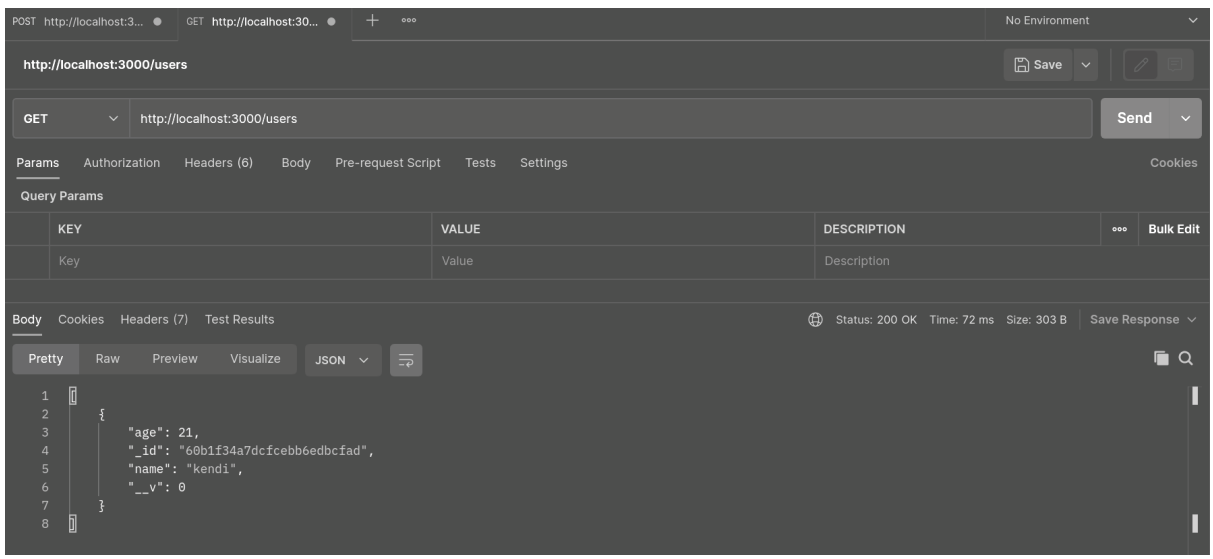
```
$ node server.js
```

▶ **Step 6 :** Testing the endpoints

- Now, let's test the two endpoints we created above.
- Open Postman and make a POST request to the http://localhost:3000/add_user endpoint.



- A new user is added to the database. You can check your collections to confirm this.
- Make a GET request to the http://localhost:3000/users endpoint.



The endpoint returns a list of all the users added to the database.

4.8 REST API

- An API, or application programming interface, is a set of rules that define how applications or devices can connect to and communicate with each other. A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.
- REST stands for representational state transfer and was developed by Roy Fielding in 2000.
- REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.
- When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it's language-agnostic, as well as readable by both humans and machines.

4.8.1 REST Design Principles

The API needs to meet the following architectural requirements to be considered a REST API.

1. **Uniform Interface :** All API requests for the same resource should look the same, no matter where the request comes from. The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI). Resources shouldn't be too large but should contain every piece of information that the client might need.
2. **Client-Server :** In REST API design, client and server applications must be completely independent of each other. The only information the client application should know is the URI of the requested resource; it can't interact with the server application in any other ways. Similarly, a server application shouldn't modify the client application other than passing it to the requested data via HTTP.
3. **Stateless :** REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it. In other words, REST APIs do not require any server-side sessions. Server applications aren't allowed to store any data related to a client request.
4. **Cacheable :** When possible, resources should be cacheable on the client or server side. Server responses also need to contain information about whether caching is allowed for the delivered resource. The goal is to improve performance on the client side, while increasing scalability on the server side.
5. **Layered System Architecture :** In REST APIs, the calls and responses go through different layers. As a rule of thumb, don't assume that the client and server applications connect directly to each other. There may be a number of different intermediaries in the communication loop. REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary.
6. **Code on Demand :** REST APIs usually send static resources, but in certain cases, responses can also contain executable code (such as Java applets). In these cases, the code should only run on-demand.

4.8.2 Rules of REST API

There are certain rules which should be kept in mind while creating REST API endpoints.

1. REST is based on the resource or noun instead of action or verb based. It means that a URI of a REST API should always end with a noun.

   Example: /api/users

2. HTTP verbs are used to identify the action. Some of the HTTP verbs are – GET, PUT, POST, DELETE, UPDATE, PATCH.

   HTTP verbs: Some of the common HTTP methods/verbs are described below:

   (i) GET : Retrieves one or more resources identified by the request URI and it can cache the information receive.

   (ii) POST : Create a resource from the submission of a request and response is not cacheable in this case. This method is unsafe if no security is applied to the endpoint as it would allow anyone to create a random resource by submission.

   (iii) PUT : Update an existing resource on the server specified by the request URI.

   (iv) DELETE : Delete an existing resource on the server specified by the request URI. It always return an appropriate HTTP status for every request.

3. A web application should be organized into resources like users and then uses HTTP verbs like - GET, PUT, POST, and DELETE to modify those resources. And as a developer it should be clear that what needs to be

Module 4

done just by looking at the endpoint and HTTP method used.

4. Always use plurals in URL to keep an API URI consistent throughout the application.
5. Send a proper HTTP code to indicate a success or error status.

URI	HTTP Verb	Description
api/users	GET	Get all users
api/users/new	GET	Show form for adding new user
api/users	POST	Add a user
api/users/1	PUT	Update a user with id = 1
api/users/1/edit	GET	Show edit form for user with id = 1
api/users/1	DELETE	Delete a user with id = 1
api/users/1	GET	Get a user with id = 1

4.8.3 Evaluating API Patterns

HTTP Methods and Status Codes

Each HTTP request includes a method, sometimes called "HTTP verbs," that provides a lot of context for each call. Following are the most common HTTP methods.

- **GET :** read data from your API
- **POST :** add new data to your API
- **PUT :** update existing data with your API
- **PATCH :** updates a subset of existing data with your API
- **DELETE :** remove data (usually a single resource) from your API

HTTP status codes help describe the response in the reverse direction.

Some common HTTP status codes include:

- **200 :** Successful request, often a GET
- **201 :** Successful request after a create, usually a POST
- **204 :** Successful request with no content returned, usually a PUT or PATCH
- **301 :** Permanently redirect to another endpoint
- **400 :** Bad request (client should modify the request)
- **401 :** Unauthorized, credentials not recognized
- **403** : Forbidden, credentials accepted but don't have permission
- **404 :** Not found, the resource does not exist
- **410 :** Gone, the resource previously existed but does not now
- **429 :** Too many requests, used for rate limiting and should include retry headers
- **500 :** Server error, generic and worth looking at other 500-level errors instead
- **503 :** Service unavailable, another where retry headers are useful

Use Friendly Endpoint Names

- A typical design pattern with REST APIs is to build your endpoints around resources. These are the "nouns" to HTTP method verbs. Your API design will be much easier to understand if these names are descriptive.
- For example, if you're working on a cookbook API, you might include the following endpoint: /recipes/
- As you add new recipes, you would POST them to the endpoint. To get a list, you use the GET method on the same endpoint. To retrieve a specific recipe, you could call it by its identifier in the URL: /recipes/42
- One thing to specifically avoid with friendly REST endpoint names is describing actions. For example, a verb within the endpoint (i.e., /getRecipes/) would run counter to relying on HTTP to provide that context.

Support Use Cases with API Parameters

- When use cases are discovered after an API is built, engineers will create new endpoints to support these unearthed requirements. For example, your cookbook API may need to return only recipes from a specific category, or you want to show the recipes with the least prep time. Rather than create redundant endpoints, plan for smart parameters from the start.
- There are three common types of parameters to consider for your API:
  1. **Filtering :** Return only results that match a filter by using field names as parameters. For example: /recipes/?category=Cookies
  2. **Pagination :** Don't overload clients and servers by providing everything. Instead, set a limit and provide prev and next links in your response. Example: /recipes/?limit=100&page=3
  3. **Sorting :** Provide a way to sort or some use cases will still require paging through all results to find what's needed. Example: /recipes/?sort=prep_time
- These three approaches can be used together to support very specific queries. For example, this API request would retrieve one cookie recipe with the shortest preparation time:

  /recipes/?category=Cookies&sort=prep_time&limit=1

4.9 BUILDING RESTFUL CRUD API WITH NODE.JS, EXPRESS AND MONGODB

- **Step 1 :** Start your terminal and create a new folder for the application.

  $ mkdir node-easy-notes-app

- **Step 2 :** Initialize the application with a package.json file.

Go to the root folder of your application and type npminit to initialize your app with a package.json file.

$ cd node-easy-notes-app

$ npminit

server.js as the entry point of an application.

- **Step 3 :** Install dependencies

We will need express, mongoose and body-parser modules in our application. Let's install them by typing the following command -

$ npm install express body-parser mongoose –save

- Our application folder now has a package.json file and a node_modules folder -

```
node-easy-notes-app
    └── node_modules/
    └── package.json
```

- **Step 4 :** Setting up the web server

- Create the main entry point of our application. Create a new file named server.js in the root folder of the application with the following contents –

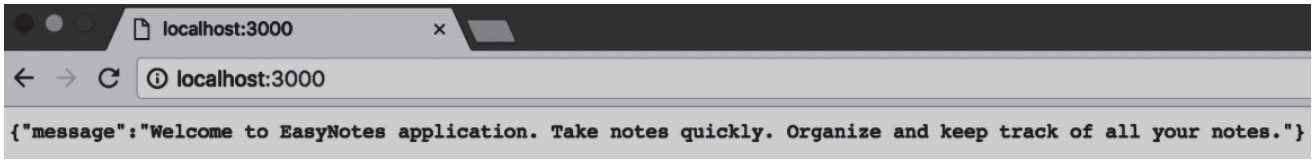
```
const express = require('express');
constbodyParser = require('body-parser');
// create express app
const app = express();
// parse requests of content-type - application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }))
// parse requests of content-type - application/json
app.use(bodyParser.json())
// define a simple route
app.get('/', (req, res) => {
res.json({"message": "Welcome to EasyNotes application. Take notes quickly. Organize and keep track of all your notes."});
});
// listen for requests
app.listen(3000, () => {
console.log("Server is listening on port 3000");
});
```

- In the above code, first we import express and body-parser modules. Express, is a web framework that we will be using for building the REST APIs, and body-parser is a module that parses the request of various content types and creates a req.body object that we can access in our routes.
- Then, We create an express app, and add two body-parser middlewares using express's app.use() method. A middleware is a function that has access to the request and response objects. It can execute any code, transform the request object, or return a response.
- Then, We define a simple GET route which returns a welcome message to the clients.
- Finally, We listen on port 3000 for incoming connections.

- Let's now run the server and go to http://localhost:3000 to access the route we just defined.

  $ node server.js

  Server is listening on port 3000



▶ **Step 5 :** Configuring and connecting to the database

- Create a new folder config in the root folder of our application for keeping all the configurations:

  $ mkdirconfig

  $ cdconfig

- Now, Create a new file database.config.js inside config folder with the following contents –

```
module.exports = {
url: 'mongodb://localhost:27017/easy-notes'
}
```

Now import the above database configuration in server.js and connect to the database using mongoose.

- Add the following code to the server.js file after app.use(bodyParser.json()) line -

```
// Configuring the database
constdbConfig = require('./config/database.config.js');
const mongoose = require('mongoose');

mongoose.Promise = global.Promise;
// Connecting to the database
mongoose.connect(dbConfig.url, {
useNewUrlParser: true
}).then(() => {
console.log("Successfully connected to the database");
}).catch(err => {
console.log('Could not connect to the database. Exiting now...',
err);
process.exit();
});
```

Run the server and make sure that you're able to connect to the database :

$ node server.js

Server is listening on port 3000

Successfully connected to the database

▶ **Step 6 :** Defining the Note model in Mongoose

Create a new folder called app inside the root folder of the application, then create another folder called models inside the app folder -

$ mkdir -p app/models

$ cd app/models

Now, create a file called note.model.js inside app/models folder with the following contents -

```
const mongoose = require('mongoose');

constNoteSchema = mongoose.Schema({
title: String,
content: String
}, {
timestamps: true
});

module.exports = mongoose.model('Note', NoteSchema);
```

The Note model contains a title and content field also timestamps option is added to the schema.

Mongoose uses this option to automatically add two new fields - createdAt and updatedAt to the schema.

▶ **Step 7 :** Defining Routes using Express

Create a new folder called routes inside the app folder.

$ mkdir app/routes

$ cd app/routes

Now, create a new file called note.routes.js inside app/routes folder with the following contents –

```
module.exports = (app) => {
const notes = require('../controllers/note.controller.js');

   // Create a new Note
app.post('/notes', notes.create);
   // Retrieve all Notes
app.get('/notes', notes.findAll);

   // Retrieve a single Note with noteId
app.get('/notes/:noteId', notes.findOne);

   // Update a Note with noteId
app.put('/notes/:noteId', notes.update);
   // Delete a Note with noteId
app.delete('/notes/:noteId', notes.delete);
}
```

Note that we have added a require statement for note.controller.js file. We'll define the controller file in the next section. The controller will contain methods for handling all the CRUD operations.

Before defining the controller, let's first include the routes in server.js. Add the following require statement before app.listen() line inside server.js file.

```
// ........

// Require Notes routes
require('./app/routes/note.routes.js')(app);

// ........
```

▶ **Step 8 :** Writing the Controller functions

Create a new folder called controllers inside the app folder, then create a new file called note.controller.js inside app/controllers folder with the following contents –

```
const Note = require('../models/note.model.js');
// Create and Save a new Note
exports.create = (req, res) => {
};
// Retrieve and return all notes from the database.
exports.findAll = (req, res) => {

};
// Find a single note with a noteId
exports.findOne = (req, res) => {
};
// Update a note identified by the noteId in the request
exports.update = (req, res) => {
};
// Delete a note with the specified noteId in the request
exports.delete = (req, res) => {
};
```

Let's now look at the implementation of the above controller functions one by one –

☞ Creating a new Note

```
// Create and Save a new Note
exports.create = (req, res) => {
    // Validate request
if(!req.body.content) {
returnres.status(400).send({
message: "Note content can not be empty"
        });
    }

    // Create a Note
const note = new Note({
title: req.body.title || "Untitled Note",
content: req.body.content
    });

    // Save Note in the database
note.save()
    .then(data => {
res.send(data);
    }).catch(err => {
res.status(500).send({
message: err.message || "Some error occurred while creating
the Note."
        });
    });
};
```

☞ Retrieving all Notes

```
// Retrieve and return all notes from the database.
exports.findAll = (req, res) => {
Note.find()
    .then(notes => {
res.send(notes);
    }).catch(err => {
res.status(500).send({
message: err.message || "Some error occurred while retrieving
notes."
        });
    });
};
```

☞ Retrieving a single Note

```
// Find a single note with a noteId
exports.findOne = (req, res) => {
Note.findById(req.params.noteId)
    .then(note => {
if(!note) {
returnres.status(404).send({
message: "Note not found with id " + req.params.noteId
            });
        }
res.send(note);
    }).catch(err => {
if(err.kind === 'ObjectId') {
returnres.status(404).send({
message: "Note not found with id " + req.params.noteId
            });
        }
returnres.status(500).send({
message: "Error retrieving note with id " + req.params.noteId
        });
    });
};
```

☞ Updating a Note

```
// Update a note identified by the noteId in the request
exports.update = (req, res) => {
    // Validate Request
if(!req.body.content) {
returnres.status(400).send({
message: "Note content can not be empty"
        });
    }
    // Find note and update it with the request body
Note.findByIdAndUpdate(req.params.noteId, {
title: req.body.title || "Untitled Note",
```

Module 4

```
content: req.body.content
    }, {new: true})
    .then(note => {
if(!note) {
returnres.status(404).send({
message: "Note not found with id " + req.params.noteId
            });
        }
res.send(note);
    }).catch(err => {
if(err.kind === 'ObjectId') {
returnres.status(404).send({
message: "Note not found with id " + req.params.noteId
            });
        }
returnres.status(500).send({
message: "Error updating note with id " + req.params.noteId
        });
    });
};
```

The {new: true} option in the findByIdAndUpdate() method is used to return the modified document to the then() function instead of the original.

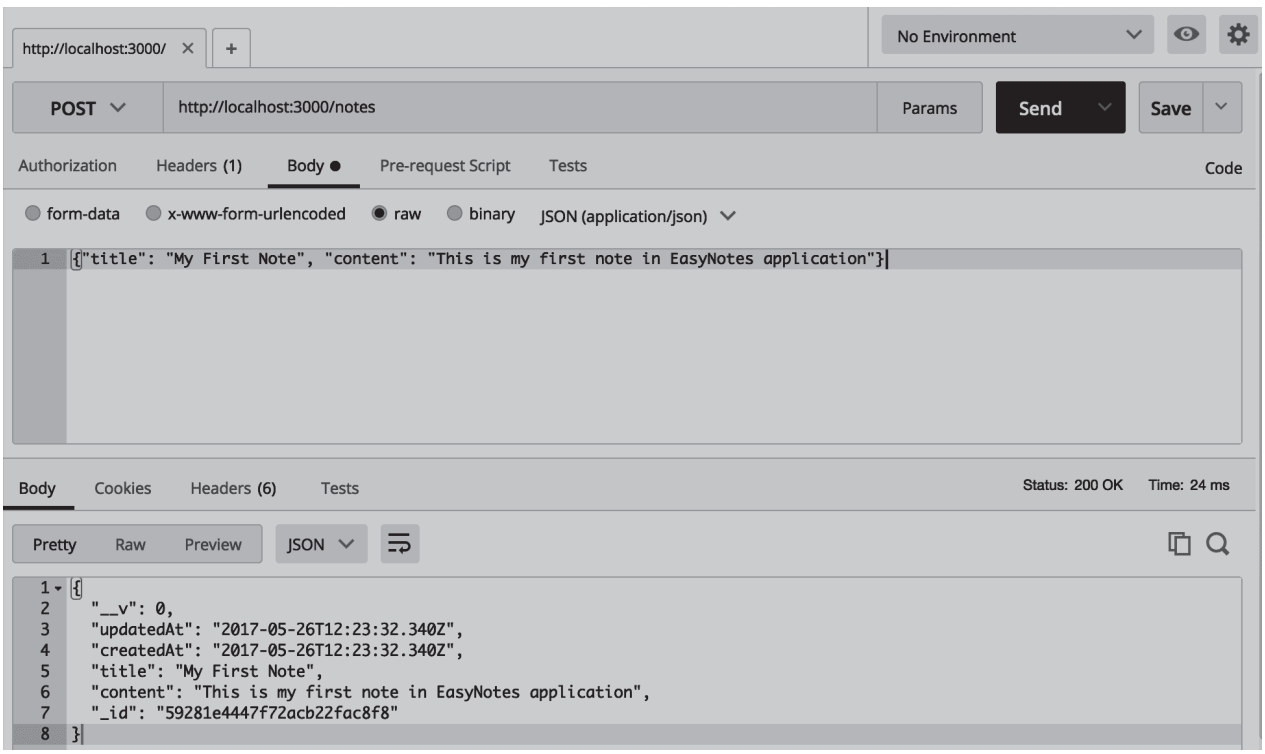
☞ **Deleting a Note**

```
// Delete a note with the specified noteId in the request
exports.delete = (req, res) => {
Note.findByIdAndRemove(req.params.noteId)
    .then(note => {
if(!note) {
returnres.status(404).send({
message: "Note not found with id " + req.params.noteId
            });
        }
res.send({message: "Note deleted successfully!"});
    }).catch(err => {
if(err.kind === 'ObjectId' || err.name === 'NotFound') {
returnres.status(404).send({
message: "Note not found with id " + req.params.noteId
            });
        }
returnres.status(500).send({
message: "Could not delete note with id " + req.params.noteId
        });
    });
};
```

