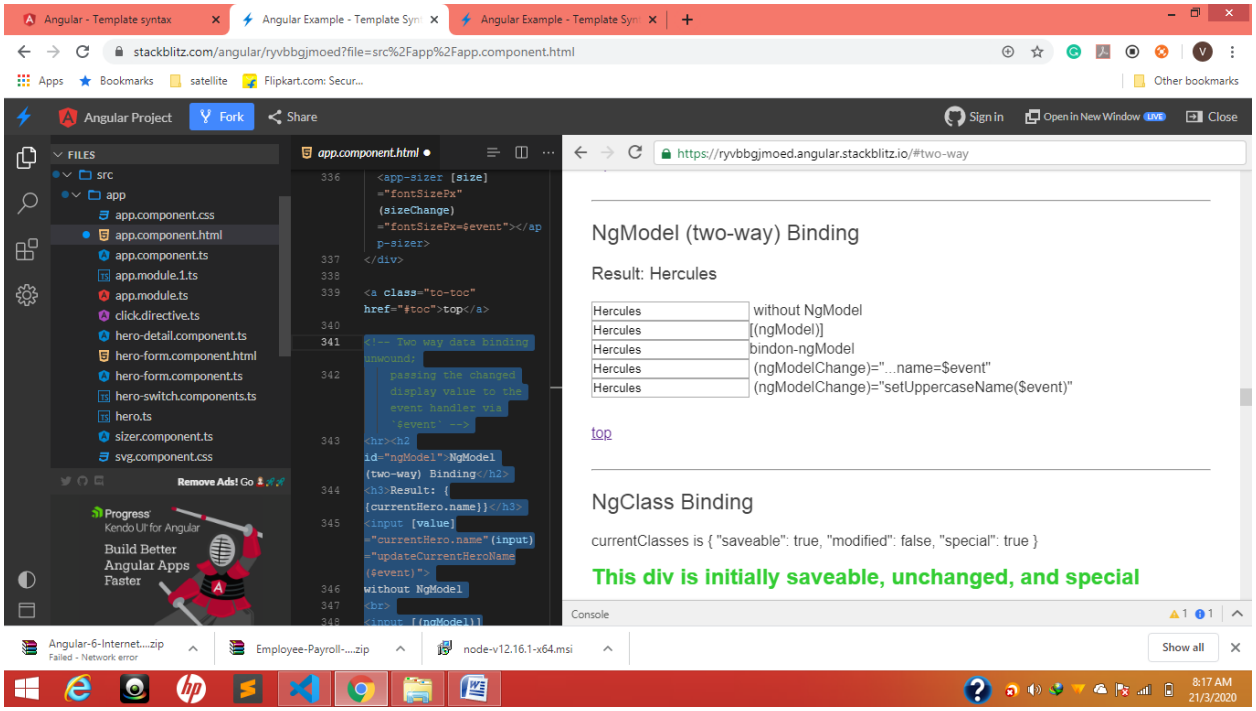
https://stackblitz.com/angular/ryvbbgjmoed?file=src%2Fapp%2Fapp.component.html

```
<hr><h2 id="enums">Enums in binding</h2>
```

It href will navigate to that

```
<a href="#enums">Enums</a><br>
```




```
<!-- Two way data binding unwound;
    passing the changed display value to the event handler via `$event` -->
<hr><h2 id="ngModel">NgModel (two-way) Binding</h2>
<h3>Result: {{currentHero.name}}</h3>
<input [value]="currentHero.name"(input)="updateCurrentHeroName($event)">
without NgModel
<br>
<input [(ngModel)]="currentHero.name">[(ngModel)]<br>
<input bindon-ngModel="currentHero.name">bindon-ngModel<br>
<input [ngModel]="currentHero.name" (ngModelChange)="currentHero.name=$event">
(ngModelChange)="...name=$event"
<br>
<input [ngModel]="currentHero.name" (ngModelChange)="setUppercaseName($event)">
(ngModelChange)="setUppercaseName($event)"
<a class="to-toc" href="#toc">top</a>
```

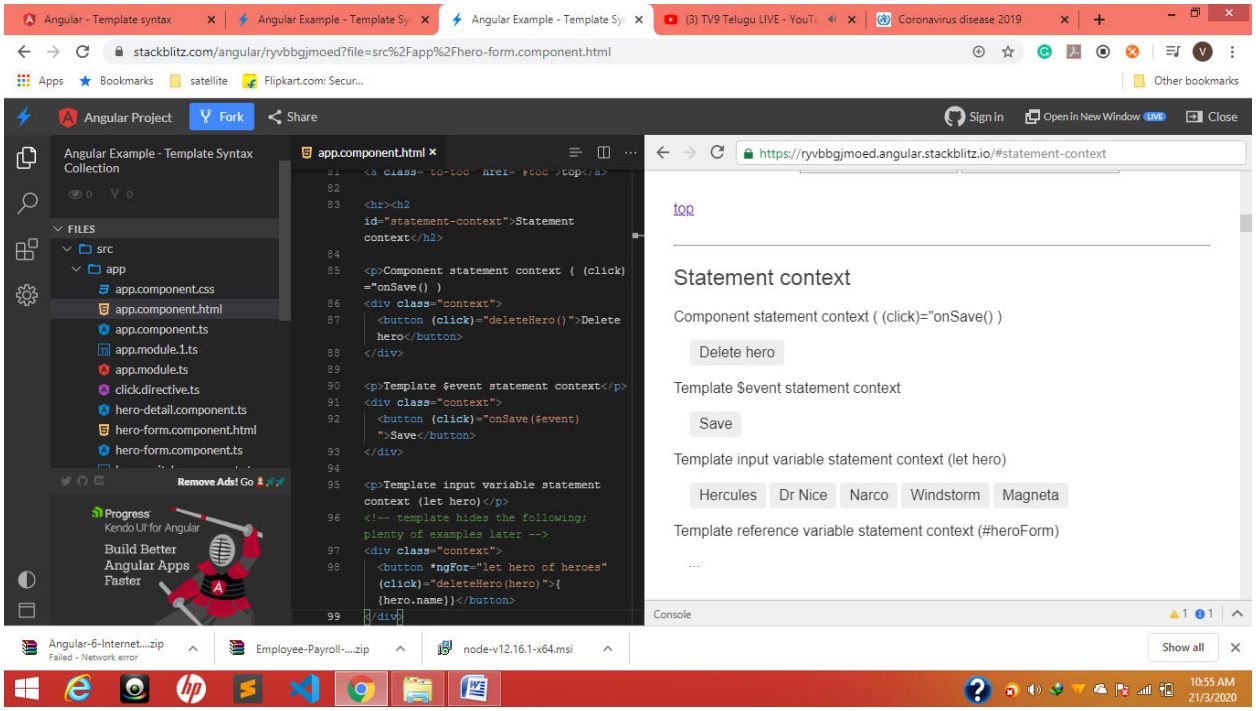
Expression context

`let customer`, or a template reference variable, `#customerInput`.

1. <li *ngFor="let customer of customers">{{customer.name}}

2.<label>Type something: <input #customerInput>{{customerInput.value}} </label>

Statement context

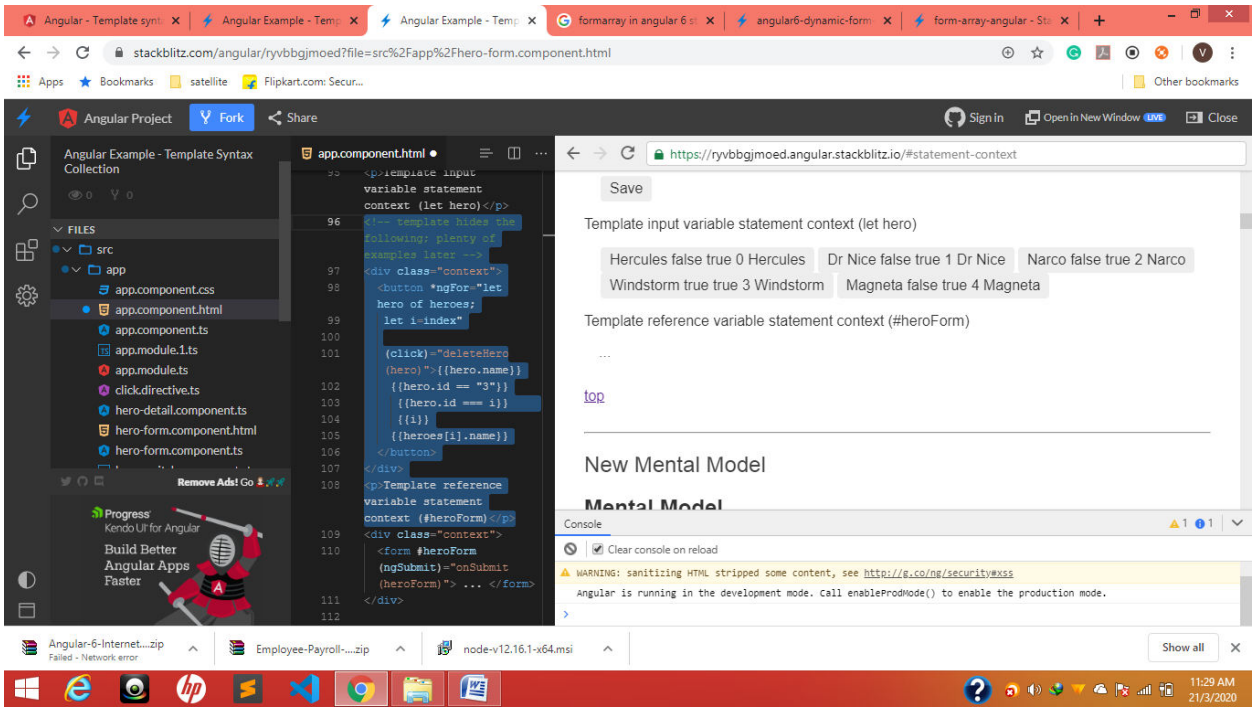



```
<div class="context">       <button (click)="deleteHero()">Delete hero</button>       </div>
```

```
deleteHero(hero?: Hero) {
    this.alert(`Delete ${hero ? hero.name : 'the hero'}.`);
  }
```

