Welcome

Wifi Password: **password**

If you have not completed your setup:
github.com/blove/angular-fundamentals

Brian Love

brian_love

Portland
Oregon
OLD TOWN

Google Developers Expert
Angular + Web Technologies

Housekeeping

Housekeeping

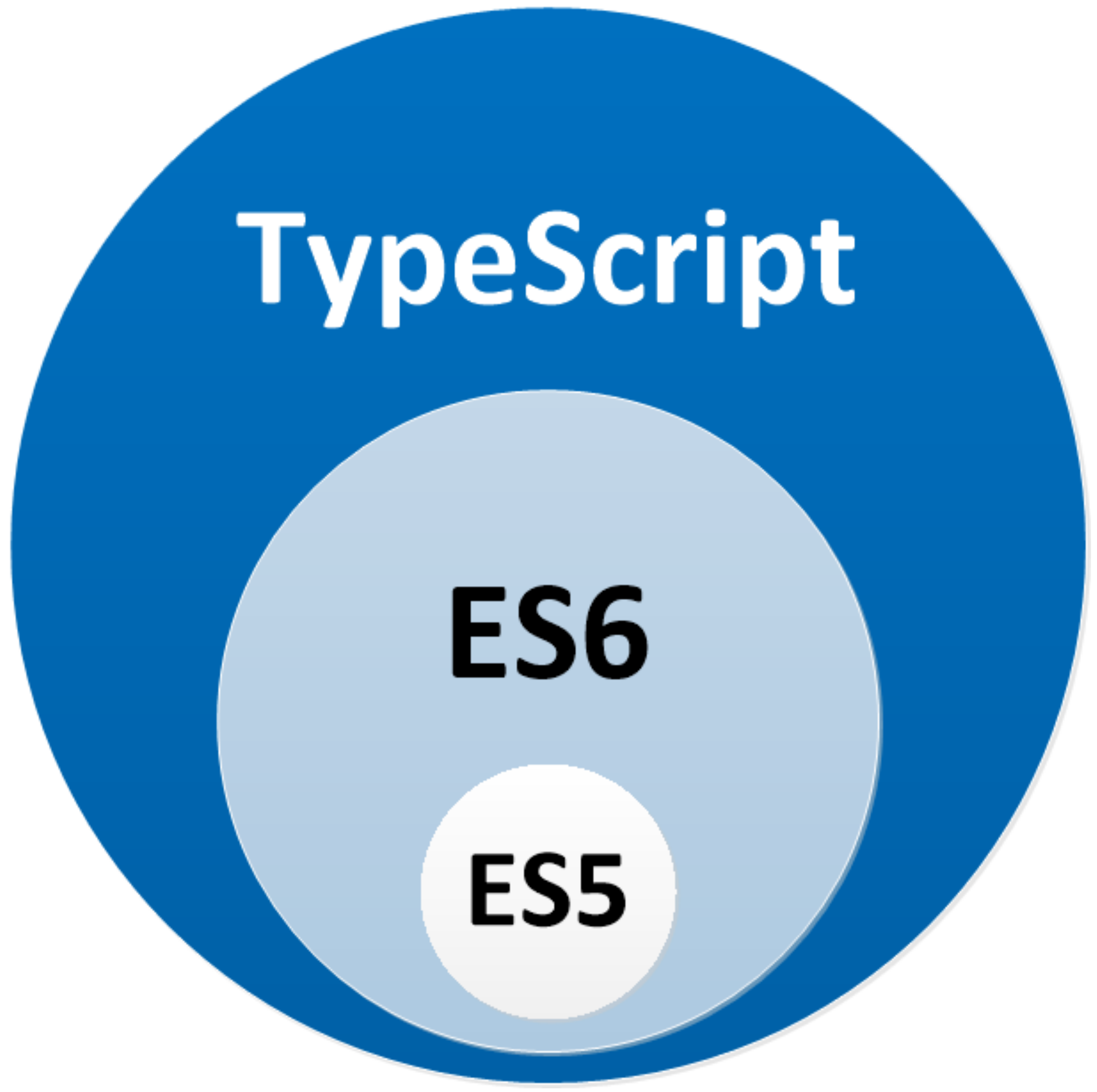
- Code of Conduct
- 20-minute break in the morning and afternoon
- 1-hour break for lunch
- Raise your hand if you have a question
- Be helpful to your neighbors
- Please silence phones
- It’s absolutely ok to step outside to take a call, etc.

Getting to Know You

- Who has JavaScript experience?
- Who has TypeScript experience?
- Who has Angular experience?
- Who has React (component architecture) experience?

TypeScript Essentials	Project Structure	Foundational Concepts	Templates & Components	CLI	Pipes
Forms	RxJS Essentials	Services	HTTP	Routing	Directives

TypeScript Essentials



Superset of JavaScript

Basic Types

```
let x: number;
let name: string;
let isActive: boolean;
let winningNumbers: Array<number>;
let losingNumbers: number[];

const y = 1;
const isAdmin = false;

let z: number | null = null;
z = 2;
```

Basic Types Exercise

⚡ https://stackblitz.com/fork/ngfs-basic-types

1. Go to index.ts
2. Follow directions in the comment at the top

Function Types

```
function square(base: number): number {
  return Math.pow(base, 2);
}

let s = square(2)
console.log(s);
```

Function Types Exercise

⚡ https://stackblitz.com/fork/ngfs-function-types

1. Go to index.ts
2. Follow directions in the comment at the top

Interfaces

```
interface User {
  firstName: string;
  lastName: string;
  fullName(): string;
  greet(): void;
}
```

Interfaces Exercise

⚡ https://stackblitz.com/fork/ngfs-interfaces

1. Go to index.ts
2. Refactor the `user` argument to the `login()` function to be a `User` interface
3. Fix the issue in the `verifyPhone()` function

