Lab 1: Create a Simple Angular Component

Exercise:

Create a simple Angular component named `hello-world` that displays "Hello, World!".

Solution:

```typescript
// hello-world.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-hello-world',
  template: `<h1>Hello, World!</h1>`,
  styles: [`h1 { color: blue; }`]
})
export class HelloWorldComponent { }
```

Lab 2: Manually Create a Component

Exercise:

Manually create a component named `greeting` that takes an input property `name` and displays "Hello, {name}!".

Solution:

```typescript
// greeting.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-greeting',
  template: `<h1>Hello, {{name}}!</h1>`,
  styles: [`h1 { color: green; }`]
})
export class GreetingComponent {
  @Input() name: string;
}
```

Lab 3: Working with Component Templates

Exercise:

Create a component named `user-card` with a template that displays a user's name and email.

Solution:

```typescript
// user-card.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-user-card',
  template: `
    <div class="card">
      <h2>{{user.name}}</h2>
      <p>{{user.email}}</p>
    </div>
```

```
  `,
  styles: [`
    .card {
      border: 1px solid #ccc;
      padding: 16px;
      border-radius: 8px;
    }
  `]
})
export class UserCardComponent {
  @Input() user: { name: string, email: string };
}
```

Lab 4: Component Styles

Exercise:

Create a component named `color-box` that has a colored background. The color should be passed as an input property.

Solution:

```typescript
// color-box.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-color-box',
  template: `<div class="color-box" [style.backgroundColor]="color"></div>`,
  styles: [`
    .color-box {
      width: 100px;
      height: 100px;
    }
  `]
})
export class ColorBoxComponent {
```

```
  @Input() color: string;
}
```

Lab 5: Data Binding - String Interpolation

Exercise:

Create a component named `current-date` that displays the current date using string interpolation.

Solution:

```typescript
// current-date.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-current-date',
  template: `<p>Today's date is {{currentDate}}</p>`,
  styles: [`p { font-size: 18px; }`]
```

```
})
export class CurrentDateComponent {
  currentDate: string = new
Date().toLocaleDateString();
}
```

Lab 6: Property Binding

Exercise:

Create a component named `image-display` that takes an image URL as input and displays the image using property binding.

Solution:

```typescript
// image-display.component.ts
import { Component, Input } from '@angular/core';
```

```
@Component({
  selector: 'app-image-display',
  template: `<img [src]="imageUrl" alt="Image" />`,
  styles: [`img { max-width: 100%; height: auto; }`]
})
export class ImageDisplayComponent {
  @Input() imageUrl: string;
}
```

Lab 7: One-Way Binding

Exercise:

Create a component named `item-list` that takes an array of items and displays them in a list using one-way binding.

Solution:

```typescript
```

```
// item-list.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-item-list',
  template: `
    <ul>
      <li *ngFor="let item of items">{{item}}</li>
    </ul>
  `,
  styles: [`ul { list-style-type: none; padding: 0; } li { padding: 4px; }`]
})
export class ItemListComponent {
  @Input() items: string[];
}
```

Lab 8: Two-Way Binding

Exercise:

Create a component named `name-editor` that includes an input field for editing a name. Implement two-way data binding to update the name in real-time.

Solution:

```typescript
// name-editor.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-name-editor',
  template: `
    <input [(ngModel)]="name" placeholder="Enter your name" />
    <p>Your name is: {{name}}</p>
  `,
  styles: [`input { margin-bottom: 8px; } p { font-weight: bold; }`]
})
```

```
export class NameEditorComponent {
  name: string = '';
}
```

Lab 9: Event Binding

Exercise:

Create a component named `button-clicker` that includes a button. Display a message each time the button is clicked using event binding.

Solution:

```typescript
// button-clicker.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-button-clicker',
```

```
  template: `
    <button (click)="handleClick()">Click me!</button>
    <p>{{message}}</p>
  `,
  styles: [`button { margin-bottom: 8px; } p { font-style: italic; }`]
})
export class ButtonClickerComponent {
  message: string = '';

  handleClick() {
    this.message = 'Button clicked!';
  }
}
```

Lab 10: Passing Data between Components (Parent to Child)

Exercise:

Create a parent component `parent-comp` that passes data to a child component `child-comp`. The child component should display the data passed from the parent.