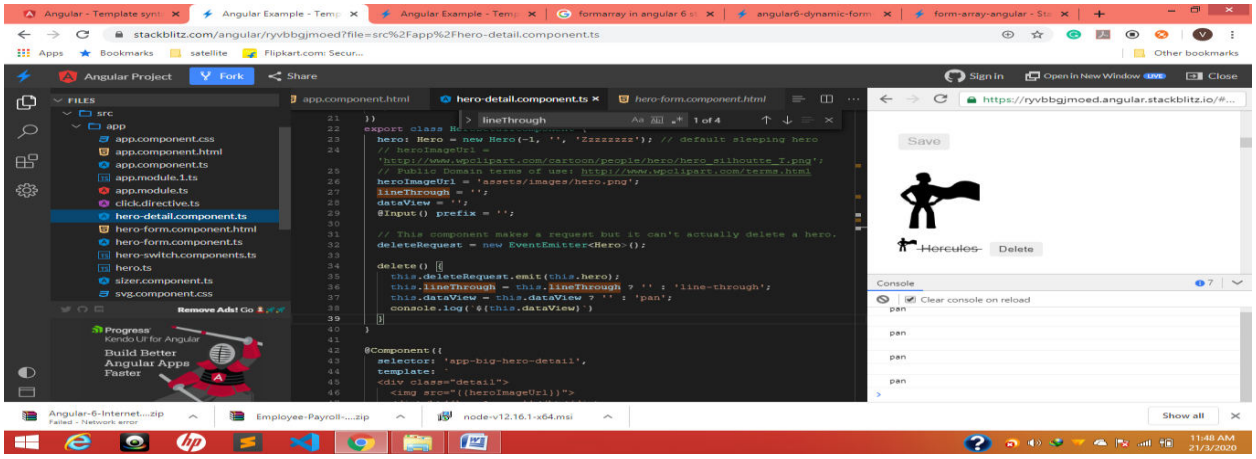
This is how we can print from ts----------('${hero.name}')-------- Html -----{{}}

```
console.log(`${hero.name}`)

 this.alert(`Delete ${hero ? hero.name : 'the hero'}.`);
  }
```




```
<!-- template hides the following; plenty of examples later -->
<div class="context">
  <button *ngFor="let hero of heroes;
   let i=index"

   (click)="deleteHero(hero)">{{hero.name}}
    {{hero.id == "3"}} // to get an idea
     {{hero.id === i}}
     {{i}}
    {{heroes[i].name}}
  </button>
</div>
<p>Template reference variable statement context (#heroForm)</p>
```

Toggle




```
heroImageUrl = 'assets/images/hero.png';
 lineThrough = '';
 dataView = '';
 @Input() prefix = '';
```

```
  // This component makes a request but it can't actually delete a hero.
  deleteRequest = new EventEmitter<Hero>();

  delete() {
    this.deleteRequest.emit(this.hero);
    this.lineThrough = this.lineThrough ? '' : 'line-through';
    this.dataView = this.dataView ? '' : 'pan';
    console.log(`${this.dataView}`)
  }
}
```

****

myClick click me

TS

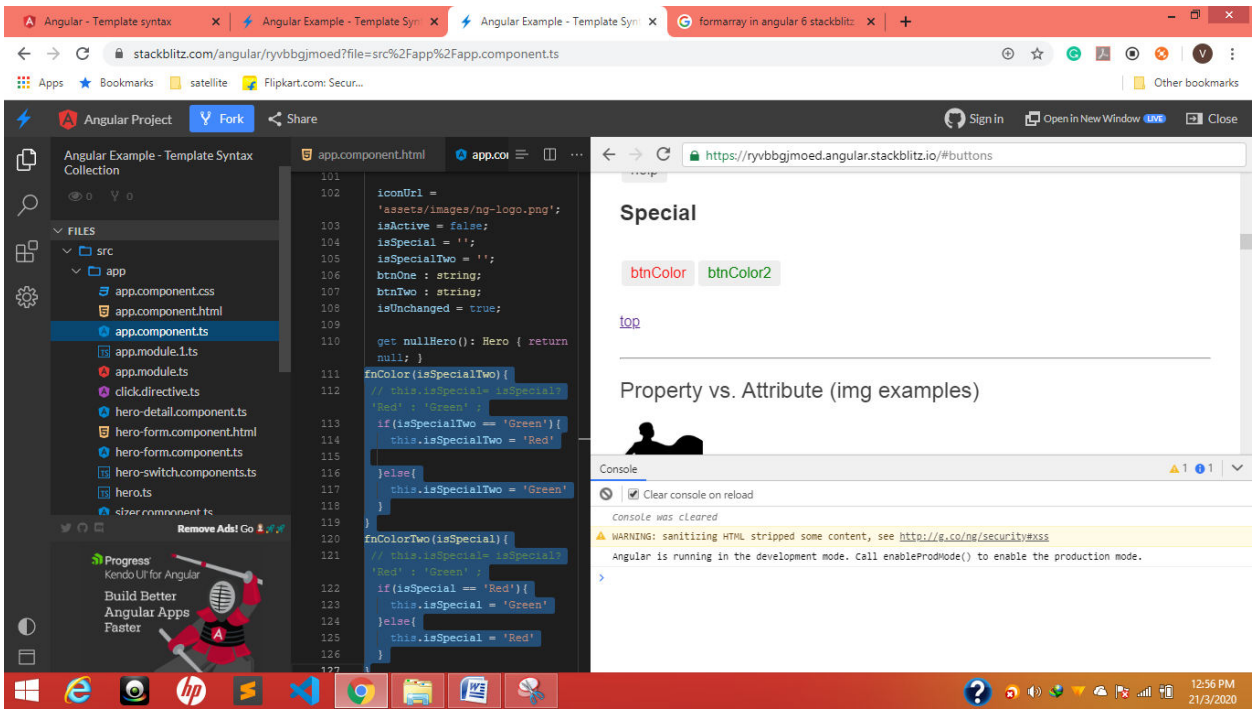
```
clicked = '';
```

HTML

```
<div (myClick)="clicked=$event" clickable>click me</div>
{{clicked}}
```

click me

Click!



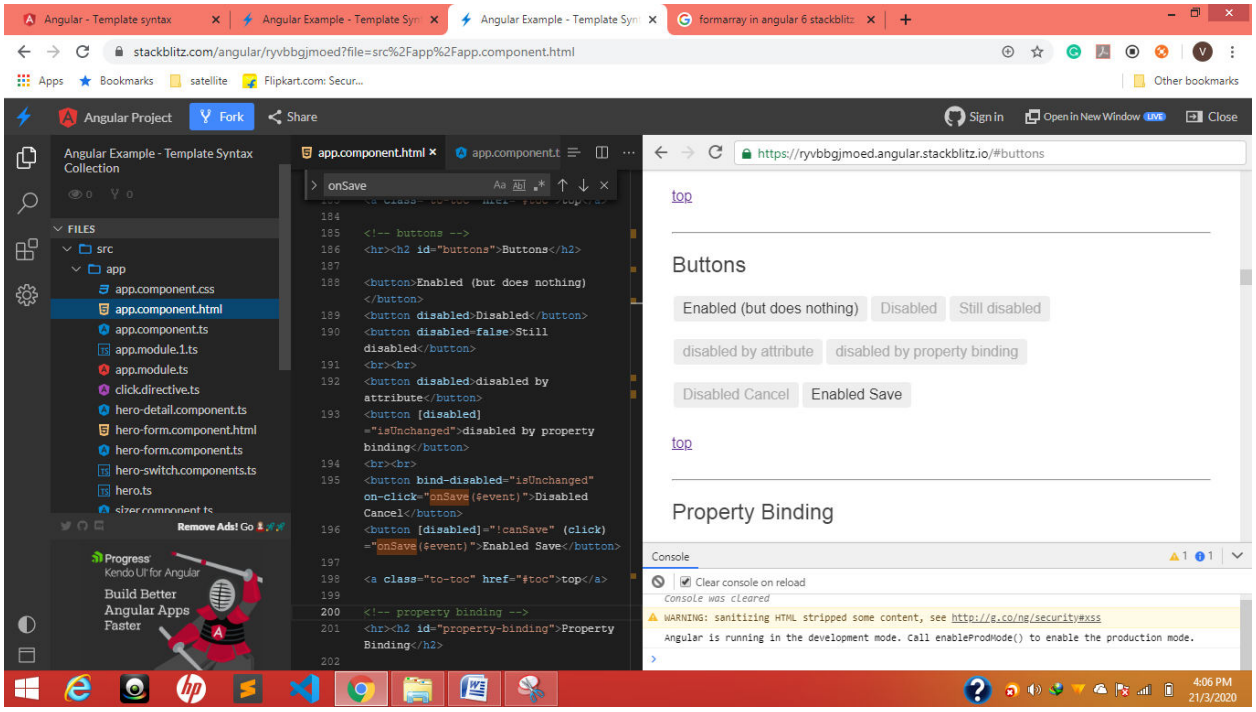

```
fnColor(isSpecialTwo){
```

```
 // this.isSpecial= isSpecial? 'Red' : 'Green' ;
  if(isSpecialTwo == 'Green'){
    this.isSpecialTwo = 'Red'

  }else{
    this.isSpecialTwo = 'Green'
  }
}
fnColorTwo(isSpecial){
 // this.isSpecial= isSpecial? 'Red' : 'Green' ;
  if(isSpecial == 'Red'){
    this.isSpecial = 'Green'
  }else{
    this.isSpecial = 'Red'
  }
}
```

