# NG

|  |
| --- |
| TypeScript |
| * Syntax looks like “Object Oriented Programming” ( Java -or- C# ) --- More Readable * Strong “Typing” --it helps in,   + Type Restriction   + Compilation Errors (Catch errors at Compile time –instead of Run Time)   + Better IDE -- Auto Complete (Code Suggestion), Refactoring * TypeScript is Superset of ES6   + TypeScript = ES6 Features + Additional Features * Transpiler converts,   + TS --> ES5   + TS --> ES6 * Transpiler (E.g. System.JS, WebPack) * tsconfig.json (TypeScript Configuration) |
| Angular |
| Angular is written in TypeScript |

|  |
| --- |
| String Interpolation: |
| @**Component**({  selector: 'string-interpolation',  template: `  <h3> StringInterpolationComponent </h3>  <p> myStr: {{myStr}} </p>  <p> getMyStrFn(): {{**getMyStrFn**()}} </p>  <p> Exp: 3+2 is: {{3+2}} </p>  {{ *<!-- write any JavaScript statement returns value -->* }}  *<!-- return value will be Stringified and rendered -->*  *<!--*  *WRONG USAGE: of 'String Interpolation' Examples (it works but NOT recommnded)*  *INSTEAD USE: Property Binding [myProperty]=".."*  *-->*  <p> myStr: <input *type*="text" *value*="{{myStr}}" /> </p>  <p> myObj: {{myObj}} </p>  <p> myArr: {{myArr}} </p>  *<!-- custom component:: 'String Interpolation' (WRONG practice) -->*  <my-custom-1 *prop1Str*="{{myStr}}" *prop2Str*="{{myStr}}"  *prop3Obj*="{{myObj}}" *prop4Arr*="{{myArr}}">  </my-custom-1>  `  })  class **StringInterpolationComponent** {  myStr : string;  myObj : **Object**;  myArr : **Array**<any>;  constructor(){    this.myStr = 'Jagadeesh';  this.myObj = { name: 'Jagdeesh', age: 22 };  this.myArr = ['Jagadeesh', 'Sundar', 'Saran'];  }  **getMyStrFn**() : string {  return this.myStr;  }  } |

|  |
| --- |
| One Way Binding :: Property Binding [] and Event Binding () |
| * [ ] — square brackets — Property Binding — we bind Input of the component * ( ) — square brackets —Event Binding — we bind output of the component |

|  |
| --- |
| Property Binding [] |
| * [ ] — square brackets — Property Binding — we bind Input of the component   @**Component**({  selector: 'property-binding',  template: `  <h3> PropertyBindingComponent </h3>  <input *type*="text" *[value]*="myStr" />  *<!-- or -->*  <input *type*="text" *bind-value*="myStr" />  *<!-- custom component:: property binding -->*  <my-custom-1 *[prop1Str]*="myStr" *[prop2Str]*="myStr"  *[prop3Obj]*="myObj" *[prop4Arr]*="myArr">  </my-custom-1>  `  })  class **PropertyBindingComponent** {  myStr : string;  myObj : **Object**;  myArr : **Array**<any>;  constructor(){  this.myStr = 'Jagadeesh';  this.myObj = { name: 'Jagdeesh', age: 22 };  this.myArr = ['Jagadeesh', 'Sundar', 'Saran'];  }  **getMyStrFn**() : string {  return this.myStr;  }  } |

|  |
| --- |
| Event Binding () |
| * ( ) — square brackets —Event Binding — we bind output of the component   @**Component**({  selector: 'event-binding',  template: `  <h3> EventBindingComponent </h3>  <p>myStr1: {{myStr1}}</p>  <input *type*="text" *[value]*="myStr" *(input)*="**myInputEventFired**($event)" />  <button *(click)*="**btnClickEventFired**()">Reset</button>  *<!-- or -->*  <input *type*="text" *[value]*="myStr" *on-input*="myInputEventFired($event)" />  <button *on-click*="btnClickEventFired()">Reset</button>  <br/><br/>  *<!-- custom component:: event binding -->*  <p>myNumbr1: {{myNumbr1}}</p>  <my-custom-2-counter *[myVal]*="myNumbr1"  *(myValChangeEvent)*="**myValChangeEventFired**($event)">  </my-custom-2-counter>  `  })  class **EventBindingComponent** {  myStr : string;  myObj : **Object**;  myArr : **Array**<any>;  myStr1 : string;  myNumbr1 : number;  constructor(){    this.myStr = 'Jagadeesh';  this.myObj = { name: 'Jagdeesh', age: 22 };  this.myArr = ['Jagadeesh', 'Sundar', 'Saran'];  this.myStr1= 'Jagadeesh1';  this.myNumbr1= 10;  }  **getMyStrFn**() : string {  return this.myStr;  }  **myInputEventFired**(event) {  *//console.log(event);*  this.myStr1 = event.target.value;  }  **btnClickEventFired**(event) {  *// console.log(event);*  this.myStr1 = 'Jagadeesh1';  }  **myValChangeEventFired**(myVal) {  *// console.log(myVal);*  this.myNumbr1 = myVal;  }  } |

