MALLA REDDY UNIVERSITY
(Telangana State Private Universities Act No. 13 of 2020 & G.O.Ms.No. 14, Higher Education (UE) Department)
Maisammaguda, Kompally, Hyderabad - 500100, Telangana State.

I-B.Tech II-Sem

Question Bank UNIT-1

SUBJECT: UI FRAMEWORKS

CODE: MR24-1CS0105

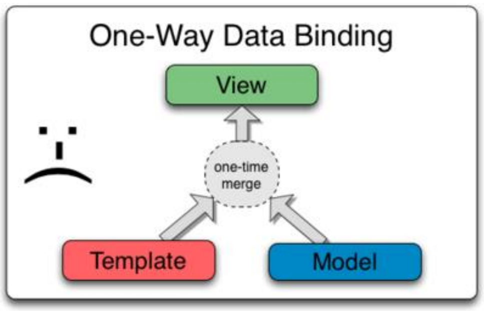
1 What is React and why is it used? Explain the features of React.

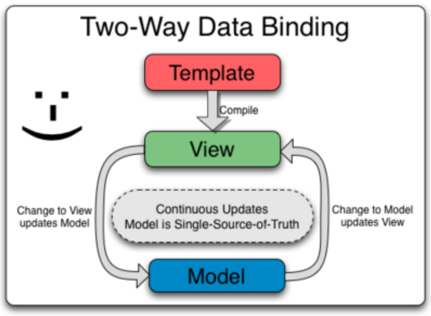
ReactJS is an open-source JavaScript library used to build a user interface for Single Page Applications (SPA). Examples for Single Page Applications are Facebook, Gmail, Twitter etc. It is responsible only for the view layer of the application. It provides developers to compose complex UIs from a small and isolated piece of code called components. ReactJS uses virtual DOM (JavaScript object), which improves the performance of the app. The JavaScript virtual DOM is faster than the regular DOM.The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.

Features of React JS:

1. **JSX:** JSX stands for JavaScript XML. It allows the user to write HTML in React by converting HTML tags into react elements.
2. **Components:** ReactJS is all about components. A component is a reusable piece of HTML code. A ReactJS application is made up of multiple components, each component is responsible for outputting a small, reusable piece of HTML code. We can reuse the components while building large scale applications.
3. **One-way Data Binding:**

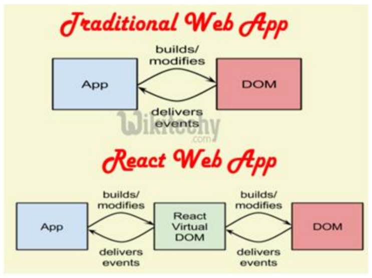
Data binding is the process that establishes a connection between the app UI and the data it displays.




Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model updates View
Model

In one-way data binding information flows in only one direction, and is when the information is displayed, but not updated where as in two-way data binding information flows in both directions and is used in situations where the information needs to be updated.

4. **Virtual DOM:** A virtual DOM object is a representation of the real DOM object. Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation.



5. **Simplicity:** ReactJS uses the JSX file, which makes the application simple and to code, as well as understand. Also, ReactJS is a component-based approach which makes the code reusable as your need.

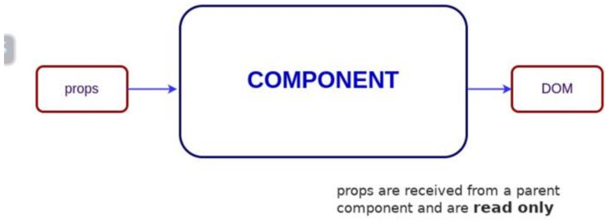
2 What is React component? Explain Class component and Functional component with examples.

Components are the fundamental building blocks of React applications. They act as independent, reusable pieces of UI that encapsulate both the logic and presentation of a particular portion of the interface. Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of our application. There are two types of components:

1. Functional Components
2. Class Components

1. Functional Component:

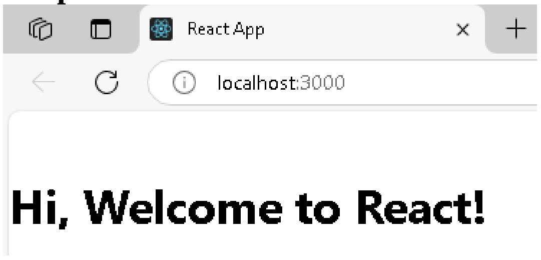
In React, function components are simply JavaScript functions that may or may not receive data as parameters. The functional component is also known as a stateless component because they do not hold or manage state.


props
COMPONENT
DOM
props are received from a parent component and are **read only**

Example:

```
import React from 'react';
function welcome() {
  return(
      <h2>Hi, welcome to React!</h2>
      )
}
export default welcome;
```

Output:

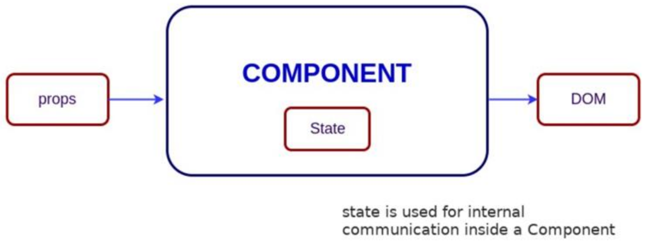


React App

localhost:3000

Hi, Welcome to React!

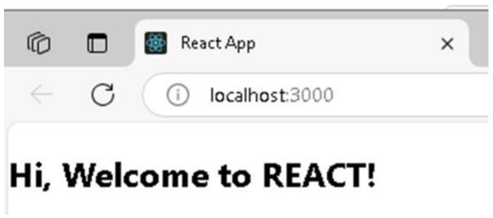
2. **Class Component:**
   - A class component must include the extends Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.
   - The component also requires a render() method, this method returns HTML.
   - The class component is also known as a stateful component because they can hold or manage local state.



Example:

```
import React, { Component } from 'react';
class mycomp extends Component {
  render() {
    return <h2>Hi, Welcome to REACT!</h2>;
  }
}
export default mycomp;
```

```
Output:
```



React App

localhost:3000

Hi, Welcome to REACT!

3 **Write the steps involved in React installation.**

Step 1: Download the Node.js source code or a pre-built installer for your platform using the following link: ***https://nodejs.org/en/download*.**

Step 2: Open command prompt and type the following to check whether it is completely installed or not: ***node –version***

Step 3: Now in the terminal run the below command: ***npm install -g create-react-app.***
It will globally install react app for you. To check everything went well run the command ***create-react-app –version***

Step 4: Create a new folder where you want to make your react app using the below command: ***mkdir myfolder***
Move inside the same folder using the command: ***cd myfolder***

Step 5: To create the default React APP run the following command: ***create-react-app myapp***

Step 6: Now open the IDE of your choice for eg. Visual studio code and then go to File -> open folder -> myfolder -> myapp and type the following command in terminal to run the app: ***npm start***

4 **Explain how State is declared in React with an example.**

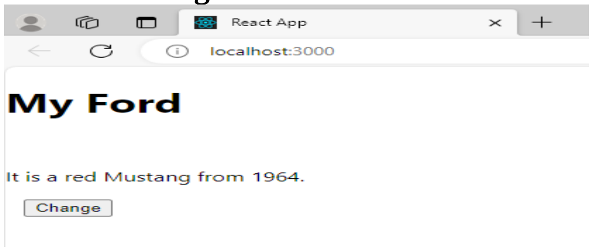
React components have a built-in state object. The state object is where the property values are stored which belong to the component. The state object is initialized in the constructor.
To change a value in the state object, **this.setState()** method is used.
constructor(): used to initialize objects i.e, State in React JS
super(): This function is used to access and call the parent class's constructor, methods, and variables.

Example:
```
import React, { Component } from 'react';
class Car extends Component {
constructor() {
      super();
      this.state = {
            brand: "Ford",
            model: "Mustang",
            color: "red",
            year: 1964 };
 }
changeColor = () => { this.setState({color: "blue"})};
render() {
  return (
  <div>
```

```
   <h1>My {this.state.brand}</h1>
     <p>   It   is   a   {this.state.color}   {this.state.model}   from
{this.state.year}. </p>
   <button type="button" onClick={this.changeColor} >Change</button>
   </div> ); } }
export default Car
```

OUTPUT:

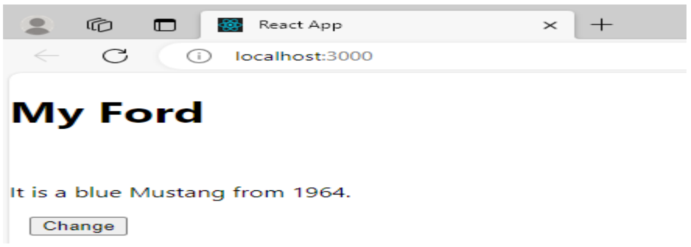
Before clicking the Button:



My Ford

It is a red Mustang from 1964.

After Clicking the Button:



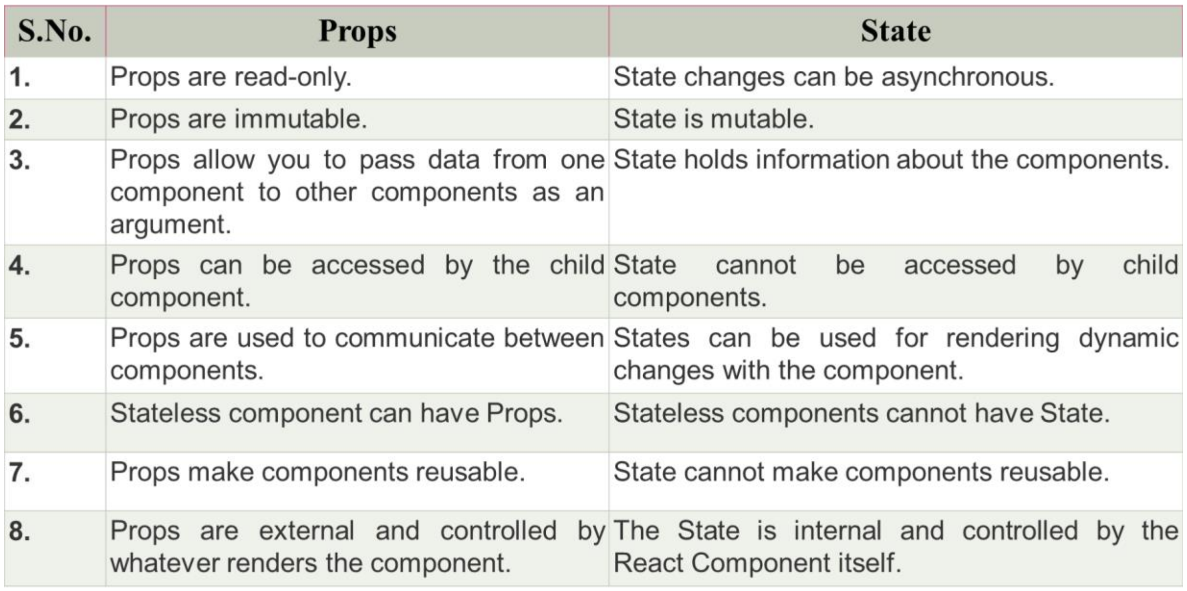
My Ford

It is a blue Mustang from 1964.

5 What are Props in React and how they differ from React States?

Props:
Props stand for "Properties." They are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from one component to other components.

Differences between State and Props:



S.No.	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.
5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
6.	Stateless component can have Props.	Stateless components cannot have State.
7.	Props make components reusable.	State cannot make components reusable.
8.	Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

6 **Explain the types of forms in React. Write a code to create a basic form in React.**

There are mainly two types of form input in React.

- ➢ Uncontrolled component
- ➢ Controlled component

Uncontrolled component:

- ➢ In the Uncontrolled Form component, the state of the input is not controlled by the React component.
- ➢ This means that when the user changes the value of the input, the state is not updated and the component is not re-rendered.
- ➢ The uncontrolled input is similar to the traditional HTML form inputs.
- ➢ The DOM itself handles the form data.
- ➢ Here, the HTML elements maintain their own state that will be updated when the input value changes.

Controlled Component

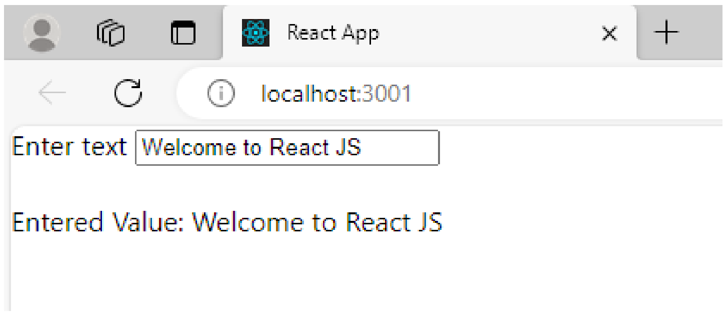
- ➢ In the ControlledForm component, the state of the input is controlled by the React component.
- ➢ This means that when the user changes the value of the input, the state is updated and the component is re-rendered.
- ➢ In the controlled component, the input form element is handled by the component rather than the DOM.

