Angular Training

(Intermediate to Advanced)

Narasimha

Sr. Corporate Trainer, Mentor

tnrao.trainer@gmail.com

Schedule for React JS Training

Day1 : Custom Pipes in Angular

Day2 : Parent-Child Communication

Day3 : Custom Directives in Angular

Day4 : Working with Reactive Forms

Day5 : Dependency Injection and Services in Angular

Day6 : RxJS Library – Observables

Day7 : Http Client – Server calls in Angular

Day8 : Routing and Security in Angular

Day2
Parent-Child Component Communication

Narasimha

Sr. Corporate Trainer, Mentor

tnrao.trainer@gmail.com

Index – Day2

- Component communication
- Data Sharing between components
- Parent- Child using @Input()
- Child - Parent using @Output

Sharing data between child and parent

- A common pattern in Angular is sharing data between a parent component and one or more child components.
- Implement this pattern with the @Input() and @Output() decorators.

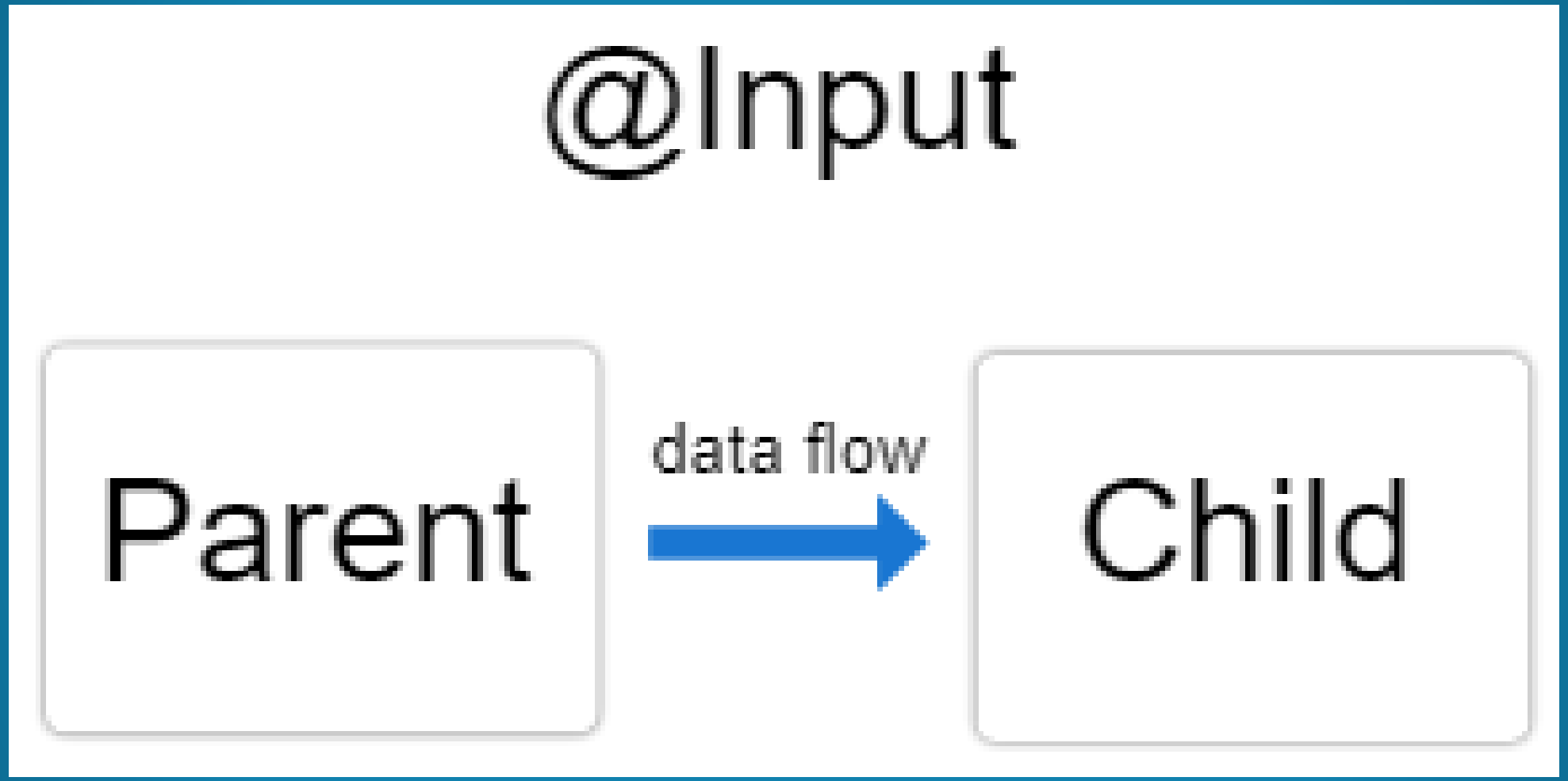
@Input and @Output

- @Input() and @Output() give a child component a way to communicate with its parent component.
- @Input() lets a parent component update data in the child component.
- Conversely, @Output() lets the child send data to a parent component.

@Input()

Narasimha

Sr. IT Trainer/Consultant


@Input
Parent
data flow
Child

@Input()

