JavaSCRIPT

Async Await

▶ Async and Await are extensions of promises.

▶ More recent additions to the JavaScript language are async functions and the await keyword, part of the so-called ECMAScript 2017 JavaScript edition

▶ async keyword, which you put in front of a function declaration to turn it into an async function.

▶ Async functions enable us to write promise based code as if it were synchronous, but without blocking the execution thread. It operates asynchronously via the event-loop. Async functions will always return a value. Using async simply implies that a promise will be returned, and if a promise is not returned, JavaScript automatically wraps it in a resolved promise with its value.

▶ Basically async functions will always return a promise and we can use await inside async.

JavaScript Modules

- Split up your code into separate components
- Therefore easier to maintain and organize
- Supported in major browsers (excl. IE)

What is jquery

- jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

jQuery selectors

▶ $(this).hide() - hides the current element.

▶ $("p").hide() - hides all <p> elements.

▶ $(".test").hide() - hides all elements with class="test".

▶ $("#test").hide() - hides the element with id="test".

More Selectors

Syntax	Description
$("*")	Selects all elements
$(this)	Selects the current HTML element
$("p.intro")	Selects all <p> elements with class="intro"
$("p:first")	Selects the first <p> element
$("ul li:first")	Selects the first element of the first
$("ul li:first-child")	Selects the first element of every
$("[href]")	Selects all elements with an href attribute
$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"
$("a[target!='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"
$(":button")	Selects all <button> elements and <input> elements of type="button"
$("tr:even")	Selects all even <tr> elements
$("tr:odd")	Selects all odd <tr> elements

The Document Ready Event

- ```
  $(document).ready(function(){

    // jQuery methods go here...

  });
  ```

- This is to prevent any jQuery code from running before the document is finished loading (is ready).

Events

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

jQuery Animations

▶ The jQuery animate() method is used to create custom animations.

▶ $(*selector*).animate({*params*},*speed*,*callback*);

▶ The required params parameter defines the CSS properties to be animated.

▶ The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

▶ The optional callback parameter is a function to be executed after the animation completes.

Compile

- ❖ The first step of execution/processing a directive for rendering.
- ❖ Loads and traverses the DOM (of template)
  - › Compiles each of the directives collected thereafter (ex: nested directives)
- ❖ Happens for both existing Angular Directives or Custom Directives

```
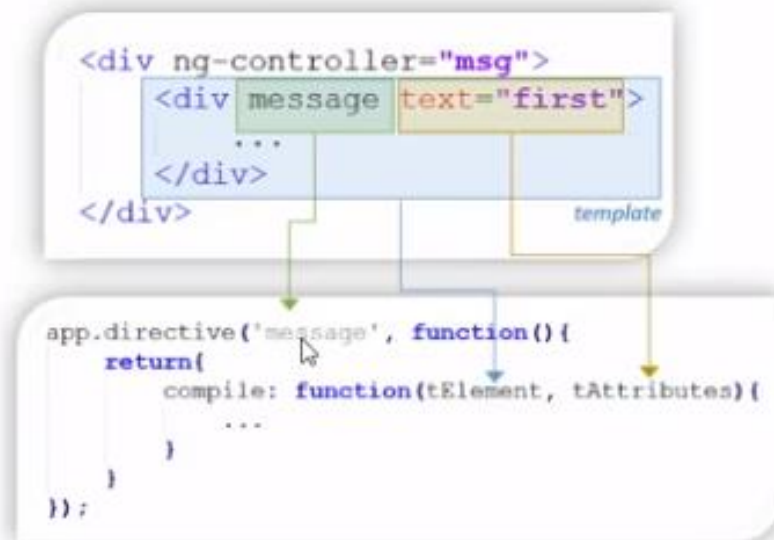<div ng-controller="msg">
    <div message text="first">
        ...
    </div>
</div>                          template
```

- • Template loaded from HTML DOM
- • Traverses the template for additional directives (if any)
- • Compiles (loads & Traverses) each of those additional directives found

Compile

❖ The first step of execution/processing a directive for rendering.