Example:

```
import React from 'react';
class App extends React.Component {
    state = { inputValue: '' };
    render() {
        return (
            <div>
```

```
                <form>
                    <label> Enter text </label>
                    <input type="text"
                        value={this.state.inputValue}
                        onChange={(e) => this.setState(
                            { inputValue: e.target.value })} />
                </form> <br />
                <div>
                    Entered Value: {this.state.inputValue}
                </div>
            </div>
        );}
}
export default App;
```

OUTPUT:



```
Enter text Welcome to React JS

Entered Value: Welcome to React JS
```

7 Explain phases of React Component Lifecycle.

In ReactJS, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component's lifecycle.
The lifecycle of the component is divided into **four phases**. They are:
1. Initial Phase
2. Mounting Phase
3. Updating Phase
4. Unmounting Phase

1. Initial Phase:

- It is the birth phase of the lifecycle of a ReactJS component. Here, the component starts its journey on a way to the DOM.
- Component contains the default Props and initial State.
- These default properties are done in the constructor of a component.
- The initial phase only occurs once and consists of the following methods.
    a. **getDefaultProps()**
    b. **getInitialState()**

2. Mounting Phase:

Mounting means putting elements into the DOM. React has four built-in methods that gets called, in this order, when mounting a component:

- componentWillMount()
- componentDidMount()
- render()

The render() method is required and will always be called, the others are optional and will be called if you define them.

3. Updating Phase:

A component is updated whenever there is a change in the component's state or props. This phase consists of the following methods:

- componentWillRecieveProps()
- shouldComponentUpdate()
- componentWillUpdate()
- render()
- componentDidUpdate()

4. Unmounting Phase:

- The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it.
- React has only one built-in method that gets called when a component is unmounted: ***componentWillUnmount()***

8. Explain React Props with an example.

Props stand for "Properties." They are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from one component to other components. Props are passed to the component in the same way as arguments passed in a function. Props are immutable so we cannot modify the props from inside the component.

Example:

App.js File:

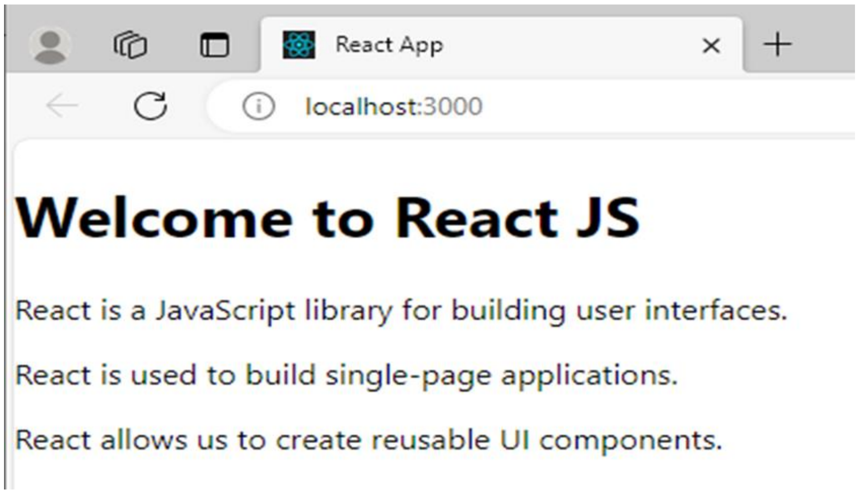
```
import React, { Component } from 'react';
class App extends React.Component {
   render() {
      return (
          <div>
            <h1> Welcome to { this.props.name } </h1>
```

```
                <p>   React  is  a  JavaScript  library  for  building  user
interfaces.</p>
            <p>React is used to build single-page applications.</p>
                <p>React  allows  us  to  create  reusable  UI  components.
</p>
          </div>
      );
   }  }
export default App;
```

index.js File:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App name="React JS" />

  </React.StrictMode>
);

reportWebVitals();
```

OUTPUT:



Welcome to React JS

React is a JavaScript library for building user interfaces.

React is used to build single-page applications.

React allows us to create reusable UI components.

9. What is React JSX? What are the advantages of React JSX? Write a program using React JSX.

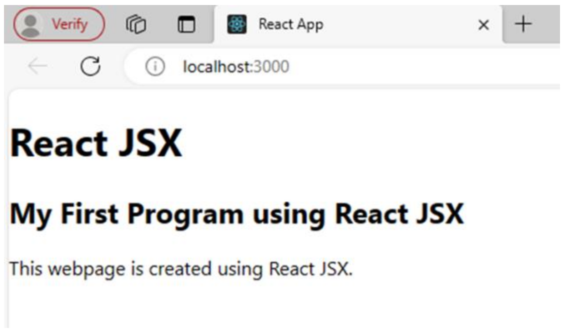
JSX stands for JavaScript XML. JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods. JSX converts HTML tags into react elements.
Key Advantages of JSX are:

- **Visually Expressive:** Writing UI elements in JSX closely resembles HTML, making it easier to visualize and understand the structure of your components.
- **Combines Logic and Markup:** JSX enables you to embed JavaScript expressions within the markup, allowing for dynamic content and seamless integration of logic and presentation.
- **Enhanced Tooling and Error Checking:** Compilers and linters can provide better error messages and type checking with JSX, improving code quality and development experience.

Program:

```
import React, { Component } from 'react';
class App extends Component{
  render(){
    return(
      <div>
        <h1>React JSX</h1>
       <h2>My First Program using React JSX</h2>
        <p>This webpage is created using React JSX.</p>
      </div>
    );
  }
}
export default App;
```

OUTPUT:



React JSX

My First Program using React JSX

This webpage is created using React JSX.

10. What is Virtual DOM? How virtual DOM is different from actual DOM?

- The virtual DOM (VDOM) is a programming concept where an ideal, or “virtual”, representation of a UI is kept in memory and synced with the “real” DOM by a library such as ReactDOM.
- In React world, the term “virtual DOM” is usually associated with React elements since they are the objects representing the user interface.
- Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation.

Differences between Virtual DOM and Actual DOM:

Virtual DOM	Real DOM
It is a lightweight copy of the original DOM	It is a tree representation of HTML elements
It is maintained by JavaScript libraries	It is maintained by the browser after parsing HTML elements
After manipulation it only re-renders changed elements	After manipulation, it re-render the entire DOM
Updates are lightweight	Updates are heavyweight
Performance is fast and UX is optimised	Performance is slow and the UX quality is low
Highly efficient as it performs batch updates	Less efficient due to re-rendering of DOM after each update

I-B.Tech II-Sem

Question Bank UNIT-2

SUBJECT: UI FRAMEWORKS

CODE: MR24-1CS0105

1 How to define an event in React? Write a program for creating an "onClick" event.

Answer

- ➢ An event is an action that could be triggered as a result of the user action or system generated event. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.
- ➢ React has its own event handling system which is very similar to handling events on DOM elements. **The react event handling system is known as Synthetic Events.** The synthetic event is a cross-browser wrapper of the browser's native event.
- ➢ Handling events with react have some syntactic differences from handling events on DOM.

These are:

1. React events are named as **camelCase** instead of **lowercase**.
   (Eg.:onClick instead of onclick)
2. With JSX, a function is passed as the **event handler** instead of a **string**.

Event declaration in React:

- ➢ **In Functional Component:**
  <button onClick={showMessage}> Hello MRU </button>
- ➢ **In Class Based Component:**
  <button onClick={this.showMessage}>Hello MRU</button>

Program

```
import React from 'react';
function event() {
  const alertBox = () => {
    alert("Welcome to React Event !");
  }
  return (
    <div>    <h1>React Event</h1>
    <h3>This Example explains onlick event in React</h3>
    <button onClick={alertBox}>Click Here</button>
    </div>
  );
}
export default event;
```

2 Write a program to display the following ordered list in React:

i. **HTML**
ii. **Java**
iii. **JavaScript**
iv. **SQL**
v. **Python**

Progarm

```
import React, { Component } from 'react';
class App extends Component{
 render(){
 const listItems = ['HTML', 'Java','JavaScript','SQL','Python'];
 const list = listItems.map((item) => <li>{item}</li>);
 return (
 <div>
 <h1>Creating an Ordered list</h1>
<h2>The list items are displayed below </h2>
<ol>{list}</ol>
 </div>
 )
}
 }
export default App;
```

3 **Explain the purpose of Keys in React lists with an example.**

Answer

- A “key” is a special string attribute you need to include when creating lists of elements in React.
- Keys allow React to keep track of elements. This way, if an item is updated or removed, only that item will be re-rendered instead of the entire list.
- Generally, the key should be a unique ID assigned to each item. As a last resort, you can use the array index as a key.
- Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity

Example

```
import React from "react";
function MyComponent() {
  const data = [
    { id: 1, name: "John" ,dob:12 },
    { id: 2, name: "Jane" ,dob:13 },
    { id: 3, name: "Alex" ,dob:14},
  ];
return (
    <div>
      {data.map((item) => (
        <div key={item.id}>
          <p>{item.dob}</p>
        </div>
      ))}
    </div>
  );
}

export default MyComponent;
```

The above program display only the dob of the data.

Output :

```
Id:1 12
Id:2 13
Id:3 14
```

4 What is the difference between key and ref in React?

Answer

Key	**Ref**
Helps React identify and track list items for efficient updates	Provides a reference to a DOM element or React component
Used in Lists (e.g., `.map()` rendering multiple elements)	Used in Direct access to DOM elements or component instances
Not accessible within the component (`props.key` is not available)	Accessible within the component using ref.current
Improves performance by avoiding unnecessary re-renders	Used for manipulating DOM elements (e.g., focusing an input, measuring dimensions)
Syntax <Component key={id} />	Syntax <input ref={inputRef} />
Example {items.map(item => <ListItem key={item.id} />)}	Example const ref = useRef(); <div ref={ref} />

5 What is React Router? Explain the purpose of React Router with an example.

- React Router is a powerful routing library built on top of React that helps you add new screens and flows to your application incredibly quickly, all while keeping the URL in sync with what's being displayed on the page.
- Routing is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications.
- When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.

Need of React Router:

- React Router plays an important role to display multiple views in a single page application. Without React Router, it is not possible to display multiple views in React applications.
- Most of the social media websites like Facebook, Instagram uses React Router for rendering multiple views.
- **react-router:** It provides the core routing components and functions for the React Router applications.
- **react-router-native:** It is used for mobile applications.
- **react-router-dom:** It is used for web applications design.

To use react routing install **react-router-dom** modules in your application.

Components in React Router

There are two types of router components:

- **<BrowserRouter>:** It is used for handling the dynamic URL.
- **<HashRouter>:** It is used for handling the static request.

Installation

Step 1:**npm install react-router-dom** in terminal

Step 2:Include import { BrowserRouter } from "react-router-dom" and < BrowserRouter >

in index.js

about.js

```
import React from 'react'
function about() {
   return (
    <div> <h2>About page</h2>  </div>
   )
}
export default about
```

contact.js

```
import React from 'react'
function about() {
   return (
    <div> <h2>Contact page</h2>  </div>
   )
}
export default about
```

nav.js

```
import { Link } from 'react-router-dom'
const Nav = () => {
   return(
      <nav>
            <Link to="/">Home</Link>
            <Link to="/about">About</Link>
      </nav>
   )
}
export default Nav;
```

app.js

```
import { Routes, Route } from "react-router-dom"
import Home from "./home"
import About from "./about"
import Nav from "./nav"
function App() {
  return (
   <div >
     <h1><center>MALLA REDDY UNIVERSITY</center></h1>
   <Nav/>
   <Routes>
     <Route path="/" element={ <Home/> } />
     <Route path="about" element={ <About/> } />
   </Routes>
  </div>
 )
}
export default App
```

6 Explain the concept of Higher-Order Components (HOCs) in React.

Answer

- It is also known as HOC. In React, HOC is an advanced technique for reusing component logic.
- **It is a component that takes another component as input and returns a new component with extra features added to the original component.**
- A higher order component function accepts another function as an argument.
- The main goal of this is to decompose the component logic into simpler and smaller functions that can be reused as you need.

A function is said to be Higher Order Component if:

- **Case 1:** If another function is called into this function or
- **Case-2:** If another function is returned from this function

HOC.js

```
import React, {Component} from 'react';


function Hoc (HocComponent){
    return class extends Component{
        render(){
            return (
                <div>
                    <HocComponent/>


                </div>


            );
        }
```

App.js

```
import React, { Component } from 'react';
import HOC from './HOC';
  class App extends Component {
  render() {
    return (
      <div>
        <h2>HOC Example</h2>
        <h3>HOC is a component that takes another component as input and returns a new
component with extra features added to the original component</h3>
      </div>
    )
  }
}
App = HOC(App);
export default App;
```

7 Write a program to create a table in React.

Answer

