

# Nested Components

---

UI-Angular



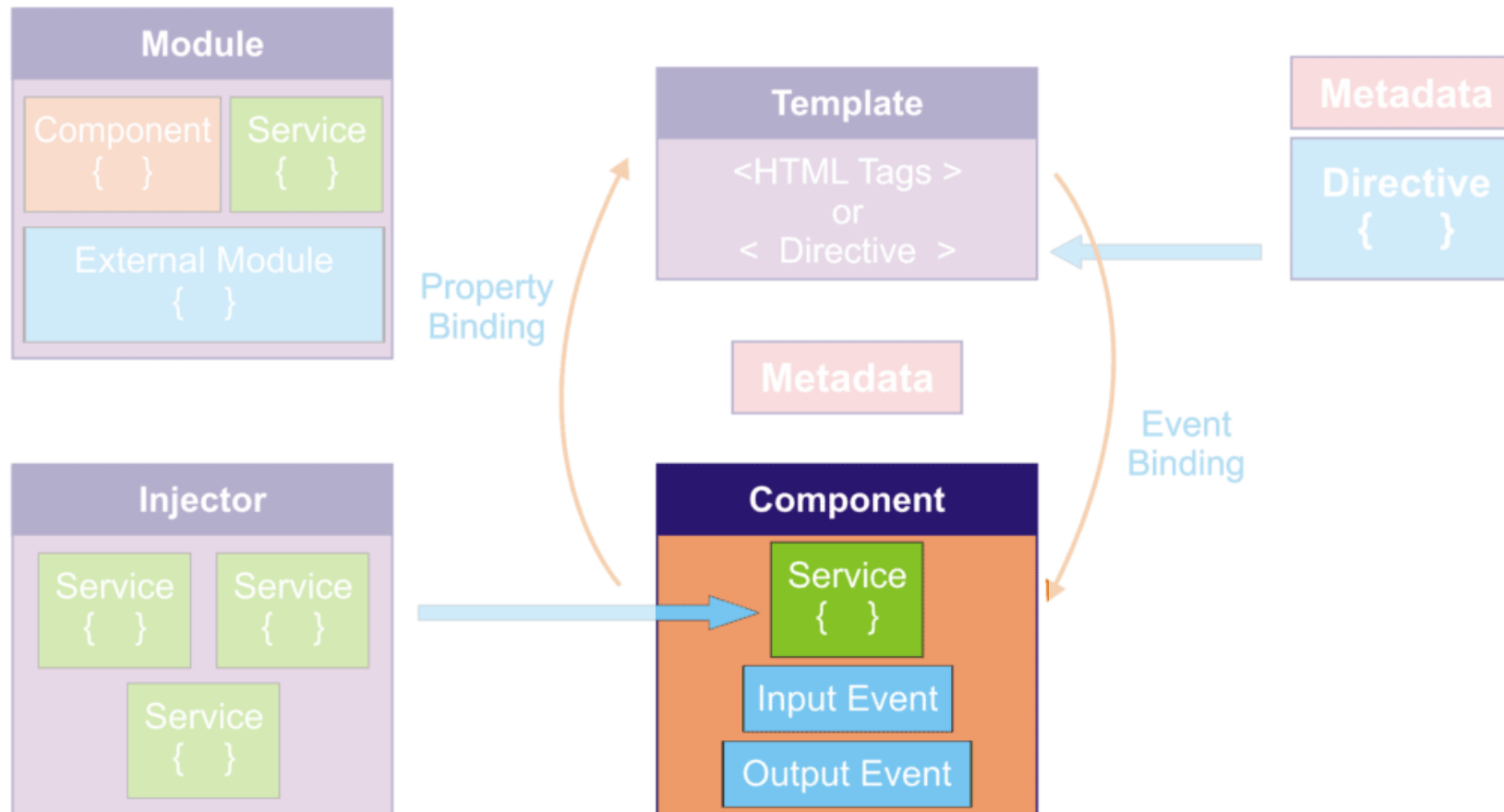
# מה נלמד היום?