HTML

```
button [style.color]="isSpecial" (click)="fnColor(isSpecialTwo)" >
btnColor</button>
<button [style.color]="isSpecialTwo" (click)="fnColorTwo(isSpecial)">
btnColor2 </button>
```

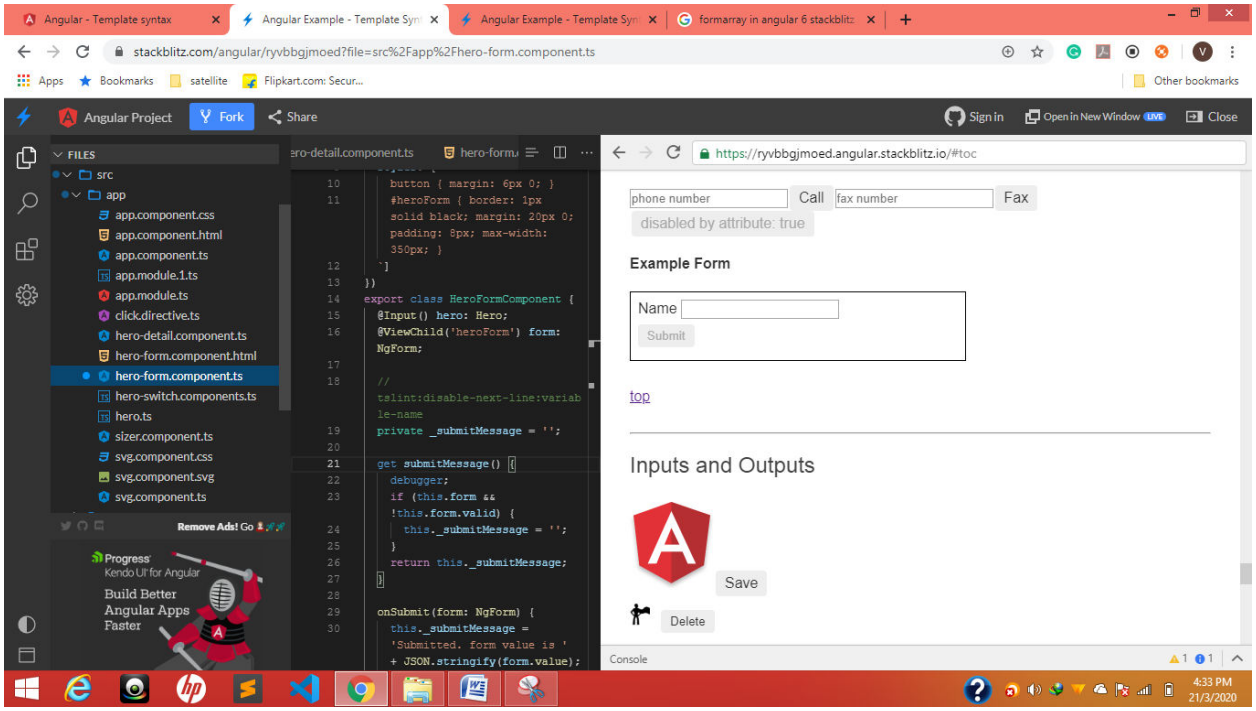
when two things think X



These many ways we can disable the things

```
<!-- buttons -->
<hr><h2 id="buttons">Buttons</h2>

<button>Enabled (but does nothing)</button>
<button disabled>Disabled</button>
<button disabled=false>Still disabled</button>
<br><br>
<button disabled>disabled by attribute</button>
<button [disabled]="isUnchanged">disabled by property binding</button>
<br><br>
<button bind-disabled="isUnchanged" on-click="onSave($event)">Disabled Cancel</button>
<button [disabled]="!canSave" (click)="onSave($event)">Enabled Save</button>
```

```
<a class="to-toc" href="#toc">top</a>
```



One text--one button with validation

```
@Input() hero: Hero;
@ViewChild('heroForm') form: NgForm;

// tslint:disable-next-line:variable-name
private _submitMessage = '';

get submitMessage() {
  debugger;
  if (this.form && !this.form.valid) {
    this._submitMessage = '';
  }
  return this._submitMessage;
}

onSubmit(form: NgForm) {
  this._submitMessage =  'Submitted. form value is ' + JSON.stringify(form.value);
}
}
```

```
<div id="heroForm">
  <form (ngSubmit)="onSubmit(heroForm)" #heroForm="ngForm">
    <div class="form-group">
      <label for="name">Name
        <input class="form-control" name="name" required [(ngModel)]="hero.name">
      </label>
    </div>
    <button type="submit" [disabled]="!heroForm.form.valid">Submit</button>
  </form>
  <div [hidden]="!heroForm.form.valid">
    {{submitMessage}}
  </div>
```

```
</div>
```

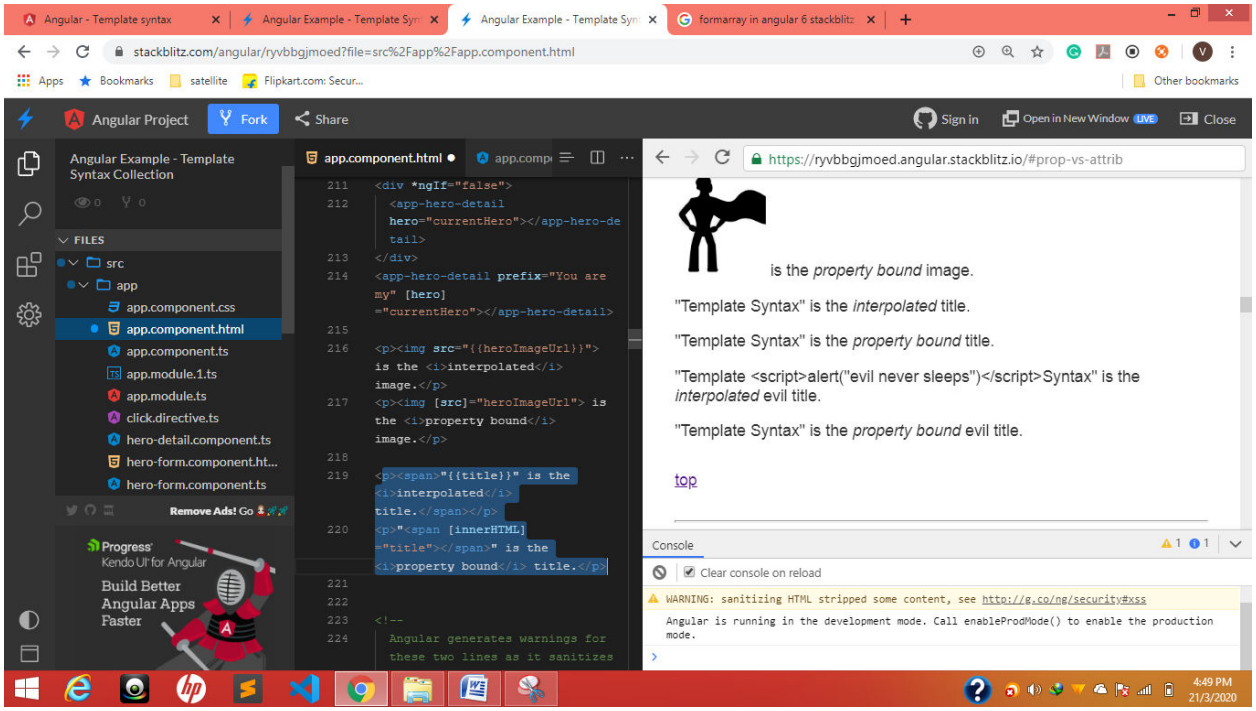
Check

```
<input class="form-control" name="name" required [(ngModel)]="hero.name">
```

```
<button type="submit" [disabled]="!heroForm.form.valid">Submit</button>
```

```
  <div [hidden]="!heroForm.form.valid"> {{submitMessage}}<div>
```

Property Binding




```
<p><span>"{{title}}" is the <i>interpolated</i> title.</span></p>
<p>"<span [innerHTML]="title"></span>" is the <i>property bound</i> title.</p>
```