```
import React  from 'react';
import './App.css'

function App() {
    return (
    <div >
        <h1>Table using React</h1>
        <table>
            <tr>
                <th>Name</th>
                <th>Age</th>
                <th>Gender</th>
            </tr>
            <tr>
                <td>Anish</td>
                <td>19</td>
                <td>Male</td>
            </tr>
             <tr>
                <td>Megha</td>
                <td>19</td>
                <td>Female</td>
            </tr>
            <tr>
                <td>Subham</td>
            <td>25</td>
            <td>Male</td>
          </tr>
        </table>
</div>
);
}

export default App;
```

8. What are React Hooks? Write a program using useState Hook in React.

Answer

- Hooks are the new feature introduced in the React 16.8 version.
- It allows you to use state and other React features without writing a class.
- It does not work inside classes.
- If you write a function component, and then you want to add some state to it, previously you do this by converting it to a class.
- But, now you can do it by using a Hook inside the existing function component

Rules of Hooks

1. Only call Hooks at the top level:

Do not call Hooks inside loops, conditions, or nested functions. Hooks should always be used at the top level of the React functions. This rule ensures that Hooks are called in the same order each time a components renders.

2. Only call Hooks from React functions:

You cannot call Hooks from regular JavaScript functions. Instead, you can call Hooks from React function components. Hooks can also be called from custom Hooks.

Progarm

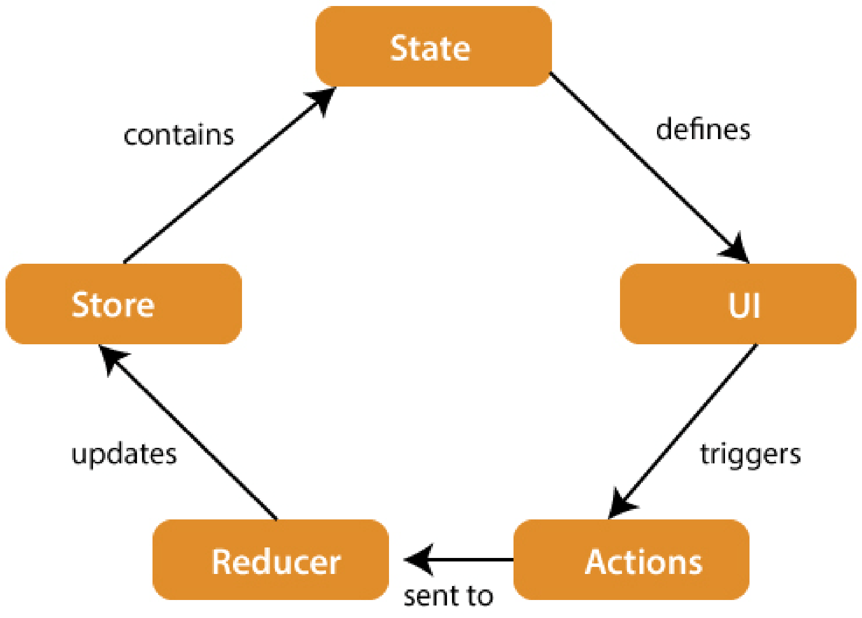
```
import React, { useState } from 'react';


function CountApp() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);


  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
export default CountApp;
```

9. What is React Redux? Explain the components of Redux architecture.

Answer

- Redux is an open-source JavaScript library used to manage application state. React uses Redux for building the user interface.
- React Redux is the official React binding for Redux. It allows React components to read data from a Redux Store, and dispatch **Actions** to the **Store** to update data.
- Redux helps apps to scale by providing a sensible way to manage state through a unidirectional data flow model. React Redux is conceptually simple.
- It subscribes to the Redux store, checks to see if the data which your component wants have changed, and re-renders your component.
- React Redux is the official **UI bindings** for react Application. It is kept up-to-date with any API changes to ensure that your React components behave as expected.
- It implements many performance optimizations internally, which allows to components re-render only when it actually needs.

Redux architecture.



The components of Redux architecture are explained below.

STORE: A Store is a place where the entire state of your application lists. It manages the status of the application and has a dispatch(action) function. It is like a brain responsible for all moving parts in Redux.

ACTION: Action is sent or dispatched from the view which are payloads that can be read by Reducers. It is a pure object created to store the information of the user's event. It includes information such as type of action, time of occurrence, location of occurrence, its coordinates, and which state it aims to change.

REDUCER: Reducer read the payloads from the actions and then updates the store via the state accordingly. It is a pure function to return a new state from the initial state.

10. What is the importance of code splitting in React? Explain code splitting with an example.

Answer

- The React app bundles their files using tools like Webpack or Browserfy.
- Bundling is a process in which multiple files are merged into a single file, which is called a bundle.
- The bundle is responsible for loading an entire app at once on the webpage.
- Code-Splitting is a feature supported by Webpack and Browserify, which can create multiple bundles that can be dynamically loaded at runtime.
- Code splitting uses React.lazy and Suspense tool/library, which helps to load a dependency lazily and only load it when needed by the user.
- When building large-scale applications, the JavaScript bundle can become quite large, which can impact the load time of the application.
- Code splitting is a technique where we split our code into various bundles which can then be loaded on demand or in parallel.
- This can significantly reduce the load time of our application.

React.lazy

- **React.lazy** is a function that lets you render a dynamic import as a regular react component.
- It makes it possible to load components lazily, which can be very useful for reducing the size of the initial JavaScript bundle that the user's browser has to download and parse.
  Using React.lazy function:
  const ExampleComponent = React.lazy(() => **import**('./ExampleComponent'));

<suspense>

- <suspense> component is responsible for handling the output when the lazy component is fetched and rendered

Program
About.js
```
import React from 'react'

function about() {
       return (
        <div>
          <h2>About page </h2>
```

```
            </div>
        )
    }
    export default about
```

home.js

```
import React from 'react'

function Home() {
      return (
        <div>
          <h1>Home page</h1>
        </div>
      );
    }
    export default Home;
```

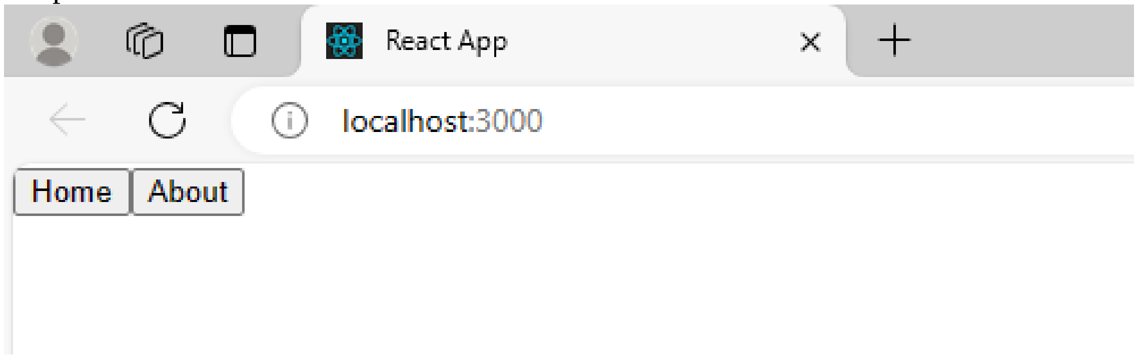
check.js file:

```
import React, {useState, lazy, Suspense} from 'react';
const About=lazy(() => import('./about'));
const Home=lazy(() => import('./home'));
function Selection() {
    const [selectedTab,setSelectedTab] = useState('');
    return(
        <div>
            <div>

            <button onClick={ () => setSelectedTab ('homeTab')}> Home</button>
            <button onClick={ () => setSelectedTab ('aboutTab')}> About</button>
            </div>
            <Suspense fallback={<div> <h1>Loading.... </h1> </div>}>
                {selectedTab === 'homeTab' && <Home/>}
                {selectedTab === 'aboutTab' && <About/>}
            </Suspense>
        </div>
    )
}    export default Selection;
```

App.js

```
import React, {Component} from 'react';
import Selection from './check.js';

class App extends Component {
    render (){
        return (
            <div>
            < Selection />

            </div>
        );
    }
}

export default App;
```

Output



- Initially only five files will render in before clicking the button.
- When the user clicks on Home Button, home.js file is added to the existing list
- when the user clicks on the about button, about.js file is rendered

Unit – 3

Answer

21. What is Angular? Explain the features of Angular.

Ans

- Angular is a JavaScript **(actually a TypeScript based open-source full-stack web application)** framework which makes you able to create reactive **Single Page Applications** (SPAs).
- Angular is completely based on components. It consists of several components which forms a tree structure with parent and child components.
- Angular's versions beyond 2+ are generally known as **Angular** only. The very first version Angular 1.0 is known as **AngularJS.**

Angular Features

1. **Two-Way Data Binding:** AngularJS introduced two-way data binding, enabling automatic synchronisation between the model and the view, creating a seamless and responsive user interface with minimal code.
2. **MVC Architecture:** AngularJS follows the Model-View-Controller (MVC) architectural pattern, providing a structured approach to application design and development, enhancing code organisation and maintainability.
3. **Dependency Injection:** AngularJS facilitates dependency injection, allowing components to be injected with their dependencies, which promotes reusability, testability, and modular development.
4. **Directives for Extending HTML:** Directives in AngularJS extend the HTML vocabulary, enabling the creation of custom, reusable components and behaviours, resulting in a modular and extensible development approach.
5. **Modular Design:** AngularJS promotes modular design, allowing the development of independent and reusable components that can be combined to build complex applications, enhancing scalability and maintainability.
6. **Active Community Support:** The AngularJS community provides active support, fostering collaboration, knowledge sharing, and the development of a rich ecosystem of tools, resources, and best practices.

22. State the differences between Angular and React.

Ans

Sl.No	ANGULAR	REACT
1	Angular is a TypeScript based JavaScript framework. It is written in TypeScript.	React is not a framework. It is a JavaScript library for building user interfaces.
2	Angular is developed and maintained by Google. It was released in 2010 by google	React is developed and maintained by Facebook. It was released in 2013 by facebook.
3	Angular is a full MVC (Model, View, and Controller) framework.	React is a simple JavaScript library
4	Angular uses regular DOM. The regular DOM updates the entire tree structure of HTML tags.	React uses virtual DOM which makes it amazing fast.
5	Angular provides two-way data binding.	React provides one-way data binding.
6	Angular is fast as compared to old technologies	React is faster than Angular.
7	The size of Angular is large, so it takes longer load time and performance on mobile.	The size of React is smaller than Angular, so it is a little bit faster.
8	**Companies using Angular -** Google, Nike, Forbes	**Companies using React -** Facebook, Airbnb, Uber, Netflix, Instagram, WhatsApp

23. Explain how to generate a component in Angular with an example.

Ans

Components are the main building blocks for Angular applications. Each component consists of:

- An HTML template that declares what renders on the page
- A TypeScript class that defines behavior

- A CSS selector that defines how the component is used in a template
- Optionally, CSS styles applied to the template

Creating a component using the Angular CLI

- From a terminal window, navigate to the directory containing your application.
- Run the following command

ng generate component <component-name>

ng g c <component-name> → Shortcut Command

where <component-name> is the name of your new component.

By default, this command creates the following:

1. A **directory** named after the component
2. A component file, <component-name>.**component.ts**
3. A template file, <component-name>.**component.html**
4. A CSS file, <component-name>.**component.css**
5. A testing specification file, <component-name>.**component.spec.ts**

How to use Component

Navigate to the directory containing your app and type the following:

```
ng g c mycomp
```

Example:

mycomp.component.ts file:

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

@Component({
  selector: 'app-mycomp',
  standalone: true,
  imports: [],
  templateUrl: './mycomp.component.html',
  styleUrl: './mycomp.component.css'
})

export class MycompComponent {
}
```

<u>mycomp.component.html file:</u>

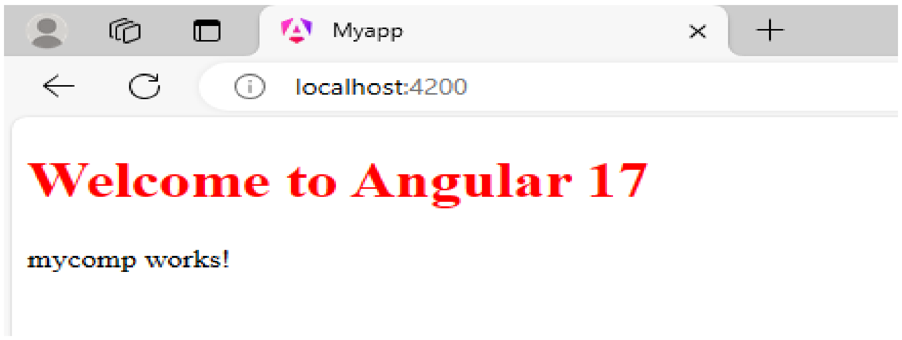
```
<p> my comp work! </p>
```

<u>app.component.ts file:</u>

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

import { RouterOutlet } from '@angular/router';

import {MycompComponent} from './mycomp/mycomp.component'


@Component({

  selector: 'app-root',

  standalone: true,

  imports: [RouterOutlet, MycompComponent],

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

  styles: ['h1 { color: red; }']