---

- Child/Nested Components
- Component Communication
  - @Input
  - @Output, EventEmitter
  - Template Reference Variable
  - @ViewChild

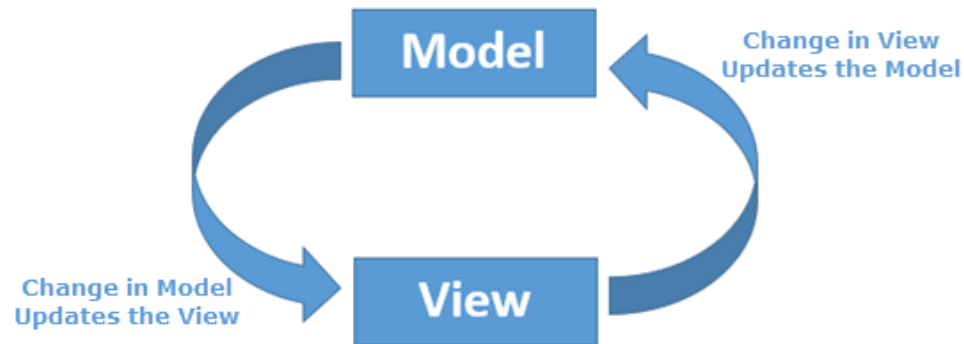
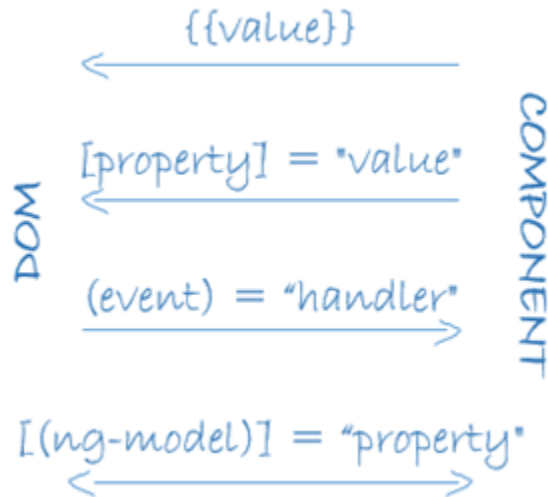


# ארכיטקטורת יישום Angular

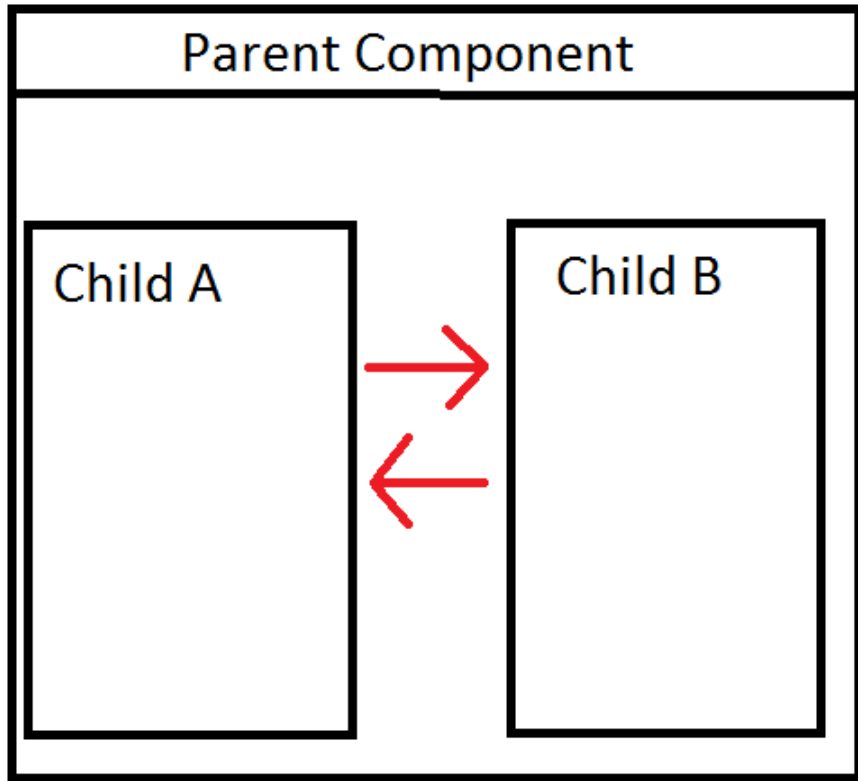


# Child/Nested Components

- הקומפוננטות הן אבני הבניין של יישום אנגולר.
- למדנו על טכניקות Data-Binding להעברת מידע בתוך הקומפוננטה
  - One-way Data Binding
  - Two-way Data Binding



# Child/Nested Components



- כל קומפוננטה עומדת בפניה עצמה וניתן להשתמש בה בשנית בחלקים אחרים של היישום.
- ניתן ליצור קומפוננטות מקוננות, כלומר קומפוננטה בתוך קומפוננטה ולהעביר מידע בין קומפוננטות

