M1:INTRO TO WEB X

Discuss Clickstream Analysis

Clickstream analysis is a method used in web analytics to track and analyze the sequence of clicks or interactions that a user performs while navigating through a website or application. It provides valuable insights into user behavior, helping businesses and website owners understand how visitors engage with their online content

VUTTBECE

V:Visits
U:Unique visitors
T:Time on page
T:Time on site
B:Bounce rate
E:Exit rate
C:Conversion rate
E:Engagement

Discuss the strategy to choose web analytics tool.

Define Objectives and Goals:

- Clearly outline your **business objectives and goals**. Understand what you want to achieve with web analytics. Whether it's increasing website traffic, improving user engagement, or optimizing conversions, having clear goals will guide your tool selection.

Identify Key Features:

- List the essential features you need in a web analytics tool. Common features include **page views, user demographics, conversion tracking, e-commerce tracking, custom reporting, and real-time analytics.** Prioritize features based on your business requirements.

Consider Ease of Use:

- Choose a tool that aligns with the **technical expertise of your team.** Some tools are more user-friendly and intuitive, while others may have a steeper learning curve. Ensure that the selected tool can be easily adopted and used effectively by your team.

Scalability:

- Evaluate whether the web analytics **tool can scale with your business as it grows.** Consider factors such as the tool's ability to handle increased data volume, support additional websites or applications, and accommodate more users.

Integration Capabilities:

- Check the tool's compatibility with other tools and platforms used in your organization, such as content management systems, customer relationship

management (CRM) software, or marketing automation tools. Integration capabilities are crucial for a seamless workflow.

Data Accuracy and Reliability:

- **Ensure that the web analytics tool provides accurate and reliable data.** Look for features that prevent data discrepancies, such as filtering options, exclusion of internal IP addresses, and support for various tracking methods (e.g., cookies, JavaScript tags).

Cost and Budget:

- Understand the pricing structure of the web analytics tool. Consider not just the initial costs but also any additional fees for increased usage, premium features, or support. Align the **tool's pricing** with your budget constraints.

Customer Support and Training:

- Assess the level of customer support and training provided by the analytics tool vendor. Consider factors such as available documentation, tutorials, customer forums, and the responsiveness of customer support. Adequate support is essential for resolving issues and optimizing tool usage.

Security and Compliance:

- Ensure that the web analytics tool adheres to **data security standards** and compliance regulations relevant to your industry. Protecting user data and maintaining compliance with privacy laws should be a priority.

Trial Period and Testing:

- Whenever possible, take advantage of trial periods or free versions of web analytics tools. Test the tool in a real-world scenario to evaluate its performance, features, and compatibility with your specific requirements before making a final decision.

Future Roadmap and Updates:

- Check the vendor's commitment to ongoing development and updates. A tool that receives regular updates and improvements is more likely to stay relevant and meet evolving business needs.

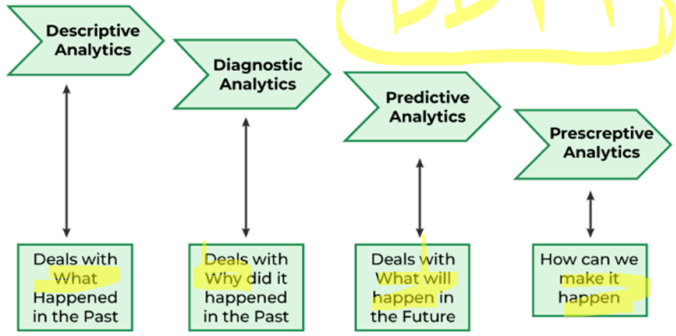
Explain factors for Measuring the success of a website

Traffic metrics: Unique visitors, pageviews, sessions.
User engagement: Bounce rate, time on page, scroll depth.
Conversion rates: Conversion rate, click-through rate (CTR).
E-commerce metrics: Revenue, average order value (AOV), shopping cart abandonment rate.
Content effectiveness: Popular pages, exit pages, dwell time.
Search engine performance: Keyword rankings, organic traffic, backlink quality.
Social media impact: Social shares, social engagement, referral traffic.
Mobile responsiveness: Mobile traffic, mobile conversion rates, page load speed.
User satisfaction and feedback: Surveys, Net Promoter Score (NPS).
Security and reliability: Uptime, security incidents.
Cost-related metrics: Return on Investment (ROI), Cost per Acquisition (CPA).
Accessibility and inclusivity: Accessibility compliance, inclusive design.
Brand perception: Brand mentions, brand sentiment.