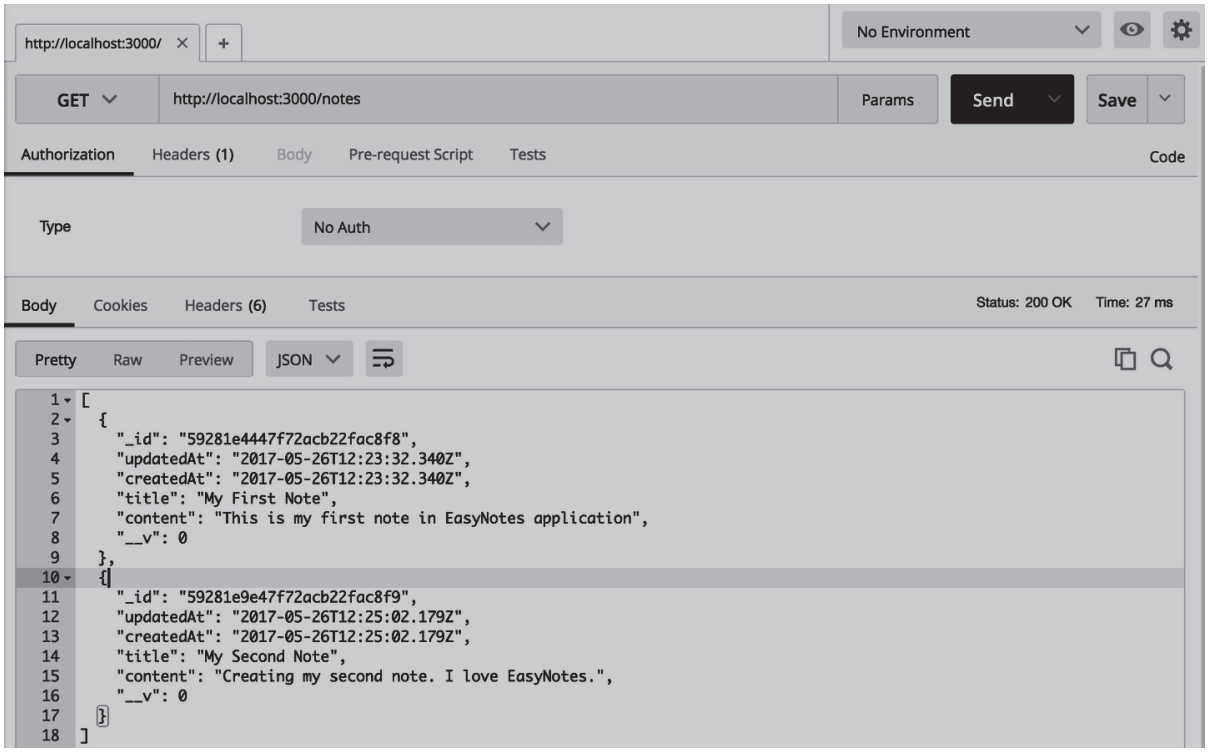
▶ **Step 9:** Testing our APIs

Let's now test all the APIs one by one using postman

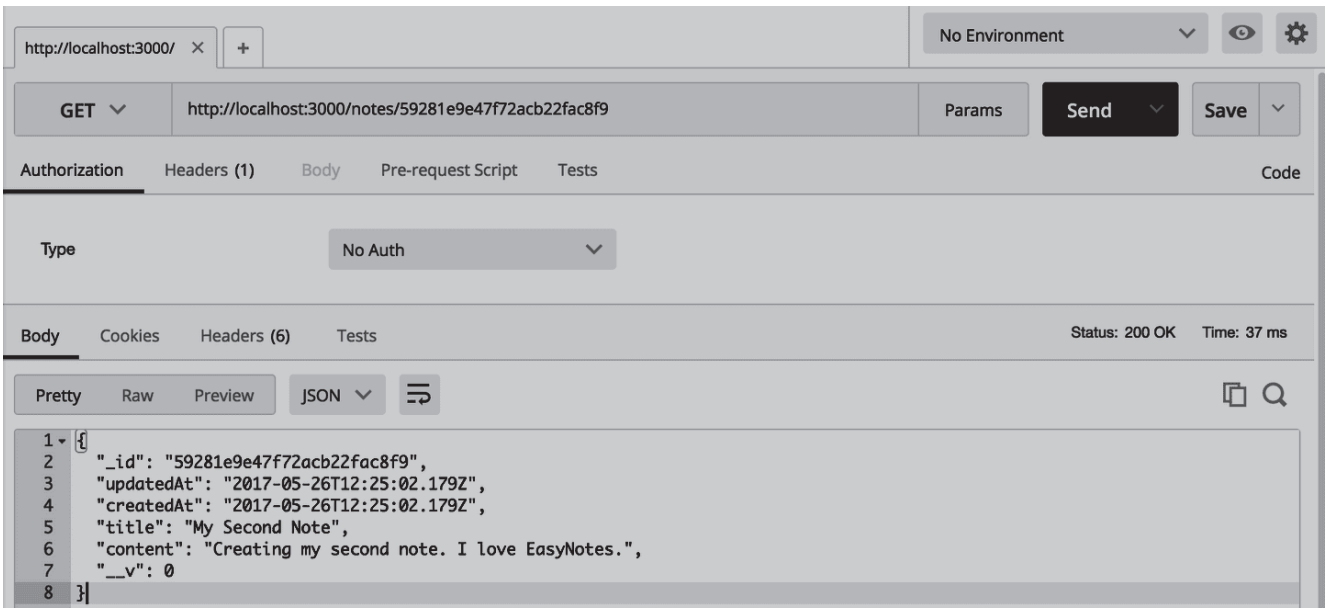
Creating a new Note using POST /notes API



Retrieving all Notes using GET /notes API

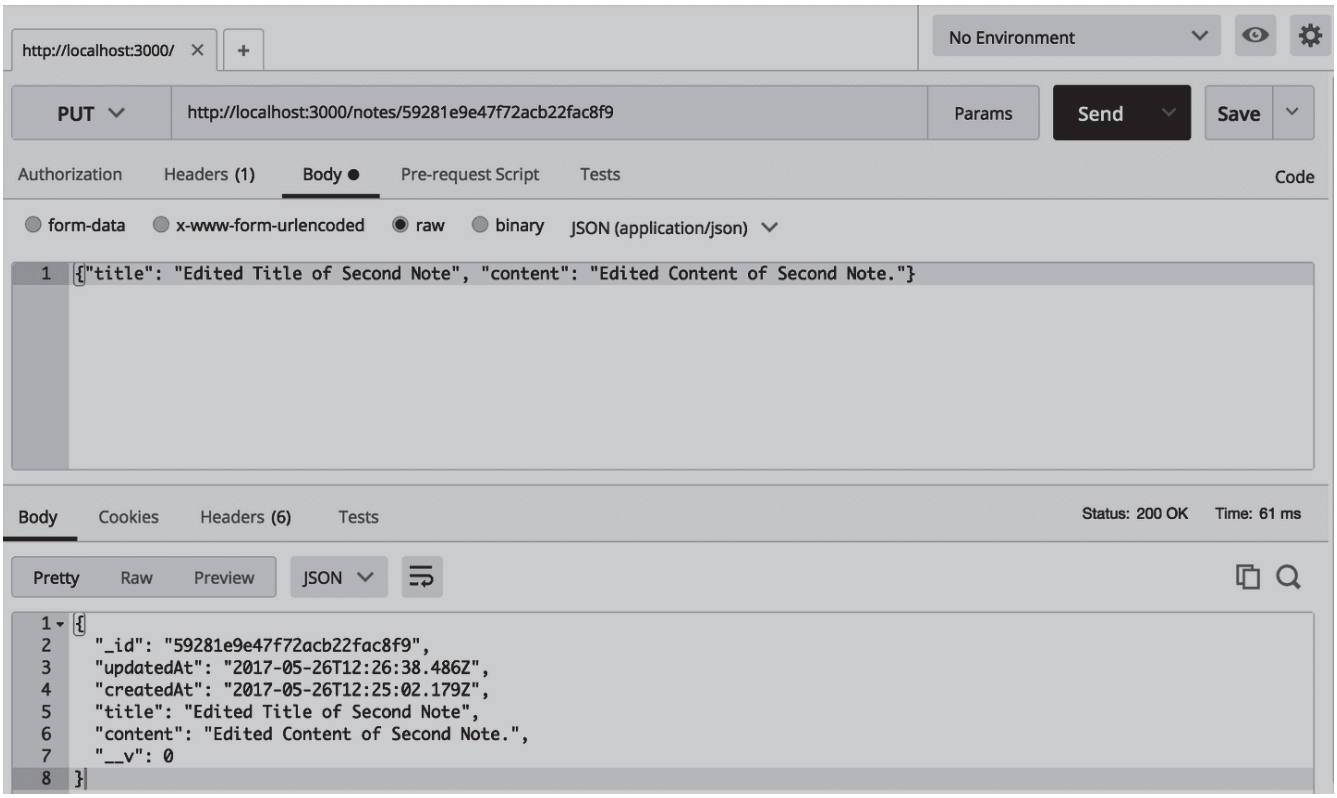


Retrieving a single Note using GET /notes/:noteId API

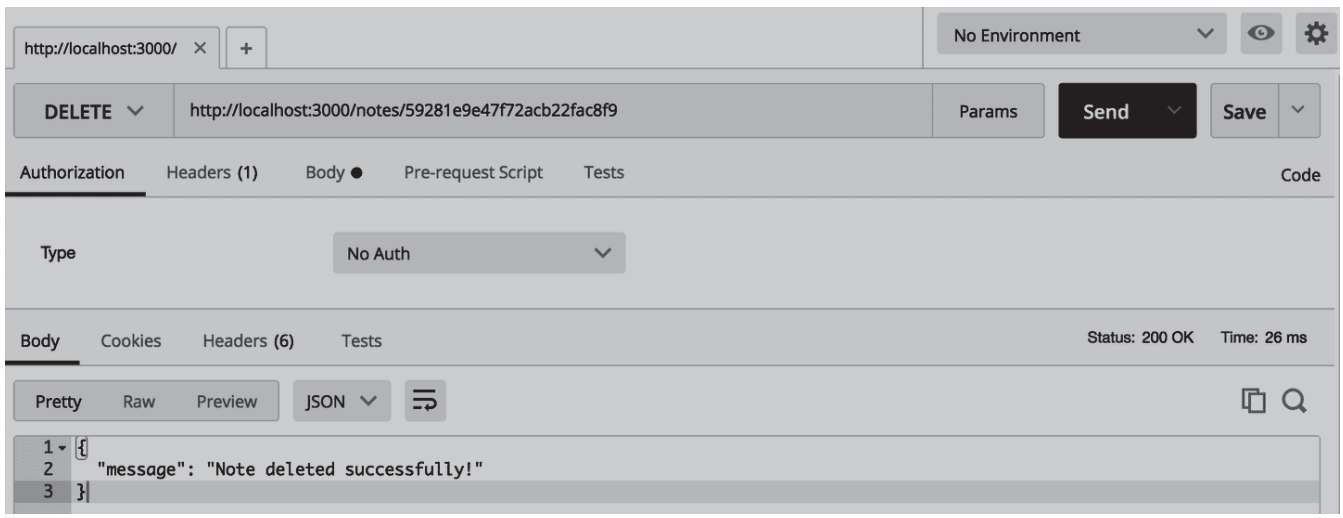


Module 4

Updating a Note using PUT /notes/:noteId API



Deleting a Note using DELETE /notes/:noteId API



4.10 MONGODB VS SQL DATABASES

NoSQL Database (MongoDB)	SQL Database
MongoDB is an open-source database developed by MongoDB, Inc.	MySQL is a popular open-source relational database management system (RDBMS) that is developed, distributed and supported by Oracle Corporation.
In MongoDB, each individual records are stored as 'documents'.	In MySQL, each individual records are stored as 'rows' in a table.
Documents belonging to a particular class or group as stored in a 'collection'. Example : collection of users.	A 'table' is used to store rows (records) of similar type.
Non-relational database	Relational database
Supports JSON query language	Supports SQL query language
Collection based and key-value pair	Table based
Field based	Column based
No support for foreign key	Support foreign key
No Support for triggers	Support for triggers
Contains dynamic schema	Contains schema which is predefined
Best fit for hierarchical data storage	Not fit for hierarchical data storage
Horizontally scalable - add more servers	Vertically scalable - increasing RAM

Review Questions

1. What is MongoDB?
2. What are the advantages of MongoDB?
3. What is a Document in MongoDB?
4. What is a Collection in MongoDB?
5. What are the features of MongoDB?
6. What are the data types in MongoDB?
7. Explain working of MongoDB in detail.
8. List out applications of MongoDB.
9. Explain MongoDB CRUD Operations with an example.
10. Write a short note on : Mongoose schemas and Models.
11. Explain RSET API in detail.
12. Enlist different types of HTTP requests supported by REST API.
13. List out HTTP response codes returned by REST API.
14. Explain API design patterns.

Module 4

Chapter Ends…

❑❑❑

Note

MODULE 5

CHAPTER 5

Flask

University Prescribed Syllabus w.e.f Academic Year 2021-2022

Introduction, Flask Environment Setup, App Routing, URL Building, Flask HTTP Methods, Flask Request Object, Flask cookies, File Uploading in Flask.

5.1 INTRODUCTION

5.1.1 Introduction to Flask

Flask is a popular Python framework for developing web applications. It is a framework with a built-in development server and a debugger.

Flask is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, and upload handling, various open authentication technologies, and several common framework-related tools.

Flask is Python's micro-framework for web app development. It was developed by **Armin Ronacher**, who led an international team of Python enthusiasts called Pocco. Flask consists of Werkzeug WSGI toolkit and Jinja2 template engine. Both were also developed by Pocco. It was initially released in April 2010. Currently, the latest version as of writing this article is 2.0.2.

What is WSGI?

- Web Server Gateway Interface (WSGI) is the standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.
- Werkzeug is one of the most advanced WSGI modules that contain various tools and utilities that facilitate web application development. Flask implements Werkzeug.

What is Jinja 2?

- Jinja 2 is a template rendering engine. It renders the web pages for the server with any specified custom content given to it by the webserver. Flask renders its HTML based templates using Jinja 2.

5.1.2 Features of Flask

1. It is a lightweight framework.
2. Flask gives independent or full control to the developer for creating applications.
3. Flask provides a development server and a debugger.
4. It uses Jinja2 templates.
5. It is compliant with WSGI 1.0.
6. It provides integrated support for unit testing.
7. Many extensions are available for Flask, which can be used to enhance its functionalities.
8. Secure cookies
9. Support URL routing
10. Unicode support
11. Request dispatching

5.1.3 Advantages of Flask

Flask is a lightweight Web Server Gateway Interface (WSGI) web application framework. It has minimal or no external libraries. It is used to create a simple website and then scale it up to complex applications. Below are some of the advantages of Flask.

1. Adaptable to the latest technologies
2. Independent framework enables experimentation with architecture, libraries.
3. Suitable for small case projects.
4. Requires small codebase size for simple functions
5. Ensures scalability for simplistic applications
6. Easy to build a quick prototype
7. Routing URL functions through Werkzeug makes the process easier.
8. Hassle-free application development and maintenance.
9. Database integration is easy
10. Extensible and easy core system.
11. The power of the framework lies in its minimalistic features.
12. Flexible and allow full control access.

5.1.4 What is Micro-framework?

- In contrast to a traditional web framework, a micro-framework is a minimalistic framework where developers are provided with a lot of freedom to build the web application layer.
- As compared to an enterprise framework, a developer would not need to set up many things in a micro-framework to get the web app hosted and running.

- This is incredibly useful in cases of small web app development where the needs are not the same as an enterprise-level framework which would save lots of development maintenance time and money as a consequence.

Why Flask is called a micro-framework?

- Flask is known as a micro-framework because it is lightweight and only provides components that are essential. It only provides the necessary components for web development, such as routing, request handling, sessions, and so on.
- For the other functionalities such as data handling, the developer can write a custom module or use an extension.

5.2 FLASK ENVIRONMENT SETUP

Step 1 : Install Python

To install flask on the system, we need to have python 2.7 or higher version installed on our system.

Step 2 : Install virtual environment (virtualenv)

- Virtualenv is considered as the virtual python environment builder which is used to create the multiple python virtual environment side by side. It can be installed by using the following command.

```
$ pip install virtualenv
```

- Once it is installed, we can create the new virtual environment into a folder as given below.

```
$ mkdir new
$ cd new
$ virtualenv venv
```

- To activate the corresponding environment, use the following command on the Linux operating system.

```
$ venv/bin/activate
```

- On windows, use the following command.

```
$ venv\scripts\activate
```

Step 3: Install Flask

- Install the flask by using the following command.

```
$ pip install flask
```

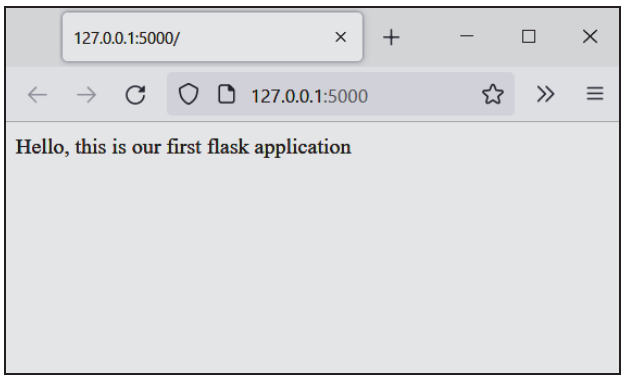
First Flask application

app.py

```
from flask import Flask
app = Flask(__name__) #creating the Flask class object
@app.route('/')
def home():
    return "Hello, this is our first flask application";
if __name__ =='__main__':
    app.run(debug = True)
```

Run above python code on the command line and check the result. Since it is a web application, therefore it is to be run to on the browser at http://localhost:5000.

Output



- To build the python web application, we need to import the Flask module. An object of the Flask class is considered as the WSGI application.
- We need to pass the name of the current module, i.e. __name__ as the argument into the Flask constructor.
- The route() function of the Flask class defines the URL mapping of the associated function. The syntax is given below.

```
app.route(rule, options)
```

- It accepts the following parameters.
  1. **rule :** It represents the URL binding with the function.
  2. **options :** It represents the list of parameters to be associated with the rule object.
- As we can see here, the / URL is bound to the main function which is responsible for returning the server response. It can return a string to be printed on the browser's window or we can use the HTML template to return the HTML file as a response from the server.
- Finally, the run method of the Flask class is used to run the flask application on the local development server.

Module 5

- The syntax is given below.

```
app.run(host, port, debug, options)
```

  - **Host :** The default hostname is 127.0.0.1, i.e. localhost.
  - **Port :** The port number to which the server is listening to. The default port number is 5000.
  - **Debug :** The default is false. It provides debug information if it is set to true.
  - **Options :** It contains the information to be forwarded to the server.

5.3 FLASK APP ROUTING

- App routing is used to map the specific URL with the associated function that is intended to perform some task. It is used to access some particular page.
- If we visit the particular URL mapped to some particular function, the output of that function is rendered on the browser's screen. To bind a function to an URL path we use the app.route decorator.

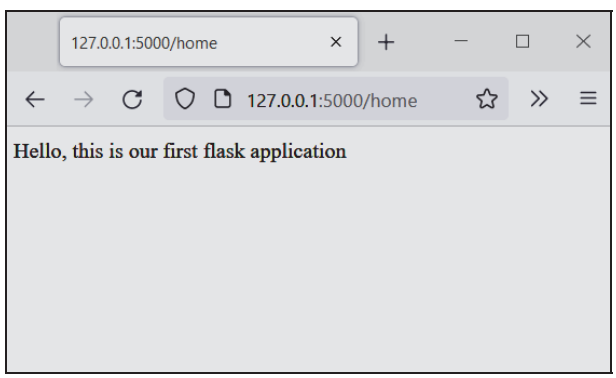
Example

```
from flask import Flask
app = Flask(__name__)
@app.route('/home')
def home():
    return " Hello, this is our first flask application ";
if __name__ =="__main__":
    app.run(debug = True)
```

- Run above python code on the command line and check the result. Since it is a web application, therefore it is to be run to on the browser at

  http://localhost:5000/home.