# Nested Components

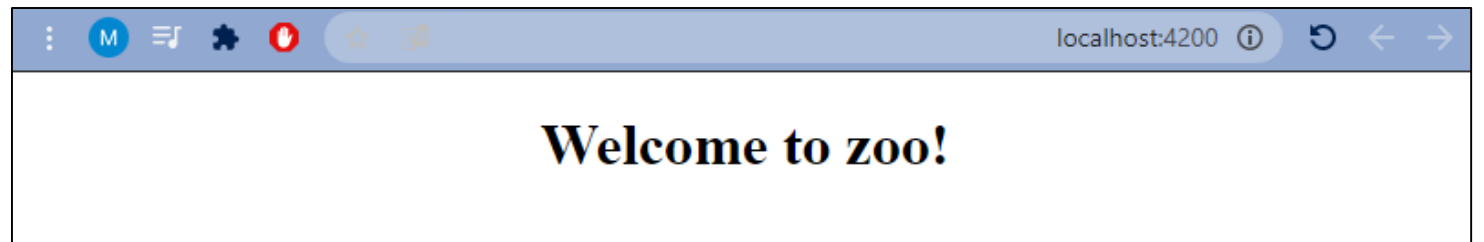
## יצירת קומפוננטה מקוננת דוגמה

■ שלב 1 – יצירת יישום אנגולר (אם לא קיים)

> ng new zoo

```
TS app.component.ts •
src > app > TS app.component.ts > ...
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'zoo';
10 }
```

```
<> app.component.html •
src > app > <> app.component.html > ...
1  <div [style.text-align]='center'>
2    <h1>
3      Welcome to {{title}}!
4    </h1>
5  </div>
```



## יצירת קומפוננטה מקוננת דוגמה

■ שלב 2 – יצירת קומפוננטות חדשות

> ng g c giraffes

```
<> giraffes.component.html ●  
src > app > giraffes > <> giraffes.component.html > ...  
1 <p>The giraffes eat {{food}}</p>
```

> ng g c giraffe

```
<> giraffe.component.html ✕  
src > app > giraffe > <> giraffe.component.html > .  
1 <p>giraffe works!</p>
```

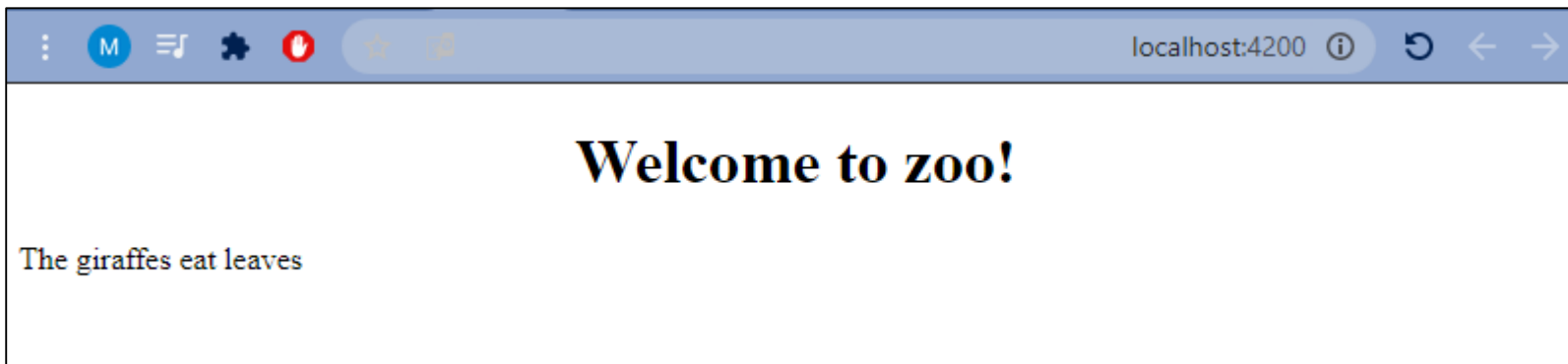
# Nested Components

## יצירת קומפוננטה מקוננת דוגמה

■ שלב 3 – שיבוץ הקומפוננטה giraffes בתוך הקומפוננטה הראשית

```
<> app.component.html •
src > app > <> app.component.html > ...
1  <div [style.text-align]='center'>
2    <h1>
3      Welcome to {{title}}!
4    </h1>
5  </div>
6  <app-giraffes></app-giraffes>
```

```
<> giraffes.component.html •
src > app > giraffes > <> giraffes.component.html > ...
1  <p>The giraffes eat {{food}}</p>
```