Solution:

```typescript
// parent-comp.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-parent-comp',
  template: `
    <app-child-comp [childData]="parentData"></app-child-comp>
  `,
  styles: [`:host { display: block; padding: 16px; }`]
})
export class ParentCompComponent {
  parentData: string = 'Data from parent';
```

```
}

// child-comp.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-child-comp',
  template: `<p>{{childData}}</p>`,
  styles: [`p { font-size: 18px; }`]
})
export class ChildCompComponent {
  @Input() childData: string;
}
```

Lab 11: Passing Data between Components (Child to Parent)

Exercise:

Create a child component `child-input` that emits an event when a button is clicked. The parent component `parent-listener` should listen for this event and update a message.

Solution:

```typescript
// child-input.component.ts
import { Component, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-child-input',
  template: `<button (click)="sendMessage()">Send Message</button>`,
  styles: [`button { padding: 8px 16px; }`]
})
export class ChildInputComponent {
  @Output() messageEvent = new EventEmitter<string>();

  sendMessage() {
```

```
    this.messageEvent.emit('Hello from Child');
  }
}

// parent-listener.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-parent-listener',
  template: `
    <app-child-input
(messageEvent)="receiveMessage($event)"></app-
child-input>
    <p>{{message}}</p>
  `,
  styles: [`p { font-weight: bold; }`]
})
export class ParentListenerComponent {
  message: string = '';

  receiveMessage(event: string) {
```

```
    this.message = event;
  }
}
```

Lab 12: Using ngFor Directive

Exercise:

Create a component named `product-list` that takes an array of products and displays them in a list using the `ngFor` directive.

Solution:

```typescript
// product-list.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-product-list',
```

```
  template: `
    <ul>
      <li *ngFor="let product of products">{{product.name}} - {{product.price | currency}}</li>
    </ul>
  `,
  styles: [`ul { list-style-type: none; padding: 0; } li { padding: 4px; }`]
})
export class ProductListComponent {
  @Input() products: { name: string, price: number }[];
}
```

Lab 13: Using ngIf Directive

Exercise:

Create a component named `user-status` that takes a boolean input `isLoggedIn` and displays a different message based on the user's login status using the `ngIf` directive.

Solution:

```typescript
// user-status.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-user-status',
  template: `
    <p *ngIf="isLoggedIn; else loggedOut">Welcome, User!</p>
    <ng-template #loggedOut><p>Please log in.</p></ng-template>

  `,
  styles: [`p { font-size: 18px; }`]
})
```

```
export class UserStatusComponent {
  @Input() isLoggedIn: boolean;
}
```

Lab 14: Using ngClass Directive

Exercise:

Create a component named `highlight-text` that takes a boolean input `isHighlighted` and applies a CSS class to highlight the text based on the value using the `ngClass` directive.

Solution:

```typescript
// highlight-text.component.ts
import { Component, Input } from '@angular/core';

@Component({
```

```
  selector: 'app-highlight-text',
  template: `<p [ngClass]="{'highlight': isHighlighted}">Highlight me!</p>`,
  styles: [`.highlight { background-color: yellow; }`]
})
export class HighlightTextComponent {
  @Input() isHighlighted: boolean;
}
```

Lab 15: Using ngStyle Directive

Exercise:

Create a component named `dynamic-styles` that takes a boolean input `isSpecial` and applies different styles based on the value using the `ngStyle` directive.

Solution:

```typescript
```

```
// dynamic-styles.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-dynamic-styles',
  template: `<p [ngStyle]="{'font-weight': isSpecial ? 'bold' : 'normal', 'color': isSpecial ? 'red' : 'black'}">Style me dynamically!</p>`,
  styles: [`p { font-size: 18px; }`]
})
export class DynamicStylesComponent {
  @Input() isSpecial: boolean;
}
```

Lab 16: Using Custom Pipes

Exercise:

Create a custom pipe named `reverseStr` that reverses a given string and use it in a component.