- To use the @Input() decorator in a child component class, first import Input decorator and then decorate the property with @Input()

greeting.component.ts

```
import { Component, Input } from '@angular/core';
export class GreetingComponent {
        @Input() uname = ''; // decorate the property with @Input()
}
```

@Input() – Configure Child

greeting.component.ts

```
import { Component, Input } from '@angular/core';
export class GreetingComponent {
        @Input() uname = ''; // decorate the property with @Input()
}
```

greeting.component.html

```
<h3>Welcome to {{uname}}</h3>
```

Configure Parent component

app.component.ts

```
import { Component, Input } from '@angular/core';
export class AppComponent {
                user1:string  = "John";
}
```

app.component.html

```
<app-greeting  uname="Narasimha"></app-greeting>
<app-greeting  [uname]="user1"></app-greeting>
```

Parent & Child

greeting.component.ts

```
import { Component, Input } from '@angular/core';
export class GreetingComponent {
        @Input() uname = '';
}
```

greeting.component.html

```
<h3>Welcome to {{uname}}</h3>
```

app.component.ts

```
import { Component, Input } from '@angular/core';
export class AppComponent {
        user1:string  = "John";
}
```

app.component.html

```
<app-greeting  uname="Scott"></app-greeting>
<app-greeting  [uname]="user1"></app-greeting>
```