# Nested Components

## יצירת קומפוננטה מקוננת דוגמה

■ שלב 4 – שיבוץ הקומפוננטה giraffe בתוך הקומפוננטה giraffes

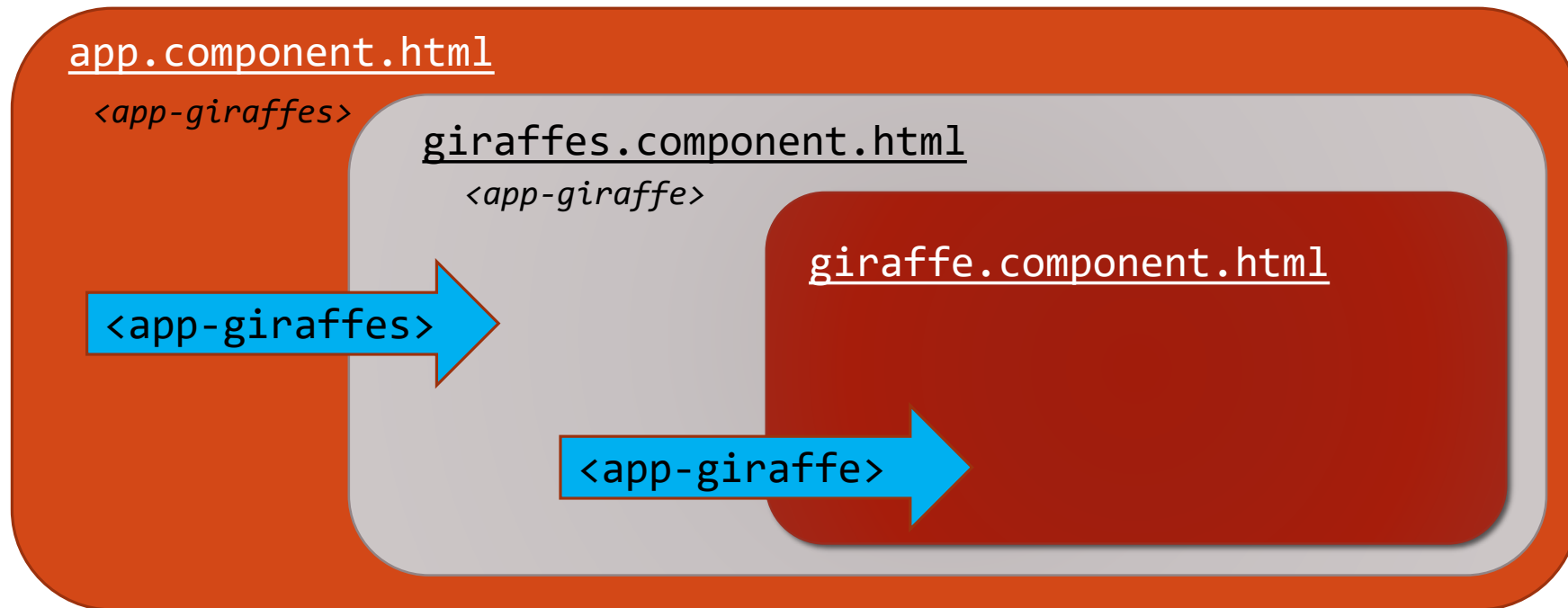
```
<> giraffes.component.html ●
src > app > giraffes > <> giraffes.component.html > ...
1  <p>The giraffes eat {{food}}</p>
2  <app-giraffe></app-giraffe>
```

```
<> giraffe.component.html ✕
src > app > giraffe > <> giraffe.component.html > ...
1  <p>giraffe works!</p>
```