})

export class AppComponent {

  title = 'myapp';

}
```

<u>app.component.html file:</u>

```
<h1> Welcome to Angular 17 </h1>

<app-mycomp> </app-mycomp>
```



Welcome to Angular 17

mycomp works!

24. Write the classification of Angular directives. Explain ngIf directive with an example.

Ans

Directives are classes that add additional behavior to elements in your Angular applications.

Angular built-in directives are used to manage forms, lists, styles, and what users see.

Types of Directives:

1. Structural Directives - ngIf (ngIf, ngIf-else, ngIf-then-else), ngFor, ngSwitch
2. Attribute Directives- ngClass, ngStyle, ngModel
3. Component Directives

ngIf directive

`*ngIf`: Conditionally adds or removes an element from the DOM.

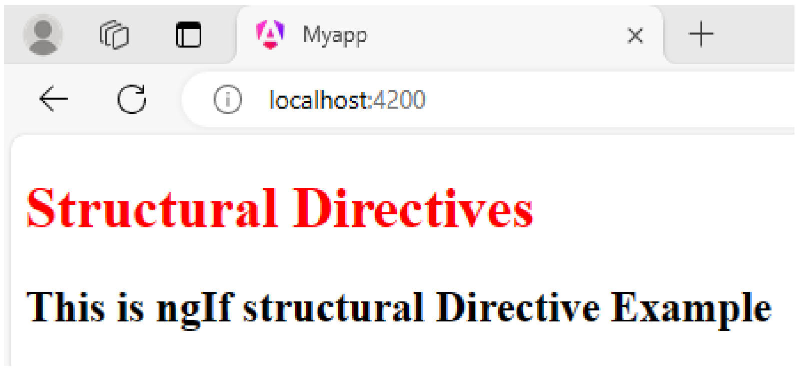
app.component.ts file:

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

import {CommonModule} from '@angular/common';


@Component({

  selector: 'app-root',

  standalone: true,

  imports: [CommonModule],

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

  styles: ['h1 { color: red; }']

})
```

```
export class AppComponent {
  data="abc"
}
```

app.component.html file:

```
<h1> Structural Directives </h1>

<h2 *ngIf="data">This is ngIf structural Directive Example</h2>
```



Structural Directives

This is ngIf structural Directive Example

25. Explain the following structural directives in Angular with an example.

i. ngFor

ii. ngSwitch

Ans

i. ngFor

- The ngForOf directive is generally used in the shorthand form *ngFor.
- In this form, the template to be rendered for each iteration is the content of an anchor element containing the directive.

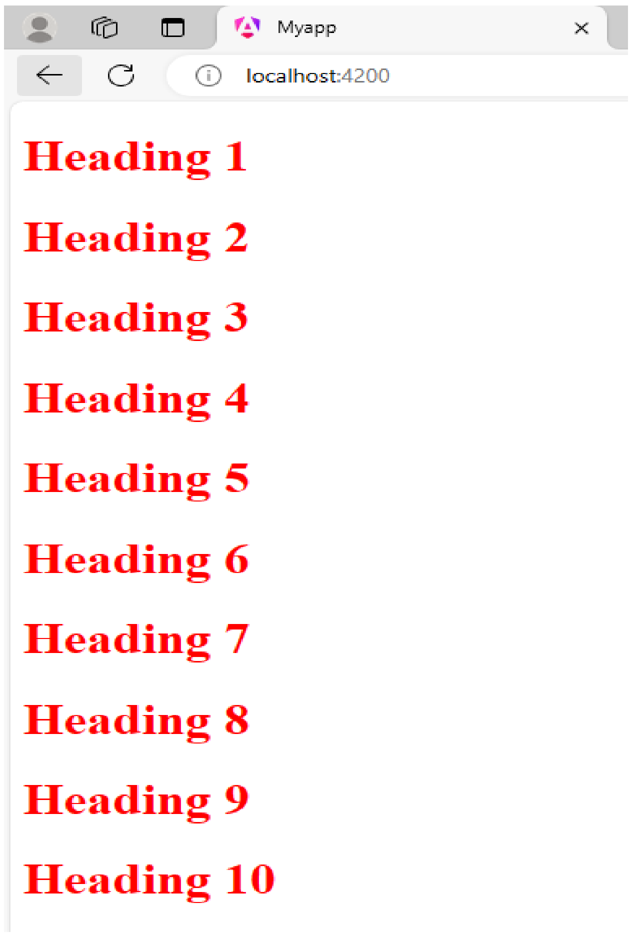
Example

app.component.ts file:

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

import {CommonModule} from '@angular/common';


@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule],

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

  styles: ['h1 { color: red; }']

})

export class AppComponent {


 nums=[1,2,3,4,5,6,7,8,9,10]

}
```

app.component.html file:

```
<h1 *ngFor ="let n of nums">Heading {{n}}</h1>
```



Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Heading 7

Heading 8

Heading 9

Heading 10

ii. ngSwitch

- The [ngSwitch] directive on a container specifies an expression to match against.

- The expressions to match are provided by ngSwitchCase directives on views within the container.
- Every view that matches is rendered.
- If there are no matches, a view with the ngSwitchDefault directive is rendered.

```
Switch (expression) {
    case value1:
            statement1;
            break;
    case value2:
            statement2;
            break;
    case value3:
            statement3;
            break;
    default:
            statement;
}
```

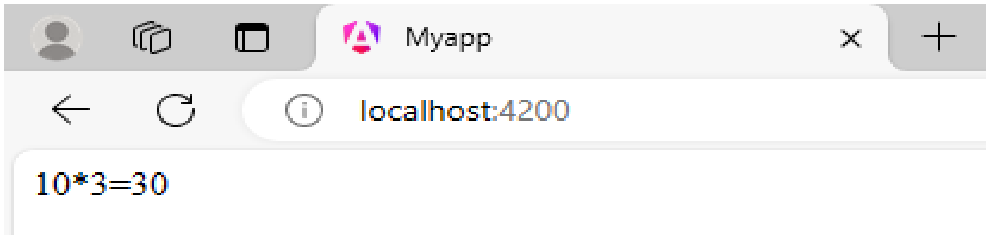
Example:

app.component.ts file:

```
import { Component } from '@angular/core';
import {CommonModule} from '@angular/common';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './app.component.html',
  styles: ['h1 { color: red; }']
})
export class AppComponent {
num1: number = 10;
num2: number = 3;
operation: string = '*';
}
```

app.component.html file:

```
<div [ngSwitch]="operation">
  <div *ngSwitchCase="'+'">{{num1}}+{{num2}}={{num1+num2}}</div>
  <div *ngSwitchCase="'-'">{{num1}}-{{num2}}={{num1-num2}}</div>
  <div *ngSwitchCase="'*'">{{num1}}*{{num2}}={{num1*num2}}</div>
  <div *ngSwitchCase="'/'">{{num1}}/{{num2}}={{num1/num2}}</div>
  <div *ngSwitchDefault="'+'">{{num1}}+{{num2}}={{num1+num2}}</div>
</div>
```



```
10*3=30
```

26. Write down the steps for Routing in Angular and write a program to create a simple webpage using Angular Router.

Ans

- Angular Router is used to handle the navigation from one view to the next.
- The Router enables navigation by interpreting a browser URL as an instruction to change the view.

Basic Steps in Routing:

1. Creating the components which are to be linked through Routing
2. Configuring the Routes
3. Adding Routes in template.

Adding Router Outlet and RouterLink

Example:

create a small app with **3 pages**:

- **Home**
- **About**
- **Contact**

And use the Angular **Router** to navigate between them.

src/app/app-routing.module.ts

```
import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { AboutComponent } from './about/about.component';

import { ContactComponent } from './contact/contact.component';



const routes: Routes = [

  { path: '', component: HomeComponent },

  { path: 'about', component: AboutComponent },

  { path: 'contact', component: ContactComponent },

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }
```

src/app/app-routing.module.html

```html
<nav>
  <a routerLink="">Home</a>
  <a routerLink="about">About</a>
  <a routerLink="contact">Contact</a>
</nav>
<hr>
<router-outlet></router-outlet>
```

home.component.html

```html
<h2>Welcome to the Home Page</h2>
```

about.component.html

```html
<h2>About Us</h2>
<p>This is a demo Angular app using routing.</p>
```

contact.component.html

```html
<h2>Contact Page</h2>
<p>Email us at contact@example.com</p>
```

27. Write down the installation steps of Angular.

Ans

Steps to Install Angular JS:

1. Install Node.js and npm

Angular requires **Node.js** and **npm** (Node Package Manager).

- Download and install the latest **LTS (Long-Term Support)** version of Node.js
- from: ☞ https://nodejs.org/
- Verify the installation by running the following commands in the terminal or command prompt:

```
node -v   # Check Node.js version
npm -v    # Check npm version
```

---

2. Install Angular CLI

Angular CLI (Command Line Interface) helps in creating and managing Angular projects.

- Install it globally using npm:

npm install -g @angular/cli

- Verify the installation:

ng version

---

3. Create a New Angular Project

- Run the following command and replace my-angular-app with your project name:

ng new my-angular-app

- Follow the prompts (e.g., selecting routing and CSS options).

---

4. Navigate to the Project Directory
cd my-angular-app

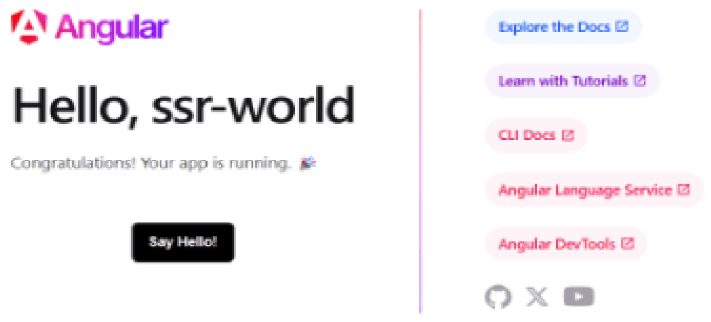
---

5. Serve the Application

- Start the development server:

ng serve or ng s

- Open a browser and go to:
  👉 http://localhost:4200


Angular

Hello, ssr-world

Congratulations! Your app is running. 🎉

28. Write a program in Angular using ngStyle directive.

Ans

- ➢ **Attribute directives changes the appearance or behavior of a DOM element.**
- ➢ **These directives look like regular HTML attributes in templates**
- ➢ Based on the component state, dynamic styles can be applied by using ngStyle
- ➢ Many inline styles can be set by binding to ngStyle.

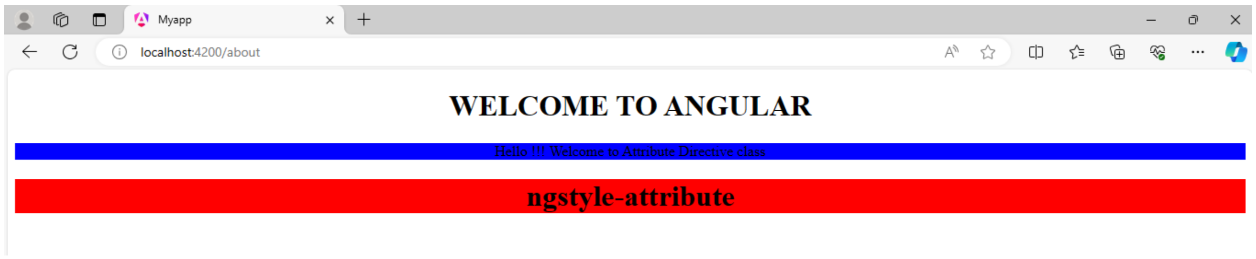
Edit src/app/app.component.ts:

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './app.component.html',
})
```

```
export class AppComponent {
  myvar='AIML'
}
```
Edit src/app/app.component.html:

```
<div style = "text-align: center;">
<h1> WELCOME TO ANGULAR</h1>
<div style ="background-color: blue;"> Hello !!! Welcome to Attribute Directive class</div>
<h1 [ngStyle]="{'background-color': myvar=='CSE' ? 'green':'red'}">ngstyle-attribute</h1>
```

O/P:



WELCOME TO ANGULAR

Hello !!! Welcome to Attribute Directive class

ngstyle-attribute

29. What are attribute directives in Angular? Write a program in Angular using ngClass directive.

Ans

Attribute directives in Angular are used to **change the appearance, behavior, or layout** of an element — **without changing the DOM structure** (i.e., they don't add or remove elements like structural directives do).

They're called *attribute* directives because you use them as if you're applying an attribute to an HTML element.