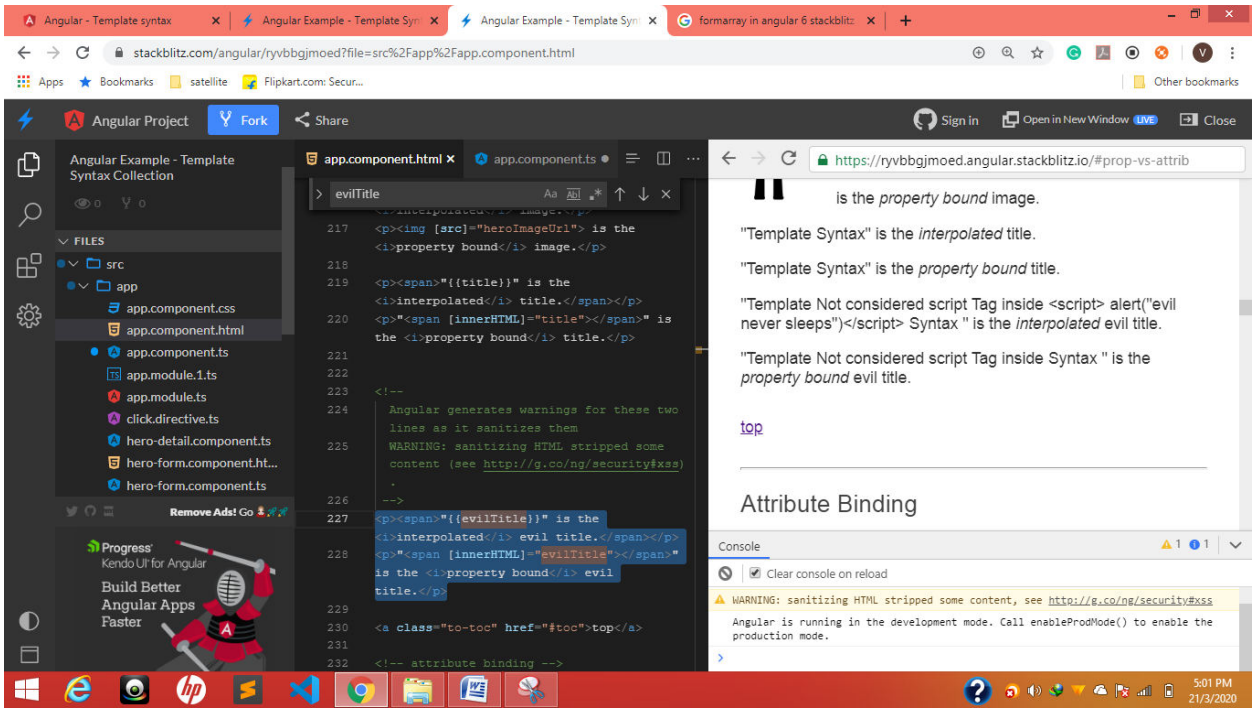
Instead of {{}} we can print by using property binding "<span [innerHTML]="title">"

2.

```
evilTitle = `Template Not considered script Tag inside
  <script> alert("evil never sleeps")</script>
  Syntax `;
```

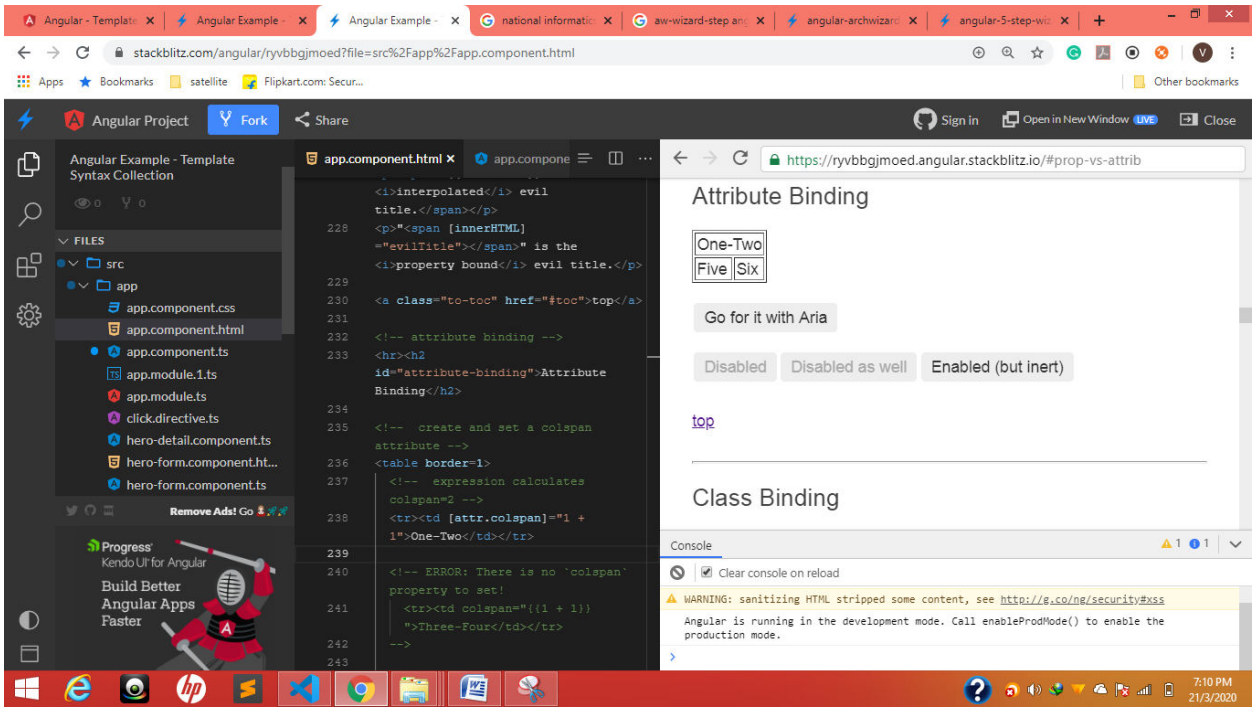
We want to print only Text not <script> one then we have to go propery binding

```
<p><span>"{{evilTitle}}" is the <i>interpolated</i> evil title.</span></p>
<p>"<span [innerHTML]="evilTitle"></span>" is the <i>property bound</i> evil title.</p>
```




```
<p><span>"{{evilTitle}}" is the <i>interpolated</i> evil title.</span></p>
<p>"<span [innerHTML]="evilTitle"></span>" is the <i>property bound</i> evil title.</p>
```

Attribute Binding

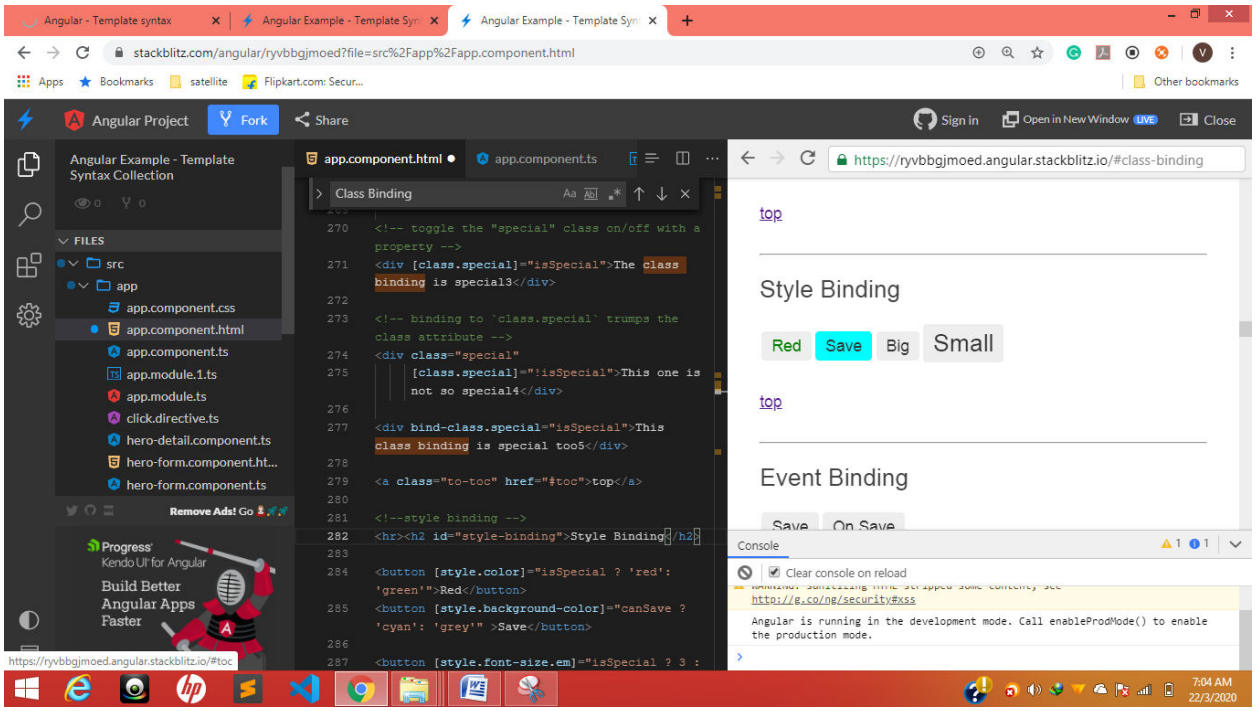


If we want to combine two columns dynamically in a table

```
<table border=1>
  <!--  expression calculates colspan=2 -->
  <tr>     <td [attr.colspan]="1 + 1">One-Two</td> </tr>
  <tr><td>Five</td><td>Six</td></tr>
</table>
```