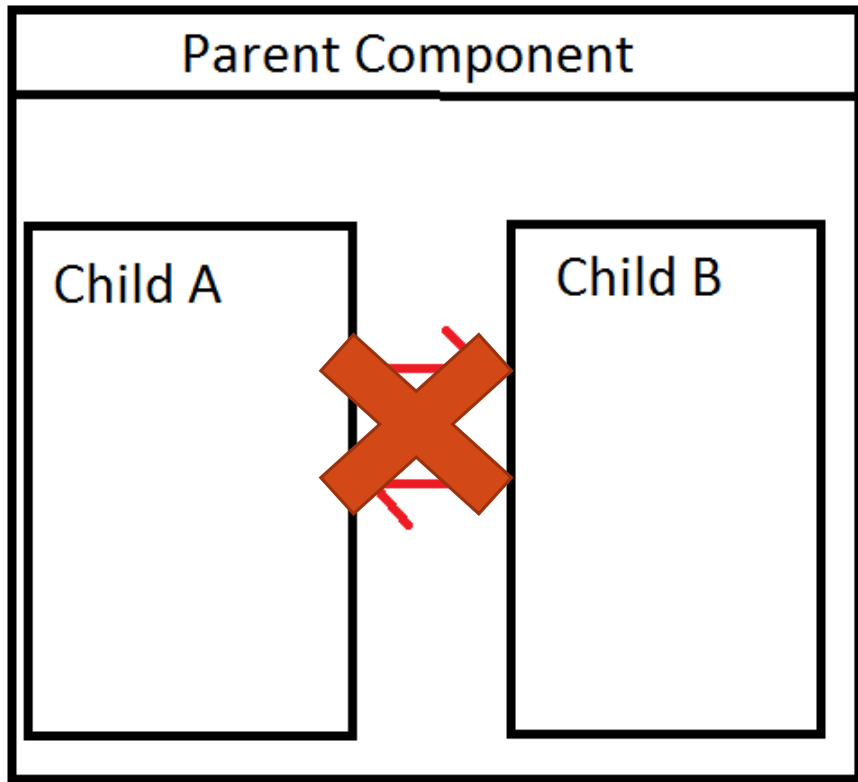


# Nested Components

## יצירת קומפוננטה מקוננת דוגמה



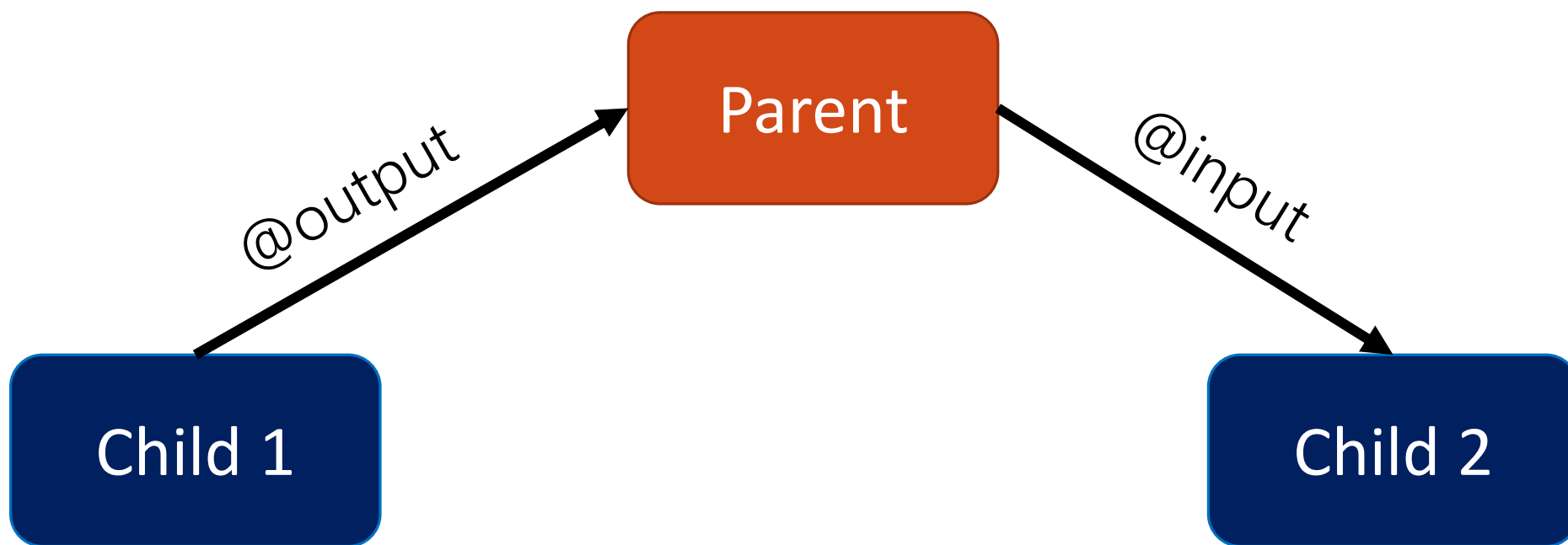
# Component Communication



- ברוב יישומי האנגולר הקומפוננטות מתקשרות זו עם זו, אחרת הן חסרות משמעות
- העברת מידע בין קומפוננטות אחיות מתבצעת דרך קומפוננטת האב
- כלומר ניתן להעביר מידע:
  - מילד לאב
  - מאב לילד