Output



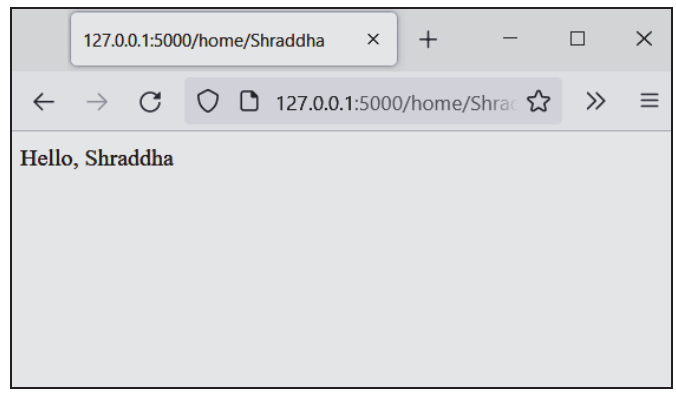
Dynamic URLs

- We can also build dynamic URLs by using variables in the URL. To add variables to URLs, use <variable_name> rule. The function then receives the <variable_name> as keyword argument.

Example

```
from flask import Flask
app = Flask(__name__)
@app.route('/home/<name>')
def home(name):
    return "Hello, " + name;
if __name__ =="__main__":
    app.run(debug = True)
```

- It will produce the following result on the web browser.

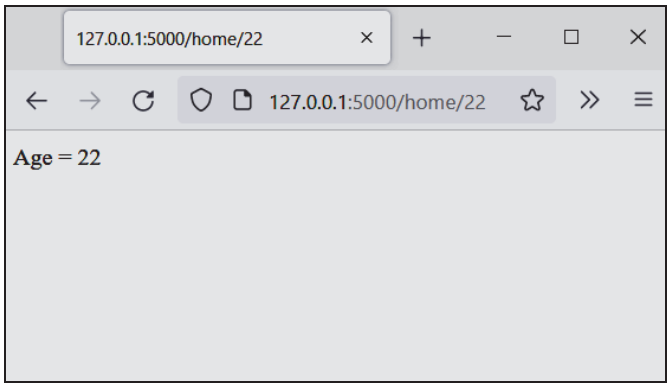


- The converter can also be used in the URL to map the specified variable to the particular data type. For example, we can provide the integers or float like age or salary respectively. To convert use <converter:variable_name>.

Example

```
from flask import Flask
app = Flask(__name__)
@app.route('/home/<int:age>')
def home(age):
    return "Age = %d"%age;
if __name__ =="__main__":
    app.run(debug = True)
```

Output



The following converters are used to convert the default string type to the associated data type.

- **string :** It is the default type and it accepts any text without a slash.
- **int :** It is used to convert the string to the integer.
- **float :** It is used to convert the string to the float.
- **path :** It can accept the slashes given in the URL.
- **uuid :** It accepts UUID strings.

The add_url_rule() function

The URL mapping can also be done using the add_url_rule() function. This function is mainly used in the case if the view function is not given and we need to connect a view function to an endpoint externally by using this function.

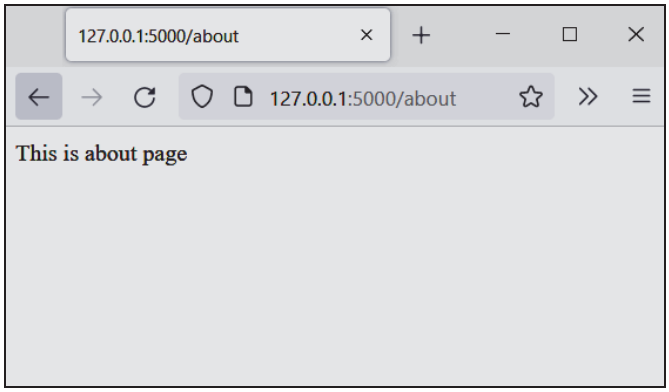
Syntax

```
add_url_rule(<url rule>, <endpoint>, <view function>)
```

Example

```
from flask import Flask
app = Flask(__name__)
def about():
    return "This is about page";
app.add_url_rule("/about","about",about)
if __name__ =="__main__":
    app.run(debug = True)
```

Output



5.4 FLASK URL BUILDING

- The url_for() function is used to build a URL to the specific function dynamically. The first argument is the name of the specified function, and then we can pass any number of keyword argument corresponding to the variable part of the URL.
- This function is useful in the sense that we can avoid hard-coding the URLs into the templates by dynamically building them using this function.

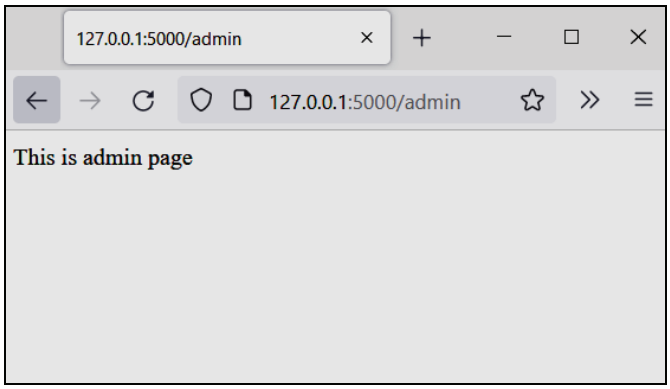
Example

```
from flask import *
app = Flask(__name__)
@app.route('/admin')
def admin():
    return 'This is admin page'
@app.route('/librarian')
def librarian():
    return 'This is librarian page'
@app.route('/student')
def student():
    return 'This is student page'
@app.route('/user/<name>')
def user(name):
    if name == 'admin':
        return redirect(url_for('admin'))
    if name == 'librarian':
        return redirect(url_for('librarian'))
    if name == 'student':
        return redirect(url_for('student'))
if __name__ =='__main__':
    app.run(debug = True)
```

Module 5

- The above example describes the library management system which can be used by the three types of users, i.e., admin, librarian, and student.
- There is a specific function named user() which recognizes the user the redirect the user to the exact function which contains the implementation for this particular function.

Output



- For example, the URL http://localhost:5000/user/admin is redirected to the URL http://localhost:5000/admin, the URL localhost:5000/user/librarian, is redirected to the URL http://localhost:5000/librarian, the URL http://localhost:5000/user/student is redirected to the URL http://localhost/student.

☞ **Benefits of the Dynamic URL Building**

1. It avoids hard coding of the URLs.
2. We can change the URLs dynamically instead of remembering the manually changed hard-coded URLs.
3. URL building handles the escaping of special characters and Unicode data transparently.
4. The generated paths are always absolute, avoiding unexpected behavior of relative paths in browsers.
5. If your application is placed outside the URL root, for example, in /myapplication instead of /, url_for() properly handles that for you.

⏭ 5.5 FLASK HTTP METHODS

- HTTP methods are the standard way of sending information to and from a web server. A website runs on a server or multiple servers and simple returns information to a client (web-browser).
- Information is exchanged between the client and the server using an HTTP protocol that has a few different methods.
- All web frameworks including flask need to provide several HTTP methods for data communication.
- The method is the type of action you want the request to perform and is sent from the client to the server on every request.
- Different methods for retrieving data from a specified URL are defined in HTTP protocol.
- There are several HTTP methods :
  - GET - Used to fetch the specified resource
  - POST - Used to create new data at the specified resource
  - PUT - Used to create new data or replace existing data at the specified resource
  - PATCH - Used to create new data or update/modify existing data at the specified resource
  - DELETE - Used to delete existing data at the specified resource
- Requesting a URL is an example of a GET request, where your browser makes a request for resources at a specified location (the URL) and the server returns some HTML. GET requests are "safe" as they aren't able to modify state or data on the server.
- An example use case of a POST request would be creating a new account on a website or application, whereby the resource doesn't already exist.
- By default, routes created with @app.route(/example) only listen and respond to GET requests and have to be instructed to listen and respond to other methods using the methods keyword & passing it a list of request methods.

5.5.1 POST Method

To handle the POST requests at the server, let us first create a form to get some data at the client side from the user, and we will try to access this data on the server by using the POST request.

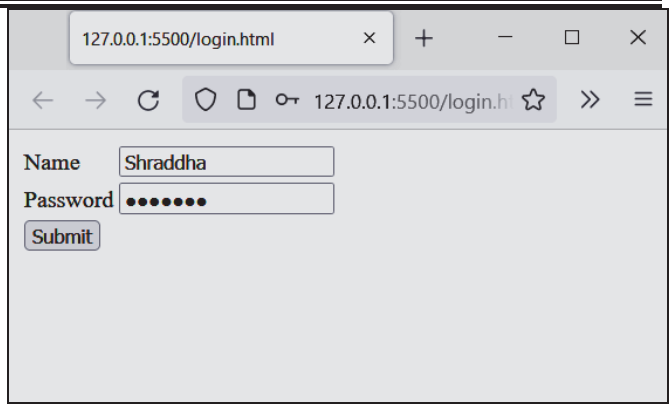
login.html

```
<html>
    <body>
    <form action = "http://localhost:5000/login" method =
    "post">
        <table>
        <tr><td>Name</td>
    <td><input type ="text" name "uname"></td></tr>
        <tr><td>Password</td>
        <td><input type ="password" name
        ="pass"></td></tr>
        <tr><td><input type = "submit"></td></tr>
    </table>
    </form>
</body>
</html>
```

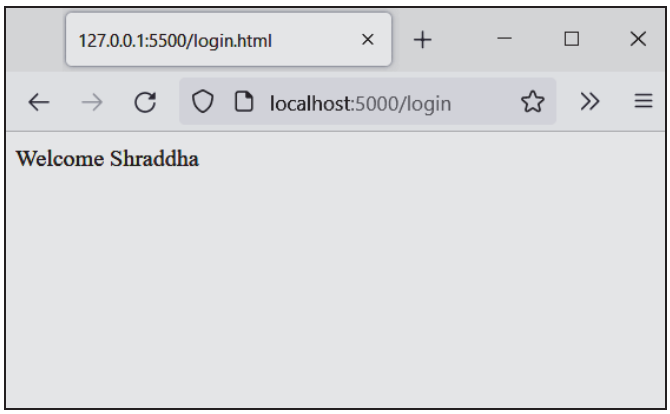
post.py

```
from flask import *
app = Flask(__name__)
    @app.route('/login',methods = ['POST'])
def login():
      uname=request.form['uname']
      password=request.form['pass']
      if uname=="Shraddha" and password=="saidham":
         return "Welcome %s" %uname
    if __name__ == '__main__':
    app.run(debug = True)
```

Now, start the development server by running the script using python pos.py and open login.html on the web browser as shown in the following image.



Give the required input and click Submit, we will get the following result.



Hence, the form data is sent to the development server by using the post method.

5.5.2 GET Method

Let's consider the same example for the Get method. However, there are some changes in the data retrieval syntax on the server side. First, create a form as login1.html.

login1.html

```
<html>
    <body>
      <form action = "http://localhost:5000/login1" method
= "get">
        <table>
        <tr><td>Name</td>
        <td><input type ="text" name
="uname"></td></tr>
        <tr><td>Password</td>
        <td><input type ="password" name
="pass"></td></tr>
        <tr><td><input type = "submit"
value="Submit"></td></tr>
```

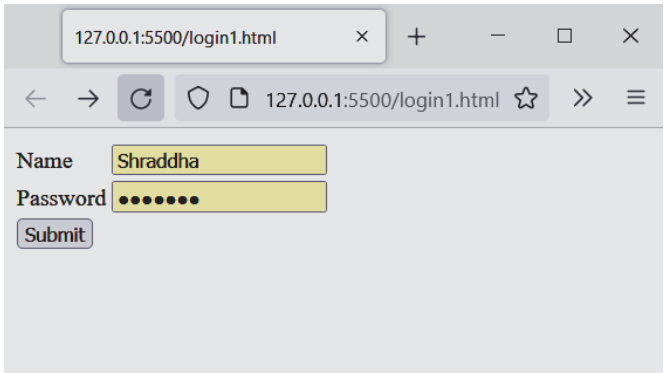
Module 5

```
    </table>
     </form>
  </body>
</html>
```

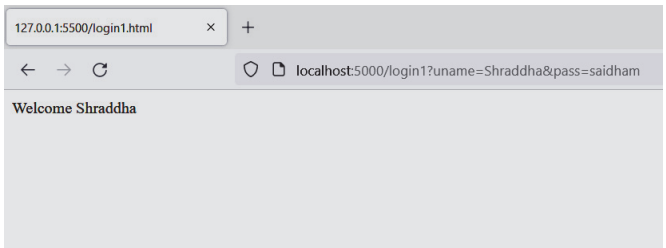
get.py

```
from flask import *
app = Flask(__name__)
@app.route('/login1',methods = ['GET'])
def login():
      uname=request.args.get('uname')
      password=request.args.get('pass')
      if uname=="Shraddha" and password=="saidham":
         return "Welcome %s" %uname
if __name__ == '__main__':
  app.run(debug = True)
```

Now, open the HTML file, login.html on the web browser and give the required input.



Now, click the submit button.



As we can check the result, the data sent using the get() method is retrieved on the development server.

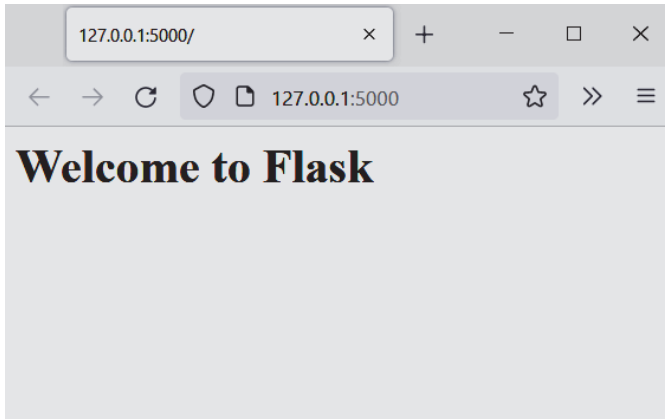
5.6 FLASK TEMPLATES

- Flask facilitates us to return the response in the form of HTML templates. There are different ways using which we can return the HTML response from the web applications.
- **Example :** The following flask script contains a view function, i.e., the message() which is associated with the URL '/'. Instead of returning a simple plain string as a message, it returns a message with <h1> tag attached to it using HTML.

script.py

```
from flask import *
app = Flask(__name__)
@app.route('/')
def message():
      return "<html><body><h1>Welcome to
Flask</h1></body></html>"
if __name__ == '__main__':
  app.run(debug = True)
```

Output



5.6.1 Rendering external HTML files

- Flask facilitates us to render the external HTML file instead of hardcoding the HTML in the view function. Here, we can take advantage of the jinja2 template engine on which the flask is based.
- Flask provides us the render_template() function which can be used to render the external HTML file to be returned as the response from the view function.
- **Example :** To render an HTML file from the view function, let's first create an HTML file named as message.html.

message.html

```
<html>
<head>
<title>Message</title>
</head>
<body>
<h1> Welcome to Flask </h1>
```

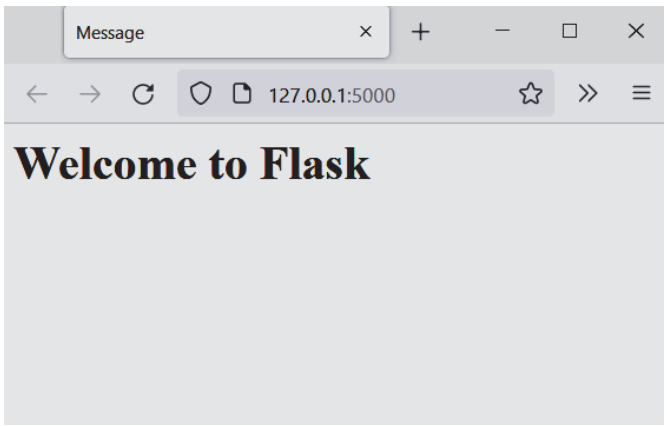
```
</body>
</html>
```

script.py

```
from flask import *
app = Flask(__name__)

@app.route('/')
def message():
        return render_template('message.html')
if __name__ == '__main__':
  app.run(debug = True)
```

Here, we must create the folder **templates** inside the application directory and save the HTML templates referenced in the flask script in that directory.

Output



5.6.2 Delimiters

- Jinga 2 template engine provides some delimiters which can be used in the HTML to make it capable of dynamic data representation. The template system provides some HTML syntax which are placeholders for variables and expressions that are replaced by their actual values when the template is rendered.
- The jinga2 template engine provides the following delimiters to escape from the HTML.
  - {% ... %} for statements
  - {{ ... }} for expressions to print to the template output
  - {# ... #} for the comments that are not included in the template output
  - # ... ## for line statements
- **Example :** Consider the following example where the variable part of the URL is shown in the HTML script using the {{ ... }} delimiter.

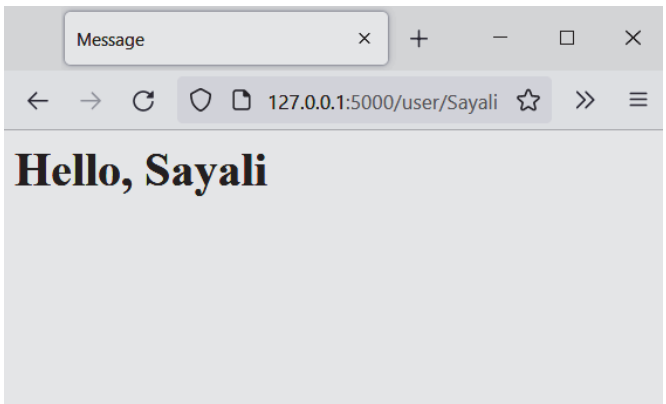
message.html

```
<html>
<head>
<title>Message</title>
</head>
<body>
<h1>Hello, {{ name }}</h1>
</body>
</html>
```

script.py

```
from flask import *
app = Flask(__name__)
@app.route('/user/<uname>')
def message(uname):
        return render_template('message.html',name=uname)
if __name__ == '__main__':
  app.run(debug = True)
```

Output



The variable part of the URL http://localhost:5000/user/Sayali is shown in the HTML script using the {{name}} placeholder.

5.7 FLASK REQUEST OBJECT

- In the client-server architecture, the request object contains all the data that is sent from the client to the server. In order to process the request data, it should be imported from the Flask module.
- Attributes of request object are listed below:
  - **form :** It is a dictionary object containing key and value pairs of form parameters and their values.

Module 5

- **args :** It is parsed contents of query string which is part of URL after question mark (?).
- **cookies :** It is the dictionary object holding Cookie names and values.
- **files :** It contains the data pertaining to uploaded file.
- **method :** It is the current request method (get or post).