❖ Loads and traverses the DOM (of template)
  › Compiles each of the directives collected thereafter (ex: nested directives)

❖ Happens for both existing Angular Directives or Custom Directives

❖ "compile" function: executes code during compilation phase of a template

```
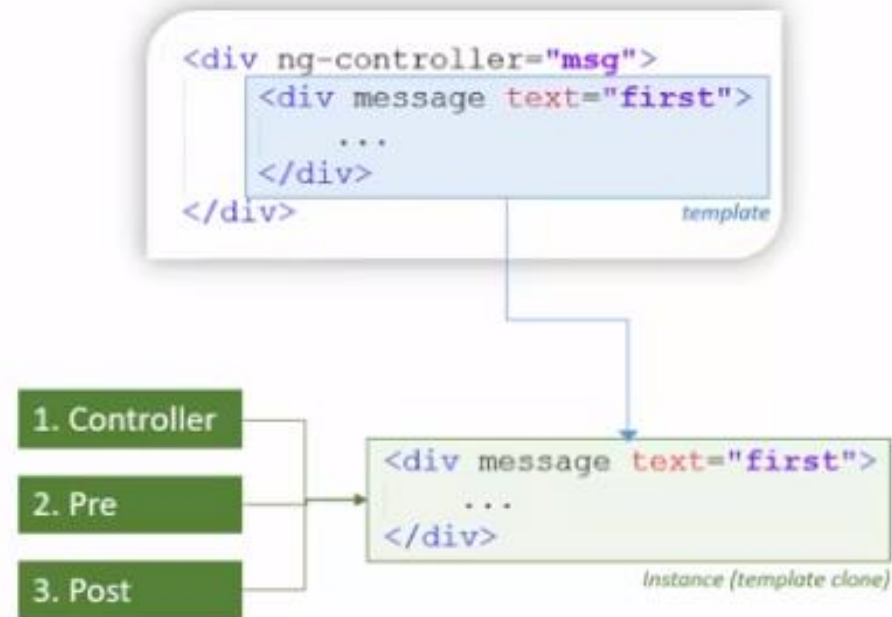<div ng-controller="msg">
    <div message text="first">
    ...
    </div>
</div>                        template
```

```
app.directive('message', function(){
    return{
        compile: function(tElement, tAttributes){
            ...
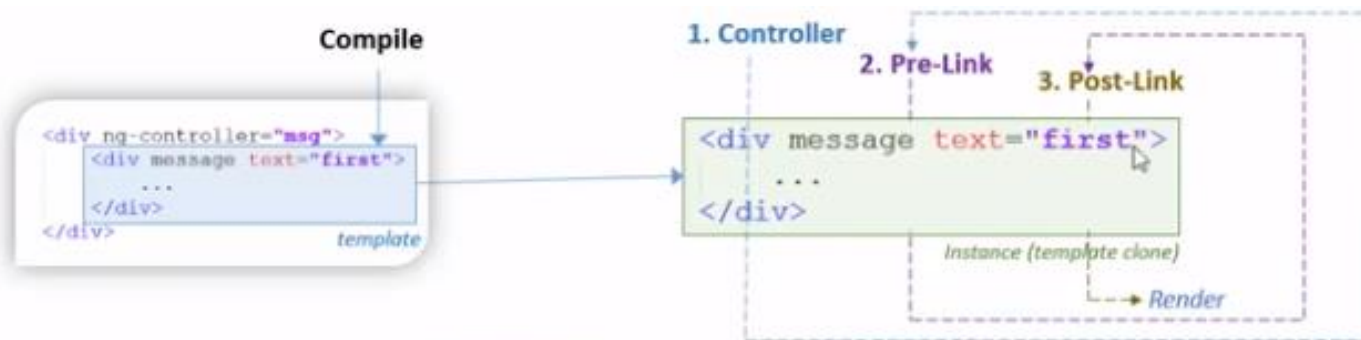        }
    }
});
```