# Component Communication

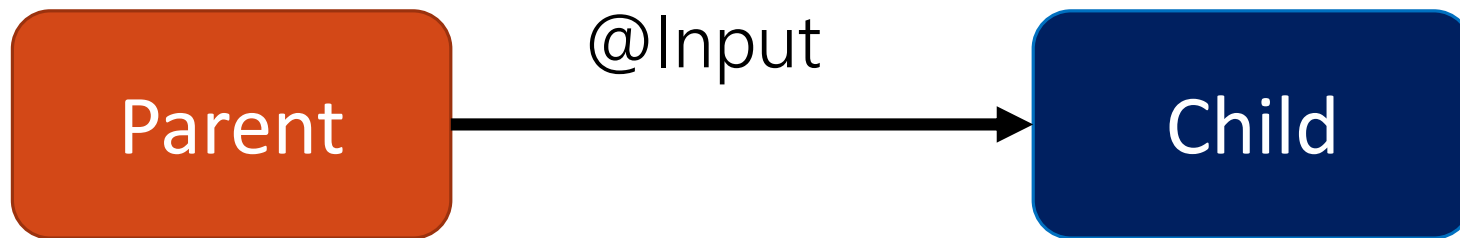
---



הקומפוננטות child1 ו-child2 הן קומפוננטות אחיות כי הן משובצות לאותה קומפוננטת אב

## העברת מידע מאב לילד

- ב-Angular קומפוננטת ה-parent יכולה לתקשר עם קומפוננטת ה-child על ידי:  
הגדרת המאפיין שלו (Property Binding).
- העברת המידע מתבצעת דרך תגית קומפוננטת ה-child
- קומפוננטת ה-child מקבלת את המידע באמצעות הדקורטור @Input



## @Input

### שינויים בקומפוננטת ה-child:

- ייבוא של המודול @Input מ- @angular/Core
- ייבוא הנתונים בקוד המחלקה ע"י שימוש בדקורטור @Input

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

@Component({
  selector: 'child-component',
  template: `<h2> Child Component </h2>
             <p> current count is {{ count }} </p>
  `
})
export class ChildComponent {
  @Input() count: number
}
```

# Component Communication

## @Input

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

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <button (click)="increment()">Increment</button>
    <button (click)="decrement()">decrement</button>
    <child-component [count]=Counter></child-component>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Passing Data to Child Component!'
  Counter = 5

  increment() {
    this.Counter++
  }
  decrement() {
    this.Counter--
  }
}
```

- שינויים בקומפוננטת ה-parent:
- העברת המידע לקומפוננטת ה-child (צורת property binding)





# Component Communication

## @Input

אפשר גם אחרת...

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

@Component({
  selector: 'child-component',
  template: `<h2>Child Component</h2>
    <p>current count is {{ count }}</p>
  `
})
export class ChildComponent {
  @Input() count: number
}
```



```
@Component({
  selector: 'child-component',
  inputs: ['count'],
  template: `<h2>Child Component</h2>
    current count is {{ count }}
  `
})
export class ChildComponent {}
```

## @Input

- עד כה ראינו כיצד ניתן להעביר מידע מקומפוננטה האב לקומפוננטה הילד באמצעות @input ו-property binding
- העברת המידע אינה מספיקה – קומפוננטת הילד צריכה לדעת מתי המידע משתנה ולהגיב בהתאם
- שתי טכניקות לזיהוי שינוי במידע:
  - ngOnChanges
  - Setter & Getter

# Component Communication

## @Input קומפוננטת האב (אין שינוי)

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

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <button (click)="increment()">Increment</button>
    <button (click)="decrement()">decrement</button>
    <child-component [count]=Counter></child-component>` ,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Passing Data to Child Component!'
  Counter = 5

  increment() {
    this.Counter++
  }
  decrement() {
    this.Counter--
  }
}
```

לצורך הדוגמה, בשתי טכניקות לזיהוי שינוי במידע:

- ngOnChanges
- Setter & Getter

זוהי קומפוננטת האב

# Component Communication

## שימוש ב-`ngOnChanges` בקומפוננטת הילד `@Input`

```
import { Component, Input, OnChanges, SimpleChanges, SimpleChange } from '@angular/core';
```

```
@Component({  
  selector: 'child-component',  
  template: `<h2>Child Component</h2>  
             current count is {{ count }}`  
})  
export class ChildComponent implements OnChanges {  
  @Input() count: number;
```

```
  ngOnChanges(changes: SimpleChanges) {  
    for (let property in changes) {  
      if (property === 'count') {  
        console.log('Previous:', changes[property].previousValue)  
        console.log('Current:', changes[property].currentValue)  
        console.log('firstChange:', changes[property].firstChange)  
      }  
    }  
  }  
}
```

יש  
שינוי?