Compliance and legal metrics: Privacy compliance, legal compliance.
Innovation and adaptability: Technology updates, adaptability.

Different Types of Analytics

- Understanding and leveraging different types of analytics provided added value to a business to improve its organization-level operational capabilities.
- Choosing the right forms of data, analytics can make most of the unstructured or structured data they own
- Types of Analytics
  - Descriptive
  - Diagnostic
  - Predictive
  - Prescriptive.



Web 3.0:

Definition:

- Web 3.0, often referred to as the "Semantic Web" or the "Decentralized Web," is the next evolution of the World Wide Web. It represents a vision for the future of the internet that emphasizes decentralized protocols, increased user control over data, and enhanced machine-to-machine communication.

Key Characteristics:

- Decentralization: Web 3.0 envisions a more decentralized internet, moving away from central control of data and services. Blockchain technology and decentralized protocols play a significant role in achieving this.
- Interoperability: Improved interoperability between different platforms and applications is a key focus. This means that data and services should seamlessly work together, breaking down silos that exist on the current web.
- User Empowerment: Web 3.0 aims to give users greater control over their data. Users should have ownership of their personal information and decide who gets access to it. This shift challenges the current model where large tech companies often control and monetize user data.
- Machine Learning Integration: Advanced machine learning and artificial intelligence are expected to play a larger role in Web 3.0, enabling more intelligent and context-aware applications.

- Semantic Understanding: The Semantic Web is a critical component of Web 3.0, where data is not only linked but also given meaning, allowing machines to understand and interpret content in a more human-like way.

Semantic Web:

Definition:

- The Semantic Web is a concept introduced by Sir Tim Berners-Lee, the inventor of the World Wide Web. It involves enhancing the web by adding semantic metadata to web content, enabling machines to understand and interpret the meaning of information.

Key Characteristics:

- Semantic Markup: The Semantic Web relies on the use of standardized semantic markup languages (such as **RDF - Resource Description Framework**) to annotate web content with machine-readable metadata.
- Linked Data: Data on the Semantic Web is not only presented in a human-readable form but is also linked in a way that machines can traverse and understand relationships between different pieces of information.
- **Ontologies:** Ontologies are used to define relationships and vocabularies, creating a shared understanding of terms and concepts. This helps machines interpret data in a way that goes **beyond simple keyword matching.**
- Machine Understanding: The goal is to enable machines to understand the context, meaning, and relationships within data, making it possible to provide more intelligent and context-aware services.

Applications:

- Improved Search: The Semantic Web aims to enhance search engines by enabling them to provide more accurate and contextually relevant results based on a deeper understanding of user queries.
- Data Integration: It facilitates the integration of data from different sources, making it easier for machines to combine and analyze information from diverse datasets.
- Automated Reasoning: The Semantic Web supports automated reasoning, allowing machines to make logical inferences based on the relationships defined in ontologies.

Explain Characteristics of Semantic Web

The Semantic Web is characterized by several key features that **aim to enhance the World Wide Web by making information more meaningful and understandable to both humans and machines.** Here are the main characteristics of the Semantic Web:

Semantic Markup:

- The Semantic Web uses standardized **semantic markup languages**, such as **RDF** (Resource Description Framework), to annotate web content with metadata.

This metadata provides a layer of meaning to the data, allowing machines to understand the **context and relationships within the information.**

Linked Data:

- Data on the Semantic Web is not isolated but is interconnected through a system of links, **creating a web of linked data.** This interconnectivity enables machines to traverse and explore relationships between different pieces of information, forming a more comprehensive understanding.

Ontologies:

- Ontologies are formalized frameworks that **define the relationships and vocabularies used to describe concepts and entities on the Semantic Web.** They provide a shared understanding of terms, allowing for more precise and standardized communication between applications and systems.

Machine Readability:

- The Semantic Web focuses on making **data machine-readable.** This means that information is not only presented in a way that humans can understand but is also structured and encoded in a manner that machines can interpret and process.