Link

- ❖ The next phase after Compile
  - › Carried out in various steps
- ❖ Works on an instance of a Template (previously "compiled")
- ❖ 3 internal phases of template instance (executed in sequence):
  - › Controller (of directive)
  - › Pre
  - › Post (aka. "Link")
- ❖ Gets rendered after Link phase

```
<div ng-controller="msg">
    <div message text="first">
        . . .
    </div>
</div>
                            template
```

1. Controller
2. Pre
3. Post

```
<div message text="first">
    . . .
</div>
                    Instance (template clone)
```

Compile & Link

Compile

- Loads & traverses Template DOM
- Executes only once
- Returns link function
    - Or object (containing pre, post (or link) etc.)
- No Scope present
- No instances/clones (of template) created yet
- Do operations which can be shared among all instances/clones of template
- Can manipulate DOM of template
- Cannot play with data/events of clones
- No DOM of clones available

Controller

- The first to execute for every instance/clone of template
- Creates scope (scope initialization) for the template instance
- Can manipulate data for template instance
- Not recommended to access instance DOM

Pre

- Executes immediately after controller phase (for every instance)
- Ref. to DOM template instance is available
- Scope for the instance is ready
- Instance is not linked to scope yet (no bindings are setup)
- Child elements/directives not ready
- Scope can be manipulated
- Safe to set data and even child data
- Not safe to manipulate DOM template instance
- No access to child elements

Post

- The last phase (of every instance), usually called "linking" (or link function etc.)
- Ref. to DOM template instance is available
- Scope and instance linked (and data bound)
- Child elements/directives are ready (and already linked)
- Scope can be manipulated
- Safe to attach event handlers, inspect child elements
- Safe to manipulate DOM template instance
- Not safe to set data for child elements