# Component Communication

## שימוש ב-ngOnChanges בקומפוננטת הילד @Input

```
import { Component, Input, OnChanges, SimpleChanges, SimpleChange } from '@angular/core';
```

```
@Component({  
  selector: 'child-component',  
  template: `<h2>Child Component</h2>  
             current count is {{ count }}`  
})  
export class ChildComponent implements OnChanges {  
  @Input() count: number;
```

```
  ngOnChanges(changes: SimpleChanges) {  
  
    for (let property in changes) {  
      if (property === 'count') {  
        console.log('Previous:', changes[property].previousValue)  
        console.log('Current:', changes[property].currentValue)  
        console.log('firstChange:', changes[property].firstChange)  
      }  
    }  
  }  
}
```

# Component Communication

## שימוש ב-ngOnChanges בקומפוננטת הילד @Input

```
import { Component, Input, OnChanges, SimpleChanges, SimpleChange } from '@angular/core';
```

```
@Component({  
  selector: 'child-component',  
  template: `<h2>Child Component</h2>  
             current count is {{ count }}`  
})  
export class ChildComponent implements OnChanges {  
  @Input() count: number;
```

```
    ngOnChanges(changes: SimpleChanges) {  
  
      for (let property in changes) {  
        if (property === 'count') {  
          console.log('Previous:', changes[property].previousValue)  
          console.log('Current:', changes[property].currentValue)  
          console.log('firstChange:', changes[property].firstChange)  
        }  
      }  
    }  
  }  
}
```


# Component Communication

## שימוש ב-ngOnChanges בקומפוננטת הילד @Input

```
import { Component, Input, OnChanges, SimpleChanges, SimpleChange } from '@angular/core';

@Component({
  selector: 'child-component',
  template: `<h2>Child Component</h2>
             current count is {{ count }}
  })
export class ChildComponent implements OnChanges {
  @Input() count: number;

  ngOnChanges(changes: SimpleChanges) {
    for (let property in changes) {
      if (property === 'count') {
        console.log('Previous:', changes[property].previousValue)
        console.log('Current:', changes[property].currentValue)
        console.log('firstChange:', changes[property].firstChange)
      }
    }
  }
}
```



# Component Communication

## שימוש ב-Getters ו-Setters בקומפוננטת הילד @Input

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

@Component({
  selector: 'child-component',
  template: `<h2> Child Component </h2>
             <p> current count is {{count}} </p>
  `
})
export class ChildComponent {
  private _count = 0
  pcountChanged = false

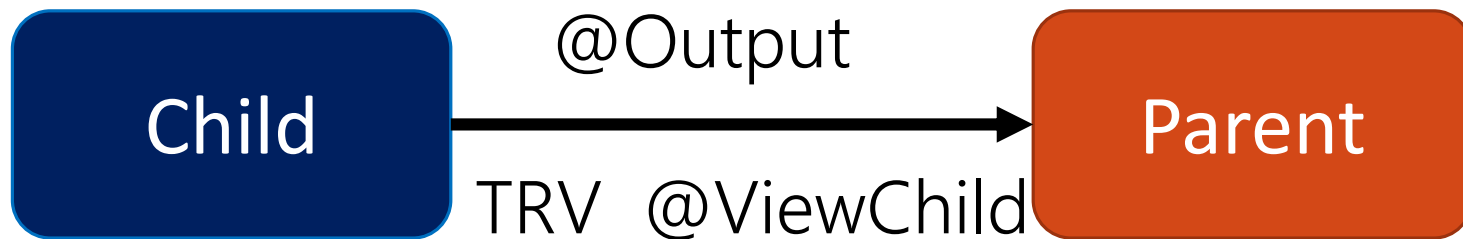
  @Input()
  set count(count: number) {
    console.log('previous item = ', this._count)
    this._count = count
    this.pcountChanged = true
  }
  get count(): number { return this._count }
}
```



## העברת מידע מילד לאב

ב- Angular קומפוננטת ה-child יכולה לתקשר עם קומפוננטת ה-parent על ידי:

- יצירת אירוע (@Output, EventEmitter)
- שימוש במשתנים מקומיים (Template Reference Variable)
- שימוש ב- @ViewChild



## יצירת אירוע

- כאשר קומפוננטת ה-child רוצה להעביר מידע לקומפוננטת ה-parent, היא יוצרת אירוע המעיד על השינוי
- קומפוננטת ה-child משתמשת במחלקה EventEmitter שמתקשרת עם הדקורטור @Output לצורך יצירת האירוע

### Child's component

```
countChanged: EventEmitter<number> = new EventEmitter() // creating the event
this.countChanged.emit(this.count) // raising the event with relevant data
```

## יצירת אירוע

- קומפוננטת ה-parent מקשיבה לאירוע שיצרה קומפוננטת ה-child ומגיבה אליו

### Parent's component

