1.0) AIM: Understanding and implementing Signals in Angular to manage reactive state and automatically update the view when data changes.

2.0) SCOPE: To demonstrate the concept of Signals in Angular by creating a simple web application that stores and updates reactive data. Signals in Angular allow you to hold a value and automatically update the UI whenever the value changes, simplifying reactivity.

3.0) FACILITIES/ APPARATUS:

i)Hardware: I3 Processor, 8GB RAM, HD Monitor, Windows 10, Internet connectivity

ii) Software: Updated Web browser, Node js, Angular CLI, VS Code.

4.0) THEORY:

In Angular, signals are a new way to manage and track reactive data. A signal holds a value, and whenever that value changes, Angular automatically updates the parts of the UI that depend on it — without needing @Input(), @Output(), or RxJS for simple cases.

Working with signals

1. **Create a signal** → Use signal(initialValue) to store data.
2. **Read a signal** → Just call it like a function: mySignal().
3. **Update a signal** → Use .set(newValue) or .update(...).
4. **Auto reactivity** → The UI refreshes whenever the signal changes.

Example:

```
counter = signal(0);
counter.update(v => v + 1);
```

5.0) Procedure

1. Set up the Angular Project:

Create a new Angular component using Angular CLI:

ng new signal-demo

Navigate to the project directory:

cd signal-demo

2. Create a Component for the Experiment:

Generate a new component using Angular CLI:

ng generate component signal

This command will create a new component with the necessary files:

signal.ts, signal.html, and signal.css.

3. Implement signal in the Component TypeScript file:

Open the signal.ts file in your text editor.

Define the class:

4. Bind the signal to the template using interpolation and event binding:

Open the signal-demo.html file. use signal to display the title property in a heading and bind it to an input element and implement signal.

```
<div style="text-align: center; margin-top: 50px;">
  <h1>Angular Signals Example</h1>

  <!-- Reading a signal by calling it like a function -->
  <h2>Counter Value: {{ counter() }}</h2>

  <!-- Buttons to update signal -->
  <button (click)="increment()">Increment</button>
  <button (click)="reset()">Reset</button>
</div>
<div style="text-align:center; margin-top:50px;">
  <h1>Working with Signals in Angular</h1>

  <!-- Display the signal value -->
  <h2>Welcome to {{ name() }}</h2>

  <!-- Input to change the signal -->
  <input
    type="text"
    [value]="name()"
    (input)="updateName($event.target.value)"
    placeholder="Enter new name"
    style="padding: 5px;"
  />

  <br><br>

  <!-- Buttons -->
  <button (click)="resetName()">Reset</button>
</div>
```

The [innerText] binds the title property to the h1 tag's text content, and [value] binds the same property to the input element's value.

5.Updations in .ts File of component

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

@Component({
  selector: 'app-signal',
  standalone: true, // if you're using standalone components
  templateUrl: './signal.html',
  styleUrls: ['./signal.css']
})
export class SignalComponent {
  // Create a signal with initial value
  counter = signal(0);
  name = signal('Angular Signals');

  // Increment method
  increment() {
    this.counter.update(value => value + 1);
  }

  // Reset method
  reset() {
    this.counter.set(0);
  }

  // Method to update signal value from input
  updateName(newName: string) {
    this.name.set(newName);
  }

  // Reset the name
  resetName() {
    this.name.set('Angular Signals');
  }
}
```

6. Update the App Component:

Open the app.component.html file.Replace its content with the following to include the newly created component:

<app-signal-demo></app-signal-demo>

7. Run the Angular Application

ng serve

Open a web browser and navigate to **http://localhost:4200** .You should see the heading and input field displaying the text signal.

6.0) Conclusion

The experiment successfully demonstrated the use of Signals in Angular. Signals simplify reactivity by automatically updating the UI whenever the stored value changes. This feature reduces boilerplate code and makes state management more intuitive.