Interoperability:

- By employing common standards and ontologies, the Semantic Web promotes interoperability, allowing different systems and applications to **exchange and integrate data seamlessly.** This interoperability is crucial for creating a more connected and unified web.

Automated Reasoning:

- The Semantic Web supports automated reasoning, enabling machines to make logical inferences based on the relationships and rules defined in ontologies. This ability to infer new information enhances the capabilities of applications and services built on the Semantic Web.

Smart Applications:

- Applications on the Semantic Web are designed to be "smarter" in understanding user intent and context. By leveraging semantic data, these applications can provide more relevant and personalized results, leading to a more intelligent and user-friendly web experience.

Data Integration:

- The Semantic Web facilitates the integration of data from diverse sources. Different datasets, even if they use different schemas, can be linked and queried together, allowing for a more comprehensive and holistic analysis of information.

Improved Search and Discovery:

- Semantic technologies enhance search engines by enabling them to understand the meaning behind user queries. This leads to more accurate and contextually relevant search results, improving the overall search and discovery experience for users.

Decentralization of Information:

- The Semantic Web promotes the decentralization of information by **allowing data to be distributed across various sources.** This decentralization aligns with the broader concept of decentralization in Web 3.0.

Explain Components of Semantic Web

RDF (Resource Description Framework):

- RDF is a fundamental framework for **representing information on the Semantic** Web. It provides a standardized way to describe resources and their relationships using **subject-predicate-object triples**. RDF forms the basis for expressing data in a machine-readable format.

Ontologies:

- Ontologies define the **relationships and vocabularies used to describe concepts and entities on the Semantic Web**. They provide a formalized, shared understanding of terms, enabling common interpretation and communication between different systems. Common ontology languages include OWL (Web Ontology Language) and RDFS (Resource Description Framework Schema).

SPARQL (SPARQL Protocol and RDF Query Language):

- SPARQL is a query language and protocol used to **retrieve and manipulate data stored in RDF format.** It allows for querying RDF datasets, making it a crucial component for accessing and extracting information from the Semantic Web.

OWL (Web Ontology Language):

- OWL is a powerful ontology language that allows for the **creation of complex ontologies with rich semantics.** It supports the definition of classes, properties, and individuals, enabling the representation of detailed relationships and constraints within a domain.
- Linked Data:
    - Linked Data principles emphasize the creation of a **web of interconnected datasets using standardized URIs and RDF.** By linking data across different sources, Linked Data enables a more comprehensive and interconnected knowledge graph on the Semantic Web.
- Inference Engines:
    - Inference engines support automated reasoning on the Semantic Web. These engines use ontologies and rules to make logical inferences, deducing new information from existing data. Inference engines enhance the capabilities of applications and services by providing a higher level of intelligence.

Explain Semantic Web Stack,

The Semantic Web Stack, also known as the Semantic Web Layer Cake, is a conceptual representation that illustrates the architecture and layers of technologies that contribute to the development and functioning of the Semantic Web.

URI (Uniform Resource Identifier):

- The foundation of the Semantic Web Stack. URIs uniquely identify resources, providing a means for referencing and linking data on the web. They serve as the basis for creating globally unique identifiers.

Resource Description Framework (RDF):

- RDF is the core data model of the Semantic Web. It allows the representation of information using triples (subject-predicate-object), providing a standardized way to describe resources and their relationships.

RDF Schema (RDFS) and Web Ontology Language (OWL):

- RDFS and OWL are ontology languages that build on RDF. They allow the definition of classes, properties, and relationships, enabling the creation of more sophisticated and expressive ontologies. OWL, in particular, provides more advanced constructs for reasoning and inference.

SPARQL (SPARQL Protocol and RDF Query Language):

- SPARQL is a query language and protocol specifically designed for querying RDF data. It allows for the retrieval and manipulation of data stored in RDF format, facilitating the extraction of information from the Semantic Web.

Linked Data:

- Linked Data principles emphasize the use of URIs to identify resources and the interlinking of datasets. By following Linked Data principles, data can be seamlessly integrated and traversed across the web, forming a connected network of information.

Rules and Logic:

- This layer involves the use of rules and logic to make inferences and draw conclusions from the data. Inference engines use rules specified in ontologies and other logical constructs to deduce new information.