|  |
| --- |
| Custom Component |
| Breaking Big Application into many small piece of Components   * Makes it easier to manage, architect the application * Reduces complexity * Re Use |

|  |
| --- |
| Custom Component (with Property Binding) @Input |
| *<!-- custom component -->*  <my-custom-1 *[prop1Str]*="myStr" *[prop2Str]*="myStr"  *[prop3Obj]*="myObj" *[prop4Arr]*="myArr">  </my-custom-1>  @**Component**({  selector: 'my-custom-1',  template: `  <h3> MyCustom1Component </h3>  <p>prop1Str: {{prop1Str}}</p>  <p>prop2Str: {{prop1Str}}</p>  <p>prop3Obj: {{prop3Obj | json}}</p>  <p>prop3Obj.name: {{prop3Obj.name}}</p>  <p>prop4Arr: {{prop4Arr | json}}</p>    `  })  class **MyCustom1Component** {  @**Input**() prop1Str: string;  @**Input**('prop2Str') prop2: string;  @**Input**() prop3Obj: **Object**;  @**Input**() prop4Arr: **Array**<any>;    *// without @Input, we cannot accept values from other components*  myInternalProperty: string;  constructor() { }  } |

|  |
| --- |
| Custom Component (with Event Binding) @Output |
| ..  @**Component**({  selector: 'my-custom-2-counter',  template: `  <h3> MyCustom2CounterComponent </h3>  <p>  <button *(click)*="**decreaseFn**()">Decrease</button>  {{myVal}}  <button *(click)*="**increaseFn**()">Increase</button>  </p>  `  })  class **MyCustom2CounterComponent** {    @**Input**()  myVal: number = 0;  @**Output**()  myValChangeEvent: **EventEmitter**<number> = new **EventEmitter**<number>();  **increaseFn**() {  this.myVal++;  this.myValChangeEvent.**emit**(this.myVal);  }  **decreaseFn**() {  this.myVal--;  this.myValChangeEvent.**emit**(this.myVal);  }  } |

|  |
| --- |
| Custom Component |
| .. |

|  |
| --- |
| Two Way Binding |
| ..  @**Component**({  selector: 'twoway-binding',  template: `  <h3> TwoWayBindingComponent </h3>  <p>myStr1: {{myStr1}}</p>  <input *type*="text" *[(ngModel)]*="myStr1" />  `  })  class **TwoWayBindingComponent** {  myStr1: string;  constructor() {  this.myStr1 = 'Jagadeesh1';  }  }  we must import in our ‘FormsModule’ – in order to use *[(ngModel)]* directive  import { FormsModule } from '@angular/forms';  @**NgModule**({  imports: [  BrowserModule,  FormsModule  ],  declarations: [  AppRootComponent  ],  providers: [],  bootstrap: [ AppRootComponent ]  })  export class **AppRootModule** { } |

|  |
| --- |
| Looping -using \*ngFor Directives : |
| <ul>  <li \**ngFor*="let user of users; index as i;">  {{ i }} -- {{ item.prop1 }}  </li>  </ul>  <ul>  <li \**ngFor*="let user of userObservable | async as users;  index as i; first as isFirst; last as isLast; even as isEven; odd as isOdd;">  <p>{{user.name}}</p>  <p> Index: {{i}}</p>  <p> Total Users: {{users.length}} </p>  <p \**ngIf*="isFirst">This is First Element</p>  <p \**ngIf*="isLast">This is Last Element</p>  <p \**ngIf*="isEven">This is Event Element</p>  <p \**ngIf*="isOdd">This is Odd Element</p>  </li>  </ul> |

|  |
| --- |
| Domain Model |
| ..  class **User** {  id: string;  name: string;  age: number;  hideAge: boolean;  constructor(id: string, name: string) {  this.id = id;  this.name = name;  this.hideAge = true;  }  **toggleAge**() {  this.hideAge = !this.hideAge;  }  } |

|  |
| --- |
| Custom Component |
| .. |