Exports

```
export interface User {
  firstName: string;
  lastName: string;
  fullName(): string;
  greet(): void;
}
```

Imports

```
import { User } from './user';

const users: User[];

login(user: User): boolean;
```

Exports and Imports Exercise

⚡ https://stackblitz.com/fork/ngfs-exports-imports

1. Go to index.ts file
2. Follow directions in the comment at the top

Classes: properties

```
class Administrator implements User {
  firstName: string;
  lastName: string;

  constructor(firstName: string, lastName: string) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
}
```

Classes: constructor property assignment

```
class Administrator implements User {
  constructor(
    public firstName: string,
    public lastName: string
  ) {}
}
```

Classes: methods

```
class Administrator implements User {
  // code omitted

  fullName(): string {
    return `${this.firstName} ${this.lastName}`;
  }

  greet(): void {
    console.log(`Hi ${this.fullName()}`);
  }
}
```

Classes: new up

```
let brian: Administrator;
brian = new Administrator('Brian', 'Love');
```

Classes Exercise

⚡ https://stackblitz.com/fork/ngfs-classes

1. Go to the donut.interface.ts file and note the `Donut` interface shape
2. Go to index.ts
3. Follow directions in comment at the top
4. If you have time, try the extra credit steps

Spread Operator: Object

```
let user = {
  firstName: 'Brian',
  lastName: 'Love'
};

// object mutation
user['twitter'] = 'brian_love';

// spread operator
newUser = {
  ...user,
  twitter: 'brian_love'
};
```

Spread Operator: Object Exercise

⚡ https://stackblitz.com/fork/ngfs-spread-operator-object

1. Go to index.ts file
2. Follow directions in the comment at the top

Spread Operator: Array

```
let films = ['The Force Awakens', 'The Last Jedi'];

// array mutation
films.push('The Rise of Skywalker');

// spread operator
updatedFilms = [
  ...films,
  'The Rise of Skywalker'
];
```

Spread Operator: Array Exercise

⚡ https://stackblitz.com/fork/ngfs-spread-operator-array

1. Go to index.ts file
2. Follow directions in the comment at the top

Project Structure

```
➜  git clone https://github.com/blove/angular-fundamentals.git
➜  cd angular-fundamentals
➜  npm install
➜  npm start
```

Project Structure

```
├── angular.json
├── package.json
├── src
|   ├── app
|   |   ├── app.component.spec.ts
|   |   ├── app.component.ts
|   |   └── app.module.ts
|   ├── assets
|   ├── browserslist
|   ├── environments
|   |   ├── environment.prod.ts
|   |   └── environment.ts
|   ├── favicon.ico
|   ├── index.html
|   ├── main.ts
|   ├── polyfills.ts
|   ├── styles.css
|   ├── tsconfig.app.json
|   ├── tsconfig.spec.json
|   └── tslint.json
├── tsconfig.json
└── tslint.json
```

Foundational Concepts

Component Metadata

Foundational Concepts

Component Metadata

```
@Component({
  selector: 'app-root',
  template: `
    <h1>Welcome to the Angular Fundamentals Workshop!</h1>
  `
})
export class AppComponent {}
```

Component Metadata

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {}
```

Template Strings

Foundational Concepts

Template Strings

```
@Component({
  selector: 'app-root',
  template: `
    <h1>{{ name }}</h1>
    <p>Age: {{ age - 10 }} 😜</p>
    <p>Follow me: {{ twitter }}</p>
    <p>
      Enjoys knitting?
      {{ likesKnitting ? 'sweeeet' : ‘meh' }}
    </p>
  `
})
export class AppComponent {}
```

Helpful Hints

- If the `ng serve` process is still running use:
  - `ctrl+c` on Windows
  - `cmd+c` on macOS
- Checkout a branch via: `git checkout`
- Stash (set aside) changes via: `git stash`