Order of execution
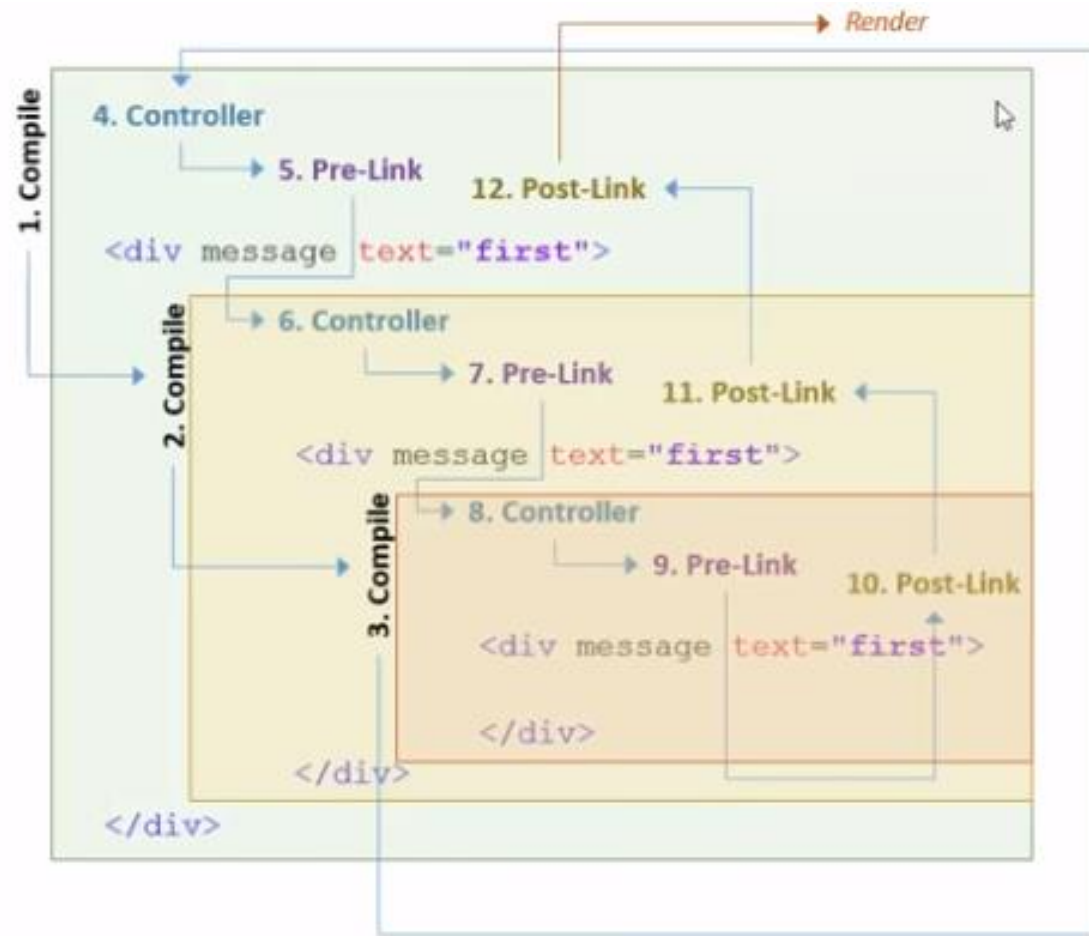
Watchers

❖ Nothing but monitoring: keeping track of (or watching) scope variables and the changes of their values.

❖ Watch:
  › keeps track of a variable and its value
  › Watched by angular framework (specifically, *digest cycle loop*)
  › Some variables may not be watched
  › If the value gets changed, angular performs necessary updates (propagations to DOM, UI update etc.)
  › Angular can execute our custom functions (watch listeners) on value changes

❖ Used heavily in data-binding
  › Bindings are associated with Watches.
  › Even, internally in AngularJs Framework.

```
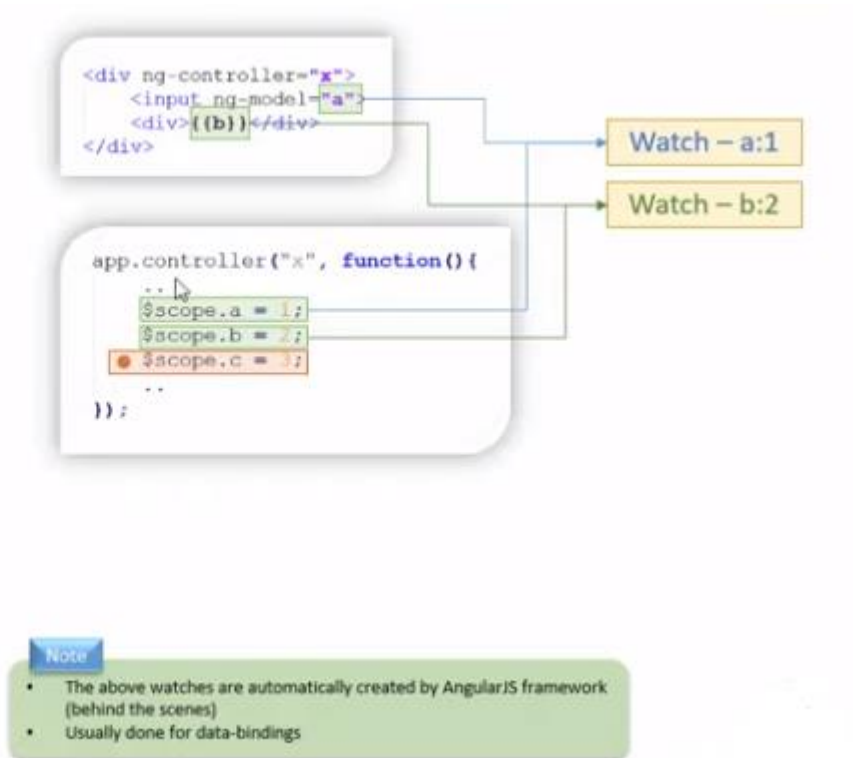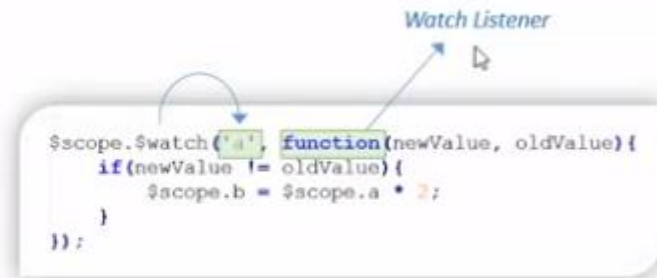<div ng-controller="x">
    <input ng-model="a">
    <div>{{b}}</div>
</div>
```