```
<child-component [count]=Counter (countChanged)="countChangedHandler($event)"></child-component>
```



# Component Communication

## יצירת אירוע

```
import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'child-component',
  template: `<h2>Child Component</h2>
    <button (click)="increment()">Increment</button>
    <button (click)="decrement()">Decrement</button>
    <p> current count is {{ count }} </p>
  `
})
export class ChildComponent {
  @Input() count: number;

  @Output() countChanged: EventEmitter<number> = new EventEmitter()

  increment() {
    this.count++;
    this.countChanged.emit(this.count)
  }
  decrement() {
    this.count--;
    this.countChanged.emit(this.count)
  }
}
```

### שינויים בקומפוננטת ה-child:

- יצירת אירוע של EventEmitter
- סימון האירוע בדקורטור @Output (Event Binding)
- הפעלת האירוע והעברת המידע הרלוונטי לקומפוננטת ה-parent

# Component Communication

## יצירת אירוע

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

```
@Component({
  selector: 'app-root',
  template: `
    <h1>Welcome to {{title}}!</h1>
    <p> current count is {{Counter}} </p>
    <child-component [count]=Counter (countChanged)="countChangedHandler($event)"></child-component>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Component Interaction'
  Counter = 5

  countChangedHandler(count: number) {
    this.Counter = count
    console.log(count)
  }
}
```

שינויים בקומפוננטת ה-parent:

- הגדרת קוד הפונקציה המטפלת באירוע
- קישור הפונקציה המטפלת באירוע לקומפוננטת ה-child

## שימוש במשתנים מקומיים

- ב-parent template ניתן לגשת למשתנים ולפונקציות של קומפוננט ה-child ע"י שימוש במשתנה מקומי המייצג "מופע" של הקומפוננט
- המשתנה המקומי הוא מסוג Template Reference Variable כלומר reference לאלמנט DOM.
- תבנית המשתנה המקומי:  
*#varName*
- דוגמה:  
`<child-component #child></child-component>`

# Component Communication

## שימוש במשתנים מקומיים

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

@Component({
  selector: 'child-component',
  template: `<h2>Child Component</h2>
    <p> current count is {{ count }} </p>
  `,
})
export class ChildComponent {
  count = 0

  increment() {
    this.count++
  }
  decrement() {
    this.count--
  }
}
```

שינויים בקומפוננטת ה-child:

▪ אין @input, @output, eventEmitter

## שימוש במשתנים מקומיים

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

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}!</h1>
    <p> current count is {{child.count}} </p>
    <button (click)="child.increment()">Increment</button>
    <button (click)="child.decrement()">decrement</button>
    <child-component #child></child-component>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Parent interacts with child via local variable'
}
```

שינויים בקומפוננטת ה-parent:  
■ הגדרת TRV בתגית קומפוננטת הילד



## שימוש ב-@ViewChild

- הזרקת מופע של קומפוננט ה-child לקומפוננט ה-parent בתור @ViewChild היא טכניקה נוספת בה משתמש ההורה לגישה למשתנים ולפונקציות של קומפוננט ה-child.
- השינוי הוא בקומפוננט ה-parent בלבד ועליה להכיל:
  - ייבוא המודול ViewChild
  - ייבוא קומפוננט ה-child
  - יצירת משתנה מחלקה שיכיל את מופע ה-child וסימונו ב-@ViewChild
- כעת ניתן להשתמש במופע ה-child (המשתנים והפונקציות שלו)



## שימוש בדקורטור @ViewChild

```
import { Component, ViewChild } from '@angular/core';
import { ChildComponent } from './child.component';

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <p> current count is {{child.count}} </p>
    <button (click)="increment()">Increment</button>
    <button (click)="decrement()">decrement</button>
    <child-component></child-component>`
})
export class AppComponent {
  title = 'Parent calls an @ViewChild()';

  @ViewChild(ChildComponent) child: ChildComponent

  increment() {
    this.child.increment()
  }
  decrement() {
    this.child.decrement()
  }
}
```

שינויים בקומפוננטת ה-parent:

# Component Communication

לסיכום:

- קומפוננט ה-parent יכולה לתקשר עם קומפוננט ה-child על ידי:
  - הגדרת המאפיין שלו (Property Binding)
  - העברת המידע מתבצעת דרך תגית קומפוננט ה-child
  - קומפוננט ה-child מקבלת את המידע באמצעות הדקורטור @Input
- קומפוננט ה-child יכולה לתקשר עם קומפוננט ה-parent על ידי:
  - יצירת אירוע (@Output, EventEmitter)
  - שימוש במשתנים מקומיים (Template Reference Variable)
  - שימוש ב- @ViewChild