Trust and Proof:

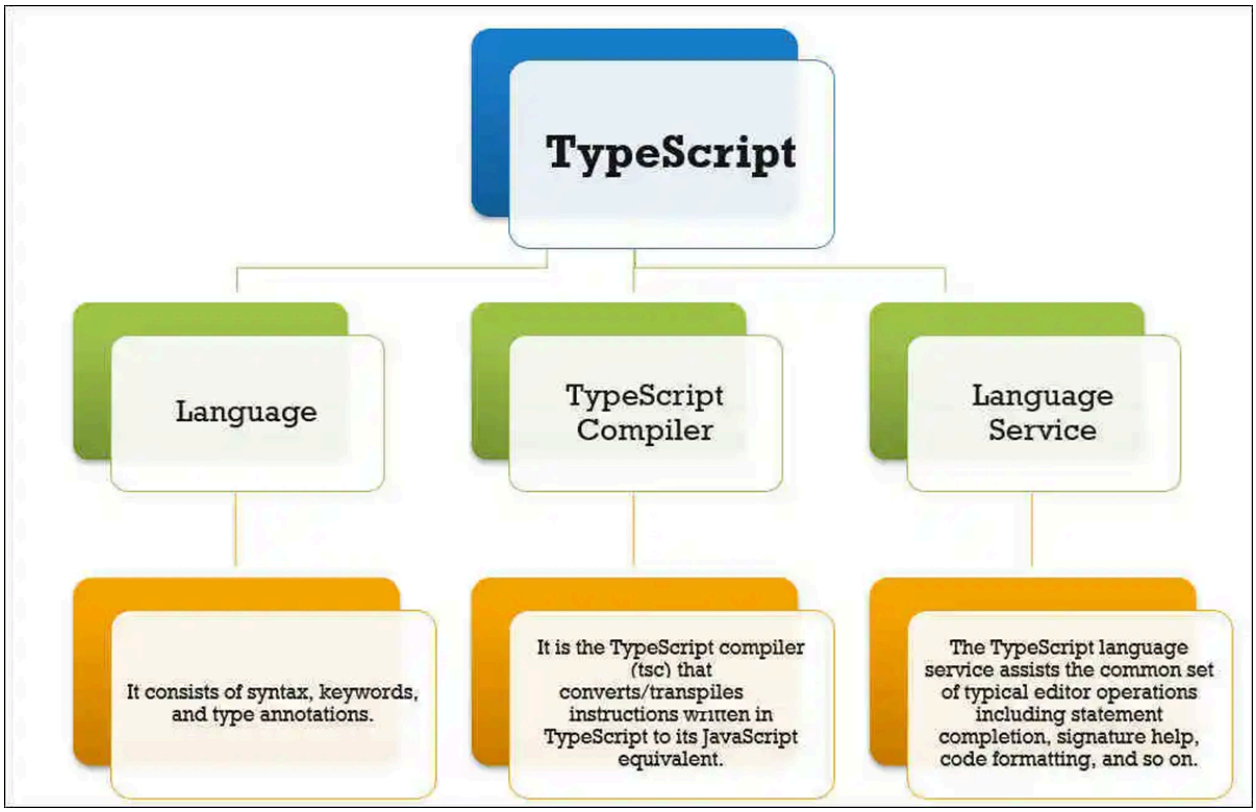
- Trust and proof mechanisms are introduced to assess the reliability and validity of information on the Semantic Web. This layer involves establishing trust in data sources and providing proofs or evidence to support the authenticity of claims.

Natural Language Processing (NLP) and Machine Learning:

- Integrating natural language processing and machine learning techniques into the Semantic Web Stack enhances the understanding and interpretation of unstructured data. This layer allows for more advanced semantic analysis and reasoning.