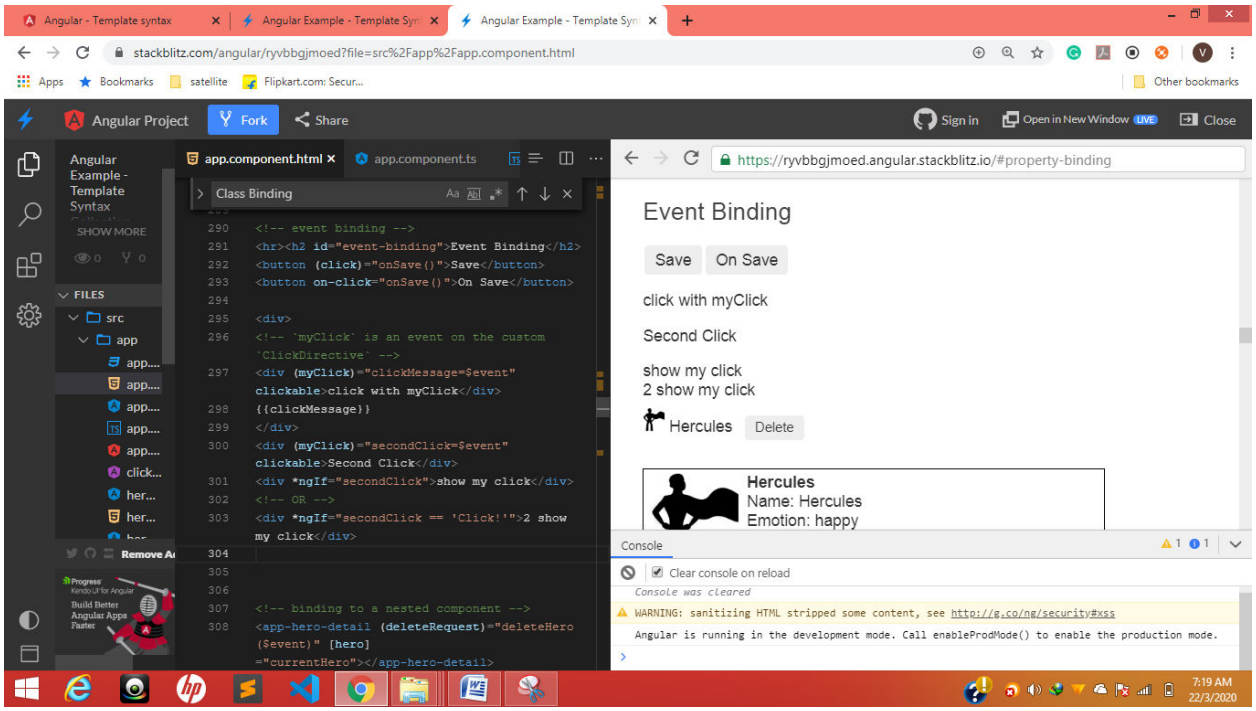
[attr.colspan]='1 + 1'

Style binding



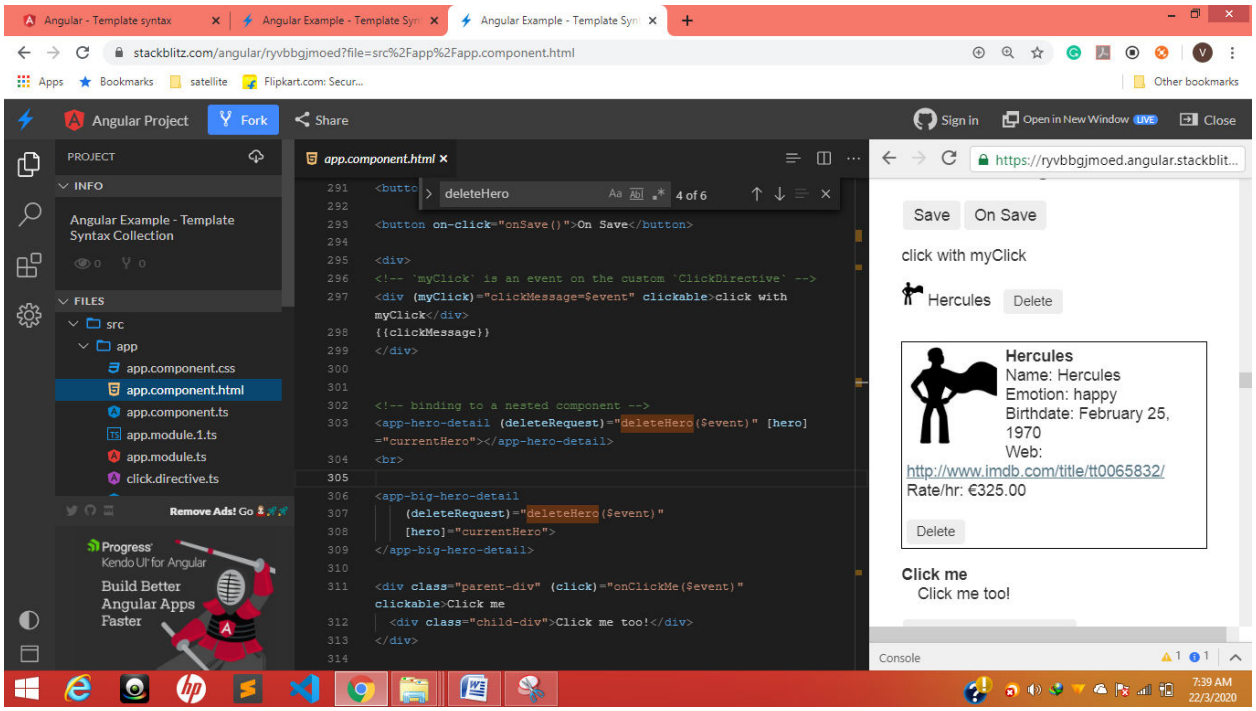

```
<!--style binding -->
<hr><h2 id="style-binding">Style Binding</h2>
<button [style.color]="isSpecial ? 'red': 'green'">Red</button>
<button [style.background-color]="canSave ? 'cyan': 'grey'" >Save</button>
<button [style.font-size.em]="isSpecial ? 3 : 1" >Big</button>
<button [style.font-size.%]="!isSpecial ? 150 : 50" >Small</button>
```

Event binding



(click), on-click,(myClick)—it can be used for toggle hide and show directly

```
<!-- event binding -->
<hr><h2 id="event-binding">Event Binding</h2>
<button (click)="onSave()">Save</button>
<button on-click="onSave()">On Save</button>

<div>
<!-- `myClick` is an event on the custom `ClickDirective` -->
<div (myClick)="clickMessage=$event" clickable>click with myClick</div>
{{clickMessage}}
</div>
<div (myClick)="secondClick=$event" clickable>Second Click</div>
<div *ngIf="secondClick">show my click</div>
<!-- OR -->
<div *ngIf="secondClick == 'Click!'">2 show my click</div>
```



@Output (deleteRequest) from chaild

```
<app-hero-detail (deleteRequest)="deleteHero($event)" [hero]="currentHero"></app-hero-detail>
```

```
deleteHero(hero?: Hero) {
    this.alert(`Delete ${hero ? hero.name : 'the hero'}.`);
  }
```

From app-hero

```
@Output() deleteRequest = new EventEmitter<Hero>();

  delete() {
    this.deleteRequest.emit(this.hero);
  }
```

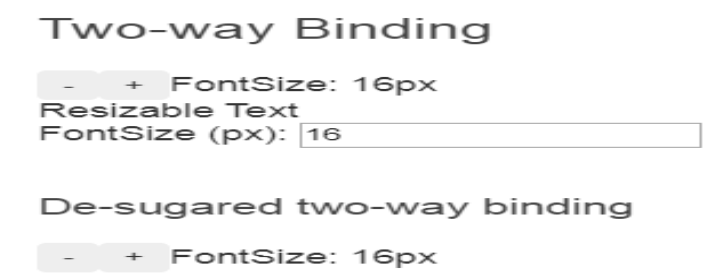
Parent Click me

Child No click event but works Click me too!

```
<div class="parent-div" (click)="onClickMe($event)" clickable>Parent Click me
  <div class="child-div">Child No click event but works Click me too!</div>
</div>
```

Save-once save-twice

```
<!-- Will save only once -->
<div (click)="onSave()" clickable>
  <button (click)="onSave($event)">Save, no propagation</button>
</div>

<!-- Will save twice -->
<div (click)="onSave()" clickable>
  <button (click)="onSave()">Save w/ propagation</button>
</div>
```

