Angular Change Detection

Angular updates variables and HTML using Change Detection Mechanism.
There are two types of Change Detection Strategy in Angular:

1. Default
2. OnPush

You can see its behavior with `@Input` and `@Output` emitters:

Effect on Child Component

We use **OnPush** strategy for performance reasons as it input to child will not trigger change detection automatic. It will have same effect on primitives(strings, boolean, numbers) but different for reference types(Objects, Arrays).

If you have child component where inputs are passed and output emitters:

parent.component.ts

```
name: {firstname: string, lastname: string} = {
    firstname: '',
    lastname: 'Dhamecha'
  };
```

parent.component.html

```
<input type="text" [(ngModel)]="name.firstname" />
<app-child [name]="name"></app-child>
```

Note: OnPush will on parent will not affect anything

child.component.ts

```
@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrl: './child.component.scss',
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class ChildComponent {

  @Input() name!: {firstname: string, lastname: string};
}
```

child.component.html

```
<p>
    Full Name: {{name?.firstname + ' ' + name?.lastname}}
</p>
```

Here, changing the value in the input field will not trigger the change detection cycle in the child. Which means, it will not change **name.firstname** in the **ChildComponent**.

Objects vs Primitives

If you pass Object reference, just like above example, it wont work as you saw but if you pass primitive like string then it will have no effect and will trigger change detection cycle on **ChildComponent**.

But what if you pass both, for example:

parent.component.html
```
<app-child [name]="name" [fname]=”name.firstname”></app-child>
```

child.component.ts
```
@Input() fName!: string;
```

Then, it will trigger the change detection mechanism and will also update the reference type as well, since change detection ran, which means, if you pass both then will trigger if only reference type, then it will not detect as **reference** is not changed.

Now, coming to previous example with only reference type passed, then if you set it to another object altogether or set it to **null**, then it will trigger change detection cycle in the **ChildComponent**.

This also applies to **Arrays**, if you push to an array, it wont trigger, if you assign a new array then it will.

Note: *It wont even trigger Input **set/get** neither it will be caught in **ngOnChanges.***

child.component.ts
```
  private _name!: {firstname: string, lastname: string};
  @Input() set name(name: {firstname: string, lastname: string}) {
    console.log('INPUT SET: ', name);
    this._name = name;
  }

  get name() {
    console.log("INPUT GET: ", this._name);
    return this._name;
  }

  constructor(private cdr: ChangeDetectorRef) {}

  ngOnChanges(changes: SimpleChanges): void {
    console.log('CHANGES: ', changes);
  }
```

DetectNow

If you want to manually trigger and detect changes (update the input variables) then inject **ChangeDetectorRef** and have a reference and then call **detectChanges()** in the **ChildComponent.**

```
this.cdr.detectChanges();
    OR
this.cdr.markForChanges();
```

Behavior with @Output

Lets consider,

parent.component.html

```
<app-child [name]="name" (onChangeName)="onChangeName($event)"></app-child>
```

child.component.ts

```
@Output()
onChangeName: EventEmitter<{firstname: string, lastname: string}> = new
EventEmitter();

changeName() {
    this.onChangeName.emit({firstname: 'Sharvil', lastname: 'Dhamecha'});
}
```

Now, suppose, you call the function **changeName()**, it will emit the value, which will trigger change detection cycle and update the UI, which means, if you have @Output, then it will trigger the Change Detection Cycle and will update the reference type variables.

Simple Events Also Trigger Change Detection

Simple events like **click, keyup,** etc, also triggers change detection cycle.