M2:TYPESCRIPT.ts

Explain TypeScript Internal Architecture



Built-in Data Type	keyword	Description
Number	number	It is used to represent both Integer as well as Floating-Point numbers
Boolean	boolean	Represents true and false
String	string	It is used to represent a sequence of characters
Void	void	Generally used on function return-types
Null	null	It is used when an object does not have any value
Undefined	undefined	Denotes value given to uninitialized variable
Any	any	If variable is declared with any data-type then any type of value can be assigned to that variable

Example:

let a: null = null; let b: number = 123; let c: number = 123.456; let d: string = ‘Geeks’; let e: undefined = undefined; let f: boolean = true; let g: number = 0b111001; // Binary let h: number = 0o436; // Octal let i: number = 0xadf0d; // Hexa-Decimal

Variables in TypeScript:

Variable Declaration with Type Annotation:

- let age: number;
- let name: string;
- let isStudent: boolean;

Variable Declaration with Type Inference:

- let height = 175;
- let greeting = "Hello, TypeScript!";
- let isActive = false;

Operators in TypeScript:

Arithmetic Operators:

- +, -, *, /, %

Comparison Operators:

- ===, !==, >, <, >=, <=

Logical Operators:

- && (and), || (or), ! (not)

Assignment Operators:

- +=, -=, *=, /=

Type Assertion Operator:

- <type>value (Angle-bracket syntax)
- value as type (as syntax)

String Operators:

- + (String concatenation)
- Template literals: `Hello, ${fullName}!`

TypeScript Functions:

Functions in TypeScript are similar to functions in JavaScript, but **TypeScript introduces static typing, allowing you to declare the types of parameters and the return type**. This enhances code r**eadability, helps catch errors early in the development process**, and improves tooling support.

1. Function Declaration:

```
function add(x: number, y: number): number {
   return x + y;
}
```

2. Function Expression:

```
const multiply = function (x: number, y: number): number {
    return x * y;
};
```

3. Arrow Function:

```
const divide = (x: number, y: number): number => x / y;
```

4.Generics in Functions:

```
function identity<T>(arg: T): T {
    return arg;
}

const result: number = identity<number>(42);
```

Class Object:

```
class Person {
    firstName: string;
    lastName: string;

    constructor(firstName: string, lastName: string) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
}

// Creating an object of the class
const person1 = new Person("John", "Doe");

// Accessing properties and methods
console.log(person1.firstName); // "John"
console.log(person1.getFullName()); // "John Doe"
```

TypeScript Modules:

Exporting

```
// math.ts
export function add(x: number, y: number): number {
    return x + y;
}

export function subtract(x: number, y: number): number {
    return x - y;
}
```

—----------

Importing

```
// app.ts
import { add, subtract } from './math';

console.log(add(5, 3)); // 8
console.log(subtract(8, 3)); // 5
```

Inheritance

```
class Animal {
    name: string;

    constructor(name: string) {
        this.name = name;
    }

    makeSound(): void {
        console.log("Some generic sound");
    }
}

class Dog extends Animal {
    breed: string;

    constructor(name: string, breed: string) {
        super(name);
        this.breed = breed;
    }

    makeSound(): void {
        console.log("Woof! Woof!");
    }
}

// Creating an object of the derived class
const myDog = new Dog("Buddy", "Golden Retriever");

// Accessing properties and methods
console.log(myDog.name); // "Buddy"
myDog.makeSound(); // "Woof! Woof!"
```

Function Overloading

```
function greet(name: string): string;
function greet(firstName: string, lastName: string): string;

function greet(arg1: string, arg2?: string): string {
    if (arg2) {
        return `Hello, ${arg1} ${arg2}!`;
    } else {
        return `Hello, ${arg1}!`;
    }
}

console.log(greet("John")); // "Hello, John!"
console.log(greet("John", "Doe")); // "Hello, John Doe!"
```

Explain features in TS.

TypeScript (TS) is a superset of JavaScript that adds static typing and other features to the language. Here are some key features of TypeScript:

1. Static Typing:

- TypeScript introduces static typing, allowing developers to declare the types of variables, parameters, and return values. This helps catch type-related errors during development and provides better tooling support.

2. Interfaces:

- Interfaces in TypeScript allow developers to define the structure of objects, providing a way to enforce a specific shape for classes or objects. This enhances code maintainability and helps catch errors early.

3. Classes and Objects:

- TypeScript supports object-oriented programming features such as classes, inheritance, and encapsulation. This makes it easier to structure and organize code in a more modular and reusable way.

4. Generics:

- TypeScript introduces generics, allowing developers to write reusable and type-safe code. Generics enable the creation of functions and classes that can work with a variety of data types.