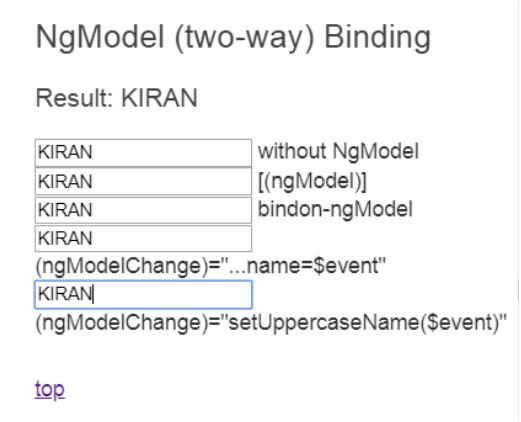


Sizer component ts'

```
<hr><h2 id="two-way">Two-way Binding</h2>
<div id="two-way-1">
  <app-sizer [(size)]="fontSizePx"></app-sizer>
  <div [style.font-size.px]="fontSizePx">Resizable Text</div>
  <label>FontSize (px): <input [(ngModel)]="fontSizePx"></label>
</div>
<br>
<div id="two-way-2">
  <h3>De-sugared two-way binding</h3>
  <app-sizer [size]="fontSizePx" (sizeChange)="fontSizePx=$event"></app-sizer>
</div>

<a class="to-toc" href="#toc">top</a>
```

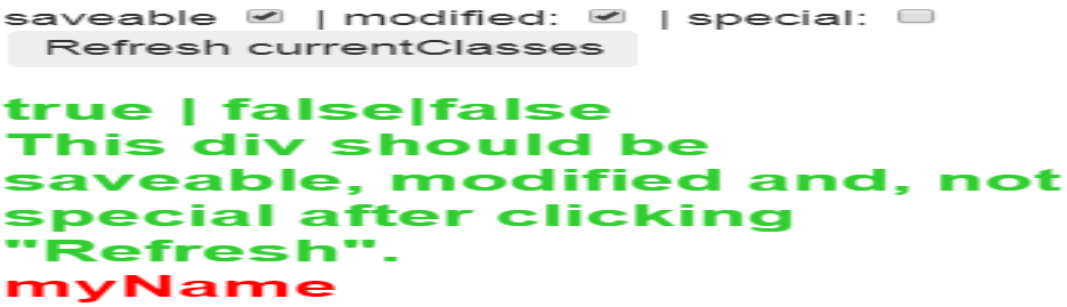
Sizer component

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

@Component({
  selector: 'app-sizer',
  template: `
  <div>
    <button (click)="dec()" title="smaller">-</button>
    <button (click)="inc()" title="bigger">+</button>
    <label [style.font-size.px]="size">FontSize: {{size}}px</label>
  </div>`
})
export class SizerComponent {
  @Input()  size: number | string;
  @Output() sizeChange = new EventEmitter<number>();

  dec() { this.resize(-1); }
  inc() { this.resize(+1); }

  resize(delta: number) {
    this.size = Math.min(40, Math.max(8, +this.size + delta));
    this.sizeChange.emit(this.size);
  }
}
```




```
<hr><h2 id="ngModel">NgModel (two-way) Binding</h2>

<h3>Result: {{currentHero.name}}</h3>

<input [value]="currentHero.name" (input)="updateCurrentHeroName($event)"> without NgModel
<br>
<input [(ngModel)]="currentHero.name">[(ngModel)]
<input bindon-ngModel="currentHero.name">bindon-ngModel
<br>
<input [ngModel]="currentHero.name"(ngModelChange)="currentHero.name=$event"> (ngModelChange)="...name=$event"
<br>
<input [ngModel]="currentHero.name"(ngModelChange)="setUppercaseName($event)">(ngModelChange)="setUppercaseName($event)"

<a class="to-toc" href="#toc">top</a>
```

ngClass binding



When deselected for first loaded time

Ts file

```
currentClasses: {};
  setCurrentClasses() {
    // CSS classes: added/removed per current state of component properties
    this.currentClasses =  {
      saveable: this.canSave,
      modified: !this.isUnchanged,
      special:  this.isSpecial
    };
  }
```

html

```
<p>currentClasses is {{currentClasses | json}}</p>
<div [ngClass]="currentClasses">This div is initially saveable, unchanged, and special</div>

<!-- not used in chapter -->
<br>
<label>saveable   <input type="checkbox" [(ngModel)]="canSave"></label> |
<label>modified:  <input type="checkbox" [value]="!isUnchanged" (change)="isUnchanged=!isUnchanged"></label> |
<label>special:   <input type="checkbox" [(ngModel)]="isSpecial"></label>
<button (click)="setCurrentClasses()">Refresh currentClasses</button>
<br><br>
<div [ngClass]="currentClasses">
  {{canSave}} | {{isUnchanged}}|{{isSpecial}}<br>
  This div should be {{ canSave ? "": "not"}} saveable,
                {{ isUnchanged ? "unchanged" : "modified" }} and,
                {{ isSpecial ? "": "not"}} special after clicking "Refresh".</div>
<br><br>
```

For any canSave ? 'true': 'false'

```
<br>
<label>saveable   <input type="checkbox" [(ngModel)]="canSave"></label> |
<label>modified:  <input type="checkbox" [value]="isUnchanged"
(change)="isUnchanged=!isUnchanged"
(change)="isSpecial=!isSpecial"></label> |
<label>special:   <input type="checkbox" [(ngModel)]="isSpecial"></label>
<button (click)="setCurrentClasses()">Refresh currentClasses</button>
<br><br>
<div [ngClass]="currentClasses">
  {{canSave}} | {{isUnchanged}}|{{isSpecial}}<br>
  This div should be {{ canSave ? "": "not"}} saveable,
                {{ isUnchanged ? "unchanged" : "modified" }} and,
                {{ isSpecial ? "": "not"}} special after clicking "Refresh".<br>
                <span [style.color]="isSpecial? 'Green' : 'Red' " >
              <!-- {{ isSpecial  ? "Green" : "Red" }}        -->
               myName Able to change colors  Toggle
                </span>
                </div>
```

Color changed toggle no need to write functions

```
<label>modified:  <input type="checkbox" [value]="isUnchanged"  (change)="isUnchanged=!isUnchanged" (change)="isSpecial=!isSpecial"></label>
                <span [style.color]="isSpecial? 'Green' : 'Red' " > myName Able to change colors  Toggle </span>
```

This can control with `[(ngModel)]="isSpecial"` or `(change)="isSpecial=!isSpecial"`

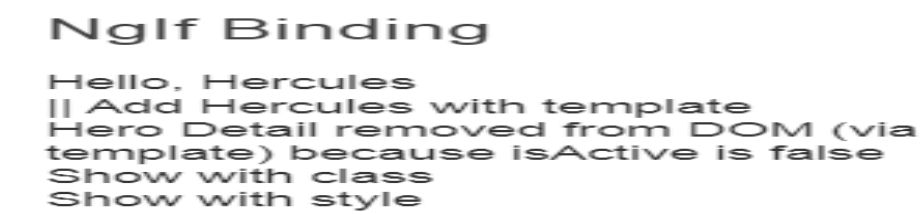
```
<label>special:   <input type="checkbox"
[(ngModel)]="isSpecial"
></label>
<div [ngClass]="isSpecial ? 'special' : ''">This div is special</div>

<div class="bad curly special">Bad curly special</div>
<div [ngClass]="{'bad':false, 'curly':true, 'special':isSpecial}">Curly special</div>

<a class="to-toc" href="#toc">top</a>
```

ngStyle binding

```
currentStyles: {};
  setCurrentStyles() {
    // CSS styles: set per current state of component properties
    this.currentStyles = {
      'font-style':  this.canSave      ? 'italic' : 'normal',
      'font-weight': !this.isUnchanged ? 'bold'   : 'normal',
      'font-size':   this.isSpecial    ? '24px'   : '12px'
    };
  }
```


```
<br>
<label>italic: <input type="checkbox" [(ngModel)]="canSave"></label> |
<label>normal: <input type="checkbox" [(ngModel)]="isUnchanged"></label> |
<label>xlarge: <input type="checkbox" [(ngModel)]="isSpecial"></label>
<button (click)="setCurrentStyles()">Refresh currentStyles</button>
<br><br>
<div [ngStyle]="currentStyles">
  This div should be {{ canSave ? "italic": "plain"}},
                {{ isUnchanged ? "normal weight" : "bold" }} and,
                {{ isSpecial ? "extra large": "normal size"}} after clicking "Refresh".</div>

<a class="to-toc" href="#toc">top</a>
```

Here only ngModel is used for toggle

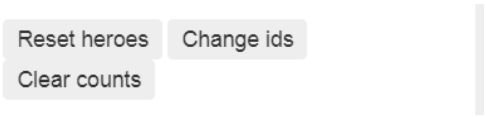



```
<div *ngIf="currentHero">Hello, {{currentHero.name}}</div>||
<div *ngIf="nullHero">Hello, {{nullHero.name}}</div>
<!-- isSpecial is true -->
<div [class.hidden]="!isSpecial">Show with class</div>
<div [class.hidden]="isSpecial">Hide with class</div>
<!-- HeroDetail is in the DOM but hidden -->

<div [style.display]="isSpecial ? 'block' : 'none'">Show with style</div>
<div [style.display]="isSpecial ? 'none'  : 'block'">Hide with style</div>
```

**On **

[style.display]…….[class.hidden]……..*ngIf

```
<div *ngFor="let hero of heroes; let i=index">{{i + 1}} - {{hero.name}}</div>
```




```
resetHeroes() {
    console.log(Hero)
    console.log(Hero.heroes)
    this.heroes = Hero.heroes.map(hero => hero.clone());
    this.currentHero = this.heroes[0];
    this.hero = this.currentHero;
    this.heroesWithTrackByCountReset = 0;
  }
```

without trackBy

(10) Hercules
(21) Dr Nice
(32) Narco
(43) Windstorm
(54) Magneta
Hero DOM elements change #1 without trackBy