- **Example :** In the following example, the / URL renders a web page customer.html that contains a form which is used to take the customer details as the input from the customer. The data filled in this form is posted to the /success URL which triggers the print_data() function. The print_data() function collects all the data from the request object and renders the result_data.html file which shows all the data on the web page.

script.py

```
from flask import *
app = Flask(__name__)

@app.route('/')
def customer():
  return render_template('customer.html')

@app.route('/success',methods = ['POST', 'GET'])
def print_data():
  if request.method == 'POST':
      result = request.form
      return render_template("result_data.html",result =
result)

if __name__ == '__main__':
  app.run(debug = True)
```

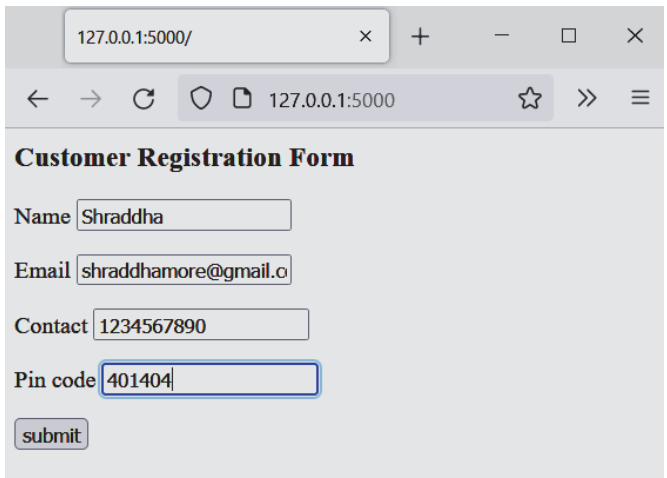
customer.html

```
<html>
  <body>
      <h3>Customer Registration Form</h3>
      <form action = "http://localhost:5000/success" method
= "POST">
          <p>Name <input type = "text" name = "name"
/></p>
        <p>Email <input type = "email" name = "email"
/></p>
        <p>Contact <input type = "text" name = "contact"
/></p>
        <p>Pin code <input type ="text" name = "pin"
/></p>
        <p><input type = "submit" value = "submit"
/></p>
      </form>
  </body>
</html>
```

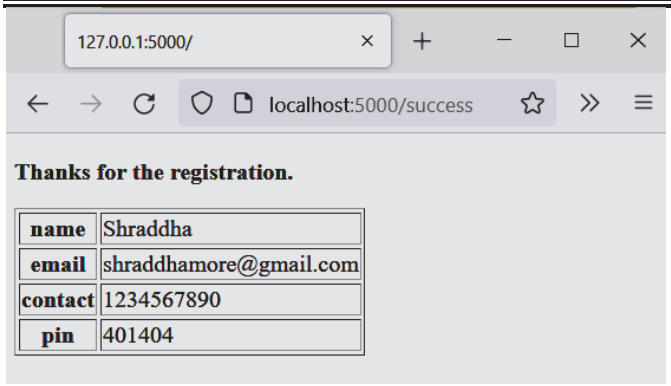
result_data.html

```
<!doctype html>
<html>
  <body>
      <p><strong>Thanks for the
registration.</strong></p>
      <table border = 1>
        {% for key, value in result.items() %}
          <tr>
            <th> {{ key }} </th>
            <td> {{ value }} </td>
          </tr>
        {% endfor %}
      </table>
  </body>
</html>
```

- To run this application, we must first run the script.py file using the command python script.py. This will start the development server on localhost:5000 which can be accessed on the browser as given below.



- Now, hit the submit button. It will transfer to the /success URL and shows the data entered at the client.



5.8 FLASK COOKIES

- The cookies are stored in the form of text files on the client's machine. Cookies are used to track the user's activities on the web and reflect some suggestions according to the user's choices to enhance the user's experience.
- Cookies are set by the server on the client's machine which will be associated with the client's request to that particular server in all future transactions until the lifetime of the cookie expires or it is deleted by the specific web page on the server.

Setting a Cookie

- In Flask, we use set_cookie() method of the response object to set cookies. The syntax of set_cookie() method is as follows:

```
set_cookie(key, value="", max_age=None)
```

- The key is a required argument and refers to the name of the cookie. The value is data you want to store in the cookie and it defaults to empty string. The max_age refers to the expiration time of the cookie in seconds, if not set the cookie will cease to exist when the user closes the browser.

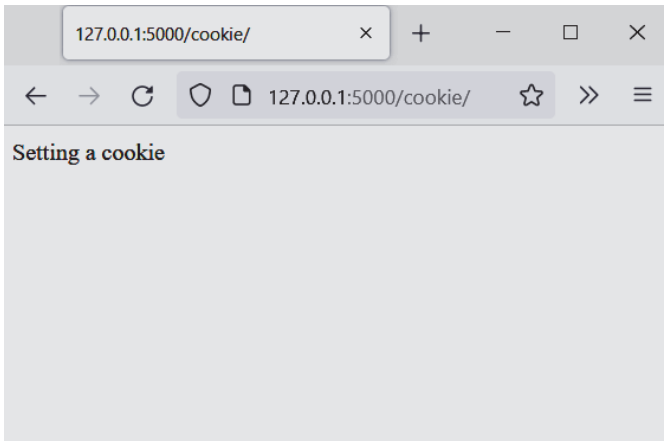
Accessing Cookies

- To access the cookie, we use the cookie attribute of the request object. The cookie attribute is a dictionary like attribute which contains all the cookies sent by the browser.

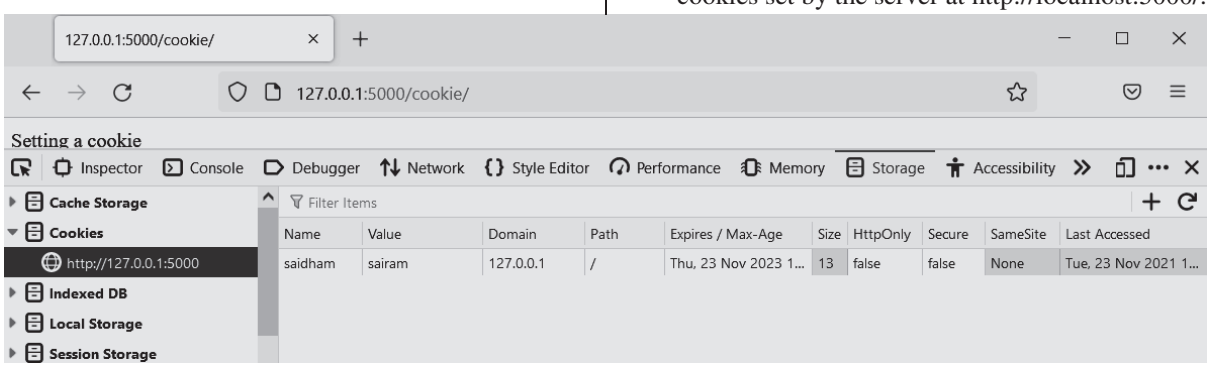
Example

```
from flask import *
 app = Flask(__name__)
@app.route('/cookie/')
def cookie():
      if not request.cookies.get('saidham'):
         res = make_response("Setting a cookie")
         res.set_cookie('saidham', 'sairam',
max_age=60*60*24*365*2)
      else:
        res = make_response("Value of cookie saidham is
{}".format(request.cookies.get('foo')))
      return res
 if __name__ == '__main__':
      app.run(debug = True)
```

Start the server and visit http://localhost:5000/cookie/.



- In above example we are creating a cookie named saidham with the value sairam that will last for 2 years.
- To view the cookie set by the server open Storage Inspector in Firefox by hitting Shift + F9. A new window will appear at the bottom of the browser window. On the left side, select "Cookies" storage type and then click on http://localhost:5000/ to view all cookies set by the server at http://localhost:5000/.



Module 5

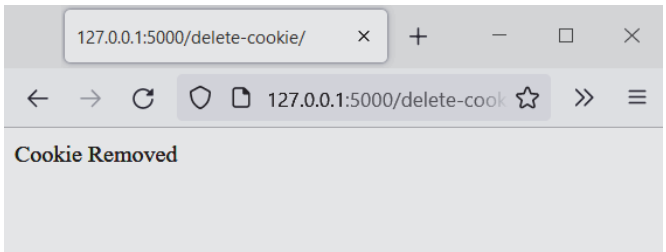
Deleting Cookies

To delete a cookie call set_cookie() method with the name of the cookie and any value and set the max_age argument to 0.

Example

```
from flask import *
app = Flask(__name__)
 @app.route('/delete-cookie/')
def delete_cookie():
      res = make_response("Cookie Removed")
      res.set_cookie('saidham', 'sairam', max_age=0)
      return res
if __name__ == '__main__':
      app.run(debug = True)
```

Visit http://localhost:5000/delete-cookie/ and you will get the following response:



5.9 FLASK FILE UPLOADING

- File uploading is the process of transmitting the binary or normal files to the server. Flask facilitates us to upload the files easily.
- It requires an HTML form whose enctype property is set to "multipart/form-data" to publish the file to the URL. The server-side flask script fetches the file from the request object using request.files[] Object. On successfully uploading the file, it is saved to the desired location on the server.
- Each uploaded file is first saved on a temporary location on the server, and then will actually be saved to its final location.
- The name of the target file can be hard-coded or available from the filename property of file request.files object. However, it is recommended that the secure_filename() function be used to obtain a secure version of it.
- The default upload folder path and maximum size of uploaded files can be defined in the configuration settings for the Flask object.
- Define the path to the upload folder

```
app.config['UPLOAD_FOLDER']
```

- Specifies the maximum size (in bytes) of the files to be uploaded

```
app.config['MAX_CONTENT_PATH']
```

- **Example :** In this example, we will provide a file selector(file_upload_form.html) to the user where the user can select a file from the file system and submit it to the server.
- At the server side, the file is fetched using the request.files['file'] object and saved to the location on the server.
- Since we are using the development server on the same device, hence the file will be uploaded to the directory from where the flask script upload.py is executed.

upload.py

```
from flask import *
app = Flask(__name__)

@app.route('/')
def upload():
      return render_template("file_upload_form.html")

@app.route('/success', methods = ['POST'])
def success():
      if request.method == 'POST':
         f = request.files['file']
         f.save(f.filename)
         return render_template("success.html", name =
f.filename)
if __name__ == '__main__':
      app.run(debug = True)
```

file_upload_form.html

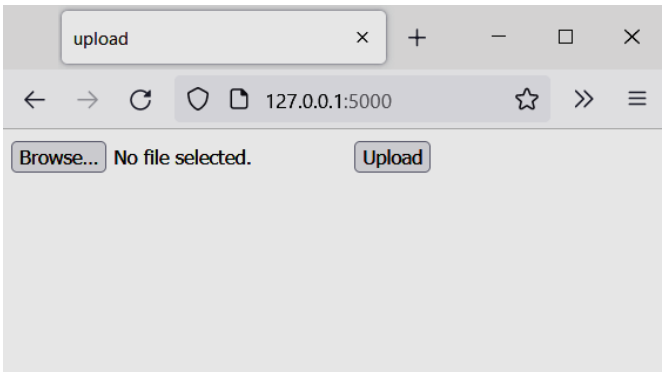
```
<html>
<head>
   <title>upload</title>
</head>
<body>
      <form action = "/success" method = "post"
enctype="multipart/form-data">
         <input type="file" name="file" />
         <input type = "submit" value="Upload">
      </form>
```

```
</body>
</html>
```

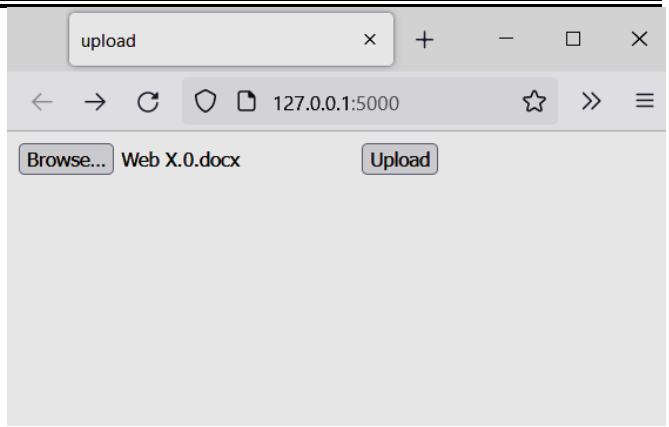
success.html

```
<html>
<head>
<title>success</title>
</head>
<body>
<p>File uploaded successfully</p>
<p>File Name: {{name}}</p>
</body>
</html>
```

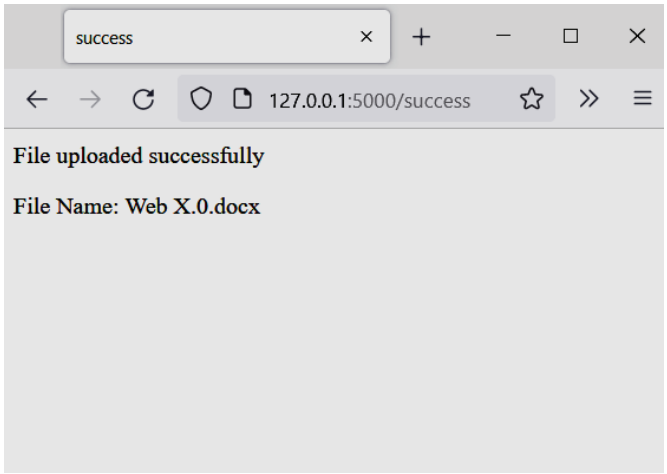
- HTML form is shown to the user so that the user can browse the file system for the file which will be uploaded to the development server.



- Here, the user has chosen a file named as galexy.jpg which will be uploaded to the server.



- On successfully uploading the file, a success message is shown to the user with the name of the uploaded file.



5.10 SELF-LEARNING : FLASK VS DJANGO

Parameter	Django	Flask
Type of framework	Django is a full-stack web framework that enables ready to use solutions with its batteries-included approach.	Flask is a lightweight framework that gives abundant features without external libraries and minimalist features.
Working of Framework/ Data Model	Django follows an object-oriented approach that enables object-relational mapping (linking databases and tables with classes)	Flask works on a modular approach that enables working through outsourced libraries and extensions.
Project Layout	Django is suitable for multiple page applications.	Flask is suitable for only single-page applications.
Bootstrapping Tool	Django-admin is the in-built bootstrapping tool of Django that allows the creation of web applications without any external input.	Flask does not come with an in-built bootstrapping tool.
Database Support	Django supports the most popular relational database management systems like MySQL, Oracle etc.	Flask does not support the basic database management system and uses SQLAlchemy for database requirements.

Parameter	Django	Flask
Flexibility	Django is less flexible because of its in-built features and tools. Developers cannot make changes to the modules.	Flask is a micro-based framework with extensible libraries making itself a flexible framework for developers.
Template Engine	Django is inspired by the Jinja2 template but has its built-in model view template that makes the development process easier.	Flask used Jinja2 template design
Control	Developers do not have full control over the modules and functions of Django because of built-in libraries.	Flask allows developers full control over the creation of applications with no dependencies from external libraries.
Working Style	The working style of Django is Monolithic	The working style of Flask is diversified style.
Debugger	Django does not support any virtual debugging.	Flash has an in-built debugger that offers virtual debugging
Routing and Views	Django framework supports the mapping of URL to views through a request.	Flask web framework allows mapping of URL to class-based view with Werkzeug.
Structure	Django framework structure is more conventional.	Flask web framework structure is random.
HTML	Django supports dynamic HTML pages	Flask framework does not support dynamic HTML pages
Usage	Django is suitable for high-end technology companies like Instagram, Udemy, Coursera etc.	Flask is suitable for companies and projects that want experimentation with the module, architecture of the framework like Netflix, Reddit, Airbnb, etc.

Review Questions

Q. 1 What is Python Flask?

Q. 2 How to start Python Flask app?

Q. 3 What are the features of Python Flask?

Q. 4 What are the advantages of Python Flask?

Q. 5 What is app routing in Python Flask?

Q. 6 Explain URL building in Python Flask.

Q. 7 What are the HTTP methods provided by Python Flask?

Q. 8 Explain Flask templates with an example.

Q. 9 Describe Flask Request object in detail.

Q. 10 How to set, access and delete cookies in Python Flask ?

Q. 11 How to handle file uploading in Python Flask?

Q. 12 Differentiate between Flask and Django frameworks.

Chapter Ends…

❑❑❑

MODULE 6

CHAPTER 6

Rich Internet Application

University Prescribed Syllabus w.e.f Academic Year 2021-2022

AJAX : Introduction and Working

Developing RIA using AJAX Techniques : CSS, HTML, DOM, XML HTTP Request, JavaScript, PHP, AJAX as REST Client

Introduction to Open Source Frameworks and CMS for RIA : Django, Drupal, Joomla

Self-learning Topics : Applications of AJAX in Blogs, Wikis and RSS Feeds

⏭ 6.1 INTRODUCTION TO AJAX

- AJAX stands for Asynchronous JavaScript and XML.
- AJAX or Asynchronous JavaScript and XML is a set of web development techniques which allows web applications to work asynchronously.
- It is used to communicate with the server without refreshing the web page and thus increasing the user experience and better performance.
- Ajax is made up of the following technologies:
  (1) XHTML and CSS for presenting information.
  (2) Document Object Model (DOM) for dynamically interacting with and displaying the presented information.
  (3) XMLHttpRequest object to manipulate data asynchronously with the web server.
  (4) XML, HTML, and XSLT for data interchange and manipulation.
  (5) JavaScript for binding data requests and information display.
- AJAX allows the web pages to be modified asynchronously by exchanging small bunch of information with the server.
- This means that it is easily possible to make changes in part of a web page, without reloading the entire page. The conventional web pages, (which are not using AJAX) reload the whole page if there is need to make changes in the content.

☞ Examples of Applications using AJAX

- Google Maps, Gmail, Youtube, and Facebook tabs.
- You may have seen the live score of cricket match on different websites like www.yahoo.com.
- There are updations of each and every ball. In such case only small part of webpage is needed to be changed.
- Only that much information is sent by the server and updated in the webpage rather than updating the entire page. This is implemented by the AJAX.
- Just consider if it is done with conventional web pages without AJAX, then it will take lot much of duration every time to reload the whole page.

☞ Features of Ajax

- User Friendly
- It make web page faster
- Independent of server technology
- Increase the Performance of web page
- Support for Live data binding
- Support for the Data View control
- Support for Client-side template rendering
- Rich and, responsive user interfaces
- Reduced consumption of server resources
- No need to pushing on a submit button and reloading the complete website.
- No need to reload the whole page only some part of page is reloaded which is required to reload.
- Apart from obtaining the XMLHTTP object, all processing is same for all browser types, because JavaScript is used.
- Using AJAX, develop faster and more interactive web applications.
- Not require to completely reload page due to this server uses less bandwidth.

For understanding the working of AJAX, let's first understand the classical model for web development i.e. synchronous model.

Synchronous (Classic Web-Application Model)

The synchronous model can be classified into three stages :

(1) On the client side, if the user does any activity on the browser, the browser sends a HTTP request to the server and waits for a response.

(2) The server does some processing on the request of the user and sends a response to the user.

(3) When the response is received at the user end, the browser reloads the webpage to show the information.

Fig. 6.1.1 illustrate the classical model or a synchronous model.

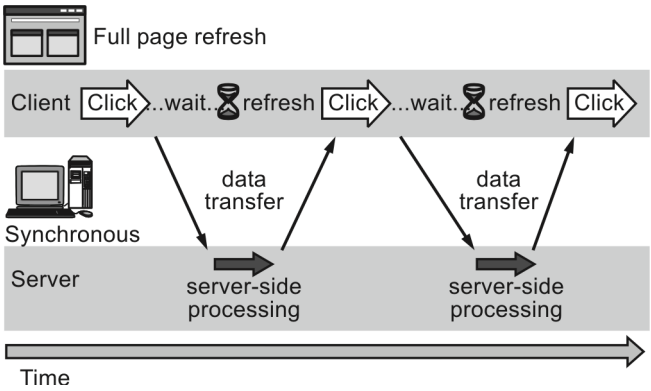


Fig. 6.1.1 : Synchronous Model

- Fig. 6.1.1 depicts a synchronous model, where the request blocks the client until the particular operation completes such as wait or refresh.
- Fig. 6.1.2 shows the process and format of data transfer involved in a synchronous model.

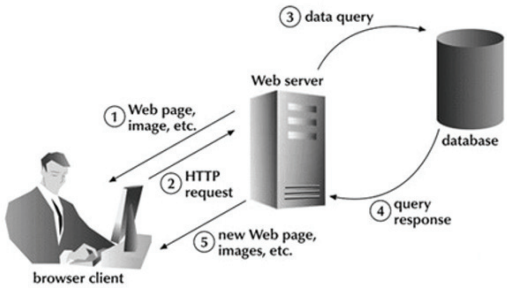


Fig. 6.1.2 : Process involved in the synchronous model

- In Fig. 6.1.2, the browser directly sends a request to the server in the form of an HTTP request. After processing the request, the server sends the response to the browser in the form of HTML and CSS data.

Asynchronous (AJAX Web-Application Model)

- AJAX asynchronous model eliminates the waiting process of the client after each event, by introducing an AJAX engine between the user and the server. This AJAX communicates with the server on the user's behalf.
- Fig. 6.1.3 shows the AJAX asynchronous model.