5. Enums:

- Enums in TypeScript provide a way to define a set of named constant values. This helps improve code readability by giving meaningful names to numeric values.

6. Modules:

- TypeScript supports the concept of modules, allowing developers to organize code into separate files and namespaces. This enhances code organization and supports better code sharing and reuse.

7. Arrow Functions:

- TypeScript supports arrow functions, providing a more concise syntax for writing functions. Arrow functions also capture the this value from the surrounding context, eliminating the need for the function keyword.

JavaScript Vs TypeScript

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.	In this, number, string are the objects.	In this, number, string are the interface.

7.	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:number, b:number) {
		return a + b;
		}
		var sum = addNumbers(15, 25);
		console.log('Sum of the numbers is: ' + sum);

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;
		}

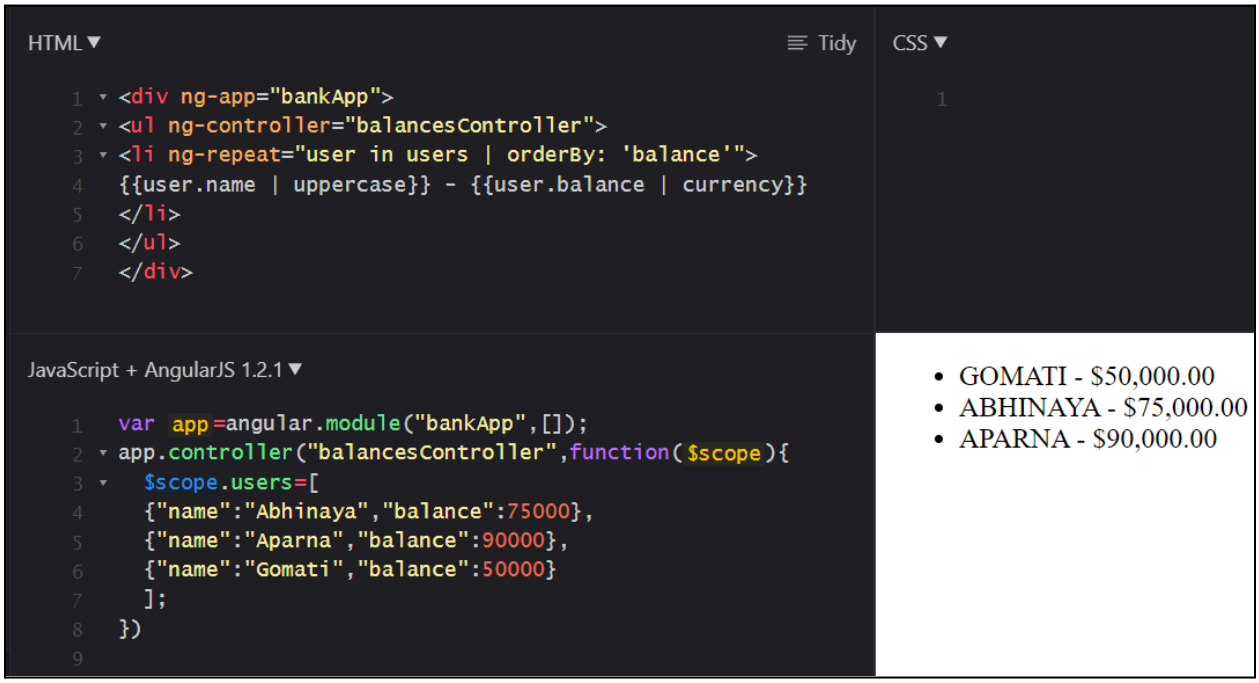
M3:ANGULAR

Discuss the Need of AngularJS in web sites

- Declarative UI: AngularJS simplifies UI development by using a declarative approach in HTML.
- Two-Way Data Binding: Automatic synchronization between the model and view, reducing manual DOM manipulation.
- Modularity and Dependency Injection: Encourages modular code and facilitates dependency injection for better maintainability.
- Dynamic Templating: Supports dynamic views with expressions, filters, and directives.
- Directives: Custom HTML attributes or elements extend HTML functionality, enabling reusable components.
- Testing Support: Designed for testability, providing features like dependency injection and unit testing support.
- SPA Development: Well-suited for Single Page Application (SPA) development with routing capabilities.
- Community and Ecosystem: Large and active community support, access to resources, and third-party tools.
- Official Documentation and Support: Comprehensive documentation and official support from Google.
- Cross-Browser Compatibility: Abstracts away complexities, ensuring a consistent development experience across browsers.