Directive	**Description**
`ngClass`	Adds/removes CSS classes dynamically
`ngStyle`	Adds/removes styles dynamically
`ngModel`	Two-way data binding (used in forms)

Example:

Edit src/app/app.component.ts:

```
import { Component } from '@angular/core';
import {CommonModule} from '@angular/common';
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './app.component.html',
  styles: './app.component.css'
})
export class AppComponent {
}
```

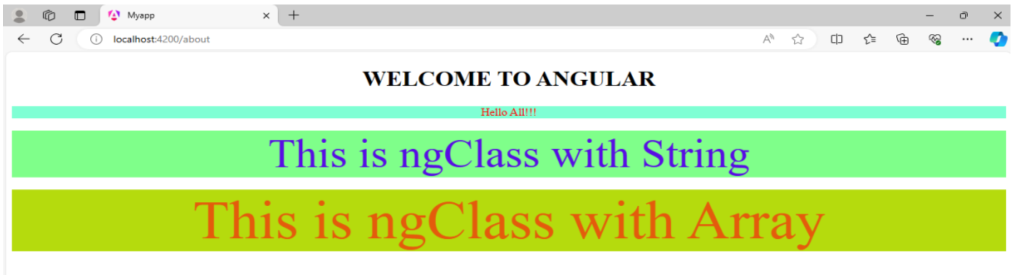
Edit src/app/app.component.html:

```
<div style = "text-align: center;">
 <h1> WELCOME TO ANGULAR</h1>
 <! ----- ngClass with String >
 <div class="one">Hello All!!!</div> <br>
 <div [ngClass]="'one three'">This is ngClass with String</div> <br>
 <! ----- ngClass with Array >
 <div [ngClass]="['one', 'four']">This is ngClass with Array</div>
```

Edit src/app/app.component.css:

```
.one{

  background-color: aquamarine;

  color: red;

  }

  .two{

  background-color: #ff7fb4;

  }

  .three{

  background-color: rgb(127, 255, 138);

  color: rgb(88, 12, 228);

  font-size: 60px;

  }

  .four{

  background-color: rgb(181, 219, 13);

  color: rgb(228, 91, 12);

  font-size: 80px;

  }
```

O/P:


WELCOME TO ANGULAR

Hello All!!!

This is ngClass with String

This is ngClass with Array

30. Write a program in Angular to display the message "Welcome to Angular" and style the text by applying any three different CSS styles.

Ans

app.component.ts

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


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Welcome to Angular';
}
```

app.component.html

```
<div class="welcome-message">
  {{ title }}
</div>
```

app.component.css

```
.welcome-message {
  color: white;                /* Text color */
  font-size: 24px;             /* Font size */
  background-color: #007acc;  /* Background color */
  padding: 20px;
  border-radius: 10px;
  text-align: center;
  font-family: Arial, sans-serif;
}
```

Output:

Welcome to Angular

(Blue background, white text, big font, centered)

Unit- 4

Answer

31. Write the differences between template driven forms and Reactive forms in Angular.

Ans

Feature	Template-Driven Forms	Reactive Forms
Form Creation	Defined mostly in the template (HTML)	Defined mostly in the component class (TS)
FormControl & FormGroup	Angular creates them automatically	You explicitly define them using code
Simplicity	Easy and simple to use for basic forms	More powerful and scalable
Control Access	Accessed via template reference variables	Accessed via component code
Validation	Defined in the HTML using directives	Defined in the component using validators
Testing	Harder to test due to reliance on DOM	Easier to test since logic is in TS
Data Flow	Two-way binding via `[(ngModel)]`	Unidirectional, reactive programming
Form Updates	Angular updates the model automatically	You have explicit control over the model
Module Required	`FormsModule`	`ReactiveFormsModule`

32. Write a program to create a Template driven form in Angular.

Ans

App.component.ts:

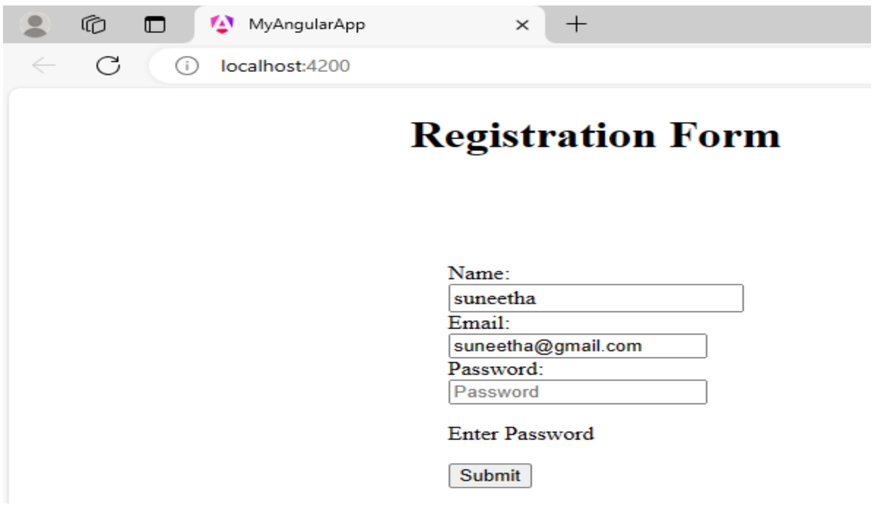
```
import { Component } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';
@Component({

  selector: 'app-root',

  standalone: true,
  imports: [CommonModule,FormsModule],
  templateUrl: './app.component.html',
  styleUrl:'./app.component.css',
})
export class AppComponent {
  user:any={
    name:'',
    email:'',
    password:'',
  }
  onSubmit(){
    console.log(this.user);
  }  }
```

App.component.html:

```
<h1 style="text-align: center;"> Registration Form </h1>
<div
style="display: flex;
align-items: center;
justify-content: center;
height: 50vh;" >

<form action="" (submit)="onSubmit()">
<div>
Name:<br><input type="text" placeholder="Name" name="name" [(ngModel)]="user.name"/>
    @if (!user.name) {
```

```
<p>Name is Required</p>

    }

</div>

<div>

    Email: <br><input type="email" placeholder="Email" name="email"
[(ngModel)]="user.email"/>

    @if (!user.email) {

        <p>Email is Required</p>

    }

</div>
<div>

    Password: <br><input type="password" placeholder="Password" name="password"
[(ngModel)]="user.password"/>

    @if (!user.password) {

        <p>Enter Password</p>

    }

</div>


<button type="submit">Submit</button>

</form>
</div>
```

O/P:



Registration Form

Name:
suneetha
Email:
suneetha@gmail.com
Password:
Password

Enter Password

Submit

33. Write a program to create a Reactive form in Angular.

Ans

app.component.ts file:

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

import { CommonModule } from '@angular/common';

import { FormControl, FormGroup,  ReactiveFormsModule, Validators } from
'@angular/forms';


@Component({

  selector: 'app-root',

  standalone: true,

  imports: [CommonModule,ReactiveFormsModule],

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

  styleUrl:'./app.component.css',

})

export class AppComponent {

  userForm: FormGroup = new FormGroup({

    firstName: new FormControl("", [Validators.required]),
    lastName: new FormControl("",[Validators.required]),
    email: new FormControl("", [Validators.required, Validators.email]),
  })
```

app.component.html:

```
<h1> Reactive Form </h1>
<form action="" [formGroup]="userForm">
  <div>
    <label>First Name</label> <br>
    <input type="text" formControlName="firstName">
    <span *ngIf="userForm.controls['firstName'].errors?. ['required']">This is required </span>
  </div>

  <div>
    <label>Last Name</label> <br>
    <input type="text" formControlName="lastName">
    <span *ngIf="userForm.controls['lastName'].errors?. ['required']">This is required  </span>
  </div>
```

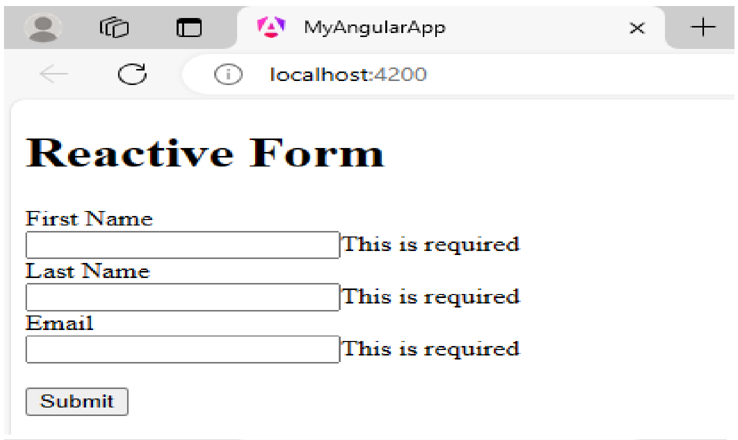
app.component.html file:

```
<div>
    <label>Email</label> <br>
    <input type="text" formControlName="email">
    <span *ngIf="userForm.controls['email'].errors?. ['required']">This is required </span>
    <span *ngIf="userForm.controls['email'].errors?. ['email']">Enter valid email</span>
  </div>
<br>
```

<button type="submit">Submit</button>

</form>

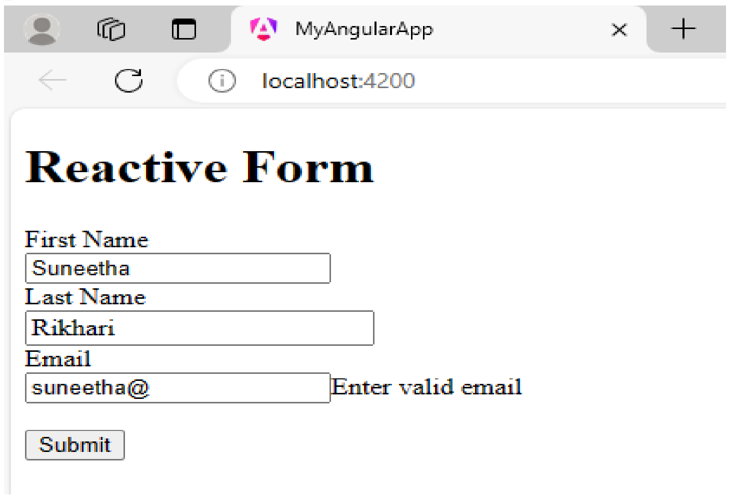
O/P:



Reactive Form

First Name
This is required
Last Name
This is required
Email
This is required

Submit



Reactive Form

First Name
Suneetha
Last Name
Rikhari
Email
suneetha@
Enter valid email

Submit

34. Define Angular Pipes. Explain built in Angular pipes with examples.

Ans

- Angular Pipes are a way to transform the format of output data for display.
- The data can be strings, currency amounts, dates, etc.
- Pipes are simple functions that *accept an input and return a transformed value* in a more technical understanding.
- They do not alter the data but change into the required format to display in the browser. Angular provides many built-in pipes for typical data transformation.
- You can also create custom pipes if you want to do custom transformation

Syntax:

{{title | uppercase}}

- Pipe takes integers, strings, arrays, and date as input separated with |.
- It transforms the data in the format as required and displays the same in the browser.

There are two types of pipes:

- Built in pipes
- Custom pipes

Built in pipes: Angular built-in pipes are pipes that are provided by the angular library.

Angular provides some built-in pipes:

Lowercase pipe

Uppercase pipe

Date pipe

Currency pipe

Json pipe

Percent pipe

Decimal pipe

Slice pipe

App.component.ts file

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

import { CommonModule } from '@angular/common';
@Component({

  selector: 'app-root',

    standalone: true,

    imports: [CommonModule],

    templateUrl: './app.component.html',
    styleUrl:'./app.component.css',
})

export class AppComponent {

  title = 'my-first-app';

  todaydate = new Date();

  val = {name: 'Alex', age: '25', address:{a1: 'Paris', a2: 'France'}};

  months = ['Jan', 'Feb', 'Mar', 'April', 'May', 'Jun',

    'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec'];

}
```

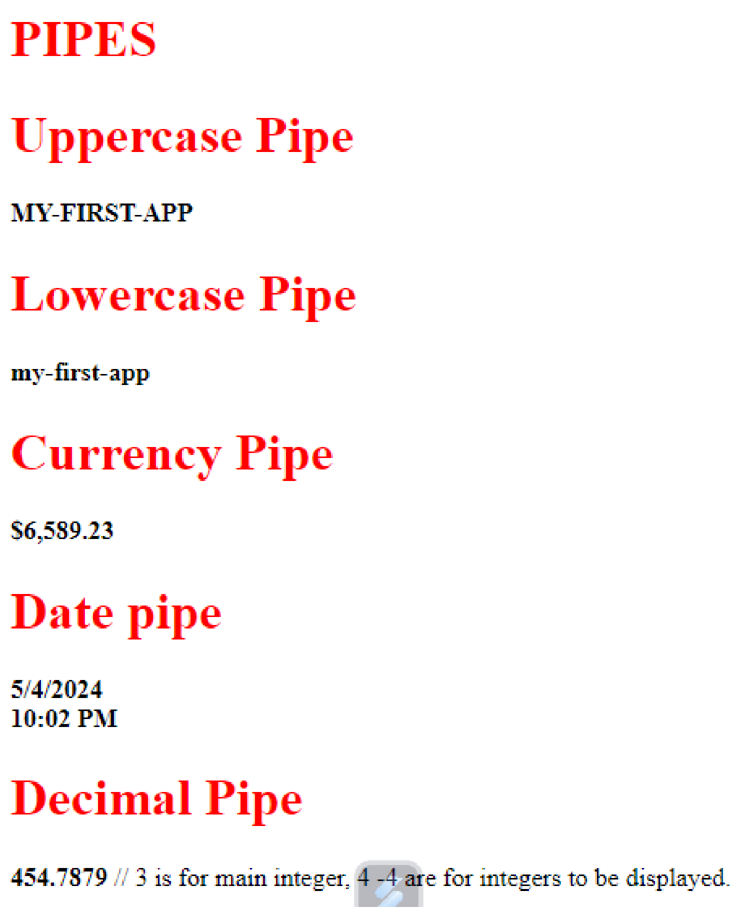
App.component.html file

```
<h1>PIPES</h1>

   <h1>Uppercase Pipe</h1>

   <b>{{title | uppercase}}</b><br/>

   <h1>Lowercase Pipe</h1>

   <b>{{title | lowercase}}</b>

   <h1>Currency Pipe</h1>

   <b>{{6589.23 | currency:""}}</b><br/>

   <h1>Date pipe</h1>

   <b>{{todaydate | date:'d/M/y'}}</b><br/>

   <b>{{todaydate | date:'shortTime'}}</b>

   <h1>Decimal Pipe</h1>

   <b>{{ 454.78787814 | number: '3.4-4' }}</b>

         // 3 is for main integer, 4 -4 are for integers to be displayed.