```
➜  git stash
➜  git checkout 1-template-strings
➜  npm start
```

Template Strings Exercise

- Go to app.component.ts
- Add properties:
    - Favorite food
    - Favorite place to go on vacation
    - The name of your best friend
- Update template to output the new properties

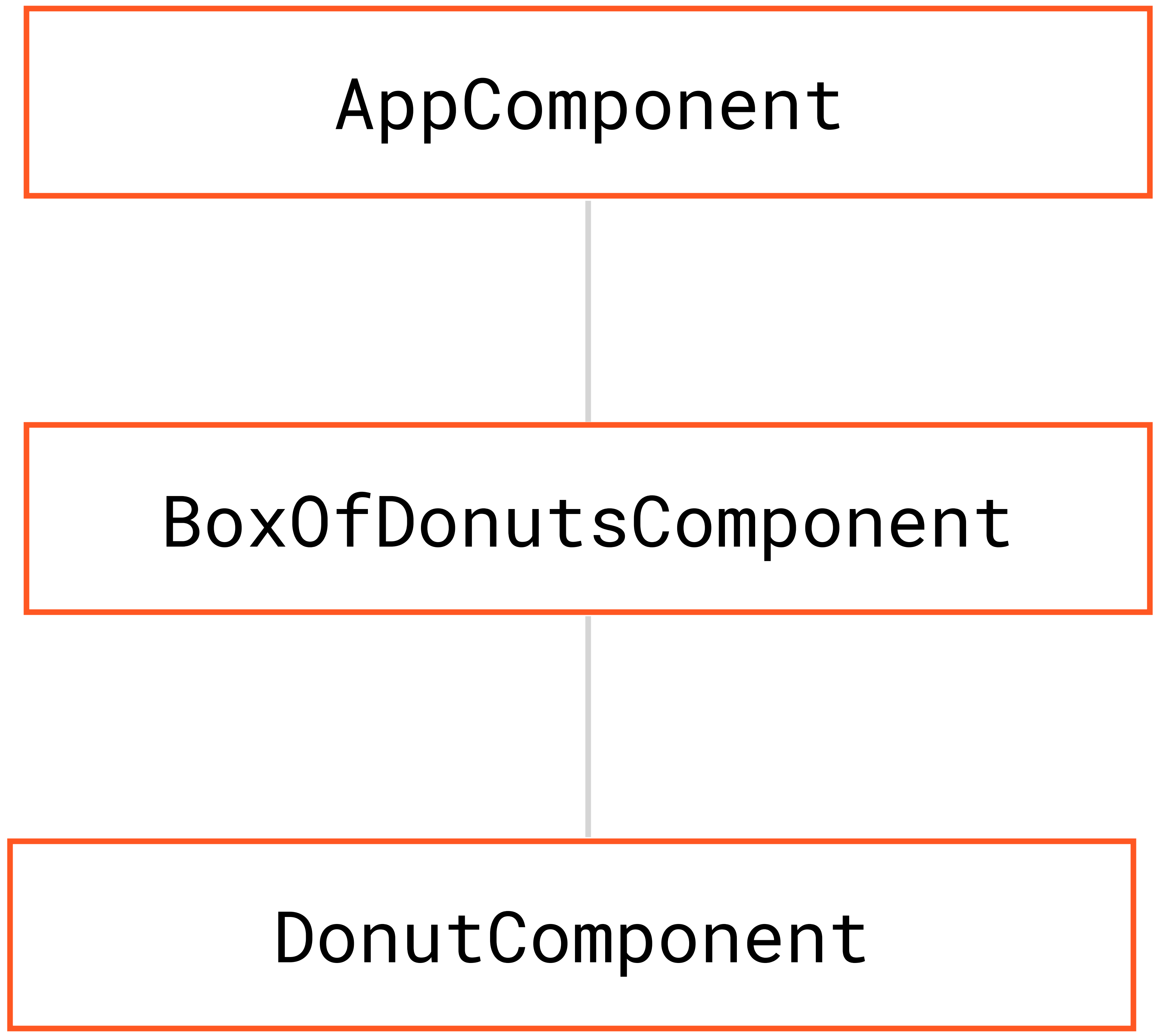
Component Tree & Property Binding

Foundational Concepts

Component Tree

Root Component
Child Component
Child Component
Grandchild Component
Grandchild Component
Grandchild Component

Component Tree



Property Binding

- One-way communication from parent to child component
- Simple or complex data
- Set properties of an `HTMLElement`

Property Binding

```
@Component({
  template: `
    <app-donut [donut]="donut"></app-donut>
  `
})
export class BoxOfDonutsComponent {
  donut: Donut = { name: 'Chocolate glazed' };
}
```

Property Binding

```
import { Component, Input } from '@angular/core';
import { Donut } from './models/donut.interface';

@Component({
  selector: 'app-donut',
  template: `
    <p>Donut name: {{ donut.name }}</p>
  `
})
export class DonutComponent {
  @Input() donut: Donut;
}
```

Component Tree Exercise

⚡ git checkout 2-component-tree

1. Add `icing: boolean` property to donut.component.ts
2. Display `icing` value in the donut template
3. Add `size: number` input to box-of-donuts.component.ts
4. Display the `size` value in the box-of-donuts' template
5. In app.component.ts specify `size` input binding into the `<box-of-donuts>` component

Structural Directives

Foundational Concepts

Structural Directives

- Modify the DOM **structure** by adding, removing, or updating elements
- Shorthand syntax is prefixed by an asterisk (*)
- Built-in structural directives:
    - NgIf
    - NgFor
    - NgSwitch

NgIf Structural Directive

```
@Component({
  template: `
    <p>
      Donut name: {{ donut.name }}
      <span *ngIf="icing">with icing</span>
    </p>
  `
})
export class DonutComponent {
  @Input() donut: Donut;

  @Input() icing: boolean;
}
```

NgIf Exercise

⚡ git checkout 3-structural-directives

1. Update the size paragraph in box-of-donuts.component.ts, to show:
   1. “small box” when 4 or fewer
   2. “box” when 5 to 7
   3. “large box” when 8 or more logic to box-of-donuts
2. In box-of-donuts.component.ts add a new paragraph with the text “Box Is Full” when the length of the `donuts` array equals the `size` of the box.

NgFor Structural Directive

```
@Component({
  template: `
    <app-donut *ngFor="let donut of donuts" [donut]="donut"></app-donut>
  `
})
export class BoxOfDonutsComponent {
  donuts: Donut[] = [
    { name: 'Bacon glazed', icing: true },
    { name: ‘Sriracha Infused', icing: false }
  ];
}
```

NgFor Demo

⚡ git checkout 3-structural-directives

1. Open box-of-donuts.component.ts
2. Use the `NgFor` structural directive to iterate over the `array` of donuts

NgFor Exercise

⚡ git checkout 3-structural-directives

1. Define an array of numbers on app-root (1-10)
2. Use `NgFor` to iterate over the array
3. Add an `NgIf` to the `NgFor`, if the number is even, make the number bold
4. Extra credit: open box-of-donuts.component.ts and use the `NgFor` structural directive to iterate over the `donuts`

HTMLElement Property Binding

Foundational Concepts

`HTMLElement` Property Binding

```
<button [disabled]="btnDisabled"></button>
```

`HTMLElement` Property Binding

```
@Component({
  template: `
    <button [disabled]="btnDisabled">Save Changes</button>
  `
})
export class AppComponent {
  btnDisabled = true;

  toggle(): void {
    this.btnDisabled = !this.btnDisabled;
  }
}
```

`HTMLElement` Property Binding