Effect on Instance Variables

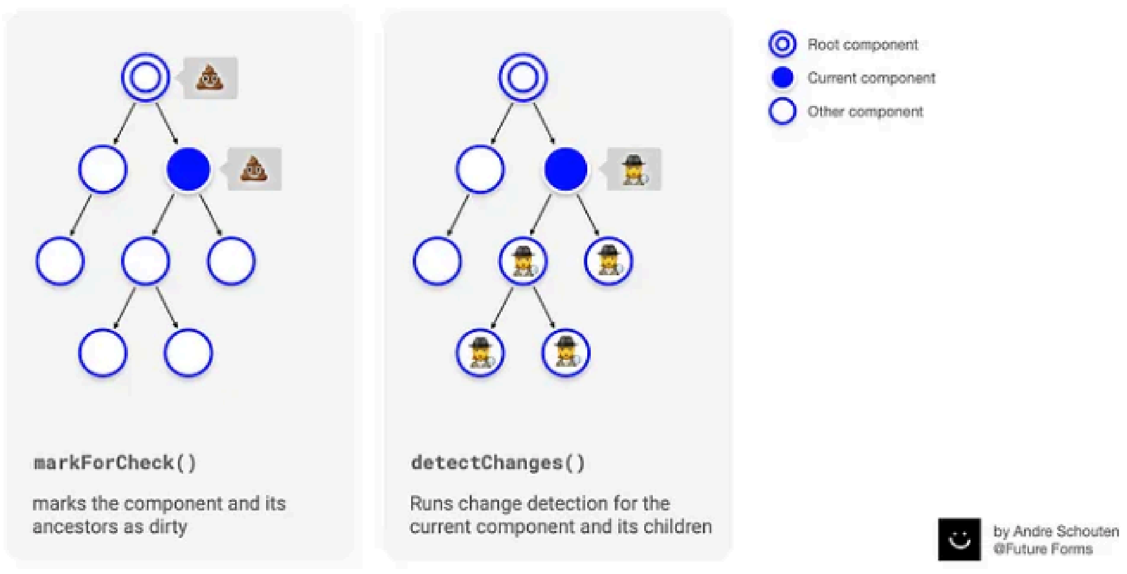
If you change the instance variables(not **@Input**) then also it will **not** trigger Change Detection Cycle if strategy is **OnPush**. Both for reference and primitive types. For example,

```
  public course: any = {
    id: 1,
    name: 'Change Detection',
    description: 'Will explain Angular Change Detection Strategy'
  };

  count: number = 0;
  setInterval(() => {
    this.count++;
  }, 2000);

  setTimeout(() => {
    this.course.name = 'Angular Change Detection';
  }, 3000);
```

DetectNow vs MarkForCheck


Root component
Current component
Other component
markForCheck()
marks the component and its ancestors as dirty
detectChanges()
Runs change detection for the current component and its children
by Andre Schouten
@Future Forms

This diagram shows a in a very simple way, whats the difference. Lets see an example,

Suppose,
child.component.ts

```
  private _name!: {firstname: string, lastname: string};

  @Input() set name(name: {firstname: string, lastname: string}) {
    console.log('INPUT SET: ', name);
    this._name = name;
    setTimeout(() => {
      this._name.firstname = 'Neha';
      this.cdr.markForCheck();
      // this.cdr.detectChanges();
    }, 3000);
  }
```

- In the above, example, **markForCheck()** will trigger, the parent Change Detection.

- In the above, example, **detectChanges()** will **not** trigger if your *parent.component.ts*, has **OnPush** then parent component Change Detection wont trigger, if **Default** then parent component will trigger on **detectChanges()**.

Change Detection with Observables (| async)

If any Observables emit, then it will trigger change detection cycle with **Async Pipe**.

Attribute Decorator

```
<app-child type=”beginner”></app-child>
```

If you know, your attribute won’t change its value, then taking it as @Input will allow Change Detection to check this value everytime. Instead you may want to take its value only once, just like an attribute of an element. You can do this like this,

```
constructor(@Attribute(‘type’) private type: string) {}
```

DoCheck

ngDoCheck() is a hook where it is being trigger whenever change detection is triggered on a component even if Change Detection Strategy is **OnPush**, which means, even if variables are not updated, still it is triggered. You can put your detection manually over here also like this,

```
ngDoCheck(): void {
    this.cdr.markForCheck();
}
```

This will behave in same way as if its **Default**.

It is also the best place where you want to implement custom change detection logic, like for example, if you want to update only if some condition satisfies.

😵

For example, if you are getting some data from a bunch of API, may be processing it in component.ts and when ready, you change a variable may be **this.isLoaded = true**, then only render it on HTML.

🤯