   <h1>Json Pipe</h1>
```

```
  <b>{{ val | json }}</b>

  <h1>Percent Pipe</h1>

  <b>{{00.54565 | percent}}</b>

  <h1>Slice Pipe</h1>

  <b>{{months | slice:2:6}}</b>

  // here 2 and 6 refers to the start and the end index
```

App.component.css file

```
h1{ color: red;}
```



PIPES

Uppercase Pipe

MY-FIRST-APP

Lowercase Pipe

my-first-app

Currency Pipe

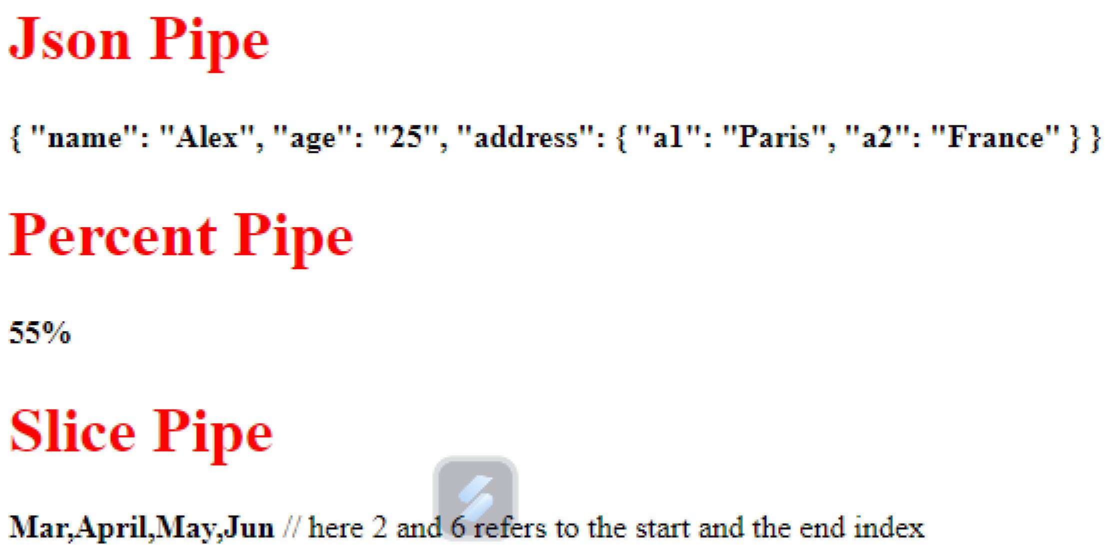
$6,589.23

Date pipe

5/4/2024
10:02 PM

Decimal Pipe

454.7879 // 3 is for main integer, 4 -4 are for integers to be displayed.



Json Pipe

{ "name": "Alex", "age": "25", "address": { "a1": "Paris", "a2": "France" } }

Percent Pipe

55%

Slice Pipe

Mar,April,May,Jun // here 2 and 6 refers to the start and the end index

35. Write a program using the following pipes in Angular: a) Lowercase pipe b) Uppercase pipe c) Date pipe d) Currency pipe

For answer refer the above answer (34th question answer).

36. Write a program to create a custom pipe in Angular.

Ans

- Create custom pipes to encapsulate transformations that are not provided with the built-in pipes.
- Then, use your custom pipe in template expressions, the same way you use built-in pipes—to transform input values to output values for display.
- Generate pipe using command
    - ng g p percentage<name of the pipe>

<u>app.component.ts file:</u>

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

import { CommonModule } from '@angular/common';

import { PercentagePipe } from './percentage.pipe';


@Component({

  selector: 'app-root',
  standalone: true,
  imports: [CommonModule,PercentagePipe],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
value:number=156
total:number=600
}
```

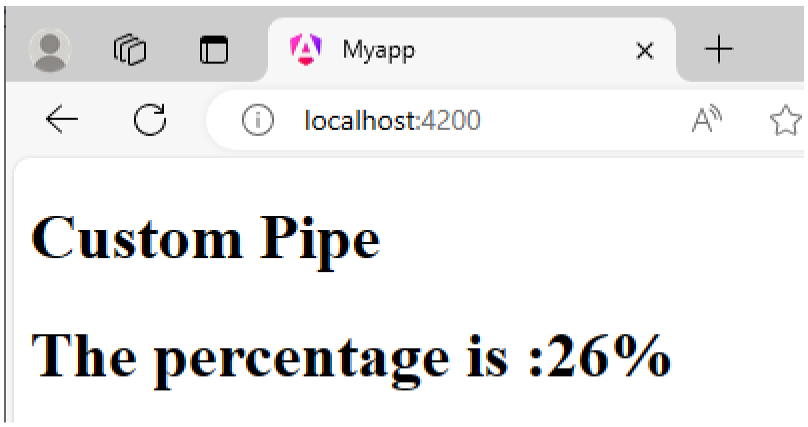
percentage.pipe.ts file:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'percentage',
  standalone: true
})
export class PercentagePipe implements PipeTransform {


    transform(value: number, total: number, ): any {


    return (value/total*100) + '%';
  }

}
```

app.component.html:

```
<h1>Custom Pipe</h1>
```

```
<h1>The percentage is :{{value| percentage:total}}</h1>
```



Custom Pipe

The percentage is :26%

37. Explain Angular services and Dependency injections.

Ans

- Angular Service is a mechanism used to share common functionality between the components.
- Services provides an abstraction layer that handles an application's business logic, which usually includes communicating with a backend and parsing/returning data or datasets.
- Services are used when you need to use the same code across multiple components.
- This is done by creating a single reusable data service and inject it into the component that needs it.
- Navigate to the app folder and type the following command to generate service:
- ng g s nameofService

Example:

message.service.ts file:

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class MessageService {
  mymessage="Hello !! Welcome to Angular Services Class"

    getmessage() {
    return this.mymessage;
  }
}
```

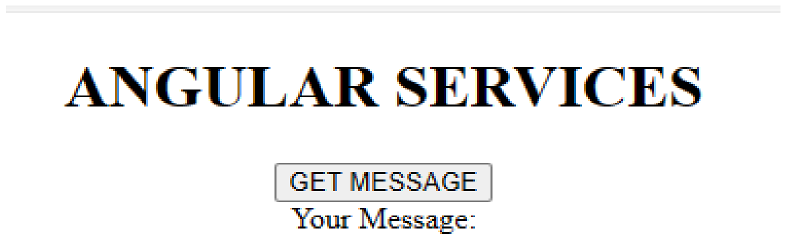
app.component.html :

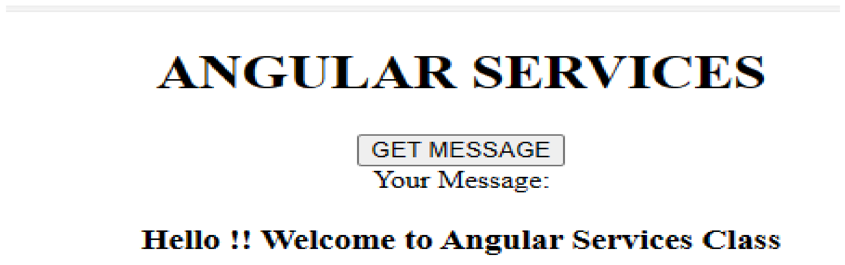
```
<div style="text-align: center;">
<h1>ANGULAR SERVICES</h1>
<button (click)="getmessage()" > GET MESSAGE</button> <br>
Your Message: <h3> {{msg}}</h3>
</div>
```

app.component.ts:

```
import { Component } from '@angular/core';
import { MessageService } from './message.service';
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [],
  templateUrl: './app.component.html',
  styleUrl:'./app.component.css',
})
export class AppComponent {
 msg: string=""
 constructor( private messageService:MessageService) {
 }
 getmessage(){

this.msg=this.messageService.getmessage();


 }

}
```

O/P:





Hello !! Welcome to Angular Services Class

38. What are Angular HTTP services? Explain HTTP services.

Ans

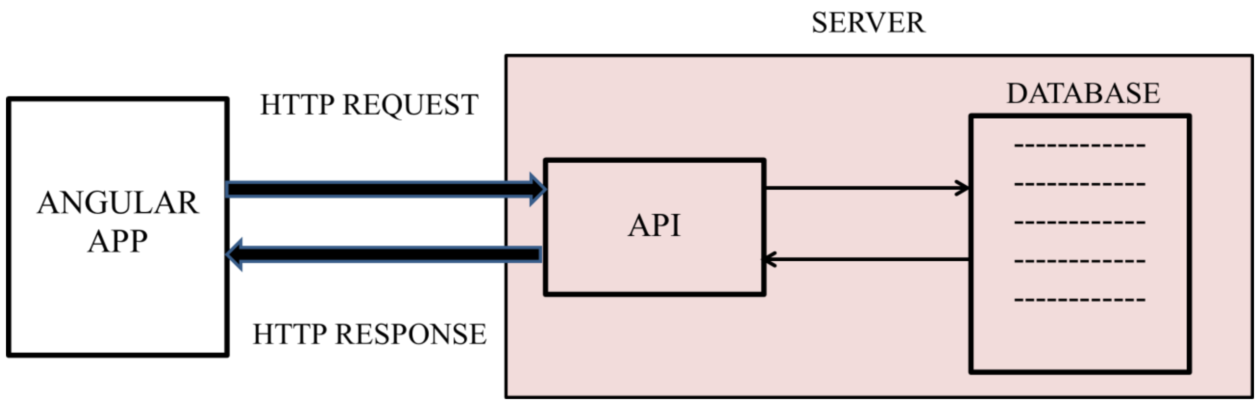
- Most front-end applications need to communicate with a server over the HTTP protocol, to download or upload data and access other back-end services.
- Angular provides a client HTTP API for Angular applications (the HttpClient service class in @angular/common/http.)

HTTP client service features:

The HTTP client service offers the following major features:

1. The ability to request typed response values
2. Streamlined error handling

3. Request and response interception
4. Robust testing utilities



- Angular cannot be connected directly to the database , because it is highly insecure to put the data in the User Interface code.
- Hence, angular communicates with the database through API (Application Programming Interface)
- Angular sends the HTTP request to the API and as a response API sends the JSON data. (not HTML data).The JSON data is converted into JavaScript objects and is then used in the frontend application.
- Angular is used for creating frontend application, it is not used for creating API's. API's are created using server side programming languages like python, node, PHP etc.
- API contains the code to interact with the database like fetching, updating, deleting the data etc.

39. Explain the core parts of Angular HTTP request services.

Ans

Following are the HTTP request core parts:

1. URL
2. HTTP Verbs
3. Headers
4. Body

5. URL:

1. URL

The most important part of an HTTP request is the URL to which we are sending the request. If we are making an HTTP requestto an API, in that case this URL is also called as API end point.

For example: *yourdomain.com/api/products/id*

2. HTTP Verbs:

- When connecting with the API's, it is not just the URL which matters, but also the HTTP verbs we are using.
- HTTP verbs define, which type of request you want to send to the API endpoint.
- Some of the HTTP verbs available to perform specific tasks are listed below:

GET – used to read data from server

POST- used to create new data on the server.

PUT- used to update existing data on the server.

DELETE- used to delete data on the server.

PATCH- used to update partial data on the server.

3. HTTP Request Headers:

- When making HTTP request to the server, along with API end point and HTTP verbs, we also need to send some additional metadata to the server with the request we are sending.
- HTTP request Headers are optional and usually clients will set some default headers fro an HTTP request.
- However, it is also possible to set HTTP request headers of our own when sending request to the server

4. Request Body:

- For some of the HTTP verbs, we also need to specify the body i.e, the data which we want to send with the request.
- Not all the requests need to have a body. HTTP requests like POST, PUT, PATCH etc, need to have a body. And requests like GET need not to have a body.