Watch – a:1

Watch – b:2

```
app.controller("x", function(){
    ..
    $scope.a = 1;
    $scope.b = 2;
  ● $scope.c = 3;
    ..
});
```

Note
- The above watches are automatically created by AngularJS framework (behind the scenes)
- Usually done for data-bindings

Watch Listener

❖ We can add our own "watch" (manually) using "$watch" function available through "$scope" object.
  › Available at $rootScope as well

❖ Used when AngularJS does not consider scope variables for watching
  › Ex: scope variables not being used in data-binding

❖ Watch Listener: a function which gets executed when a "watch" detects a value change.

Watch Listener

```
$scope.$watch('a', function(newValue, oldValue){
    if(newValue != oldValue){
        $scope.b = $scope.a * 2;
    }
});
```

Reference Listener

❖ $watch - Reference watch
❖ $watch with "true" – Equality watch


```
$scope.o = {
    a: 1,
    b: 2,
    c: 4
};

$scope.$watch('o', function(newValue, oldValue){
    if(newValue != oldValue){
        $scope.o.c = $scope.o.a * $scope.o.b;
    }
});

$scope.$watch('o', function(newValue, oldValue){
    if(newValue != oldValue){
        $scope.o.c = $scope.o.a * $scope.o.b;
    }
}, true);
```

Equality Listener

- ❖ $watch - <u>Reference watch</u>
- ❖ $watch with "true" – <u>Equality watch</u>
- ❖ $watchGroup – Reference watch for multiple variables
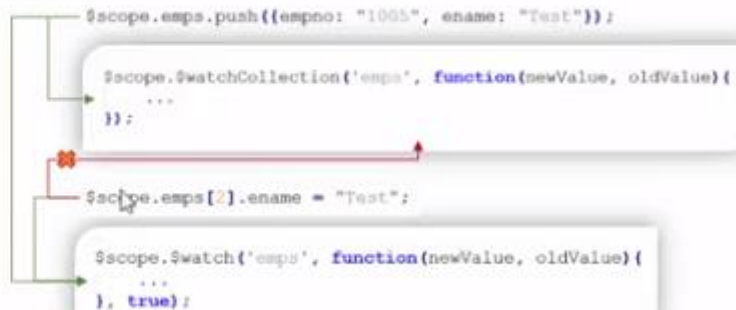
```
$scope.a = 1;
$scope.b = 2;
$scope.c = 4;

$scope.$watchGroup(['a', 'b'], function(newValue, oldValue){
    if(newValue != oldValue){
        $scope.c = $scope.a * $scope.b;
    }
});
```