```
<textarea [rows]="numRows" [cols]="numCols"></textarea>

<img [src]="user.avatarUrl" />

<div class="content" [hidden]="showContent">...</div>

<div [id]="user.id>...</div>
```

`HTMLElement` Property Binding Exercise

⚡ git checkout 4-property-bindings

1. Open donut.component.ts and add a new `img` element. Use an attribute binding to set the `src` attribute to the `fileName` property on the `donut` object.
2. Add an attribute binding to set the `alt` attribute to the `name` property on the `donut` object.
3. Hide the icing label if the `donut` has a `fileName`

Event Binding

Foundational Concepts

Event Binding

```
<button (click)="onSave()">Save Changes</button>
```

Event Binding: Event Object

```
<button (click)="onSave($event)">Save Changes</button>
```

Event Binding: Event Object

```
@Component({
  template: `
    <button (click)="onSave($event)">Save Changes</button>
  `
})
export class AppComponent {
  onSave(event: MouseEvent): void {
    console.log('Save button clicked.');
  }
}
```

Event Binding Exercise

⚡ git checkout 5-event-bindings

1. Open box-of-donuts.components.ts and add a button to the template that will toggle the `showName` boolean property value
2. Open donut.component.ts and add a `showName` property that is an input
3. Open box-of-donuts.component.ts and add an input binding for `showName` to the `<app-donut>` element
4. Open donut.component.ts and hide the name using a `hidden` attribute binding when `showName` is `false`

Styling

Styling

```
.btn {
  font-size: 18px;
  background: #dfe;
}
```

Styling: URL

```
@Component({
  template: `
    <button class="btn">Save Changes</button>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  ...
}
```

Styling: Inline

```
@Component({
  template: `
    <button class="btn">Save Changes</button>
  `,
  styles: [
    `
      .btn {
        font-size: 18px;
      }
    `
  ]
})
export class AppComponent { }
```

Styling: Host Pseudo Selector

```css
:host {
  display: flex;
  justify-content: stretch;
}
```

Styles Exercise

⚡ git checkout 6-styles

1. In styles.css copy styles for box-of-donuts and paste inline into the box-of-donuts.component.ts styles property in the component metadata
2. Open donut.component.ts and add a `styles: []` property to the component metadata
3. Copy styles from styles.css into the `styles` property in donut.component.ts
4. Copy styles from styles.css into app.component.css. Use the `:host` pseudo selector

Style Binding

Styling

Style Binding: Style Property

```
@Component({
  selector: 'app-root',
  template: `
    <div class="label" [style.font-weight]="fontWeight">
      Jeans with rainbow frosted flair
    </div>
  `
})
export class AppComponent {
  fontWeight: string | number;
}
```

Style Binding: Style Property

```
@Component({
  selector: 'app-root',
  template: `
    <div class="progress-bar">
      <div class="bar" [style.width.%]="progress"></div>
    </div>
  `
})
export class AppComponent {
  progress = 25;
}
```

Style Binding: NgStyle Directive

```
@Component({
  template: `
    <div
      class="label"
      [ngStyle]="{ 'fontWeight': fontWeight; color: color }"
    >
      Jeans with rainbow frosted flair
    </div>
  `
})
export class AppComponent {
  color = '#999';
  fontWeight = bold;
}
```

Style Binding

```
@Component({
  template: `
    <div class="progress-bar">
      <div class="bar" [ngStyle]="{ 'width.%': progress }"></div>
    </div>
    <img
      [src]="src"
      [ngStyle]="{ 'max-height.px': size, 'max-width.px': size }"
    />
  `
})
export class AppComponent {
  progress = 25;
  size: number;
  src: string;
}
```

Style Bindings

⚡ git checkout 6-styles

1. Open donut.component.ts and add the `NgStyle` directive to the `div.name` element, specifying the `font-weight` of `bold` when the donut name includes “chocolate”

Hint: use the following regular expression to test the donut name:
`/chocolate/i.test(donut.name)`

Class Binding

Styling

Class Binding: Class Attribute

```
@Component({
  template: `
    <button [class.active]="isActive">Save Changes</button>
  `
})
export class AppComponent {
  isActive = false;

  toggleActive(): void {
    this.isActive = !this.isActive;
  }
}
```

Class Binding: NgClass Directive

```
@Component({
  template: `
    <button [ngClass]="{ active: isActive }">Save Changes</button>
  `
})
export class AppComponent {
  isActive = false;

  toggleActive(): void {
    this.isActive = !this.isActive;
  }
}
```

Class Binding Exercise

⚡ git checkout 6-styles

1. Open box-of-donuts.component.ts
2. Add a `selectedDonut: Donut` property
3. Add a `click` event binding to the app-donut element and set the value of the `selectedDonut`
4. Add a `selected` class using a class attribute binding to `<app-donut>` when the current `donut` is strictly equal to the `selectedDonut`
5. Now, add the `selected` class using the `NgClass` Directive

View Encapsulation

Styling

View Encapsulation

- Emulated (default)
- None
- ShadowDom

Custom Event Binding

Custom Event Binding

```
@Component({
  selector: 'app-options'
})
export class OptionsComponent {
  @Output() clear = new EventEmitter();
  @Output() optionChange = new EventEmitter<Option>();
  @Output() save = new EventEmitter<Option[]>();
}
```

Custom Event Binding