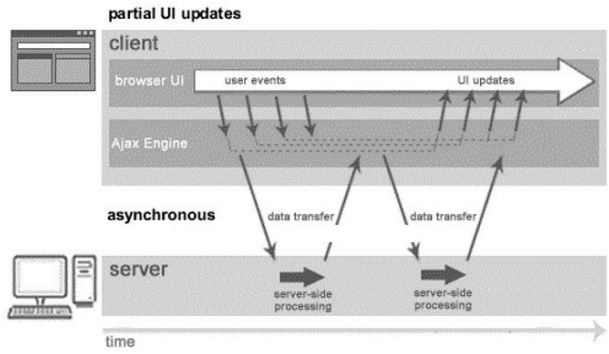


Fig. 6.1.3 : AJAX Asynchronous model

- Fig. 6.1.3 illustrate the AJAX asynchronous model, where the request doesn't block the client. This means that the browser responds to the client's request instantaneously. Fig. 6.1.4 depicts the process involved in AJAX asynchronous model.

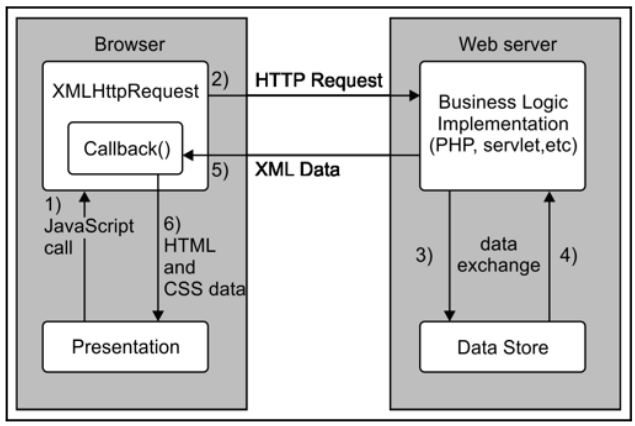


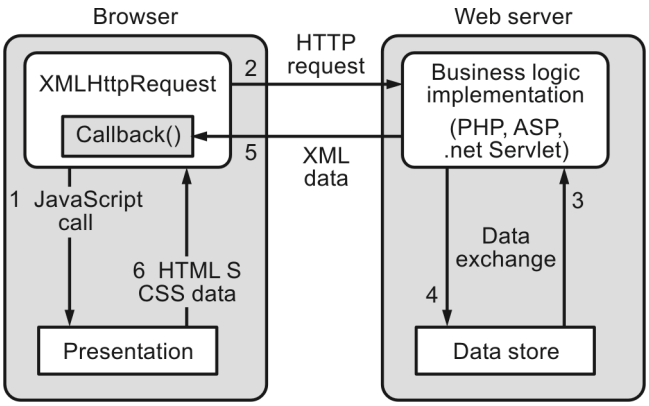
Fig. 6.1.4 : Process involved in AJAX Asynchronous model

- An AJAX asynchronous model generates a user action by invoking JavaScript call.
- This JavaScript call is handled by the AJAX engine, which converts this call to XMLHttpRequest. AJAX engine sends request to the server.
- The server does some processing according to the request and sends a response to the AJAX engine in the form of XML data. AJAX engine receives XML data and converts it to HTML and CSS and shows it on the webpages.

Module 6

⏭ 6.2 WORKING OF AJAX

- Ajax communicates with the server by using XMLHttpRequest Object. User send request from User Interface and JavaScript call goes to the XMLHttpRequest Object after that XMLHttp request is sent to the XMLHttpRequest Object.
- At that time server interacts with the database using php, servelet, ASP.net etc.
- The data is retrieved then the server sends data in the form of XML or JSON data to the XMLHttpRequest Callback function. Then HTML and CSS displayed the data on the browser.



(1F1)Fig. 6.2.1 : Working of AJAX

The functioning is as follows :

(1) User sends a request from the UI and a javascript call goes to XMLHttpRequest object.

(2) HTTP Request is sent to the server by XMLHttpRequest object.

(3) Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.

(4) Data is retrieved.

(5) Server sends XML data or JSON data to the XMLHttpRequest callback function.

(6) HTML and CSS data is displayed on the browser.

- In Ajax model, there is an Ajax engine involved in between the user and the server, which eliminates the to and fro from the user to the server and back.
- This Ajax engine is written in JavaScript and is in a hidden frame. It handles the user front by communicating to the user as well as handles the server front by itself.
- In the working of AJAX, the two aspects are very important: AJAX Request and AJAX Response.

☞ AJAX Request

- AJAX sends a request to a server by using open() and send() methods of the XMLHttpRequest object.
- Following code snippet illustrates how an AJAX request can be sent to the server.

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

- In the above code snippet, the **GET** is the request type, **ajax_info** is the location of the server type and **true** shows that the request is asynchronous.
- Following are the methods used to send a request in AJAX.

(1) Open(method, url, async): Specifies the type of request

- method : the type of request: GET or POST
- url : the server (file) location
- async : true (asynchronous) or false (synchronous)

(2) send() : Sends the request to the server (used for GET)

(3) send(string): Sends the request to the server (used for POST)

☞ AJAX Response

- The readyState property holds the status of the XMLHttpRequest.
- The onreadystatechange property defines a function to be executed when the readyState changes as shown in the following code snippet.

```
var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
```

- In the above code snippet, the response is decided by two properties of XMLHttpRequest: readyState and status. When the status is 200 and readyState is 4, the response is ready.

- The status property and the statusText property holds the status of the XMLHttpRequest object.

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200 : "OK" 403 : "Forbidden" 404 : "Page not found"
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Server Response Properties

There are two server response properties, which are as follows:

(1) responseText : Returns the response from the server as a JavaScript string.

(2) responseXML : Returns the response from the server as an XML DOM object.

Server response Methods

- There are two server response methods, which are as follows :
  (1) getResponseHeader() : Returns specific header information from the server resource.
  (2) getAllResponseHeaders() : Returns all the header information from the server resource.

  The AJX request and response functions can be used with XMLHttpRequest API.

XMLHttpRequest API

- AJAX comprises important API as XMLHttpRequest, which transfers XML data using HTTP to and from the web server.
- XMLHttpRequest is an API, which was initially built by Microsoft.
- It is used by various scripting languages such as JavaScript, VBScript, etc.
- The XMLHttpRequest object is used to exchange data with a web server in the background.
- This means that it is possible to update parts of a web page, without reloading the whole page.

XMLHttpRequest Object Methods

Methods	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(method, url, async, user, psw)	Specifies the request method: the request type GET or POST url: the file location async: true (asynchronous) or false (synchronous) user: optional user name psw: optional password
send()	Sends the request to the server. Used for GET requests
send(string)	Sends the request to the server.
setRequestHeader()	Adds a label/value pair to the header to be sent

Program : Create AJAX Application

Create a simple XMLHttpRequest, and retrieve data from a txt file.

Module 6

1. Create a file Ajax.html

```
<!DOCTYPE html>
<html>
<body>

<div id="demo">
<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Change
Content</button>
</div>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>

</body>
</html>
```

2. Create a text file ajax_info.txt

```
AJAX is not a programming language.
AJAX is a technique for accessing web servers from a web
page.
AJAX stands for Asynchronous JavaScript and XML.
```

3. View Ajax.html on the browser as an output

4. Click the Change Content button. As soon as, the Change Content button is clicked, the text appears from the ajax_info.txt on the same page as shown in as follows :

AJAX is not a programming language.

AJAX is a technique for accessing web servers from a web page.

AJAX stands for Asynchronous JavaScript And XML.

- Thus the content is displayed on the same page, with the help of the XMLHttpRequest object.
- The contents are updated partially on the webpage instead of refreshing the whole page.

6.3 RICH INTERNET APPLICATIONS (RIA)

- RIA is the web application that offers look, feel and usability of desktop applications.
- The key attributes of RIA are performance and rich GUI (Graphical User Interface).
- RIA can use the data that is processed by both the client and the server.
- The data is exchanged in the asynchronous ways that helps in continuously updating the user interface partially.
- RIA offers a similar look and feel as that of desktop applications.
- RIAs can run faster and be more engaging.
- They can offer users a better visual experience, better accessibility, usability and more interactivity than traditional browser applications that use only HTML and HTTP.
- RIA is developed using various technologies such as Java, Silverlight, JavaFX, JavaScript, REST/WS etc.

Characteristics of RIA

1. **Performance :** RIAs can often perform better than traditional applications on the basis of the characteristics of network and applications, performance of server also improved by offloading possible processing work to the client system and also perceived performance in terms of UI responsiveness and smoother visual transitions and animations are key aspects of any RIA.
2. **Offline use :** When connectivity is unavailable, it might still be possible to use an RIA. An RIA platform let the user work with the application without

connecting to the internet and synchronizing it automatically when the user goes live.

3. **Consistency of look and feel :** With RIA tools, the user interface and experience with different browsers, devices and operating systems can be more carefully controlled and made consistent.
4. **Security :** RIAs should be as secure as any other web application, and the framework should be well equipped to enforce limitations appropriately when the user lacks the required privileges, especially when running within a constrained environment such as a sandbox.
5. **Advanced Communications** : Sophisticated communications with supporting servers through optimized network protocols can considerably enhance the user experience.
6. **Rapid Development :** An RIA Framework should facilitate rapid development of a rich user experience through its easy-to-use interfaces in ways that help developers.
7. **Direct Interaction :** An RIA can use a wider range of controls that allow greater efficiency and enhance the user experience. In RIAs, for example, users can interact directly with page elements through editing or drag-and-drop tools. They can also do things like pan across a map or other image.
8. **Better Feedback :** Because of their ability to change parts of pages without reloading, RIAs can provide the user with fast and accurate feedback, real-time confirmation of actions and choices, and informative and detailed error messages.
9. **Improved Features :** RIA allow programmers to embed various functionalities in graphics-based web pages that look fascinating and engaging like desktop applications. RIA provide complex application screens on which various mixed media, including different fonts, vector graphic and bitmap files online conferencing etc. are paused by using different modern development tools.
10. **Partial-page updating :** RIAs incorporate additional technologies, such as real-time streaming, high-performance client-side virtual machines, and local caching mechanisms that reduce latency (wait times) and increase responsiveness.

6.4 HTML

- The dynamic nature of HTML with the help of client-side scripting language and DOM helps in creating an interactive and user-friendly web application.
- These interactive web applications do not require a server side page refresh. For instance, page elements can be inserted or deleted dynamically with the help of CSS and modified through a JavaScript function.
- HTML 5 is the latest version of HTML, the universal mark-up language that is used to create webpages. HTML 5 comes loaded with features which make it quite popular among web programmers and technologists.
- It makes the content on the web more interactive and allows for cross-platform compatibility across devices.
- HTML is a mark-up language which defines the structure of a webpage. It creates a mark-up using HTML tags which tell the web browser what should be displayed on a web browser and how. HTML can be viewed in two forms :
  - **(1) Content View:** This view shows the content of a webpage without HTML tags.
  - **(2) HTML Source Code:** This view shows the content as well as the HTML tags used to create the content.
- A webpage on a browser is divided into two parts of HTML code :
  - **(1) Head :** The head portion contains the title of a webpage, the scripts used and the CSS style definition that defines the look of the webpage. The Head contains the tags that define metadata on the webpage.
  - **(2) Body :** The body of a webpage contains the content displayed on the webpage. It contains the HTML tags for creating and formatting the content of the webpage.

HTML has the following advantages :

(1) It facilitates better interactions with the content.

(2) It enables the development of mobile-friendly and interactive games.

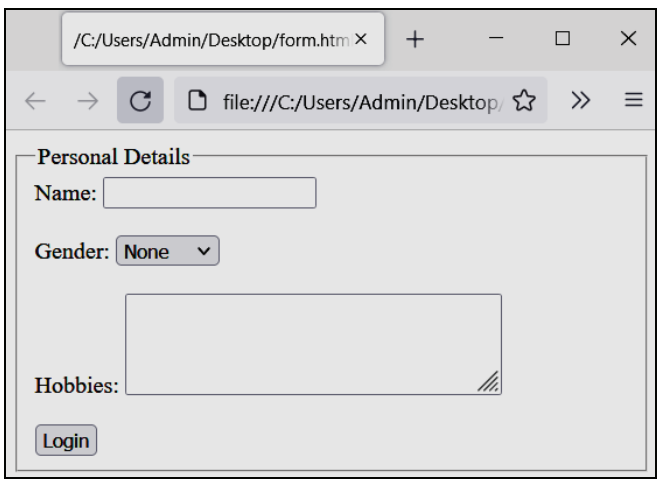
Module 6

(3) After flash became obsolete, HTML 5 has become a very powerful tool for mobile web applications development.

☞ **Program : Code to design a simple login form with the help of HTML tags.**

```
</doctype html>
<html>
<body>
<form action="" method="get">
<fieldset>
<legend>Personal Details</legend>
Name: <input type="text" name="user">
<br><br>
Gender: <select name="gender">
<option value="none" selected>None</option>
<option value="male">Male</option>
<option value="female">Female</option>
</select>
<br><br>
Hobbies: <textarea name="hobbies" rows="3" cols="30"
></textarea>
<br><br>
<input type="submit" value="Login">
</fieldset>
</form>
</body>
</html>
```

Output



6.5 CSS

- A Style Sheet is a collection of style rules that that tells a browser how the various styles are to be applied to the HTML tags to present the document.
- CSS - Cascading Style Sheet is a simple design scripting language intended to simplify the process of making the web pages attractive with high level formatting.
- CSS is used to manage the look and feel of the web pages. CSS is used to control the color of the text, the style of fonts, the way columns are sized and laid out, different background images or colors used, the spacing between paragraphs, layout designs, and variations in the display for various devices, screen sizes or resolutions and variety of other effects.
- The CSS is very easy to learn and understand but it has strong control over the presentation of web page. In general, CSS is integrated with the markup languages like HTML or XHTML.
- In simple words, CSS is nothing but declaration of style sheets which can be repeatedly used. It resembles to the function concept of C language. That means define once and use anytime anywhere repeatedly

Some of the main advantages of CSS are as follows :

(1) Time-saving : A CSS style sheet can be reused in multiple HTML pages. One style can be defined for each HTML element and applied to multiple pages.

(2) Fast page loading : A CSS style can be applied to many HTML tags. So, less code means faster page loading.

(3) Easy maintenance : If you want to apply a design change to the whole webpage, you can change the style and all the elements on the webpage will be updated automatically.

(4) Superior styles to HTML : CSS gives a webpage far better look than HTML because it has a wide array of attributes than HTML.

(5) Multiple device support : CSS helps in optimizing the content for more than one device. A website can easily be shown on different devices by using the same HTML document.

(6) Offline browsing : You can view offline website because CSS can store web applications locally with the help of an offline cache. It ensures faster loading and better performance of the website.

(8) Platform independence: It is platform independent and can support the latest browsers.

A CSS file has style rules that browser interprets and then apply to the corresponding elements in the document. A style rule has three parts which are as follows :

(1) Selector : It is an HTML tag where a style will be applied. It can be any tag like <table>, <h1>, etc.

(2) Property : It is a type of an attribute of an HTML tag. All HTML tags can be converted into CSS properties.

(3) Value : Values are assigned to the properties. For example, font-size property can have 28 px or 32 px, etc.

The following code snippet illustrates the syntax of the CSS file :

```
selector {property: value ;}
```

- The above code snippet illustrates the syntax for adding a CSS code.
- There are three ways of inserting a style sheet in the webpage :

(1) Inline Style Sheet

- The Inline style is specific for the individual tag. The HTML "style" attribute is used by the inline style to style a particular tag.
- The Inline style generally useful for an individual CSS change which we do not want to use repeatedly in the site.

(2) Internal Style Sheet

- In internal style sheet, the CSS code is usually written in the head section of the webpage.
- This simplifies the applications of styles like classes or id's in contrast to repeated use of the code. While creating an internal style sheet in the web page, we have to use the <style></style> HTML tags in the Head section of the webpage.
- The entire code of the Internal CSS style sheet is included between the <head></head> section of the websites.

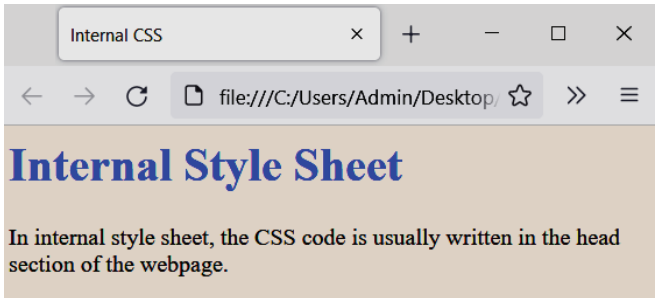
(3) External Style Sheet

- The External Style sheet is a file with extension .css which we link to website. This CSS file is used to declare the style sheets.
- Whenever any change is made in this css file, it gets reflected in all the web pages of the website. This simplifies our work and avoids making change in each and every page where the css is used.
- These are the styles which are written in a separate document or file and then linked to various web pages.
- External style sheets affects all the documents which are attached to, this means that if we have a 100-page website where all the pages use the same style sheet, just change in the style sheets will make visual change in all the web pages which are linked to it.

☞ Program : Code to add CSS code in HTML file.

```
<html>
<head>
<title>
Internal CSS
</title>
<style type="text/css">
body {background-color: linen}
h1 {color:blue;margin-left:30px>
p {color:red;font-size:18px>
</style>
</head>
<body>
<h1>Internal Style Sheet</h1>
<p>In internal style sheet, the CSS code is usually written in the head section of the webpage.</p>
</body>
</html>
```

Output



Module 6

6.6 JAVASCRIPT

- JavaScript is a protocol based, object-oriented and cross-platform language.
- All the web browsers support JavaScript, as it eliminates the requirement of a plug-in for AJAX applications. JavaScript is a lightweight, dynamic programming language.
- When it is used with HTML, it provides dynamic functionalities to websites, such as changing the content at the runtime.
- JavaScript has object-oriented capabilities; that's why it is compatible with Java. JavaScript and Java have similar names, but they are completely different from each other.
- JavaScript came in 1995 with the name LiveScript. It was invented by Brendan Eich and developed by Netscape Communication.
- It was renamed as JavaScript for taking advantage of the name Java because at that time Java was very famous for web development.
- JavaScript became ECMA (European Computer Manufacturer's Association) standard in 1997. ECMA standard specifies the core features that a scripting language should provide and how those features should be implemented.
- The name of the standard was ECMA-262, and the name of the language was ECMAScript.

JavaScript code is written between <script> and </script>. When browser finds <script> tag, it starts interpreting the code until it finds </script> tag. <script> tag has two important attributes :

(1) Language : The language attribute denotes the name of the scripting language. It's value is JavaScript.

(2) Type : The type attribute denotes the type of content between the <script> and </script> tags. Its value can be text/JavaScript.

The following code snippet demonstrates the standard JavaScript code :

```
<script language="JavaScript" type= "text/JavaScript">
Write JavaScript code here.
</script>
```

Some of the important points as related with JavaScript are illustrated as follows :

(1) JavaScript is a case-sensitive language, i.e., text and TEXT will be considered different in JavaScript.

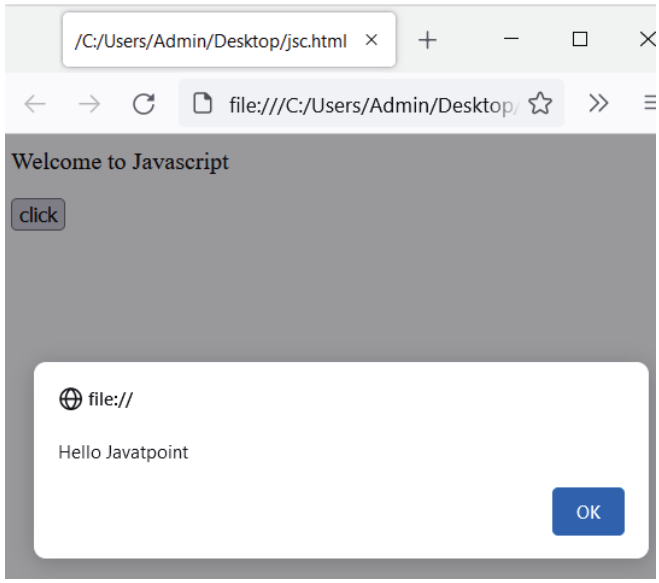
(2) JavaScript interpreter ignores the space, newline or tab occurrence in JavaScript program.