Practical
HandsOns

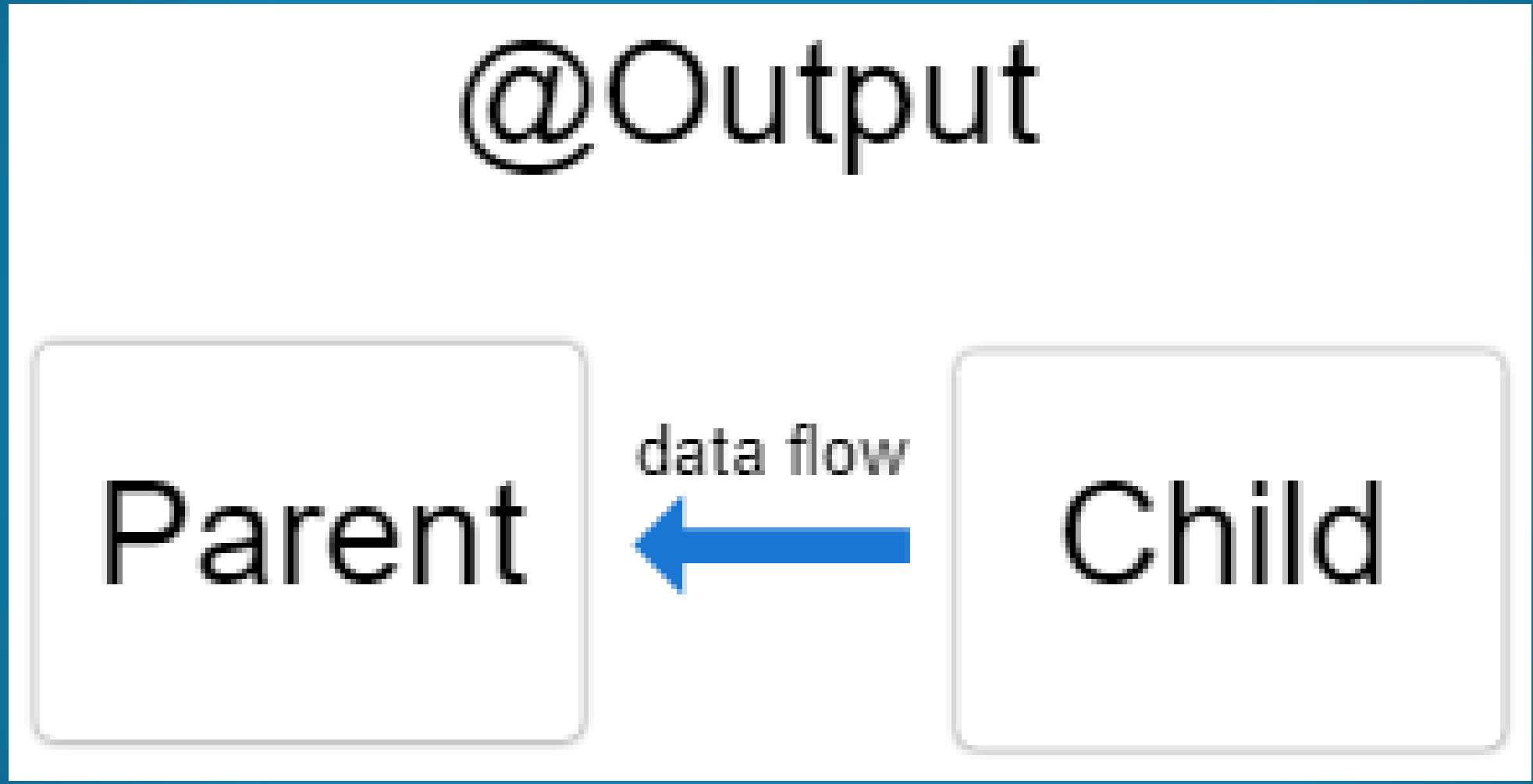
Output

Narasimha

Sr. IT Trainer/Consultant

@Output() decorator

- The @Output() decorator in a child component lets data flow from the child to the parent.
- @Output() marks a property in a child component as a doorway through which data can travel from the child to the parent.


@Output
Parent
data flow
Child

@Output() decorator

- The child component uses the @Output() property to raise an event to notify the parent of the change.
- To raise an event, an **@Output()** must have the type of **EventEmitter**,
- **EventEmitter** is a class in **@angular/core** that you use to emit custom events.

@Output() decorator --- configure child

1. import { Output, EventEmitter } from '@angular/core’;
2. @Output() onDeptRemove:EventEmitter<number> = new EventEmitter();
3. this.onDeptRemove.emit(dno);

@Output() decorator --- configure parent

1. <app-dept-details
   (onDeptRemove)="deptRemove_click_parent($event)”>
2. deptRemove_click_parent(dno:number) { … }

- The event binding, (onDeptRemove)="deptRemove_click_parent($event)”,
  - connects the event in the child - onDeptRemove,
  - to the method in the parent - deptRemove_click_parent().

- The $**event** contains the data that the user selected department in child template UI.


Practical
Hands-Ons

Narasimha

Sr. Corporate Trainer, Mentor

tnrao.trainer@gmail.com