```
@Component({
  selector: 'app-root',
  template: `
    <app-options
      (clear)="onClear()"
      (optionChange)="onOptionChange($event);
      (save)="onSave($event);
    ></app-options>
  `
})
export class OptionsComponent {
  onClear() {}
  onOptionChange(option: Option) {}
  onSave(options: Option[]) {}
}
```

Custom Event Binding Exercise

⚡ git checkout 7-custom-event-bindings

1. Open donut-wall.component.ts and add a custom `selected` event emitter
2. Add a button to the template to emit the event
3. Open app.component.ts and add the `selected` output binding to the `<app-wall-of-donuts>` element, invoking the `onSelected()` method
4. Implement the `onSelected()` method that accepts a `Donut` object

Custom Event Binding Exercise 2

⚡ git checkout 7-custom-event-bindings

1. Open box-of-donuts.component.ts and add a custom `remove` event emitter that emits the `Donut` type
2. Add a button to the template to emit the event
3. Open app.component.ts and add the `remove` output binding to the `<app-box-of-donuts>` element, invoking the `onRemove()` method
4. Implement the `onRemove()` method that accepts a `Donut` object

Component Lifecycle Hooks

Component Lifecycle

- Input binding changes: `ngOnChanges()`
- Initialization: `ngOnInit()`
- Destroy: `ngOnDestroy()`

* There are more, but let’s focus on these three

Component Lifecycle Hooks: `ngOnChanges()`

```
@Component({
  selector: 'app-root'
})
export class AppComponent implements OnChanges {
  ngOnChanges(simpleChanges: SimpleChanges) {
    // todo: check if donut input has changed
  }
}
```

Component Lifecycle Hooks: `ngOnChanges()`

```
export interface SimpleChanges {
    [propName: string]: SimpleChange;
}

export declare class SimpleChange {
    previousValue: any;
    currentValue: any;
    firstChange: boolean;
    isFirstChange(): boolean;
}
```

Component Lifecycle Hooks: `ngOnChanges()`

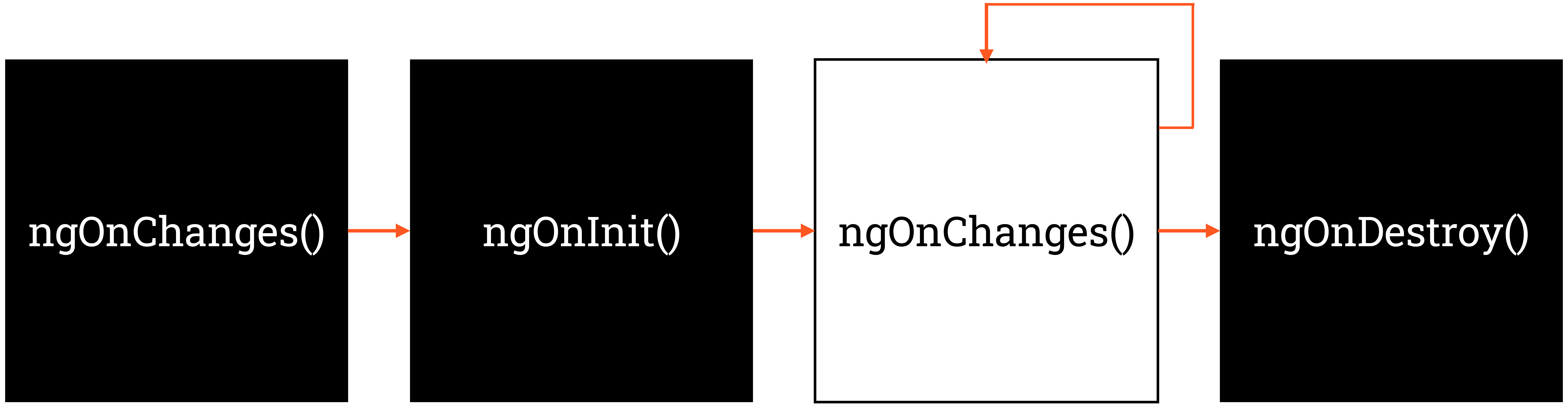
```
@Component({
  selector: 'app-root'
})
export class AppComponent implements OnChanges {
  ngOnChanges(simpleChanges: SimpleChanges) {
    if (
      simpleChanges.donut
      simpleChanges.donut && simpleChanges.donut.currentValue
      simpleChanges.donut.currentValue !== simpleChanges.donut.previousValue
    ) {
      ...
    }
  }
}
```

Component Lifecycle Hooks: `ngOnInit()`

```
@Component({
  selector: 'app-root'
})
export class AppComponent implements OnInit {
  ngOnInit() {
    // initialization code
  }
}
```

Component Lifecycle Hooks: `ngOnDestroy()`

```
@Component({
  selector: 'app-root'
})
export class AppComponent implements OnDestroy {
  ngOnDestroy() {
    // teardown code
  }
}
```


ngOnChanges()
ngOnInit()
ngOnChanges()
ngOnDestroy()

Component Lifecycle Hooks

⚡ git checkout 8-component-lifecycle-hooks

1. Open box-of-donuts.component.ts and implement the `OnChanges` interface
2. Declare the `ngOnChanges()` lifecycle method and verify that the number of donuts in the `box` does not exceed the `size`

Hint: use `this.donuts.slice(0, this.size)` to limit the `donuts` to the specified `size`

Built-in Pipes

Built-in Pipes: Titlecase

```
@Component({
  selector: 'app-root',
  template: `
    {{ firstName | titlecase }} {{ lastName | titlecase }}
  `
})
export class AppComponent {
  firstName = 'Brian';
  lastName = 'Love';
}
```

Built-in Pipes: Uppercase

```
@Component({
  selector: 'app-root',
  template: `
    {{ warning | uppercase }}
  `
})
export class AppComponent {
  warning: string;
}
```

Built-in Pipes: Number

```
@Component({
  selector: 'app-root',
  template: `
    {{ total | number: '2.1' }}
  `
})
export class AppComponent {
  total = '5'; // format to `05.1`
}
```

Built-in Pipes: JSON

```
@Component({
  selector: 'app-root',
  template: `
    <pre>{{ person | json }}</pre>
  `
})
export class AppComponent {
  person: Person
}
```

Built-in Pipes: Date