(3) The semi-colon is optional if we write each statement in a separate line.

☞ **Program : Code to add JavaScript in both head section.**

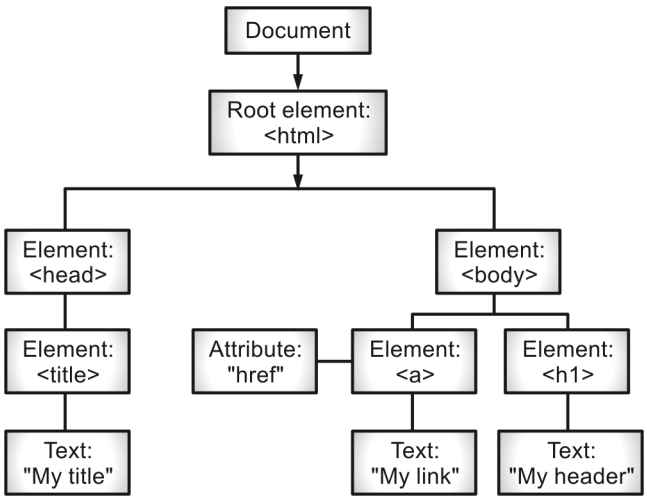
```
<html>
<head>
<script type="text/javascript">
function msg(){
 alert("Hello Javatpoint");
}
</script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

Output



6.7 DOCUMENT OBJECT MODEL (DOM)

- DOM is a standard made by World Wide Web Consortium (W3C) for web developers and browser companies. This standard defines certain rules for accessing documents.
- The W3C has three sections :
  - **(1) DOM Core :** In this section, standard is defined for modifying elements, attributes and structure of HTML document dynamically.
  - **(2) XML DOM :** It is the standard for an XML document.
  - **(3) HTML DOM :** It is the standard for HTML document. The HTML DOM defines all the elements of HTML as an object. It also defines properties and methods to access the element.
- When a web browser displays a webpage, the browser parses it. After parsing, the browser creates an object that defines the document and details how that page will be displayed. This object is known as a document object.
- DOM is represented as a tree which has its root at the top. This tree has several types of nodes such as the element node, attribute node and text node. These are HTML elements. In JavaScript, all the elements are considered as objects.
- The advantage of this tree representation of elements is to create a logical structures so that the target element can be easily accessible.
- Fig. 6.7.1 illustrate the tree structure of DOM.



(1F2)**Fig. 6.7.1 : DOM Tree Structure**

Let's understand the three types of nodes mentioned in Fig. 6.7.1.

(1) Element node : Element node occupies a major part of the DOM tree. These nodes define the structure of a document.

(2) Attribute node : Attribute nodes are treated and accessed differently. These nodes are not the child of the element nodes.

(3) Text node : Text nodes do not have a child node, and they are contained in an element node.

☞ **Program : Code to change old text with new text using DOM**

```
<html>
<body>
<H1> The old text is changed by new text.</H1>
<p id="p1">Old Text!!!</p>
<script>
document.getElementById("p1").innerHTML = "New text!!!";
</script>
</body>
</html>
```

Output

In above program, the old text has been replaced by the new text when the webpage loads.

6.8 XMLHTTP REQUEST

- The another important component of AJAX is XMLHttpRequest, which transfers XML data using HTTP to and from the web server.
- XMLHttpRequest is used by various scripting languages such as JavaScript, VBScript, etc. XMLHttpRequest object is used to transfer XML data in the background and hence, the webpage does not need to be reloaded.
- XMLHTTPRequest object is an API which is used for fetching data from the server. XMLHTTPRequest is

Module 6

basically used in Ajax programming.

- It retrieve any type of data such as JSON, XML, text etc. It request for data in background and update the page without reloading page on client side. An object of XMLHTTPRequest is used for asynchronous communication between client and server.

6.9 PHP

- The use of AJAX in web application development is to make an application faster, better and more interactive. AJAX can be integrated with many server-side programming languages such as PHP, JAVA and .NET.
- The full form of PHP is Hypertext Pre-processor. It is a server-side scripting language designed for web development. For integrating AJAX and PHP, you can either write the AJAX code (JavaScript code) in a PHP file or make a separate file for the AJAX code. When AJAX is used with PHP in a webpage, the webpage shows the result without refreshing. For example, Google map or suggestion of an input in the Google search box.

☞ **Program : Returning the text from PHP with AJAX.**

Data.html

```
<!DOCTYPE html>
<html>

<body>
    <button type="button" id="fetchBtn">Click
Me!</button>
    <p id="txt"></p>



    <script>
    let fetchBtn = document.getElementById('fetchBtn');
    fetchBtn.addEventListener('click', buttonClickHandler);

    function buttonClickHandler() {

        // Instantiate an xhr object
        var xhr = new XMLHttpRequest();

        // What to do when response is ready
        xhr.onreadystatechange = () => {
            if(xhr.readyState === 4) {
                if(xhr.status === 200) {

    document.getElementById("txt").innerHTML =
                        xhr.responseText;
                } else {
                    console.log('Error Code: ' +
xhr.status);
                    console.log('Error Message: ' +
xhr.statusText);
                }
            }
        }
        xhr.open('GET', 'data.php');

        // Send the request
        xhr.send();
    }
    </script>
</body>
</html>
```

data.php

```
<html>
<body>
<?php echo '<h1>Welcome to GfG</h1>'; ?>
</body>
</html>
```

Output

☞ **Program : The following example will demonstrate how a web page can communicate with a web server while a user type characters in an input field.**

In the example below, when a user types a character in the input field, a function called "showHint()" is executed.

The function is triggered by the onkeyup event.

Here is the HTML code:<!DOCTYPE html>

```
<html>
<head>
<script>
function showHint(str) {
 if (str.length == 0) {
  document.getElementById("txtHint").innerHTML = "";
  return;
 } else {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200) {
    document.getElementById("txtHint").innerHTML =
this.responseText;
   }
  }
  xmlhttp.open("GET", "gethint.php?q="+str, true);
  xmlhttp.send();
 }
}
</script>
</head>
<body>

<p><b>Start typing a name in the input field
below:</b></p>
<form action="">
<label for="fname">First name:</label>
<input type="text" id="fname" name="fname"
onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>

</body>
</html>
```

Code explanation

- First, check if the input field is empty (str.length == 0). If it is, clear the content of the txtHint placeholder and exit the function.
- However, if the input field is not empty, do the following:
- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a PHP file (gethint.php) on the server
- Notice that q parameter is added to the url (gethint.php?q="+str)
- And the str variable holds the content of the input field

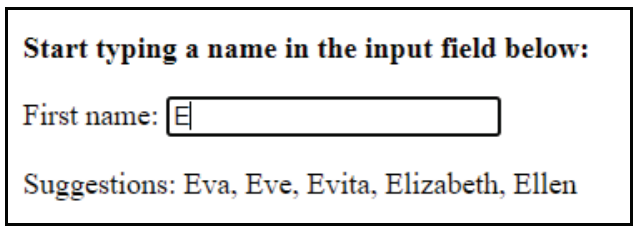
The PHP File - "gethint.php"

- The PHP file checks an array of names, and returns the corresponding name(s) to the browser:

```
<?php
// Array with names
$a[] = "Anna";
$a[] = "Brittany";
$a[] = "Cinderella";
$a[] = "Diana";
$a[] = "Eva";
$a[] = "Fiona";
$a[] = "Gunda";
$a[] = "Hege";
$a[] = "Inga";
$a[] = "Johanna";
$a[] = "Kitty";
$a[] = "Linda";
$a[] = "Nina";
$a[] = "Ophelia";
$a[] = "Petunia";
$a[] = "Amanda";
$a[] = "Raquel";
$a[] = "Cindy";
$a[] = "Doris";
$a[] = "Eve";
$a[] = "Evita";
$a[] = "Sunniva";
$a[] = "Tove";
$a[] = "Unni";
$a[] = "Violet";
$a[] = "Liza";
$a[] = "Elizabeth";
$a[] = "Ellen";
$a[] = "Wenche";
$a[] = "Vicky";

// get the q parameter from URL
$q = $_REQUEST["q"];

$hint = "";
```

```
// lookup all hints from array if $q is different from ""
if ($q !== "") {
  $q = strtolower($q);
  $len=strlen($q);
  foreach($a as $name) {
    if (stristr($q, substr($name, 0, $len))) {
      if ($hint === "") {
        $hint = $name;
      } else {
        $hint .= ", $name";
      }
    }
  }
}