Solution:

```typescript
// reverse-str.pipe.ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'reverseStr'
})
export class ReverseStrPipe implements PipeTransform {
  transform(value: string): string {
    return value.split('').reverse().join('');
  }
}

// use-reverse-str.component.ts
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-use-reverse-str',
  template: `<p>{{'Angular' | reverseStr}}</p>`,
  styles: [`p { font-size: 18px; }`]
})
export class UseReverseStrComponent { }
```

Lab 17: Creating a Simple Service

Exercise:

Create a simple service named `dataService` that provides a method to get a list of items. Use this service in a component named `data-consumer`.

Solution:

```typescript
// data.service.ts
```

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

@Injectable({
  providedIn: 'root'
})
export class DataService {
  getItems(): string[] {
    return ['Item 1', 'Item 2', 'Item 3'];
  }
}

// data-consumer.component.ts
import { Component, OnInit } from '@angular/core';
import { DataService } from './data.service';

@Component({
  selector: 'app-data-consumer',
  template: `
    <ul>
      <li *ngFor="let item of items">{{item}}</li>
    </ul>
```

```
  `,
  styles: [`ul { list-style-type: none; padding: 0; } li { padding: 4px; }`]
})
export class DataConsumerComponent implements OnInit {
  items: string[];

  constructor(private dataService: DataService) { }

  ngOnInit(): void {
    this.items = this.dataService.getItems();
  }
}
```

Lab 18: HTTP Client Module

Exercise:

Create a service named `apiService` that fetches data from an API endpoint. Use this service in a component named `api-consumer` to display the data.

Solution:

```typescript
// api.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ApiService {
  private apiUrl = 'https://jsonplaceholder.typicode.com/posts';

  constructor(private http: HttpClient) { }
```

```
  getPosts(): Observable<any[]> {
    return this.http.get<any[]>(this.apiUrl);
  }
}

// api-consumer.component.ts
import { Component, OnInit } from '@angular/core';
import { ApiService } from './api.service';

@Component({
  selector: 'app-api-consumer',
  template: `
    <ul>
      <li *ngFor="let post of posts">{{post.title}}</li>
    </ul>
  `,
  styles: [`ul { list-style-type: none; padding: 0; } li { padding: 4px; }`]
})
export class ApiConsumerComponent implements OnInit {
```

```
  posts: any[];

  constructor(private apiService: ApiService) { }

  ngOnInit(): void {
    this.apiService.getPosts().subscribe(data => {
      this.posts = data;
    });
  }
}
```

Lab 19: Routing Module

Exercise:

Set up routing for an Angular application with two components: `home` and `about`. Navigate between these components using links.

Solution:

```typescript
// 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';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

```
// home.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-home',
  template: `<h1>Home</h1>`,
  styles: [`h1 { font-size: 24px; }`]
})
export class HomeComponent { }

// about.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-about',
  template: `<h1>About</h1>`,
  styles: [`h1 { font-size: 24px; }`]
})
export class AboutComponent { }
```

Lab 20: Lazy Loading Modules

Exercise:

Create a feature module named `featureModule` with a component `featureComponent`. Configure lazy loading for this module in the main routing configuration.

Solution:

```typescript
// feature/feature.module.ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FeatureComponent } from './feature/feature.component';
import { RouterModule, Routes } from '@angular/router';
```

```
const routes: Routes = [
  { path: '', component: FeatureComponent }
];

@NgModule({
  declarations: [FeatureComponent],
  imports: [
    CommonModule,
    RouterModule.forChild(routes)
  ]
})
export class FeatureModule { }

// feature/feature.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-feature',
  template: `<h1>Feature Component</h1>`,
  styles: [`h1 { font-size: 24px; }`]
})
```

```
export class FeatureComponent { }

// 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';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'feature', loadChildren: () =>
import('./feature/feature.module').then(m =>
m.FeatureModule) }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
```

```
})
export class AppRoutingModule { }
```