```
@Component({
  selector: 'app-root',
  template: `
    {{ today | date:'short' }}
    {{ today | date:'full' }}
    {{ today | date:'MMMM d, y' }}
  `
})
export class AppComponent {
  today: Date
}
```

Built-in Pipes: Currency

```
@Component({
  selector: 'app-root',
  template: `
    {{ total | currency }}
    {{ total | currency:'EUR' }}
    {{ total | currency:'EUR':'code' }}
  `
})
export class AppComponent {
  total = 1234.567;
}
```

Built-in Pipes Exercise

⚡ git checkout 9-built-in-pipes

1. Open donut.component.ts and use the `titlecase` pipe for the `name`
2. Add the `price` of a donut using the `number` pipe
3. Now, use the `currency` pipe
4. Try specifying alternate currencies and symbols

Template-driven Forms

Forms: `FormsModule`

```
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';

@NgModule({
  imports: [BrowserModule, FormsModule],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Forms: `NgModel`

```
@Component({
  selector: 'app-root',
  template: `
    <input [(ngModel)]="firstName" />
    <input [(ngModel)]="lastName" />
    Hi {{ firstName}} {{ lastName }} 👋
  `
})
export class AppComponent {
  firstName = 'Brian';
  lastName = 'Love';
}
```

Forms Exercise 1

⚡ git checkout 10-template-driven-forms

1. Open app.module.ts and import the `FormsModule`
2. Open box-of-donuts.component.ts and add a `name` input property
3. Output the customer `name` in the `div.heading` element
4. Open app.component.ts and add a `name` property, bind to the `name` property on the `<app-box-of-donuts>` element, and add an input with the `NgModel` directive

Forms: Key Classes

FormGroup

FormControl

```
@Component({
  sel                        rm',
  tem
    <form>
      <input name="name" [(ngModel)]="donut.name" />
      <input name="price" [(ngModel)]="donut.price" />
    </form>
  `
})
export class DonutFormComponent {
  donut: Donut
}
```

Forms: `NgForm` Directive

```
@Component({
  selector: 'app-donut-form',
  template: `
    <form #donutForm="ngForm">
      <input name="name" [(ngModel)]="donut.name" required />
      <input name="price" [(ngModel)]="donut.price" required />
      <p>Valid? {{ donutForm.valid }}</p>
      <pre>{{ donutForm.value | json }}</pre>
    </form>
  `
})
export class DonutFormComponent {
  donut: Donut
}
```

Forms Exercise 2

⚡ git checkout 10-template-driven-forms

1. Open donut-wall.component.ts and add an `edit` custom event
2. Add a button to the template that emits the `edit` event when clicked
3. Open app.component.ts and add the output binding for the `edit` event to the `<app-donut-wall>` element
4. Add a `donut` property that is a reference to the donut being edited
5. Create the form to edit the donut’s `name` and `price` values
6. Check if the form is valid

Forms: `FormControl` State

State	FALSE	TRUE
Visited or Touched	`ng-untouched`	`ng-touched`
Value Changed	`ng-dirty`	`ng-pristine`
Value is Valid	`ng-invalid`	`ng-valid`

Forms Exercise 3

⚡ git checkout 10-template-driven-forms

1. Open app.component.css and add styles to modify the input element’s border for the following classes:
    1. `ng-valid`: green border
    2. `ng-invalid`: red border
2. Only apply the styles when the input is required
3. Show an alert message when the form is invalid.
4. Hide the alert when the form is valid

Forms: `NgSubmit` Event

```
@Component({
  selector: 'app-donut-form',
  template: `
    <form #donutForm="ngForm" (ngSubmit)="onSubmit(donutForm)">
      <input name="name" [(ngModel)]="donut.name" required />
      <input name="price" [(ngModel)]="donut.price" required />
      <p>Valid? {{ donutForm.valid }}</p>
      <pre>{{ donutForm.value | json }}</pre>
    </form>
  `
})
export class DonutFormComponent {
  donut: Donut

  onSubmit(donutForm: NgForm) {}
}
```

Forms Exercise 4

⚡ git checkout 10-template-driven-forms

1. Open app.component.ts and add the `ngSubmit` output binding to the form element
2. Define an `onSubmit()` method that accepts the `NgForm` instance
3. Log if the form is `valid` and the form's `value` in the console

Reactive Forms

“Why are there two form implementations in Angular?”

–Angular Developer Susie

Template-driven Forms	Reactive Forms
Abstracts away complex API	Uses robust APIs
Defined via directives	Defined explicitly in the component class
Validated via directives	Validated via functions
Relies on mutability	Relies on immutability

Forms: `ReactiveFormsModule`

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';

@NgModule({
  imports: [BrowserModule, FormsModule, ReactiveFormsModule],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Forms: `FormControl`

```
@Component({
  selector: 'app-donut-form',
  template: `
    <input [formControl]="name" />
  `
})
export class DonutFormComponent {
  name = new FormControl('');
}
```

Forms: `FormGroup`

```
@Component({
  selector: 'app-donut-form',
  template: `
    <form [formGroup]="donutFormGroup">
      <input formControlName="name" />
      <input formControlName="price" />
    </form>
  `
})
export class DonutFormComponent {
  donutFormGroup = new FormGroup({
    name: new FormControl('Chili Eclair'),
    price: new FormControl('0.5')
  });
}
```

Forms: `FormControl`

```
@Component({
  selector: 'app-donut-form',
  template: `
    <form [formGroup]="donutFormGroup">
      <input formControlName="name" />
      <input formControlName="price" />
    </form>
  `
})
export class DonutFormComponent {
  onEdit(donut: Donut): void {
    this.donutFormGroup.setValue(donut);
  }
}
```

Forms: `FormControl`

```
@Component({
  selector: 'app-donut-form',
  template: `
    <form [formGroup]="donutFormGroup">
      <input formControlName="name" />
      <input formControlName="price" />
    </form>
  `
})
export class DonutFormComponent {
  onEdit(partOfDonut: Partial<Donut>): void {
    this.donutFormGroup.patchValue(partOfDonut);
  }
}
```

"What is the difference between `setValue()` and `patchValue()`?"