// Output "no suggestion" if no hint was found or output correct values
echo $hint === "" ? "no suggestion" : $hint;
?>
```

Output



⏭ 6.10 AJAX AS REST CLIENT

- REST (Representational State Transfer) is a way of accessing the web services. REST as an architectural style does not require processing and is simpler and more flexible.
- REST is much easier and more widely used than SOAP (Simple Object Access Protocol). REST based web services can give output in any format like Comma-Separated Value (CSV), JavaScript Object Notation (JSON) and Really Simple Syndication (RSS).
- JSON is the most common output format of REST API and REST API accepts GET/POST request and gives JSON output.
- REST API can be used to fetch data from a web service. It is as simple as giving an HTTP GET or POST or PUT or DELETE request from the client to retrieve information from the server or provide information to the server.
- The most common way of implementing AJAX is JavaScript. REST client is an RIA that uses the web services or processes any request at the client –side and the result is retrieved in the JSON format.

⏭ 6.11 INTRODUCTION TO OPEN SOURCE FRAMEWORKS AND CMS FOR RIA: DJANGO, DRUPAL, JOOMLA

While developing any website or web application, it is essential to consider the tools necessary to create an RIA or a web application from the scratch, as many novice users are unfamiliar with programming languages such as PHP, HTML, CSS, and so on.

Content Management System (CMS)

- A content management system (CMS) is an application that is used to manage content, allowing multiple contributors to create, edit and publish.
- Content in a CMS is typically stored in a database and displayed in a presentation layer based on a set of templates like a website.
- A CMS will in most cases work as a web content management system (also called WCM or WCMS), which means it is designed to manage content on websites.
- This means you can control all your web-based content from this system. That includes text, graphics, video, and audio, which you control through your CMS and publish to your website.
- It not only allows you to store and manage all of your information in a one, easily accessible database, but it also allows you to accomplish the following :
  (1) Supports cross-team collaboration
  (2) Provides an easy and accessible way to update content
  (3) Increases content visibility
  (4) Improves productivity
  (5) Reduces costs
  (6) Enables you to maintain content consistency
  (7) Scales as your needs grow

CMS examples

While there are hundreds of CMS platforms, some of the more popular CMS providers are listed below :

(1) Drupal

(2) Joomla

(3) Magento

(4) Squarespace

(5) Wix

(6) Squarespace

(7) Wordpress

Framework

- Web frameworks are software frameworks that offer a standard and accessible way to build and develop web applications.
- A web framework is a code library that makes web development quicker and easier by giving basic patterns for building reliable, scalable and maintainable web applications.
- There are two main functions of frameworks: to work on the server side (backend), or on the client-side (frontend), corresponding to their type.

Server-side frameworks

- The rules and architecture of these frameworks allows you to create simple pages, landings and forms of different types.
- However, in order to build a web application with a well-developed interface, you should have a wider functionality.
- These frameworks can also form the output data and improve security in case of web attacks. All of these can definitely simplify the development process.
- Server-side frameworks work mostly on particular but important details without which an application can't work properly.
- Here are top backend frameworks and the languages they work in :

(1) Django – Python

(2) Zend – PHP

(3) Express.js – Javascript

(4) Ruby on Rails – Ruby

Client-side frameworks

- Unlike the server side, client-side frameworks have nothing to do with business logic. Their work takes place inside the browser.
- Thus, one can improve and implement new user interfaces. Numerous animated features can be created with frontend frameworks as well as SPA (single-page applications).
- Each of the client-side frameworks differs in function and use. For comparison purposes, here they are :

(1) Angular.js

(2) Ember.js

(3) Vue.js

Common Web Framework Functionalities :

- Frameworks give functionality in their code or through extensions to complete everyday operations needed to run web applications. These operations involve :

(1) URL routing

(2) Input form managing and validation

(3) HTML, XML, JSON, and other product setups with a templating engine

(4) Database connection configuration and resolute data manipulation through an object-relational mapper (ORM)

(5) Web security against Cross-site request forgery (CSRF), SQL Injection, Cross-site Scripting (XSS) and other frequent malicious attacks

(6) Session repository and retrieval

☞ **Examples of Web Frameworks**

There are several web frameworks that have been designed to make coding and app development much easier for those in the industry. A few of the most popular and successful web frameworks include :

(1) Django

(2) Ruby on Rails

(3) Express

Module 6

Difference between CMS and Framework

CMS	Framework
A CMS or content management system is a computer application that is used for creating and modifying digital content.	A framework is a software which contains a generic functionality that can be modified by additional user-written code depending on the application.
CMS helps to manage digital content.	A framework helps to organize the code to make the application development process simpler and flexible.
CMS is an open-source, so it is not as secure as frameworks.	The framework includes own custom code, so they are more secure than CMS.
It is not possible to mould CMS according to user requirement.	The framework can easily be moulded according to user requirement.
CMS is updated regularly, to secure them properly	The framework is rarely updated but it needs to be maintained.
Drupal, WordPress, and Joomla are some examples of CMS.	Django, Flask, CakePHP and CodeIgniter are some examples of frameworks.

⏭ 6.12 DJANGO FRAMEWORK

What is Django ?

- Django is a free, open source web framework written in the python programming language, which offers standard methods for fast and effective website development.
- A web framework is a software that supports the development of dynamic web sites, applications, and services.
- It provides a set of tools and functionalities that solves many common problems associated with web development, such as security features, database access, sessions, template processing, URL routing, internationalization, localization, and much more.
- It enables you to make the development process smooth and time-saving for rapid development.
- The primary goal of this high-level web framework is to create complex database-driven websites. It helps you to build and maintain quality web applications.

☞ **Features of Django Framework**

(1) Python Web-framework
(2) SEO optimised
(3) High scalability
(4) Versatile in nature
(5) Offers high security
(6) Thoroughly tested
(7) Provides rapid development

☞ **Advantages of Django**

(1) Written in Python : Django is one of the web frameworks which are written in Python programming language. Hence, it becomes easier for programmers to build web applications with clean, readable, and maintainable code by taking advantage of syntax rules of Python.

(2) Accelerates custom web application development : Django is one of the most mature web frameworks for Python. It's design rules focus extensively on reducing web application development time. The features provided by Django enable developers to build custom web applications rapidly according to varying business requirements.

(3) Designed as a batteries-included web framework : Django is one of the web frameworks that adopt the batteries-included approach. While developing a custom web application, Django provides the resources required by developers out of the box. It provides code for common operations like database manipulation, HTML templating, URL routing, session management, and security.

(4) Supports MVC programming paradigm : Django, like other modern web frameworks, supports model-view-controller (MVC) design rule. The MVC programming paradigm allows programmers to keep a web application's user interface (UI) and business logic layers separated. The approach further helps

programmers to simplify and speed up development of large web applications by separating their user interface and business logic layers.

(5) **Compatible with major operating systems and databases :** Nowadays, users access web applications on various devices and platforms. Django enhances the accessibility of web applications by supporting major operating systems like Windows, Linux and MacOS. At the same time, the ORM system provided by Django makes it easier for programmers to work with several widely used databases. They can even use the ORM system to perform common database operations and migrate from one database to another without writing additional code.

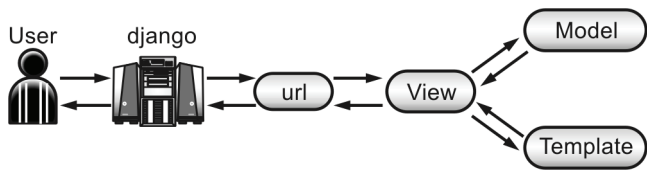
(6) **Provides robust security features :** The built-in security features provided by Django help developers to protect the web applications from a variety of targeted security attacks – cross-site scripting, SQL injection and cross-site request forgery.

(7) **Easy to extend and scale :** Django has been evolving consistently to enable programmers to build better and modern web applications. At the same time, the Django developers can easily customize, scale, and extend the web framework by making changes to its decoupled components.

(8) **Supported by a large and active community :** As an open source web framework for Python, Django helps developers to reduce web application development cost significantly. But it is supported by a large and active community of developers. The members of the Django community update new plug-ins and code snippets regularly to simplify web application development.

Django Architecture: MVC Pattern

- There are three primary components or code partitions for any website on the internet: Input Logic, Business Logic, and UI Logic.
- These partitions of code have specific tasks to achieve, the input logic is the dataset and how the data gets to organize in the database.
- It just accepts that input and transmits it to the database in the format specified.
- The main controller is the business logic, which handles the server's output in HTML or the required format. The HTML, CSS, and JavaScript pages comprise the UI Logic, as the name implies.
- When the traditional approach was used for programming all this code was written in a single file, i.e., every piece of code increases the webpage size, which is downloaded and rendered by the browser.
- This was not a big problem back in the time, the webpages were largely static and websites didn't contain much multimedia and large coding.
- Also, this architecture poses difficulty for developers while testing and maintaining the project as everything is inside one file.
- MVC pattern is a Product Development Architecture. It solves the traditional approach's drawback of code in one file, i.e., that MVC architecture has different files for different aspects of our web application/ website. The MVC pattern has three components, namely Model, View, and Controller.
- The Django architecture Fig. 6.12.1 shows the working cycle of Django MVC architecture.



(1F3)**Fig. 6.12.1 : MVC Architecture**

1. Model

- The Model is the component which contains business logic in Django architecture.
- The Model is the part of the web-app which acts as a mediator between the website interface and the database. In technical terms, it is the object which implements the logic for the application's data domain. There are times when the application may only take data in a particular dataset, and directly send it to the view (UI component) without needing any database then the dataset is considered as a model.

For example :

- When you sign up on any website you are actually sending information to the controller which then transfers it to the models which in turn applies business logic on it and stores in the database.

Module 6

2. View

- This component contains the UI logic in the Django architecture.
- View is actually the user interface of the web-application and contains the parts like HTML, CSS and other frontend technologies.
- Generally, this UI creates from the Models component, i.e., the content comes from the Models component.

For example :

- When you click on any link or interact with the website components, the new webpages that website generates is actually the specific views that stores and generates when we are interacting with the specific components.

3. Controller

- The controller as the name suggests is the main control component. What that means is, the controller handles the user interaction and selects a view according to the model.
- The main task of the controller is to select a view component according to the user interaction and also applying the model component.
- This architecture has lots of advantages and that's why Django is also based on this architecture. It takes the same model to an advanced level.

For example :

- When we combine the two previous examples, then we can very clearly see that the component which is actually selecting different views and transferring the data to the model's component is the controller.

⏭ 6.13 DRUPAL

What is Drupal?

- Drupal is a free and open source Content Management System (CMS) that allows organizing, managing and publishing your content.
- It is built on PHP based environments. Drupal CMS is more flexible if you develop a website with a content management system than any other CMS.
- Drupal is mighty and can be used for building large, complex sites. It is PHP based template and allows non-technical users to add and edit the content without any HTML or Web design knowledge.
- Furthermore, Drupal CMS makes it easy to interact with other sites or technologies as Drupal can handle complex forms and workflows.
- It is available with more than 16000 modules which can be addressed with Drupal core and add-on modules.

☞ **Features of Drupal**

- Drupal makes it easy to create and manage your site.
- Drupal translates anything in the system with built-in user interfaces
- Drupal connects your website to other sites and services using feeds, search engine connection capabilities, etc.
- Drupal is an open-source software hence requires no licensing costs.
- Drupal designs a highly flexible and creative website with adequate display quality, thus increasing visitors.
- Drupal can publish your content on social media such as Twitter, Facebook, and other social mediums.
- Drupal provides more customizable themes, including several base themes to design your themes for developing web applications.
- Drupal manages the content on informational sites, social media sites, member sites, intranets, and web applications.

☞ **Advantages of Drupal**

(1) Drupal is a flexible CMS that allows handling content types, including video, text, blog, menu handling, real-time statistics, etc.

(2) Drupal provides several templates for developing web applications. There is no need to start from scratch if you are building simple or complicated web applications.

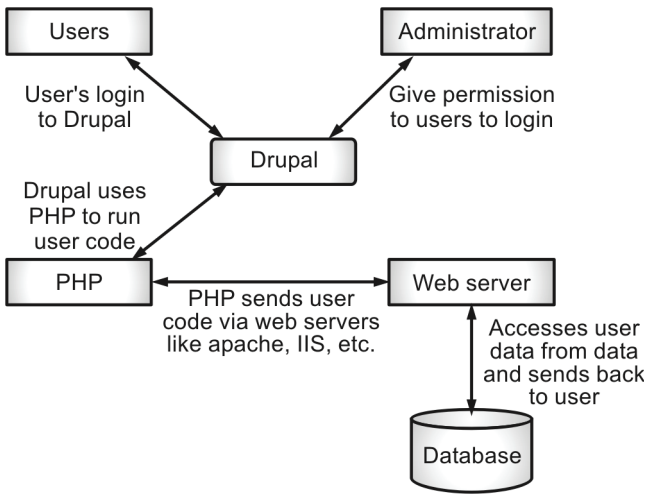
(3) Drupal is easy to manage or create a blog or website. It helps to organize, structure, find and reuse content.

(4) Drupal provides some interesting themes and templates which give your website an attractive look.

(5) Drupal has over 7000 plug-ins to boost your website. Since Drupal is open-source, you can create your plug-ins.

Architecture of Drupal

Drupal is a platform for web content management which is a powerful tool for building simple and complex sites.



(1F4)**Fig. 6.13.1 : Architecture of Drupal**

Users

- These are the users on the Drupal community.
- The user sends a request to a server using Drupal CMS and web browsers, search engines, etc. acts like clients.

(1) Administrator : Administrator can provide access permission to authorized users and will be able to block unauthorized access. Administrative account will be having all privileges for managing content and administering the site.

(2) Drupal : Drupal is a free and open source Content Management System (CMS) that allows organizing, managing and publishing your content and is built on PHP based environments. Drupal CMS is very flexible and powerful and can be used for building large, complex sites. It is very easy to interact with other sites and technologies using Drupal CMS. Further, you will be able to handle complex forms and workflows.

(3) PHP : Drupal uses PHP in order to work with an application which is created by a user. It takes the help of web server to fetch data from the database. PHP memory requirements depend on the modules which are used in your site. Drupal 6 requires at least 16MB, Drupal 7 requires 32MB and Drupal 8 requires 64MB.

(4) Web Server : Web server is a server where the user interacts and processes requests via HTTP (Hyper Text Transfer Protocol) and serves files that form web pages to web users. The communication between the user and the server takes place using HTTP. You can use different types of web servers such as Apache, IIS, Nginx, Lighttpd, etc.

(5) Database : Database stores the user information, content and other required data of the site. It is used to store the administrative information to manage the Drupal site. Drupal uses the database to extract the data and enables to store, modify and update the database.

6.14 JOOMLA

6.14.1 What is Joomla ?

- Joomla is said to be one of the best open-source content management systems (CMS) that can be used to build powerful websites and online applications.
- It is free, extendable, and separated into front-end and back-end templates (runs from administrator side). It is built on a model-view-controller framework which can be used independently of the CMS.
- Joomla is a platform which is based on PHP and MySQL. Joomla also supports third-party extensions and templates, which allow us to make further customization to meet specific requirements. It is open to anyone who wants to develop the extensions and templates.
- Joomla uses Model-View-Controller (MVC) design architecture. According to the MVC pattern when Joomla process a request, it first analyzes the URL to evaluate which component will process the request.
- The model contains the data used by the component. It is also the Model's responsibility to update the database when and where required.

Module 6

- The view is accountable for producing the output. It can contact with the model to get the needed information.
- After the view has produced the output, the component gives back the control to the Joomla framework which then executes the template.

6.14.2 Features of Joomla

1. Multilingual

- Joomla is multilingual. It supports 75 languages.

2. Responsive in nature

- Having a responsive website is a norm nowadays. Because every second person is surfing the internet through a smartphone.
- Thus, it is required to have a website that works perfectly on any device of any size. And Joomla gives you a completely responsive website. So, you never lose a potential website visitor.

3. Easy to Use

- Joomla is open source and entirely free to use.
- In fact, the completely user-friendly interface will amaze you with its WYSIWYG feature that gives you exactly the same results.
- Another factor to feel good about is that the frequent updates. Joomla brings new updates in the form of new features and functions. With every new update, it gets easier to work.

4. Security

- Security is an essential factor to consider when you create a website.
- Joomla provides you with two factor authentication to avoid the chances of hacking.
- So your site won't get hacked but in case you leave very common username and password and someone get in, you can easily restore hacked Joomla site.

5. Joomla Forum

- Anyone can contribute to Joomla forum on a volunteer basis. Joomla is an open source project which welcomes volunteers and their contributions to it.
- With whatever skillset and with as much as you can contribute to it and let others use your creation.
- If you ever stumble on any problem. Chances are that someone has already discussed that in the forum and you can easily get the solution form it.

6.14.3 Advantages of Joomla

(1) It is a freely available open-source platform.

(2) It is quite simple to install and set up.

(3) It uses WYSIWYG (What You See Is What You Get) editor, which helps to edit the content very easily.

(4) Almost all the browsers support it by default.

(5) It has an easy menu creation tool.

(6) It provides several templates and themes which are very flexible to use.

(7) It can be migrated to any server, and most of them work with any database.

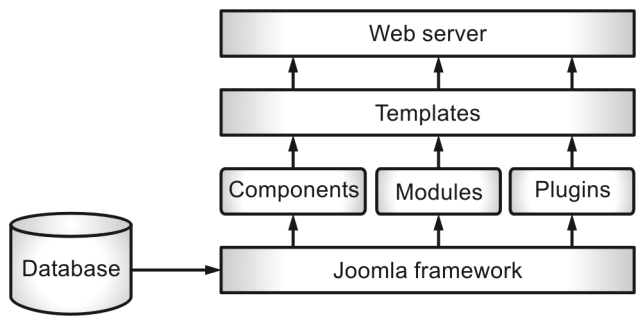
(8) It has broad community support where it is easy to ask questions and receive the solutions.

(9) It is based on PHP scripts, and so the compatibility issues will be limited while using its framework.

(10) It ensures the security and safety of the data content and does not allow to edit the data without administrator privileges.

6.14.4 Architecture of Joomla

- Joomla is written in PHP and based on MVC (Model-View-Controller) design pattern. It uses MySQL (MS SQL version 2.5 or above, and PostgreSQL version 3.0 or above) to store data.
- There are various features (e.g., page caching, blogs, polls, language internationalization support, and RSS feeds, etc.), which make Joomla an excellent choice for CMS (Content Management System).



(1F5)**Fig. 6.14.1 : Joomla architecture**

The architecture of Joomla includes the following layers :

Database

- The Database consists of data except image files and documents which can be stored, manipulated, and organized in a specific manner.
- It includes the user information, content, and other required data of the site. It also contains the administrative information so that an admin can securely access the site and manage it.
- Joomla database layer is one of the most important factors which ensure the maximum flexibility and compatibility for the extension.

Joomla Framework

- The Joomla Framework contains the collection of open-source software libraries/packages, on which Joomla content management system is built on.
- There is no need to install the Joomla Framework to use the CMS or vice-versa. Joomla Framework provides a group of files which is useful to create both web and command-line applications.
- It breaks the framework into single modular packages, and further, it helps each package to develop more easily.

Components

- Components are referred to as mini-applications which contain two parts :
  (1) Administrator
  (2) Site
- Whenever a page is loaded, the component is called to render the body of the main page.
- The Administrator part manages the different aspects of the component, and the site part helps in rendering the page when any site visitor makes a request.
- Components are known as the important functional units of Joomla.

Modules

- Modules can be defined as the lightweight extensions used to render pages in Joomla.
- They are used to display new data from the component. They can stand on its own and are managed by the '**Module Manager**', which is itself a component.
- They look like boxes, such as the login module. They also help to display the new content and images when the module is linked to Joomla components.

Plugin

- The Plugin can be explained as a very flexible and powerful Joomla extension, used to extend the framework.
- Plugins are few codes that execute on occasion of specific event triggers. It is generally used to format the output of a component or module when a page is developed.
- The plugin functions which are associated with an event are commonly executed in a sequence whenever a particular event occurs.

Templates

- Templates are used to manage the look of the Joomla websites. There are basically two types of templates available; **Front-end** and **Back-end**.
- The Front-end template is a way to manage the look of the website, which is seen by the users.
- The Back-end template is used to manage or control the functions by the administrator.
- Templates are the easiest way to build or customize the site. They are used to add maximum flexibility to make your site look attractive.

Web Server

- It is a server used to connect users to the site. It provides web pages to the client.
- The HTTP (HyperText Transfer Protocol) is used to communicate between the client and the server. Lastly, a website is that where you and your users interact with.

6.15 CMS COMPARISON - DRUPAL VS. JOOMLA VS. DJANGO

Open Source

- Drupal is an open-source platform written in PHP, and the developers distribute it under a general public license.
- The widespread use of the software began in 2003, but it became open-source in 2001. Its supported databases are MySQL and PostgreSQL.
- Joomla is also an open-source CMS. Matters Inc. developed it in August 2015. It was also written in PHP, and its data can be stored in PostgreSQL, MYSQL, or SQL Server.
- Django is also open-source software. This high-level Python CMS encourages rapid development and pragmatic design. Also, the supported databases are MySQL and PostgreSQL.

Market Share

- Out of the three content management systems, Joomla is the most popular.
- It has a 5.20% market share. Drupal follows second with a market share of 3.40%.
- Django is the least popular. The number of sites that it powers and the market share is less than 1%.

Support and Community

- All these three content management systems offer reliable ways to help users whenever they struggle with any task.
- Joomla's official site presents user forums, the community portal and training, and the primary documentation where you get help.
- Drupal also offers many sections for support, community portal, and documentation. Moreover, you can get more help by using the user guides and the numerous tutorials available on the site.
- Django also has an expansive network of experts around that world that offer users professional support. One can use the ticket system or email the professionals to get help.
- Individuals who have questions about installing, using, or contributing to the platform can also join the Django forum.

Security and Updates

- You can expect Joomla to update after every 10-40 days. Many of these updates are designed to address minor bugs.
- Significant releases come out after several months. Drupal often releases one bug-fix update and a single security update every month.
- But Drupal's more significant release comes after every six months. Django CMS, on the other hand, has not set specific release cycles.
- However, like Drupal, its security level is high. Joomla's security, however, is relatively low.

6.16 SELF-LEARNING TOPICS: APPLICATIONS OF AJAX IN BLOGS, WIKIS AND RSS FEEDS

AJAX has its practical applications and is mostly used in updating Blogs, Wikis and RSS feeds.

6.16.1 Blogs

- A blog is an abbreviation used for a weblog. The term weblog is used to describe the websites that an ongoing chronicle of information.
- It is a diary type commentary that links an article to the other websites. It is presented in the form of a list of entries in reverse chronological order.
- The blogs may include personal to political topics, one narrow subject or a whole range of subjects, or just one specific topic ranging from web designing to quantum physics.
- With the advancement of the Internet, many entrepreneurs are now trying to integrate their blogs with their traditional business websites. Though, there is a difference between traditional websites and blogs which is highlighted as follows:
  (1) Blogs are updated frequently. They have new content added several times in a week while changes in a website are made after long intervals of time.
  (2) Blogs allow reader engagement, as they have the ability for the readers to comment and have a discussion, a feature which is generally missing from a website.

☞ Types of Blogs

Web of today has millions upon millions of blogs. There are blogs maintained by individuals, there are blogs maintained by companies or organizations on the web, and then there are blogs dedicated to factual information like news, and so on. But, one thing is common – they tend to fall into these types of blogs:

(1) Personal blog : This is the most common type of blog. Maximum numbers of blogs on internet belong to this category. After all, blogging started off as a way of creating an online diary. A typical blogger may be keen on posting stories about their interest such as fishing, or dancing or collecting something.

(2) News and views : This type of blog contains factual stories about news, maintained by journalists. News and television companies such as the BBC have many professional journalists who post stories and views about the latest events. Visitors can add their own opinions as well.

(3) Company blogs : Many companies run blogs to let their customers and clients know what is going on in the company that would interest clients or customers such as new products coming up or progress being made on some project.

☞ Advantages of Blogs

Advantages of Blogs are as follows :

- Enables you to write down your thoughts on anything that interests you.
- Very quick and easy to set up, don't need much technical knowledge
- Easy and quick to update or add new posts
- People can leave comments on your blog
- If you want to read other people's blogs there are literally millions to choose from

☞ Disadvantages of Blogs

Disadvantages of Blogs are as follows:

(1) Whatever you publish is available for everyone to see. If you write a post in anger you might regret it later.

(2) Personal blogs may be biased or contain inaccurate information

(3) Blogs can be time consuming. Finding time to write regular updates can become a chore.

(4) People may leave rude or inappropriate comments

(5) There are many very dull blogs around. You may have to look at many before you find some worth reading.

6.16.2 Wikis

- Wiki is a collaborative workspace in which information is collected, shared, evaluated, organised or used to produce something new. Wiki is a server program that allows users to integrate the content of a website.
- The wiki offers a simple interface and for the user using it doesn't need to have the knowledge of HTML.
- The wiki website operates on a principle of collaborative trust that allows different users to create and edit content at the same time.
- The advantages of using wiki websites are listed as follows:
  (1) Edited by anyone.
  (2) Easy to use and learn.
  (3) Instantaneous, so do not require the publisher to create new content or update information.
  (4) Accessed by people across the globe, so many people can work on the same document.
  (5) Keep track of every edit, so it is easy to revert the change.
  (6) Widens access to the power of web publishing to the non-technical users
  (7) Organise and share information in a secure way.
- Though it offers several advantages some of the disadvantages of using wiki websites are :
  (1) As information can be accessed by anyone so, it is not possible to regulate user access.
  (2) Open to spam, if it is not managed properly.
  (3) Requires internet connectivity for collaborating.
  (4) Flexible nature can lead to disorganisation of information.

6.16.3 RSS Feeds

- RSS is an abbreviation for Really Simple Syndication. It offers a simple and easy way to distribute headlines and contents, and update notifications. It is used by computer programs that organise headlines and notices for easy reading.

Module 6

- An RSS feed is read by an RSS reader or feed reader. These readers can be either web-based, standalone desktop applications or a mobile application. The reader collaborates with the available RSS feeds which the user is subscribed to. Once the RSS feeds collaborate, it is presented with a simple interface and eliminates the need to go to each website individually and read the updates. An RSS feed is delivered in XML format that allows maximum compatibility between the readers.

☞ Advantages

RSS gives benefits to both readers (users) and web publishers.

(1) It gives you the latest updates. Whether it is about the weather, new music, software upgrade, local news, or a new posting from a rarely-updates site learn about the latest as soon as it comes out.

(2) It gives the power of subscription to the user. Users are given a free-hand on which websites to subscribe in their RSS aggregators which they can change at any time they decide differently.

(3) It saves on surfing time. Since an RSS feed provides a summary of the related article, it saves the user's time by helping s/he decide on which items to prioritize when reading or browsing the net.

(4) It is spam free. Unlike email subscriptions, RSS does not make use of your email address to send updates thus your privacy is kept safe from spam mails.

(5) Unsubscribing is hassle-free. Unlike email subscriptions where the user is asked questions on why she/he is unsubscribing and then the user would be asked to confirm unsubscribing, all you have to do is to delete the RSS feed from your aggregator.

(6) It can be used as an advertising or marketing tool. Users who subscribe or syndicate product websites receive the latest news on products and services without the website sending spam mail. This is advantageous to both the web user and the website owner since advertising becomes targeted; those who are actually interested in their products are kept posted.

☞ Disadvantages

The disadvantages of RSS use are brought about by its being a new technology and some user-preference concerns.

(1) Some users prefer receiving email updates over an RSS feed.

(2) Graphics and photos do not appear in all RSS feeds. For conciseness and ease of publication, RSS feeds do not display the photos from the original site in announcing the update except for some web-based aggregators.

(3) The identity of the source website can be confusing. Since RSS feeds do not display the actual URL or name of the website, it can sometimes get confusing on what feed a user is actually reading.

(4) Publishers cannot determine how many users are subscribed to their feed and the frequency of their visits. Moreover, they would not know the reasons why users unsubscribe which could be important in improving their advertising.

(5) RSS feeds create higher traffic and demands on the server. Most readers still prefer the whole update over a brief summary of the entry, thus they still access the site.

(6) Since it is a new technology, many sites still do not support RSS.

Descriptive Questions

Q. 1 Explain AJAX in detail.

Q. 2 Explain working of AJAX in detail.

Q. 3 With the help of diagram differentiate between synchronous and asynchronous model.

Q. 4 Write the different methods used by XMLHttpRequests.

Q. 5 Explain the different characteristics of RIA in detail.

Q. 6 List and explain the technologies used in AJAX.

Q. 7 Write a note on: Django.

Q. 8 Write a note on: Drupal.

Q. 9 Write a note on: Joomla.

Q. 10 Differentiate between Django, Drupal and Joomla.

Chapter Ends…

❑❑❑

Web X.0 (M6-104)

Multiple Choice Questions (MCQs)

Chapter 1 Introduction to WebX.0

Multiple Choice Questions

Q. 1.1 In order to identify the users, web analytics tools need to report on?

(a) User Sessions (b) Unique Users

(c) Page Views (d) Repeated Users ✓**Ans. (a)**

Q. 1.2 The most common user identification technique is via?

(a) Sessions (b) Cookies

(c) Segmentation (d) Page Views ✓**Ans. (b)**

Q. 1.3 Users use _____ to search for service on the web.

(a) Identifier (b) Clickstream

(c) Keyword (d) Request for Proposal

✓**Ans. (c)**

Q. 1.4 Which one is called content delivery network?

(a) Web 1.0 (b) Web 2.0

(c) Web 3.0 (d) Web 4.0 ✓**Ans. (a)**

Q. 1.5 _____ is also called participative social web.

(a) Web 1.0 (b) Web 2.0

(c) Web 3.0 (d) Web 4.0 ✓**Ans. (b)**

Q. 1.6 Which of the following is not the feature of Web 3.0?

(a) Semantic web (b) 2D graphics

(c) Ubiquity (d) Artificial Intelligence

✓**Ans. (b)**

Q. 1.7 _____ refers to the percentage of visitors who leave a website without visiting more than one page of the website.

(a) Time on page (b) Time on site

(c) Bounce rate (d) Conversion rate ✓**Ans. (c)**

Q. 1.8 _____ is the total number of visitors to a website who turn into the customers or subscribers.

(a) Time on page (b) Time on site

(c) Bounce rate (d) Conversion rate ✓**Ans. (d)**

Q. 1.9 _____ is used to define the time spent by a visitor on a website.

(a) Time on page (b) Time on site

(c) Bounce rate (d) Conversion rate ✓**Ans. (b)**

Q. 1.10 _____ is the number of repeated visits on the website.

(a) Recency (b) Bounce rate

(c) Conversion rate (d) Clickstream ✓**Ans. (a)**

Q. 1.11 _____ defined as the frequency of visits on website for a specific time period.

(a) Recency (b) Visitors Loyalty

(c) Length of Visit (d) Depth of Visit ✓**Ans. (b)**

Q. 1.12 Which component of the Semantic Web Stack is a SQL-like language?

(a) XML (b) SPARQL

(c) OWL (d) RIF ✓**Ans. (b)**

Q. 1.13 _____ offers a standardised to read and write all the available human languages on the web.

(a) Unicode (b) RDF

(c) URI (d) OWL ✓**Ans. (a)**

Q. 1.14 _____ is a framework that represents the details of resources in the form of a graph.

(a) Unicode (b) RDF

(c) URI (d) OWL ✓**Ans. (a)**

Q. 1.15 Which component of the Semantic Web Stack gives the description of conceptualisation?

(a) URI (b) Ontology

(c) RDF (d) Unicode ✓**Ans. (b)**

Q. 1.16 The objectives for web analytics are likely to concern:

(a) Facebook messages

(b) Personal blog activity

(c) Social Media ROI

(d) Measurement of website performance ✓**Ans. (d)**

Q. 1.17 Which of the following is the odd one out?

(a) Share of conversation (b) Bounce rate

(c) Visitors (d) Impressions

✓**Ans. (a)**

Q. 1.18 Which one of the following is mainly used in web analytics and is free of charge?

(a) Google Analytics (b) Radian6

(c) AlteranSM2 (d) Social Radar ✓**Ans. (a)**

Q. 1.19 _____ are the strings of characters used for identifying a resource.

(a) URIs (b) Literals

(c) Blank Nodes (d) Escape Characters ✓**Ans. (a)**

Q. 1.20 Which of the following N-Triples language allows to recognise strings, numbers and dates?

(a) Simple Triples (b) IRIs

(c) RDF Literals (d) RDF Blank Nodes

✓**Ans. (c)**

MCQ Ends...

❑❑❑

Chapter 2 Typescript

Multiple Choice Questions

Q. 2.1 Which of the following companies has developed and designed TypeScript?

(a) Amazon (b) TypeScript
(c) Microsoft (d) Oracle ✓**Ans. (c)**

Q. 2.2 Which of the following filenames is the extension for typescript?

(a) .tt (b) .nod (c) .txt (d) .ts ✓**Ans. (d)**

Q. 2.3 When was the first time TypeScript was made public?

(a) December 2012 (b) October 2012
(c) October 2013 (d) November 2013
✓**Ans. (b)**

Q. 2.4 The following are backported features of TypeScript, except?

(a) Classes (b) Methods
(c) Modules (d) Arrow ✓**Ans. (b)**

Q. 2.5 The _____ Service powers the interactive TypeScript experience in Visual Studio, Vs Code, Sublime, the TypeScript playground and other editor.

(a) TypeScript Language
(b) TypeScript Compiler
(c) TypeScript Main
(d) TypeScript Methods
✓**Ans. (a)**

Q. 2.6 A typescript can be installed or managed through?

(a) void (b) space
(c) npm (d) tag ✓**Ans. (c)**

Q. 2.7 _____ are the way to organize code in TypeScript.

(a) Modules (b) Classes
(c) Methods (d) Arrow ✓**Ans. (a)**

Q. 2.8 TypeScript compile down to _____.

(a) C++ (b) JavaScript
(c) HTML (d) Java ✓**Ans. (b)**

Q. 2.9 In TypeScript, weakly or dynamically typed structures are of _____ type.

(a) Complex (b) Any
(c) Dynamic (d) Var ✓**Ans. (b)**

Q. 2.10 Name of the TypeScript Compiler is _____.

(a) tsc (b) tsp
(c) tcp (d) scp ✓**Ans. (a)**

Q. 2.11 In TypeScript, integers are represented by the _____ type.

(a) int16 (b) int32
(c) int (d) number ✓**Ans. (d)**

Q. 2.12 Syntactically, TypeScript is very similar to _____.

(a) javascript (b) C#
(c) .Net (d) Jscript.Net ✓**Ans. (d)**

Q. 2.13 Which command is used in the terminal window to install TypeScript?

(a) npm install –g typescript
(b) node install –g typescript
(c) npm install –g ts
(d) node install –g ts ✓**Ans. (a)**

Q. 2.14 _____ is a structure which acts as a contract in our application.

(a) Class (b) Tuple
(c) Interface (d) Function ✓**Ans. (c)**

Q. 2.15 TypeScript is a/an _____ framework.

(a) Licensed

(b) Open-sourced

(c) Commericial

(d) Both a and b ✓**Ans. (b)**

Q. 2.16 TypeScript is a typed superset of _____.

(a) C# (b) Java

(c) JavaScript (d) ReactJS ✓**Ans. (c)**

Q. 2.17 What will be the output of the following code snippet?