Explain AngularJS modules

example:



Http example

Explain AngularJS built-in directives with example and syntax.

AngularJS provides a set of built-in directives that **enhance the functionality and behavior of HTML elements**

AngularJS provides a set of built-in directives that enhance the functionality and behavior of HTML elements. Here are some commonly used AngularJS directives along with examples and syntax:

1. ng-app:

- Purpose: Defines the root element of an AngularJS application and auto-bootstraps it.
- Example:
  <html ng-app="myApp">
  Js:
  var myApp = angular.module('myApp', []);

2. ng-controller:

- Purpose: Attaches a controller to a specific HTML element, establishing the scope for the controller.
- Example:

```
<div ng-controller="MyController">
  {{ message }}
</div>
myApp.controller('MyController', function($scope) {
    $scope.message = 'Hello, AngularJS!';
});
```

3. ng-model:

- Purpose: Binds an input, select, or textarea element to a property on the scope, enabling two-way data binding.
- Example:
  <input type="text" ng-model="username">
  {{ username }}

4. ng-repeat:

- Purpose: Iterates over a collection and repeats a set of HTML elements for each item in the collection.
- Example:

  <li ng-repeat="item in items">{{ item.name }}

  $scope.items = [
  { name: 'Item 1' },
  { name: 'Item 2' },
  { name: 'Item 3' }
  ];

5. ng-click:

- Purpose: Executes a function when the element is clicked.
- Example:
  <button ng-click="showMessage()">Click me</button>

  $scope.showMessage = function() {
  alert('Button clicked!');
  };

6. ng-disabled:

- Purpose: Disables the associated form element or button based on a given expression.

<button ng-disabled="isDisabled">Submit</button>

$scope.isDisabled = true;

7.ng-bind ng-init:

Purpose: ng-bind is an AngularJS directive used to bind the content of an HTML element to an expression, typically a model property. It is an alternative to using double curly braces {{ }} for data binding.

Inside directive:

```
<div ng-app="" ng-init=" first string variable name='your first string';
second string variable name='your second string'">
<p>My first string expression in Angular JS:<span ng-bind=
" first string variable name + second string variable name "></span>
</p>
</div>
```

Custom Directive:

Example of a Simple Directive:

js:

```
myApp.directive('myDirective', function() {
   return {
      restrict: 'E',
      template: '<div>This is my custom directive!</div>'
   };
});
```

html:

```
<my-directive></my-directive>
```

Link Function:

js:

```
myApp.directive('customColor', function() {
   return {
      restrict: 'A',
      link: function(scope, element, attrs) {
         element.css('color', attrs.customColor);
      }
   };
});
```

html:

```
<div custom-color="red">Red Text</div>
```

AngularJS Filters:

- Purpose: Filters in AngularJS are used to **format and transform data in expressions.** They can be applied to expressions in the HTML template to modify the way data is displayed.
  E.g:
  ** {{ price | currency }} **

  Common Filters:
  - currency: Formats a number as a currency.
  - date: Formats a date.
  - uppercase and lowercase: Converts text to uppercase or lowercase.
  - filter: Filters an array based on a specified criteria.
  - orderBy: Orders an array by a specified expression.

AngularJS Controllers:

- Purpose: Controllers in AngularJS are **responsible for handling user input, business logic, and managing the state of the application.** They act as an intermediary between the model and the view.
- Example:

```
myApp.controller('MyController', function($scope) {
 $scope.message = 'Hello, AngularJS!';
});
```

Responsibilities:

- Managing Scope: Controllers define properties and methods on the scope, making data available to the view.
- Handling User Input: Controllers respond to user actions such as clicks, input changes, etc.
- Business Logic: Implementing business logic and coordinating interactions with services.
- Modularization: Breaking down the application into smaller, manageable pieces.