–Angular Developer Susie

Forms: `Validators`

```
@Component({
  selector: 'app-donut-form',
  template: `
    <form [formGroup]="donutFormGroup">
      <input formControlName="name" />
      <input formControlName="price" />
    </form>
  `
})
export class DonutFormComponent {
  donutFormGroup = new FormGroup({
    name: new FormControl('Chili Eclair', Validators.required),
    price: new FormControl('0.5', [Validators.required,
Validators.min(0.5)])
  });
}
```

Forms: Validators

```
export declare class Validators {
    static min(min: number): ValidatorFn;
    static max(max: number): ValidatorFn;
    static required(control: AbstractControl): ValidationErrors | null;
    static requiredTrue(control: AbstractControl): ValidationErrors | null;
    static email(control: AbstractControl): ValidationErrors | null;
    static minLength(minLength: number): ValidatorFn;
    static maxLength(maxLength: number): ValidatorFn;
    static pattern(pattern: string | RegExp): ValidatorFn;
}
```

Forms: `NgSubmit`

```
@Component({
  selector: 'app-donut-form',
  template: `
    <form [formGroup]="donutFormGroup" (ngSubmit)="onSubmit()">
      <input formControlName="name" />
      <input formControlName="price" />
      <button type="submit" [disabled]="!donutForm.valid">Fry it!</button>
    </form>
  `
})
export class DonutFormComponent {
  onSubmit() {
    if (!this.donutFormGroup.valid) { return; }
    this.donutService.save(this.donutFormGroup.value);
  }
}
```

Forms Exercise 5

⚡ git checkout 11-reactive-forms

1. Open app.module.ts and import the `ReactiveFormsModule`
2. Open app.component.ts and modify the template-driven form to be a reactive form

Hint: You'll need to update the `edit` output binding on the `<app-donut-wall>` element to invoke an `onEdit()` method that sets the `value` of the `FormGroup`

Forms: `FormBuilder`

```
@Component({
  selector: 'app-donut-form',
  template: `
    <input [formControl]="name" />
    <input [formControl]="price" />
  `
})
export class DonutFormComponent {
  name = this.formBuilder.control('Fire and Ice', Validators.required);
  price = this.formBuilder.control('0.5', [
    Validators.required,
    Validators.min(0.5)
  ]);

  constructor(private readonly formBuilder: FormBuilder) {}
}
```

Forms: `FormBuilder`

```
@Component({
  template: `
    <form [formGroup]="donutFormGroup" (ngSubmit)="onSubmit()">
      <input formControlName="name" />
      <input formControlName="price" />
    </form>
  `
})
export class DonutFormComponent {
  donutFormGroup = this.formBuilder.group({
    name: ['Hot Chili!', Validators.required],
    price: ['1.5', Validators.required, Validators.min(0.5)]
  });

  constructor(private readonly formGroup: FormGroup) {}
}
```

Forms Exercise 6

⚡ git checkout 11-reactive-forms

1. Open app.component.ts and declare a `constructor()` function that requires the `FormBuilder` to be injected into the component
2. Declare the `donutFormGroup` using the `FormBuilder.group()` method

Forms: Custom `ValidatorFn`

```
export function usaPhoneValidator(): ValidatorFn {
  return (control: AbstractFormControl): ValidationErrors | null => {
    const valid = /(0-9){3}-(0-9){3}-(0-9){4}/.test(control.value);
    return valid ? null : { phone: { value: control.value }};
  }
}
```

Forms: Custom `ValidatorFn`

```
@Component({
  selector: 'app-customer-form',
  template: `
    <input [formControl]="phoneNo" />
    <div *ngIf="phoneNo.errors.required" class="validator">
      Phone is required
    </div>
    <div *ngIf="phoneNo.errors.phone" class="validator">
      Phone number must be a US number with the format 123-123-1234
    </div>
  `
})
export class CustomerFormComponent {
  phoneNo = new FormControl('', [Validators.required, usaFormValidator()]);
}
```

Forms Exercise 7

1. Open donut-name.validator.ts and define a new exported function named `donutNameValidator`
2. The `donutNameValidator` function has a return type of `ValidatorFn`
3. Return an arrow function that accepts a `FormControl` and returns either `null` or a `ValidationErrors` object
4. Use a regular expression to test that the donut `name` contains: coated, dipped or frosted
5. Open app.component.ts and add the custom validator to the `name` FormControl in the `donutFormGroup`
6. Show a custom error message if the `name` is invalid

Forms: FormGroup Validation

```
export function passwordMatchValidator(formGroup: FormGroup):
ValidationErrors | null
{
  const password = formGroup.get('password');
  const confirmPassword = formGroup.get('confirmPassword');

  return
    password &&
    confirmPassword &&
    password.value &&
    confirmPassword.value &&
    password.value === confirmPassword.value
  ? null
  : { invalidPasswords: true };
}
```

Forms: `FormGroup` Validation

```
@Component({
  template: `
    <form [formGroup]="form">
      <input formControlName="password" />
      <input formControlName="confirmPassword" />
      <div *ngIf="form.errors?.invalidPasswords" class="validator">
        Passwords to not match
      </div>
    </form>
  `
})
export class CustomerFormComponent {
  form = this.formBuilder.group({
    password: ['', Validators.required],
    confirmPassword: ['', Validators.required]
  }, { validators: passwordMatchValidator });
}
```