```
changeIds() {
    this.resetHeroes();
    this.heroes.forEach(h => h.id += 10 * this.heroIdIncrement++);
    this.heroesWithTrackByCountReset = -1;
  }
```

NgSwitch Binding

Printing radio button values dynamically

```
<div>

  <label *ngFor="let h of heroes">
    <input type="radio" name="heroes" [(ngModel)]="currentHero" [value]="h">{{h.name}}
  </label>
</div>
```

ngSwithch ngSwitchCase

```
<div [ngSwitch]="currentHero.emotion">
  <app-happy-hero    *ngSwitchCase="'happy'"    [hero]="currentHero"></app-happy-hero>
  <app-sad-hero      *ngSwitchCase="'sad'"      [hero]="currentHero"></app-sad-hero>

  <app-confused-hero *ngSwitchCase="'confused'" [hero]="currentHero"></app-confused-hero>
   <div *ngSwitchCase="'confused'">Are you as confused as {{currentHero.name}}?</div>

  <app-unknown-hero  *ngSwitchDefault          [hero]="currentHero"></app-unknown-hero>
</div>
```

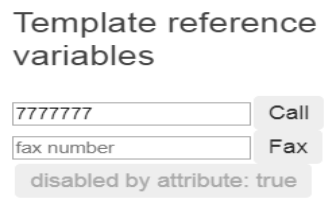
Utilizations of different components in one component

Hero.switch.component

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

@Component({
  selector: 'app-happy-hero',
  template: `Wow. You like {{hero.name}}. What a happy hero ... just like you.`
})
```

```
export class HappyHeroComponent {
  @Input() hero: Hero;
}

@Component({
  selector: 'app-sad-hero',
  template: `You like {{hero.name}}? Such a sad hero. Are you sad too?`
})
export class SadHeroComponent {
  @Input() hero: Hero;
}

@Component({
  selector: 'app-confused-hero',
  template: `Are you as confused as {{hero.name}}?`
})
export class ConfusedHeroComponent {
  @Input() hero: Hero;
}

@Component({
  selector: 'app-unknown-hero',
  template: `{{message}}`
})
export class UnknownHeroComponent {
  @Input() hero: Hero;
  get message() {
    return this.hero && this.hero.name ?
      `${this.hero.name} is strange and mysterious.` :
      'Are you feeling indecisive?';
  }
}

export const heroSwitchComponents =
  [ HappyHeroComponent, SadHeroComponent, ConfusedHeroComponent, UnknownHeroComponent ];
```

Template reference variables



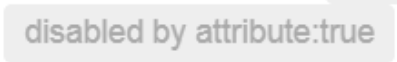
Taken Reference #phone or ref-fax or ref-phone

```
<input #phone placeholder="phone number">
<button (click)="callPhone(phone.value)">Call</button>
```

Or ref-fax

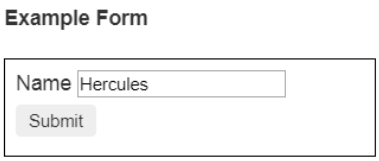
```
<input ref-fax placeholder="fax number">
<button (click)="callFax(fax.value)">Fax</button>
```


```
<button #btn  [innerHTML]="'disabled by attribute:' +btn.disabled"> </button>
```




```
<button #btn disabled [innerHTML]="'disabled by attribute:' +btn.disabled"> </button>
```

Form



HERO.FORM

```
<div id="heroForm">
  <form (ngSubmit)="onSubmit(heroForm)" #heroForm="ngForm">
    <div class="form-group">
      <label for="name">Name
        <input class="form-control" name="name" required
        [(ngModel)]="hero.name">
      </label>
    </div>
    <button type="submit" [disabled]="!heroForm.form.valid">Submit</button>
  </form>
  <div [hidden]="!heroForm.form.valid">
    {{submitMessage}}
  </div>
</div>



<!--
Copyright Google LLC. All Rights Reserved.
Use of this source code is governed by an MIT-style license that
can be found in the LICENSE file at http://angular.io/license
-->
```

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

import { Hero } from './hero';

@Component({
  selector: 'app-hero-form',
  templateUrl: './hero-form.component.html',
  styles: [`
    button { margin: 6px 0; }
    #heroForm { border: 1px solid black; margin: 20px 0; padding: 8px; max-width: 350px; }
  `]
})
export class HeroFormComponent {
  @Input() hero: Hero;
  @ViewChild('heroForm') form: NgForm;

  // tslint:disable-next-line:variable-name
  private _submitMessage = '';

  get submitMessage() {
    if (this.form && !this.form.valid) {
      this._submitMessage = '';
    }
    return this._submitMessage;
  }

  onSubmit(form: NgForm) {
    this._submitMessage =  'Submitted. form value is ' + JSON.stringify(form.value);
  }
}



/*
Copyright Google LLC. All Rights Reserved.
Use of this source code is governed by an MIT-style license that
can be found in the LICENSE file at http://angular.io/license
```

```
*/
```

Pipes

Birthdate: FEBRUARY 25, 1970
Price: $42.00

```
<hr><h2 id="pipes">Pipes</h2>

<div>Title through uppercase pipe: {{title | uppercase}}</div>

<!-- Pipe chaining: convert title to uppercase, then to lowercase -->
<div>
  Title through a pipe chain:
  {{title | uppercase | lowercase}}
</div>

<!-- pipe with configuration argument => "February 25, 1970" -->
<div>Birthdate: {{currentHero?.birthdate | date:'longDate'}}</div>

<div>{{currentHero | json}}</div>

<div>Birthdate: {{(currentHero?.birthdate | date:'longDate') | uppercase}}</div>

<div>
  <!-- pipe price to USD and display the $ symbol -->
  <label>Price: </label>{{product.price | currency:'USD':'symbol'}}
</div>

<a class="to-toc" href="#toc">top</a>
```

Non-null assertion operator *!.*

```
<div>

  <!--No hero, no text -->
  <div *ngIf="hero">
    The hero's name is {{hero!.name}}
  </div>
</div>
```

Non null can avoid error ! I guess

Enums in binding

```
export enum Color {Red, Green, Blue}
Color = Color;
  color = Color.Red;
  colorTwo = Color.Blue;

  colorToggle() {
    this.color = (this.color === Color.Red) ? Color.Blue :  Color.Red;
    this.colorTwo = (this.colorTwo === Color.Blue) ? Color.Red : Color.Blue;
  }
```

```
<!-- enums in bindings -->
<hr><h2 id="enums">Enums in binding</h2>

<p>
  The name of the Color.Red enum is {{Color[Color.Red]}}.<br>
  The current color is {{Color[color]}} and its number is {{color}}.<br>
  <button [style.color]="Color[color]" (click)="colorToggle()">Enum Toggle</button>
  <button [style.color]="Color[colorTwo]" (click)="colorToggle()">Two Enum Toggle</button>
```

```
</p>

<a class="to-toc" href="#toc">top</a>
```

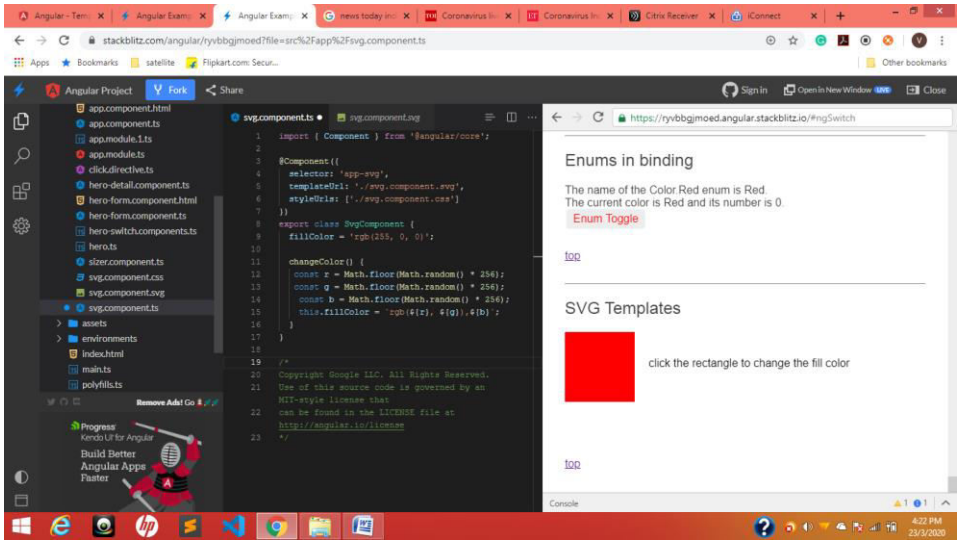
Enums in binding

The name of the Color.Red enum is Red.
The current color is Red and its number is 0.

Enum Toggle

Two Enum Toggle

Svg




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

@Component({
  selector: 'app-svg',
  templateUrl: './svg.component.svg',
  styleUrls: ['./svg.component.css']
})
export class SvgComponent {
  fillColor = 'rgb(255, 0, 0)';

  changeColor() {
   const r = Math.floor(Math.random() * 256);
   const g = Math.floor(Math.random() * 256);
    const b = Math.floor(Math.random() * 256);
    this.fillColor = `rgb(${r}, ${g}),${b}`;
  }
}

/*
Copyright Google LLC. All Rights Reserved.
Use of this source code is governed by an MIT-style license that
can be found in the LICENSE file at http://angular.io/license
*/
```

```
<svg>
  <g>
    <rect x="0" y="0" width="100" height="100" [attr.fill]="fillColor" (click)="changeColor()" />
    <text x="120" y="50">click the rectangle to change the fill color</text>
  </g> s
</svg>
```