40. Write a program using the following pipes in Angular: a) Percent pipe b) Decimal pipe c) Slice pipe

Ans

For this answer refer 34th question answer

Unit – 5

Answer

41. What is Tkinter? Write the steps to create a simple window using Tkinter.

Ans:

- **Python Tkinter** is a standard **GUI (Graphical User Interface)** library for Python which provides a fast and easy way to create desktop applications.
- Tkinter provides a variety of **widgets like buttons, labels, text boxes, menus** and more that can be used to create interactive user interfaces.
- Tkinter supports event-driven programming, where actions are taken in response to user events like **clicks or keypresses**.

Steps to create window:

1. Import the Tkinter module:

This is created using the command ***import tkinter***

2. Create the GUI application main window:

Tk() class is used to create the main window in Tkinter.

3. Add Widgets

4. Enter the main event loop:

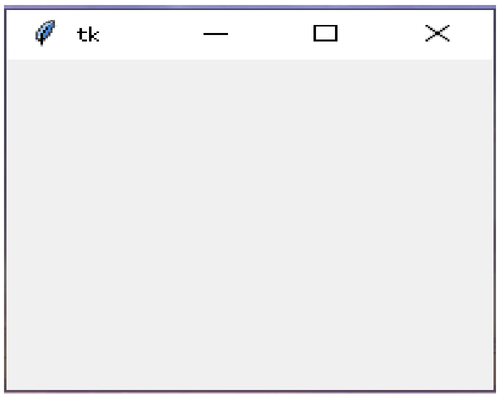
The ***mainloop()*** method is used to run application once it is ready. It is an infinite loop that keeps the application running, waits for events to occur (such as button clicks) and processes these events as long as the window is not closed.

Example:

Python Program:

```
from tkinter import *

root=Tk()

root.mainloop()
```

O/P:



42. Define Widget in Tkinter. Write a python program to create the following

a. Label Widget

b. Button Widget

Ans

- A widget is an element of a Graphical User Interface (GUI) that displays information or provides a specific way for a user to interact with the operating system or an application.
- Each widget in Tkinter is an instance of a specific class defined in the Tkinter Module.
- These classes provide methods and attributes that allow you to configure the widget's appearance, behavior, and functionality.
- Label: Syntax: *w=Label(master, option=value*)
- Button: Syntax: w=Button(master, option=value)

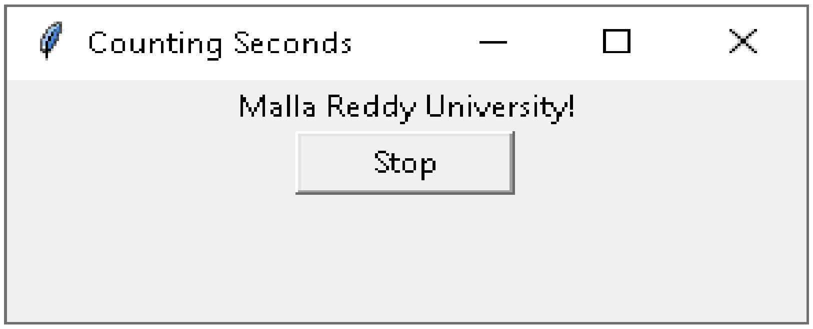
Program:

```
import tkinter as tk

r = tk.Tk()
r.title('Counting Seconds')
# Label
label = tk.Label(r, text="Malla Reddy University!")
label.pack()
# button
button = tk.Button(r, text='Stop', width=10)
```

button.pack()

r.mainloop()

O/P:



Malla Reddy University!

43. Explain the importance of Entry widget and CheckButton widget with an example.

Ans

Entry: Syntax: w=Entry(master, option=value)

CheckButton: Syntax: w = CheckButton(master, option=value)

Program:

from tkinter import *
master = Tk()

master.title("Entry and Checkbox Example")

Label(master, text='Entry Widget & Checkbox Widget').grid(row=0, columnspan=2)

#Entry

Label(master, text='First Name').grid(row=1)

Label(master, text='Last Name').grid(row=2)

e1 = Entry(master)

e2 = Entry(master)

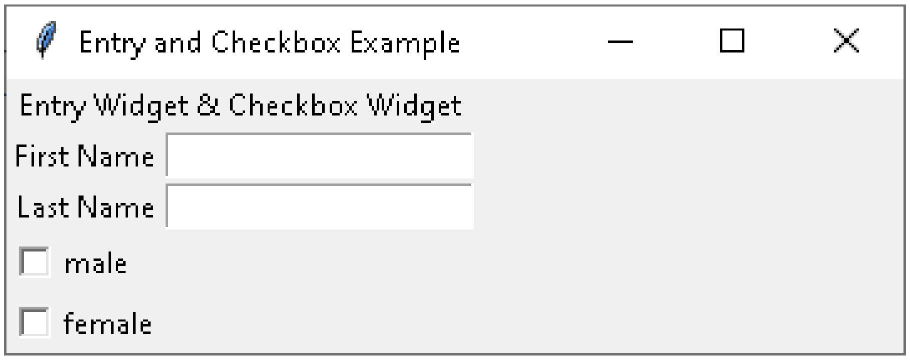
e1.grid(row=1, column=1)

e2.grid(row=2, column=1)

# checkbox

var1 = IntVar()

```
Checkbutton(master, text='male', variable=var1).grid(row=3,sticky=W)

var2 = IntVar()

Checkbutton(master, text='female', variable=var2).grid(row=4,sticky=W)


mainloop()
```



44. Define Geometry Manager classes and list the types of Geometry Manager classes. Explain any one type with an example.

Ans:

Tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows.

There are mainly three geometry manager classes:

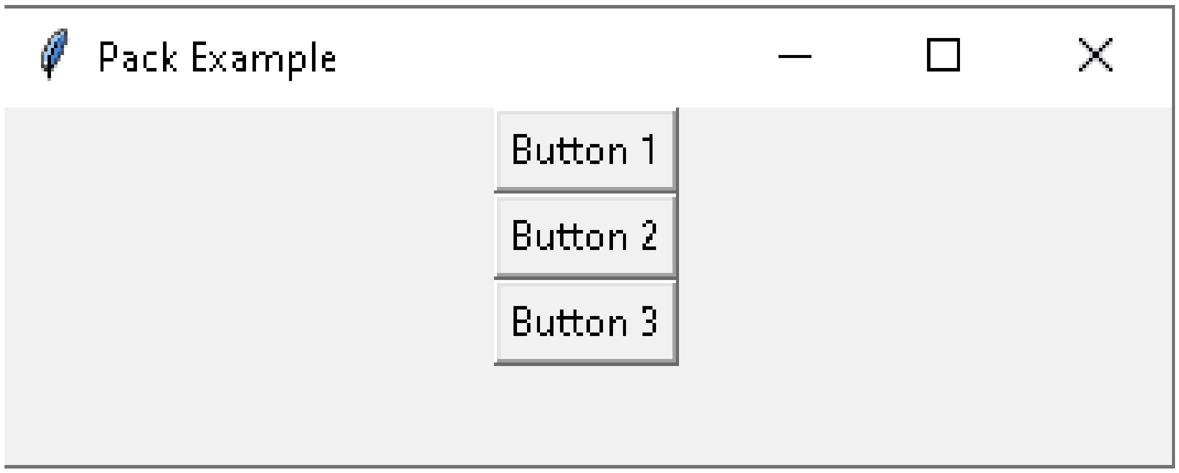
1. pack()
2. grid()
3. place()

1. pack() method:

   It organizes the widgets in blocks before placing in the parent widget. Widgets can be packed from the top, bottom, left or right. It can expand widgets to fill the available space or place them in a fixed size.

```
import tkinter as tk
root = tk.Tk()

root.title("Pack Example")
```

```
# Create three buttons

button1 = tk.Button(root, text="Button 1")

button2 = tk.Button(root, text="Button 2")

button3 = tk.Button(root, text="Button 3")


# Pack the buttons vertically

button1.pack()

button2.pack()

button3.pack()
root.mainloop()
```



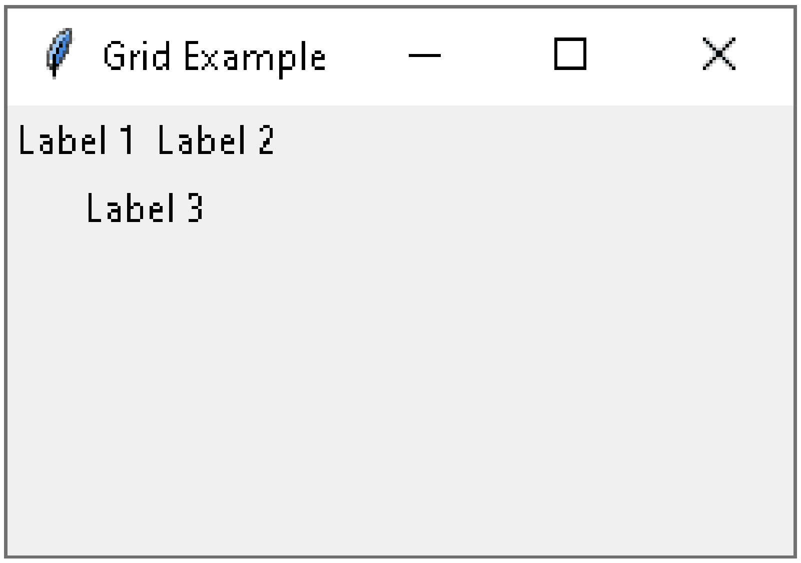
45. Explain grid() Geometry class with an example.

grid() method:

It organizes the widgets in grid (table-like structure) before placing in the parent widget. Each widget is assigned a row and column. Widgets can span multiple rows or columns using rowspan and columnspan.

```
import tkinter as tk
root = tk.Tk()

root.title("Grid Example")
```

```
# Create three labels

label1 = tk.Label(root, text="Label 1")

label2 = tk.Label(root, text="Label 2")

label3 = tk.Label(root, text="Label 3")


# Grid the labels in a 2x2 grid

label1.grid(row=0, column=0)

label2.grid(row=0, column=1)

label3.grid(row=1, column=0, columnspan=2)
root.mainloop()
```



```
Label 1 Label 2
    Label 3
```

46. Explain how events are handled in Tkinter with an example program.

Ans:

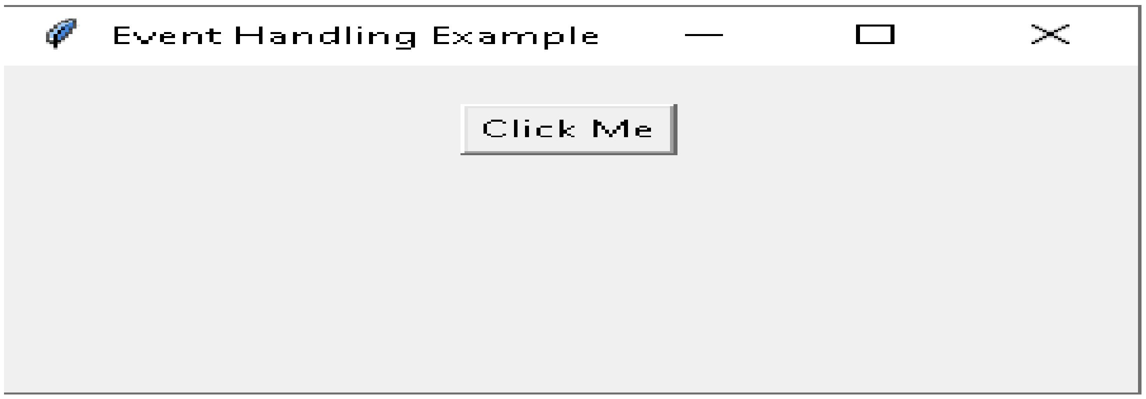
In Tkinter, events are actions that occur when a user interacts with the GUI, such as pressing a key, clicking a mouse button or resizing a window. Event handling allows us to define how our application should respond to these interactions.