Command-line Interface (CLI)

```
➜  npm install -g @angular/cli
➜  cd ~/Desktop
➜  ng new donut-shop --prefix=dnt --routing=true --style=scss
➜  ng generate component home/containers/index
➜  ng generate component home/components/wall-of-donuts

➜  ng generate module shared

➜  ng generate component shared/donut
➜  ng generate component shared/box-of-donuts
```

CLI Exercise

1. Use the Angular CLI to generate 3 new components:
    1. box-of-donuts
    2. donut-wall
    3. donut
2. You can skip tests for now via the `--no-spec` flag
3. Remove the existing declaration in the app.module.ts file
4. Copy the code out of the existing components and into each TypeScript, HTML and CSS files as appropriate
5. Delete the old files

Routing

Routing: Document Base URL

```html
<base href="/">
```

Routing: `Routes`

```
import { Route } from '@angular/router';

const routes: Route[] = [
  {
    path: 'donut-wall',
    component: DonutWallComponent
  },
  {
    path: '',
    pathMatch: 'full',
    redirectTo: '/donut-wall'
  }
];
```

Routing: `Routes`

```
import { Route, RouterModule } from '@angular/router';

const routes: Route[] = [];

@NgModule({
  imports: [RouterModule.forRoot(route)],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Routing: `RouterOutlet` Directive

```
@Component({
  selector: 'app-root',
  template: `
    <router-outlet></router-outlet>
  `
})
export class AppComponent {}
```

Routing Exercise

⚡ git checkout 12-routing

1. Open app.component.html and add a `<router-outlet>` element
2. Open app.module.ts and define a /donut-shop route for the `DonutShopComponent`
3. Redirect empty path to the /donut-shop route
4. Import the `RouterModule` and invoke the `forRoot()` static method, specifying the `routes`

Routing: Route Path Parameters

```
const routes: Route[] = [
  {
    path: 'donut-shop',
    component: DonutShopComponent
  },
  {
    path: 'kitchen/:id',
    component: KitchenComponent
  }
];

@NgModule({
  imports: [RouterModule.forRoot(route)],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Routing: `Router.navigate()`

```
export class DonutShopComponent {
  constructor(private readonly router: Router) {}

  onEdit(donut: Donut): void {
    this.router.navigate(['/kitchen', donut.id]);
  }
}
```

Routing: `Router.navigateByUrl()`

```
export class DonutShopComponent {
  constructor(private readonly router: Router) {}

  onEdit(donut: Donut): void {
    this.router.navigateByUrl(`/kitchen/${donut.id}`);
  }
}
```

Routing: `ActivatedRoute`

```
export class KitchenComponent implements OnInit {
  constructor(private readonly activatedRoute: ActivatedRoute) {}

  ngOnInit() {
    const id = this.activatedRoute.snapshot.paramMap.get('id');
  }
}
```

Routing Exercise 2

⚡ git checkout 12-routing

1. Open app.module.ts and define a route with the path of “/kitchen/:id” for the `KitchenComponent`
2. Open kitchen.component.ts and inject the `ActivatedRoute` instance
3. Use the `ActivatedRoute` to get the `id` parameter value, and then find the donut that is in the kitchen, setting the donut property
4. Inject the `Router`, and navigate back to the /donut-shop URL after the donut has been saved

Routing: Wildcard Route

```
const routes: Route[] = [
  {
    path: '**',
    component: NotFoundComponent
  }
];

@NgModule({
  imports: [RouterModule.forRoot(route)],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Routing Exercise 3

⚡ git checkout 12-routing

1. Use the Angular CLI to generate a new `NotFoundComponent`
2. Update the template to notify the user that the page cannot be found
3. Define a wildcard route and display the `NotFoundComponent`

Routing: Lazy Loading Module

```
const routes: Route[] = [
  {
    path: 'admin',
    loadChild: import('./admin/admin.module').then(
      ({ AdminModule }) => AdminModule
    )
  }
];

@NgModule({
  imports: [RouterModule.forRoot(route)],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Routing: Register Additional Routes

```
const routes: Route[] = [
  {
    path: 'index',
    component: IndexComponent
  }
];

@NgModule({
  imports: [RouterModule.forChild(route)],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Services

Services: Root Injector

```
@Injectable({
  providedIn: 'root'
})
export class DonutService {
  private readonly STORAGE_KEY = 'donuts';

  getAll(): Donut[] {
    const donuts = window.localStorage.getItem(this.STORAGE_KEY);
    return JSON.parse(donuts);
  }
}
```

Services Exercise

⚡ git checkout 13-services

1. Use the CLI to generate a new `DonutService`
2. Create a `populate()` method that populates the donuts in `localStorage` if undefined
3. Create a `getAll()` method that returns the donuts
4. Create a `save()` method that saves a donut
5. Inject the service into the `AppComponent` class and invoke the `populate()` method
6. Inject the service into the `DonutShopComponent` class and invoke the `getAll()` method
7. Inject the service into the `KitchenComponent` and invoke the `save()` method

HTTP

HTTP: `HttpClientModule`

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [HttpClientModule],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

HTTP: `HttpClient`

```
export class HttpClient {
  delete<T>(url: string, options): Observable<T>;
  get<T>(url: string, options): Observable<T>;
  put<T>(url: string, options): Observable<T>;
  post<T>(url: string, options): Observable<T>;
}
```

```
➜  git checkout 14-http
➜  cd server
➜  npm install
➜  nom start
```

HTTP Exercise

⚡ git checkout 14-http

1. Open app.module.ts and import the `HttpClientModule`
2. Open the donut.service.ts file and modify the `getAll()` and `save()` methods to use the HttpClient's `get()` and `put()` methods accordingly
3. Open the donut-shop.component.ts file and `subscribe()` to the observable stream setting the `donut` property

Staying Current

Get Involved with the Angular Community

- blog.angular.io
- meetup.com
- Podcast: The Angular Show
- YouTube: AngularAir
- Give a talk!

Thank You

@brian_love