

# @Input

Pass data from parent component  
To child component/directive

# @Input

- @Input is used for passing data from a parent component to a child component or directive

## Parent

```
@Component({
  selector: 'app-root',
  template: `
    <academeez-child [messageObj]="title">
    </academeez-child>
  `,
})
export class AppComponent {
  title = {message: 'hello child'};
}
```

## Child

```
@Component({
  selector: 'academeez-child',
  template: `
    <h1> {{ messageObj.message }} </h1>
  `,
})
export class ChildComponent {
  @Input()
  messageObj: {message: string};
}
```

# @Input - directive

- @Input can also be passed to directive

## Parent

```
@Component({
  selector: 'app-root',
  template: `
    <input child [messageObj]="title" ↗
  `,
})
export class AppComponent {
  title = {message: 'hello child'};
}
```

## Child

```
@Directive({ selector: '[child]' })
export class ChildDirective {
  @Input()
  messageObj: { message: string };
}
```

# @Input - with name

- @Input can get a name for the binding property

## Parent

```
@Component({
  selector: 'app-root',
  template: `
    <input child [messageObj]="title" ↗
  `,
})
export class AppComponent {
  title = {message: 'hello child'};
}
```

## Child

```
export class ChildComponent {
  @Input('messageObj')
  messageWrapper: {message: string};
}
```

# @Input by value by reference

- The data is passed from parent to child, if the data is primitive it is passed by value:
  - ▶ String, Number, Boolean, undefined, Symbol
- If the data type is not one of the above it will be passed by reference

# OnChanges

- This component/directive lifecycle hook will be called when a data bound @Input property is changed
- Change is comparing the old value with the new with ===
  - This means on the same reference if a change is made it won't call the hook
- The hook gets as an argument an object of type SimpleChanges where the changes in all the inputs are described

```
export class ChildComponent implements OnChanges {  
  ngOnChanges(changes: SimpleChanges): void {  
  }  
}
```

# @Input as getter

- With the OnChanges hook we can do some logic when input properties change
- Another common way to do some logic when the @Input properties change is by using getters and setters

```
export class ChildComponent {  
  private _message: string;  
  
  @Input()  
  set message(newMessage: string) {  
    // do additional logic when input change  
    this._message = newMessage;  
  }  
  
  get message() {  
    return this._message;  
  }  
}
```

# OnInit

- This lifecycle hook will trigger once
- Will trigger after OnChanges at the init of the component
- Is used for initialisation
- The @Input properties will be populated and ready for use

```
export class ChildComponent implements OnInit {  
  ngOnInit() {  
    // component initialization is here...  
  }  
}
```



# Summary

- With @Input we can send data from parent component to child component/directive
- OnChanges lifecycle hook will be called when a change is made to an @Input
- OnInit is used for initialisation
- It is also common to use getters for @Input properties to know when the variables are changing and to do additional logic on change

# Thank You

Next Lesson: `@Output`