```
let arr = [10, 20, 30, 40];
for (varval of arr) {
    console.log(val);
}
```

(a) prints 10, 20, 30, 40

(b) prints 0, 1, 2, 3

(c) prints undefined

(d) compile error ✓**Ans. (a)**

Q. 2.18 Which of the following is the valid arrow function?

(a) let sum = (x: number, y: number) =>x+y;

(b) let sum() : (x: number, y: number) : number =>x+y;

(c) let sum = (x: number, y: number) => return x+y;

(d) let sum(x: number, y: number) =>x+y;

✓**Ans. (a)**

Q. 2.19 The rest parameters in a function is used when _____.

(a) The types of parameters is not known.

(b) The number of parameters is not known.

(c) The function needs to be executed asynchronously.

(d) The function is called recursively. ✓**Ans. (b)**

Q. 2.20 Which of the following keyword is used to declare a module?

(a) export (b) namespace

(c) type (d) declare ✓**Ans. (a)**

MCQ Ends...

❑❑❑

Chapter 3 Introduction To AngularJS

Multiple Choice Questions

Q. 3.1 Which of the following statement is correct for AngularJS?

(a) AngularJS is an HTML framework
(b) AngularJS is a Java framework
(c) AngularJS is a JavaScript framework
(d) AngularJS is a SQL framework ✓**Ans. : (c)**

Q. 3.2 On which of the Architectural pattern AngularJS is based?

(a) Observer Pattern
(b) Decorator pattern
(c) MVC Architecture pattern
(d) MVVM Architectural pattern ✓**Ans. : (d)**

Q. 3.3 AngularJS is perfect for?

(a) SPAs (b) MPAs
(c) DPAs (d) CPAs ✓**Ans. : (a)**

Q. 3.4 Which of the following is the correct syntax for writing AngularJS expressions?

(a) (expression) (a) {{expression}}
(c) {{{expression}}} (d) [expression]

✓**Ans. : (b)**

Q. 3.5 Which of the following directive is used to bind the application data to the HTML view in AngularJS?

(a) ng-app directive
(b) ng-model directive
(c) ng-bind directive
(d) ng-init directive ✓**Ans. : (c)**

Q. 3.6 Which of the following syntax is correct for applying multiple filters in AngularJS?

(a) {{ expression | filter1 | filter2 | ... }}
(b) {{ expression | {filter1} | {filter2} | ... }}
(c) {{ expression - {filter1} - {filter2} - ... }}
(d) {{ {filter1} | {filter2} | ...-expression}}

✓**Ans. (a)**

Q. 3.7 What will be the output for the following code?

<div ng-app="""" ng-init=""points=[1,15,19,2,40]"">
The output is {{ points[2] }}

(a) The output is 1 (b) The output is 15
(c) The output is 19 (d) The output is 2

✓**Ans. : (c)**

Q. 3.8 Which of the following is used to share data between controller and view in AngularJS?

(a) using Model (b) using services
(c) using factory (d) using $scope

✓**Ans. : (b)**

Q. 3.9 Which of the following community Angular JS belong to?

(a) Twitter (b) Facebook
(c) Google (d) Microsoft ✓**Ans. : (c)**

Q. 3.10 Which of the following directive is used to bind the value of HTML controls to application data?

(a) ng-app (b) ng-init
(c) ng-model (d) ng-hide ✓**Ans. (c)**

Q. 3.11 How many $RootScope an AngularJS application can have?

(a) Zero (b) One
(c) Two (d) Infinity ✓**Ans. : (b)**

Q. 3.12 Which of the following components can be injected as a dependency in AngularJS?

(a) Value (b) Factory
(c) Constant (d) Application Module

✓**Ans. : (d)**

Q. 3.13 AngularJS applications are a mix of which of the following technologies?

(a) HTML and PHP
(b) HTML and JavaScript
(c) HTML and TypeScript
(d) PHP and JavaScript ✓**Ans. : (b)**

Q. 3.14 The [] parameter in the module definition can be used to define dependent modules.

(a) TRUE (b) FALSE
(c) Can be true or false (d) Can not say
✓**Ans. : (a)**

Q. 3.15 _____ in AngularJS is the synchronization between the model and the view.

(a) Scope (b) Filter
(c) Data binding (d) Service ✓**Ans. : (c)**

Q. 3.16 Which directive initializes an AngularJS application?

(a) ng-init (b) ng-app
(c) ngSrc (d) ng-start ✓**Ans. (b)**

Q. 3.17 Which of the following is not a valid Filter in AngularJs?

(a) lowercase (b) orderby
(c) email (d) currency ✓**Ans. : (c)**

Q. 3.18 How to combine filter and expression?

(a) Using the comma {{expression, filter}}
(b) Using point {{expression. filter}}
(c) Using Pipe {{expression | filter}}
(d) Using Slash {{expression / filter}} ✓**Ans. : (c)**

Q. 3.19 We need to tell AngularJS what part of our HTML page contains the AngularJS app. You do so by adding the _____ attribute to the root HTML element of the AngularJS app.

(a) ng-app (b) ag-app
(c) js-app (d) aj-app ✓**Ans. : (a)**

Q. 3.20 The _____ directive is used if you want to add or remove HTML elements from the DOM based on data in the model.

(a) ng-switch (b) ng-model
(c) ng-Disabled (d) ng-Cloak ✓**Ans. : (a)**

MCQ Ends...

❑❑❑

Chapter 4 MongoDB and Building REST API using MongoDB

Multiple Choice Questions

Q. 4.1 Which of the following format is supported by MongoDB?

(a) SQL (b) XML

(c) BSON (d) All of the mentioned

✓**Ans. : (c)**

Q. 4.2 Initial release of MongoDB was in the year?

(a) 2000 (b) 2005

(c) 2009 (d) 2011 ✓**Ans. : (c)**

Q. 4.3 MongoDB has been adopted as _____ software by a number of major websites and services.

(a) frontend (b) backend

(c) proprietary (d) all of the mentioned

✓**Ans. : (b)**

Q. 4.4 MongoDB is a _____ database that provides high performance, high availability, and easy scalability.

(a) graph (b) key value

(c) document (d) all of the mentioned

✓**Ans. : (c)**

Q. 4.5 What is the interactive shell for MongoDB called?

(a) mongo (b) mongodb

(c) dbmong (d) none of the mentioned

✓**Ans. : (a)**

Q. 4.6 After starting the mongo shell, your session will use the _____ database by default.

(a) mongo (b) masteri

(c) test (d) primary ✓**Ans. : (c)**

Q. 4.7 _____ command display the list of databases.

(a) show db (b) show dbs

(c) show data (d) display dbs ✓**Ans. : (b)**

Q. 4.8 Which of the following operation is used to switch to new database mydb?

(a) use dbs (b) use db

(c) use mydb (d) use mydbs ✓**Ans. : (c)**

Q. 4.9 Which of the following also returns a list of databases?

(a) show databases (b) show database

(c) displaydbs (d) all of the mentioned

✓**Ans. : (a)**

Q. 4.10 Command to check existence of collection is _____

(a) show collection (b) show collections

(c) show collect (d) none of the mentioned

✓**Ans. : (b)**

Q. 4.11 Which of the following method is used to query documents in collections?

(a) find (b) move

(c) shell (d) replace ✓**Ans. : (a)**

Q. 4.12 In MongoDB _____ operations modify the data of a single collection.

(a) CRUD (b) GRID

(c) READ (d) All of the mentioned

✓**Ans. : (a)**

Q. 4.13 Point out the wrong statement.

(a) Indexes cannot enforce uniqueness in collection

(b) CRUD stands for create, read, update, and delete

(c) CRUD application is the most simplest application

(d) All of the mentioned ✓**Ans. : (a)**

Q. 4.14 Which of the following operation adds a new document to the users collection?

(a) add (b) insert

(c) truncate (d) drop ✓**Ans. : (b)**

Q. 4.15 Who is credited with the invention of REST?

(a) Kohsuke Kawaguchi (b) James Gosling

(c) Roy Fielding (d) Linus Torvalds

✓**Ans. : (c)**

Q. 4.16 Which of the following best describes REST?

(a) REST is a web service standard

(b) REST is a cloud-native API framework

(c) REST is a microservices-based protocol

(d) REST is an architectural style ✓**Ans. : (d)**

Q. 4.17 How would you configure a RESTful URL parameter that supports a search for a book based on its ID?

(a) GET /{id}/books/ (b) GET /books/{id}

(c) GET /book?id={id} (d) GET /books?id={id}

✓**Ans. : (b)**

Q. 4.18 Which option is not a RESTful API constraint?

(a) Code on demand

(b) The use of a client-server model

(c)A stateless request-response cycle

(d) Service orchestration ✓**Ans. : (d)**

Q. 4.19 To create a new resource with a predefined resource URL, you should use which of the following HTTP methods?

(a) CREATE (b) POST

(c) PUT (d) OPTION ✓**Ans. : (c)**

Q. 4.20 Which of the following is true about REST?

(a) In REST architecture, a REST Server simply provides access to resources and RESTclient accesses and presents the resources.

(b) Each resource is identified by URIs/ global IDs.

(c) REST uses various representations to represent a resource like text, JSON and XML.

(d) All of the above. ✓**Ans. : (d)**

MCQ Ends...

❑❑❑

Chapter 5 Flask

Multiple Choice Questions

Q. 5.1 Flask is a web development framework created in _____ language.

(a) C (b) Java

(c) Python (d) Javascript ✓**Ans. : (c)**

Q. 5.2 WSGI stands for the?

(a) Write Server Gateway Interface

(b) Web Static Gateway Interface

(c) Web Server Gateway Interface

(d) Web Server Gateway Interact ✓**Ans. : (c)**

Q. 5.3 Flask default port is?

(a) 2000 (b) 3000

(c) 4000 (d) 5000 ✓**Ans. : (d)**

Q. 5.4 Flask is called as?

(a) Miniframework (b) Microframework

(c) Peraframework (d) Nanoframework

✓**Ans. : (b)**

Q. 5.5 The _____ decorator in Flask is used to bind URL to a function.

(a) route() (b) rend()

(c) read() (d) write() ✓**Ans. : (a)**

Q. 5.6 The _____ method of request.cookies attribute is used to read a cookie.

(a) read() (b) get()

(c) set() (d) find() ✓**Ans. : (b)**

Q. 5.7 Which extension is used to store flask app?

(a) .py (b) .html

(c) .sql (d) .php ✓**Ans. : (a)**

Q. 5.8 The server-side flask script fetches the file from the request object using _____ Object.

(a) request.files[] (b) req.files[]

(c) Both A & B (d) None of the above

✓**Ans. : (a)**

Q. 5.9 The _____ function is used to build a URL to the specific function dynamically.

(a) url_for() (b) url_in()

(c) url_loop() (d) None of the above

✓**Ans. : (a)**

Q. 5.10 Which of the following converter accepts any text without a slash in flask.

(a) string (b) float

(c) path (d) uuid ✓**Ans. : (a)**

Q. 5.11 The route() function of the Flask class accepts the following parameters.

(a) rule (b) options

(c) Both A & B (d) None of the above

✓**Ans. : (c)**

Q. 5.12 Which command is used to install virtual environment in flask.

(a) $ pip install virtualenv

(b) $ pip install virtual

(c) $ pip install virtual_env

(d) $ pip install virtual-env ✓**Ans. : (a)**

MCQ Ends...

❑❑❑

Chapter 6 Rich Internet Application

Multiple Choice Questions

Q. 6.1 Which of the following is represented as a tree-like structure which has its root at the top?

(a) AJAX (b) HTML

(c) DOM (d) JavaScript ✓ **Ans. : (c)**

Q. 6.2 Which of the following is not a framework?

(a) Code Igniter (b) Zend

(c) Django (d) Drupal ✓ **Ans. : (d)**

Q. 6.3 The full form of AJAX is

(a) Asymmetric JavaScript and XML

(b) Asynchronous JavaScript and XML

(c) Asynchronous Java and XML

(d) Asynchronous JSON and XML ✓ **Ans. : (b)**

Q. 6.4 Which of the following technology is not used by AJAX?

(a) HTML (b) XHTML

(c) C++ (d) DOM ✓ **Ans. : (c)**

Q. 6.5 Which of the following is not a standard for AJAX?

(a) Browser-based presentation using HTML and Cascading Style Sheets (CSS).

(b) Data is retrieved by parse() function.

(c) Data is stored in XML format and fetched from the server.

(d) Data is fetched at the back-end using XML Http-Request objects in the browser. ✓ **Ans. : (b)**

Q. 6.6 Which of the following provides a tree structure as a logical view of a webpage?

(a) DOM (b) CSS

(c) HTML (d) JavaScript ✓ **Ans. : (a)**

Q. 6.7 Which of the following is not a parameter of an open() function?

(a) Method (b) Url

(c) Asnc (d) Time ✓ **Ans. : (d)**

Q. 6.8 Which of the following holds the status of XML HttpRequest?

(a) Status (b) readyState

(c) statusText (d) Onreadystatechange

✓ **Ans. : (b)**

Q. 6.9 Which of the following transfers is an XML data using HTTP to and from the web server?

(a) XMLHyperRequest (b) HttpRequest

(c) XMLHttpRequest (d) XMLHttpRoute

✓ **Ans. : (c)**

Q. 6.10 The blog is an abbreviation used for:

(a) Weblogging (b) webloggers

(c) Weblog (d) Websitelog

✓ **Ans. : (c)**

Q. 6.11 Which of the following is a collaborative workspace in which information is collected, shared, evaluated, organised or used to produce something new?

(a) RSS Feeds (b) Blogs

(c) JSON (d) Wikis ✓ **Ans. : (d)**

Q. 6.12 Django is a _____ Python web framework.

(a) low-level (b) mid-level

(c) high-level (d) None of the above

✓ **Ans. : (c)**

Q. 6.13 Joomla can be installed manually from source code on a system running a web server which supports ____ applications.

(a) HTML (b) Adobe Flash Player

(c) PHP (d) Java ✓ **Ans. : (c)**

Q. 6.14 Django supports the _____ pattern.

(a) MVI (b) MVP

(c) MVC (d) MVZ ✓ **Ans. : (c)**

Q. 6.15 The full form of RIA is

(a) Random Internet Application

(b) Rich Internet Application

(c) Rich International Application

(d) Rich Internet Application ✓ **Ans. : (b)**

Q. 6.16 Which of the following is the web application that offers look, feel and usability of desktop applications?

(a) RIA (b) RSS

(c) Blogs (d) GUI ✓ **Ans. : (a)**

Q. 6.17 Which of the following is not a characteristic of RIA?

(a) Front-end use

(b) Enhanced GUI

(c) Responsive and Interactivity

(d) Data centralization ✓ **Ans. : (d)**

Q. 6.18 Which of the following types of RIA offers access to the local file system and ability to save information even when the web connection is interrupted ?

(a) Browser based (b) Desktop based

(c) Plug-in based (d) Central based

✓ **Ans. : (b)**

Q. 6.19 Which of the following HTML view shows the content of a Webpage without HTML page ?

(a) Content View (b) HTML Source code

(c) HTML View (d) Source View

✓ **Ans. : (a)**

Q. 6.20 In which of the following tags, the JavaScript is written?

(a) <JavSscript>.....</JavaScript>

(b) <Script>.....</Script>

(c) <Java>.....</Java>

(d) <JavaS>.....</JavaS> ✓ **Ans. : (b)**

Q. 6.21 Which of the following is not a advantages of CSS?

(a) Easy Maintaenance

(b) Slow Page Loading

(c) Multiple Device Support

(d) Platform Independence ✓ **Ans. : (b)**

Q. 6.22 The full form of PHP is

(a) Hypertext Preprocessor

(b) Pre-Hyper Processor

(c) Post-Hyper Processor

(d) Hyper Processor ✓ **Ans. : (a)**

MCQ Ends...

❑❑❑