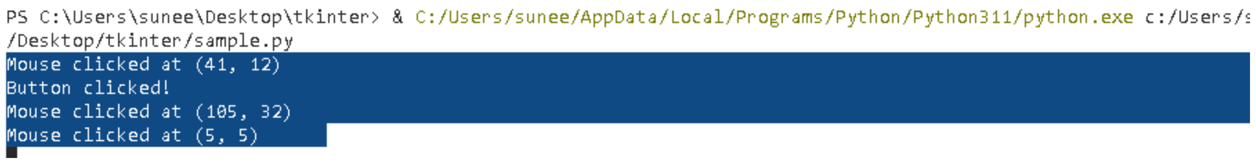
```
from tkinter import *
# Event handling functions

def on_button_click():

    print("Button clicked!")
```

```
def on_key_press(event):
    print(f"Key Pressed: {event.char}")

def on_mouse_click(event):
    print(f"Mouse clicked at ({event.x}, {event.y})")
```

# Main window

```
root = Tk()
root.title("Event Handling Example")
root.geometry("400x300")
# Button with click event
button = Button(root, text="Click Me", command=on_button_click)
button.pack(pady=20)

# Bind events
root.bind('<Key>', on_key_press)          # Capture any key press
root.bind('<Button-1>', on_mouse_click)    # Left mouse click

# Run the application
root.mainloop()
```





```
PS C:\Users\sunee\Desktop\tkinter> & C:/Users/sunee/AppData/Local/Programs/Python/Python311/python.exe c:/Users/s
/Desktop/tkinter/sample.py
Mouse clicked at (41, 12)
Button clicked!
Mouse clicked at (105, 32)
Mouse clicked at (5, 5)
```

47. Write a python program to create the following widgets in Tkinter:

a. RadioButton Widget

b. List Widget

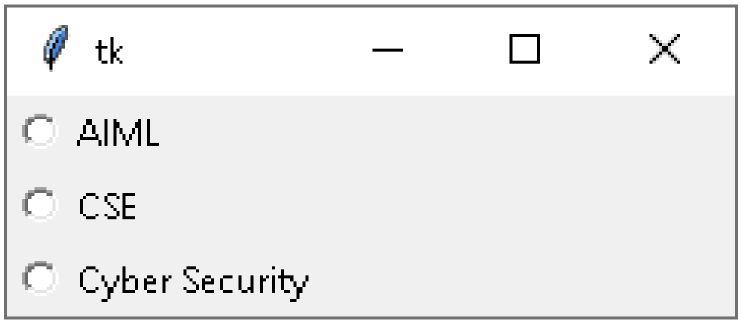
Ans

a. RadioButton Widget

<u>RadioButton</u>: Syntax: ***w = RadioButton(master, option=value)***

<u>Program:</u>

```
from tkinter import *

root = Tk()
v = IntVar()
Radiobutton(root, text='AIML', variable=v, value=1).pack(anchor=W)
Radiobutton(root, text='CSE', variable=v, value=2).pack(anchor=W)
Radiobutton(root, text='Cyber Security', variable=v, value=3).pack(anchor=W)
mainloop()
```

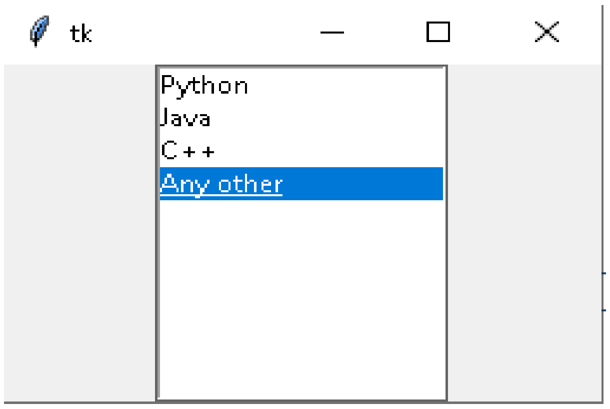


b. List Widget

<u>Listbox</u>: Syntax: ***w = Listbox(master, option=value)***

<u>Program:</u>

```
from tkinter import *

top = Tk()
Lb = Listbox(top)
Lb.insert(1, 'Python')
Lb.insert(2, 'Java')
Lb.insert(3, 'C++')
Lb.insert(4, 'Any other')
Lb.pack()
top.mainloop()
```



48. Write a program to create a scroll bar widget in Tkinter.

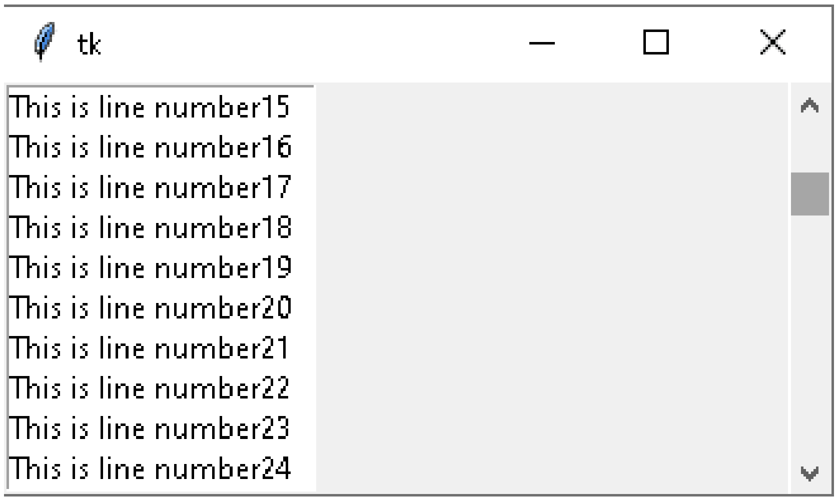
Ans

<u>Scrollbar</u>: Syntax: ***w = Scrollbar(master, option=value)***

<u>Program:</u>

```
from tkinter import *

root = Tk()
scrollbar = Scrollbar(root)
scrollbar.pack(side=RIGHT, fill=Y)
mylist = Listbox(root, yscrollcommand=scrollbar.set)
```

```
for line in range(100):
    mylist.insert(END, 'This is line number' + str(line))
mylist.pack(side=LEFT, fill=BOTH)
scrollbar.config(command=mylist.yview)
mainloop()
```



```
This is line number15
This is line number16
This is line number17
This is line number18
This is line number19
This is line number20
This is line number21
This is line number22
This is line number23
This is line number24
```

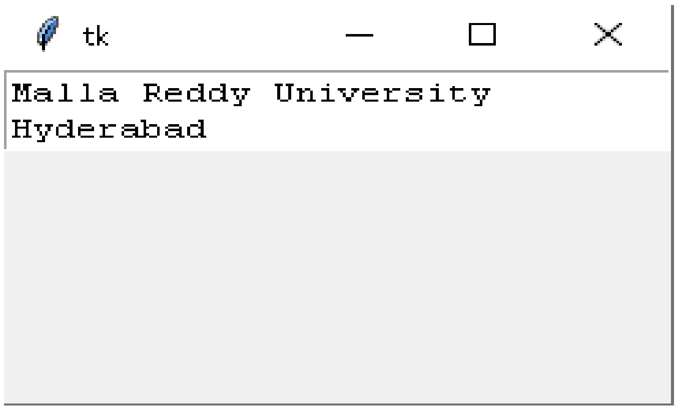
49. Write a program to design a Text Widget and Message Widget using Tkinter.

Ans

Text: Syntax: ***w =Text(master, option=value)***

Program:

```
from tkinter import *

root = Tk()
T = Text(root, height=2, width=30)
T.pack()
T.insert(END, 'Malla Reddy University\nHyderabad\n')
mainloop()
```

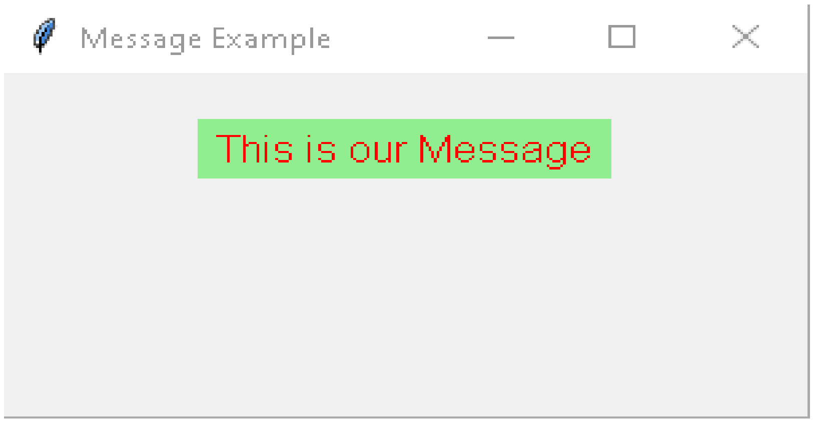


```
Malla Reddy University
Hyderabad
```

<u>Message</u>: Syntax: ***Message(master, option=value)***

<u>Program:</u>

```
from tkinter import *

main = Tk()
main.title("Message Example")
main.geometry("300x150")  # Set window size

ourMessage = 'This is our Message'
messageVar = Message(main, text=ourMessage, width=250)
messageVar.config(bg='lightgreen', fg='red',font=('Arial', 12))
messageVar.pack(padx=20, pady=20)

main.mainloop()
```

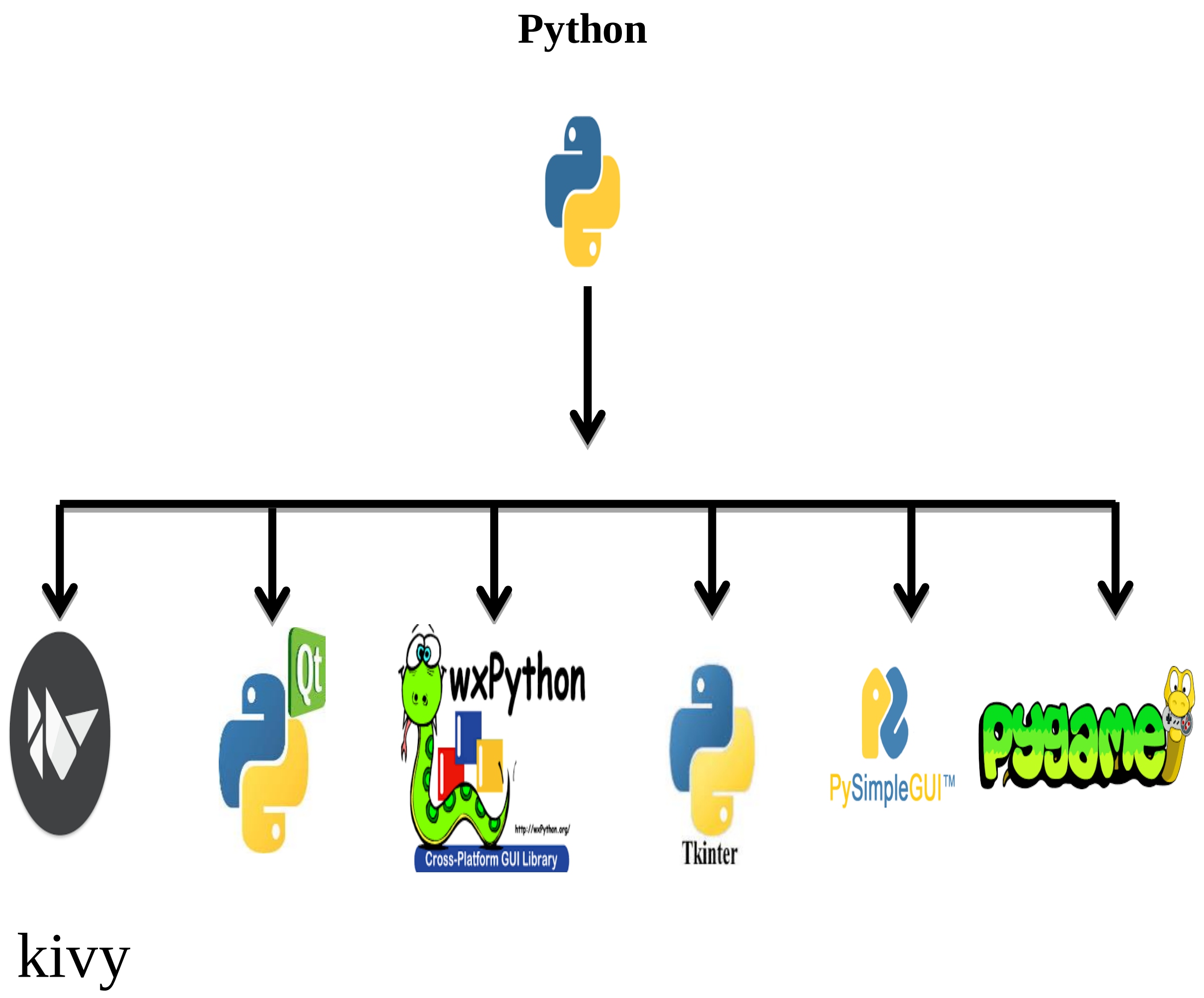


This is our Message

50. What is a GUI? List the Python libraries available for designing a GUI. Write a program to display "Hello world!!" message in Tkinter.

Ans:

- A graphical user interface (GUI) is a digital interface in which a user interacts with graphical components such as icons, buttons, and menus.
- In a GUI, the visuals displayed in the user interface convey information relevant to the user, as well as actions that they can take.
- GUIs are used on smartphones, desktops, ATMs, and self-checkout systems. They use visual elements to replace complex text-based commands.



<u>**Steps to create window:**</u>

1. Import the Tkinter module:

This is created using the command ***import tkinter***

2. Create the GUI application main window:

Tk() class is used to create the main window in Tkinter.

3. Add Widgets

4. Enter the main event loop:

The ***mainloop()*** method is used to run application once it is ready. It is an infinite loop that keeps the application running, waits for events to occur (such as button clicks) and processes these events as long as the window is not closed.

Python Program:

```
from tkinter import *

root=Tk()

---- widgets are added here----

root.mainloop()
```

OUTPUT