- Controller As Syntax:
  - The "Controller As" syntax is an alternative way to define controllers, aliasing the controller's properties and methods under a specified name.

```
myApp.controller('MyController', function() {
 this.message = 'Hello, AngularJS!';
});

<div ng-controller="MyController as ctrl">{{ ctrl.message }}</div>
```

AngularJS Scope:

- Purpose: The scope in AngularJS is an object that refers to the application model. It acts as a **bridge between the controller and the view**, providing data binding and synchronization.
- Example:

```
myApp.controller('MyController', function($scope) {
 $scope.message = 'Hello, AngularJS!';
});
```

- In this example, $scope.message is a property on the scope, and changes to this property are reflected in the associated view.
- Two-Way Data Binding:
  - Changes made to properties on the scope in the controller are automatically reflected in the view, and vice versa.
- Hierarchy:
  - AngularJS scopes form a hierarchy based on the DOM structure. Each controller creates a new scope, and child scopes inherit properties from their parent scope.
- Scope Functions:
  - Scopes can have functions that can be called from the view, promoting encapsulation of logic.
-

```
$scope.sayHello = function() {
 alert('Hello!');
};

<button ng-click="sayHello()">Say Hello</button>
```

AngularJS Dependency Injection:

- Purpose: Dependency Injection (DI) in AngularJS is a design pattern where components (services, controllers, etc.) declare their dependencies, and AngularJS provides those dependencies at runtime.
- Example:

  myApp.controller('MyController', ['$scope', 'myService', function($scope, myService) {
  // Controller logic using $scope and myService
  }]);

- In this example, the controller depends on $scope and myService. AngularJS **automatically injects these dependencies when the controller is instantiated.**

AngularJS Services:

- Purpose: Services in AngularJS are singleton objects that provide functionality to various parts of the application. They promote code organization and sharing of functionality across different components.
- **Example:**
  myApp.service('myService', function() {
  this.getData = function() {
  // Service logic to fetch data
  };
  });
- In this example, myService is a simple service with a getData method.
- Common Use Cases:
  - Data Sharing: Services are often used for sharing data between controllers.
  - Business Logic: Encapsulating business logic that can be reused across controllers.
  - HTTP Requests: Handling HTTP requests and responses.

Design a Registration/Feedback Form with Validation:

```
<form name="registrationForm" ng-submit="submitForm()" novalidate>
  <label for="name">Name:</label>
  <input type="text" id="name" name="name" ng-model="user.name" required>

  <label for="email">Email:</label>
  <input type="email" id="email" name="email" ng-model="user.email" required>

  <button type="submit" ng-disabled="registrationForm.$invalid">Submit</button>
</form>
```

Routing using ng-Route, ng-Repeat, ng-Style, ng-View:

- Routing with ng-Route:
    - AngularJS ngRoute module provides client-side routing. It allows switching between views based on the route in the URL.

```
myApp.config(['$routeProvider', function($routeProvider) {
 $routeProvider
 .when('/home', {
 templateUrl: 'views/home.html',
 controller: 'HomeController'
 })
 .when('/about', {
 templateUrl: 'views/about.html',
 controller: 'AboutController'
 })
 .otherwise({
 redirectTo: '/home'
 });
}]);
```

- **ng-Repeat:**
    - ngRepeat is used to iterate over a collection and repeat a set of HTML elements for each item in the collection.
- Example:

```
<ul>
 <li ng-repeat="item in items">{{ item.name }}</li>
</ul>
```

- **ng-Style**:ngStyle allows dynamically setting CSS styles based on expressions.
- Example:

```
<div ng-style="{ 'color': textColor, 'font-size': fontSize + 'px' }">Styled Text</div>
```

- **ng-View:**
    - ngView is a directive that **renders views based on the current route.**
- Example:

```
<div ng-view></div>
```

- In the routing configuration, the templateUrl points to different HTML files for different views.

Built-in Helper Functions:

- AngularJS provides several built-in helper functions and directives:
    - ng-show and ng-hide: Show or hide elements based on a condition.
    - ng-if: Conditionally renders or removes elements from the DOM.
    - ng-click: Defines a click event handler.
    - ng-disabled: Disables a button or input field based on a condition.
    - ng-class and ng-style: Dynamically apply CSS classes or styles.
    - ng-model: Binds an input field to a model property for two-way data binding.
    - ng-options: Generates <option> elements for a <select> dropdown.