Css

```
svg {
  display: block;
  width: 100%;
}
```

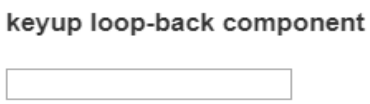
Input events


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

@Component({
  selector: 'app-click-me2',
  template: `
    <button (click)="onClickMe2($event)">No! .. Click me!</button>
    {{clickMessage}}`
})
export class ClickMe2Component {
  clickMessage = '';
  clicks = 1;

  onClickMe2(event: any) {
    let evtMsg = event ? ' Event target is ' + event.target.tagName  : '';
    this.clickMessage = (`Click #${this.clicks++}. ${evtMsg}`);
  }
}
```

(click)="onClickMe2($event)

```
(keyup)="onKey($event)
```

```
 onKey(event: KeyboardEvent) {
    this.values += (event.target as HTMLInputElement).value + ' | ';
  }
}
```




```
<input #box (keyup)="0">
   <p>{{box.value}}</p>
```

Or other

```
<input #box (keyup)="onKey(box.value)">
   <p>{{values}}</p>
```

```
values = '';
  onKey(value: string) {
    this.values += value + ' | ';
  }
```

Enter

```
<input #box (keyup.enter)="onEnter(box.value)">
    <p>{{value}}</p>
```

```
value = '';
  onEnter(value: string) { this.value = value; }
```

Type away! Press [enter] or click elsewhere when done.

```
<input #box
      (keyup.enter)="update(box.value)"
      (blur)="update(box.value)">

    <p>{{value}}</p>
```

```
  value = '';
  update(value: string) { this.value = value; }
```

Add a new hero

Add

- Windstorm
- Bombasto
- Magneta
- Tornado

```
<input #newHero (keyup.enter)="addHero(newHero.value)" (blur)="addHero(newHero.value); newHero.value='' ">

    <button (click)="addHero(newHero.value)">Add</button>

    <ul><li *ngFor="let hero of heroes">{{hero}}</li></ul>
```

```
heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];
  addHero(newHero: string) {
    if (newHero) {
      this.heroes.push(newHero);
    }
  }
```

Component Communication Cookbook

Hero.component

```
export interface Hero {
  name: string;
  id: number;
}

export const HEROES = [
  {name: 'Dr IQ',id:1234},
  {name: 'Magneta'},
  {name: 'Bombasto'}
];
```

component parent

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

import { HEROES } from './hero';

@Component({
  selector: 'app-hero-parent',
  template: `
    <h2>{{master}} controls {{heroes.length}} heroes</h2>
    <app-hero-child

      *ngFor="let hero of heroes"
      [hero]="hero"

      [master]="master" >
    </app-hero-child>
  `
})
export class HeroParentComponent {
  heroes = HEROES;
  master = 'Master';
}
```

child.component

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

import { Hero } from './hero';

@Component({
  selector: 'app-hero-child',
  template: `
    <h3>{{hero.name}} says:</h3>
    <div>
      {{hero.id}}{{hero.name}}
      </div>
    <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
  `
})
export class HeroChildComponent {
  @Input() hero: Hero;

   @Input('master') masterName: string;

}
```

masterName we will assaign without creating a variable

Source code version

ngOnChanges ("Source code version")

ngONChanges

```
/* tslint:disable:forin */
import { Component, Input, OnChanges, SimpleChange } from '@angular/core';

@Component({
  selector: 'app-version-child',
  template: `
    <h3>Version {{major}}.{{minor}}</h3>
    <h4>Change log:</h4>
    <ul>
      <li *ngFor="let change of changeLog">{{change}}</li>
    </ul>
  `
})
export class VersionChildComponent implements OnChanges {
  @Input() major: number;
  @Input() minor: number;
  changeLog: string[] = [];

  ngOnChanges(changes: {[propKey: string]: SimpleChange}) {
    let log: string[] = [];
    for (let propName in changes) {
      let changedProp = changes[propName];
      //console.log(propName)
      console.log(changedProp);
      let to = JSON.stringify(changedProp.currentValue);
      if (changedProp.isFirstChange()) {
        log.push(`Initial value of ${propName} set to ${to}`);
      } else {
        let from = JSON.stringify(changedProp.previousValue);
        log.push(`${propName} changed from ${from} to ${to}`);
      }
    }
    this.changeLog.push(log.join(', '));
  }
}
```

SimpleChange will identify the changes

Parent listens for child event ("Colonize Universe")

Should mankind colonize the Universe?

Should mankind colonize the Universe?

Agree: 1, Disagree: 0

Narco

Agree Disagree

Celeritas

Agree Disagree

Bombasto

Agree Disagree

Back to Top

Child

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

@Component({
```

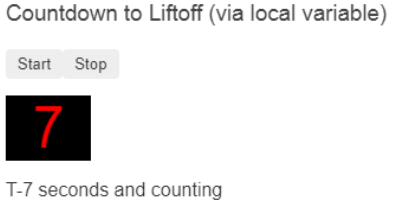
```
  selector: 'app-voter',
  template: `
    <h4>{{name}}</h4>
    <button (click)="vote(true)"  [disabled]="didVote">Agree</button>
    <button (click)="vote(false)" [disabled]="didVote">Disagree</button>
  `
})
export class VoterComponent {
  @Input()  name: string;
  @Output() voted = new EventEmitter<boolean>();
  didVote = false;

  vote(agreed: boolean) {
    this.voted.emit(agreed);
    this.didVote = true;
  }
}
```

Parent

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

@Component({
  selector: 'app-vote-taker',
  template: `
    <h2>Should mankind colonize the Universe?</h2>
    <h3>Agree: {{agreed}}, Disagree: {{disagreed}}</h3>
    <app-voter *ngFor="let voter of voters"
      [name]="voter"
      (voted)="onVoted($event)">
    </app-voter>
  `
})
export class VoteTakerComponent {
  agreed = 0;
  disagreed = 0;
  voters = ['Narco', 'Celeritas', 'Bombasto'];

  onVoted(agreed: boolean) {
    agreed ? this.agreed++ : this.disagreed++;
  }
}
```

Parent to child via *local variable*("Countdown to Liftoff")

Parent calls *ViewChild*("Countdown to Liftoff")



How to call a **child component** method in **parent component @viewchild**

```
  @ViewChild(CountdownTimerComponent)
  private timerComponent: CountdownTimerComponent;
```

```
start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
```

```
import { AfterViewInit, ViewChild } from '@angular/core';
import { Component }                from '@angular/core';
import { CountdownTimerComponent }  from './countdown-timer.component';



//// Local variable, #timer, version
@Component({
  selector: 'app-countdown-parent-lv',
  template: `
  <h3>Countdown to Liftoff (via local variable)</h3>
  <button (click)="timer.start()">Start</button>
  <button (click)="timer.stop()">Stop</button>
  <div class="seconds">{{timer.seconds}}</div>
  <app-countdown-timer #timer></app-countdown-timer>
  `,
  styleUrls: ['../assets/demo.css']
})
export class CountdownLocalVarParentComponent { }

//// View Child version
@Component({
  selector: 'app-countdown-parent-vc',
  template: `
  <h3>Countdown to Liftoff (via ViewChild)</h3>
  <button (click)="start()">Start</button>
  <button (click)="stop()">Stop</button>
  <div class="seconds">{{ seconds() }}</div>
  <app-countdown-timer></app-countdown-timer>
  `,
  styleUrls: ['../assets/demo.css']
})
export class CountdownViewChildParentComponent implements AfterViewInit {

  @ViewChild(CountdownTimerComponent)
  private timerComponent: CountdownTimerComponent;

  seconds() { return 0; }

  ngAfterViewInit() {
    // Redefine `seconds()` to get from the `CountdownTimerComponent.seconds` ...
    // but wait a tick first to avoid one-time devMode
    // unidirectional-data-flow-violation error
    setTimeout(() => this.seconds = () => this.timerComponent.seconds, 0);
  }

  start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
}



/*
Copyright Google LLC. All Rights Reserved.
Use of this source code is governed by an MIT-style license that
can be found in the LICENSE file at http://angular.io/license
*/
```

Child

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

@Component({
  selector: 'app-countdown-timer',
  template: '<p>{{message}}</p>'
})
export class CountdownTimerComponent implements OnInit, OnDestroy {

  intervalId = 0;
```

```
  message = '';
  seconds = 11;

  clearTimer() { clearInterval(this.intervalId); }

  ngOnInit()    { this.start(); }
  ngOnDestroy() { this.clearTimer(); }

  start() { this.countDown(); }
  stop()  {
    this.clearTimer();
    this.message = `Holding at T-${this.seconds} seconds`;
  }

  private countDown() {
    this.clearTimer();
    this.intervalId = window.setInterval(() => {
      this.seconds -= 1;
      if (this.seconds === 0) {
        this.message = 'Blast off!';
      } else {
        if (this.seconds < 0) { this.seconds = 10; } // reset
        this.message = `T-${this.seconds} seconds and counting`;
      }
    }, 1000);
  }
}
```