Collection Listener

❖ $watch - <u>Reference watch</u>

❖ $watch with "true" – <u>Equality watch</u>

❖ $watchGroup – Reference watch for multiple variables

❖ $watchCollection – <u>Collection watch</u>
  › Used to watch arrays
  › Detects modifications to arrays (ex: adding or deleting elements)
  › Does not detect modifications to array items (ex: modifying an object in an array)
    » To achieve this, use **Equality watch**

```
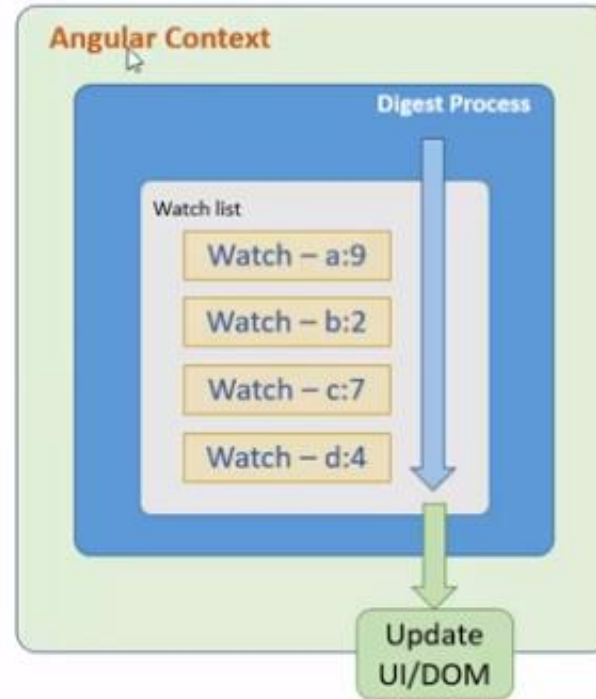$scope.emps = [
    {empno: "1001", ename: "Jag"},
    {empno: "1002", ename: "Chat"},
    {empno: "1003", ename: "Win"},
    {empno: "1004", ename: "Dhan"},
];
```

```
$scope.emps.push({empno: "1005", ename: "Test"});
```

```
$scope.$watchCollection('emps', function(newValue, oldValue){
    ...
});
```

```
$scope.emps[2].ename = "Test";
```

```
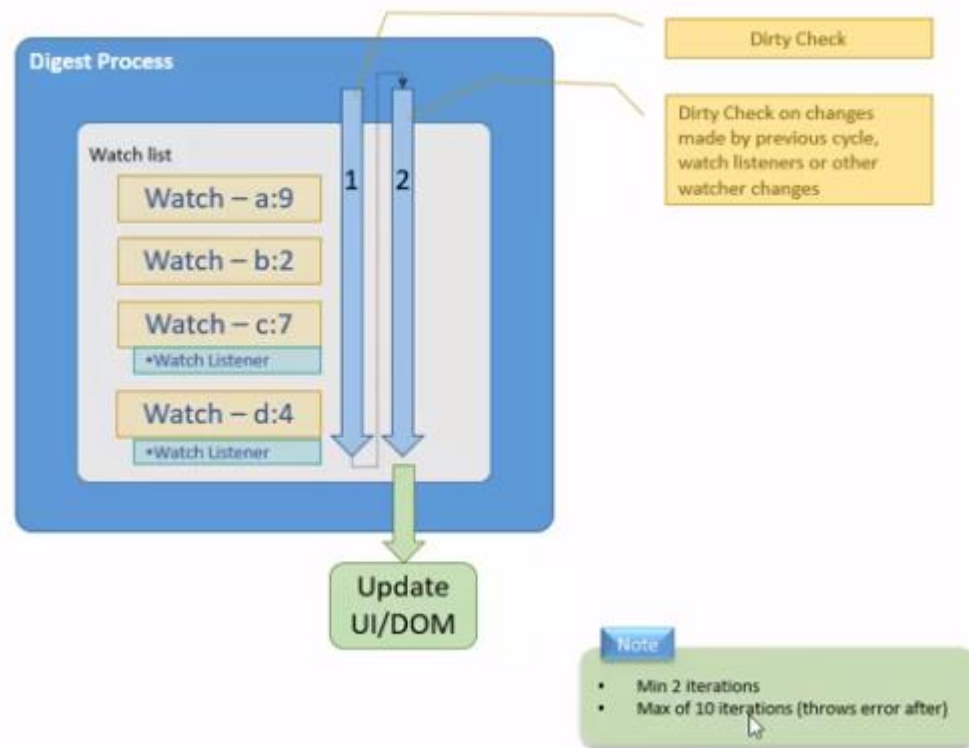$scope.$watch('emps', function(newValue, oldValue){
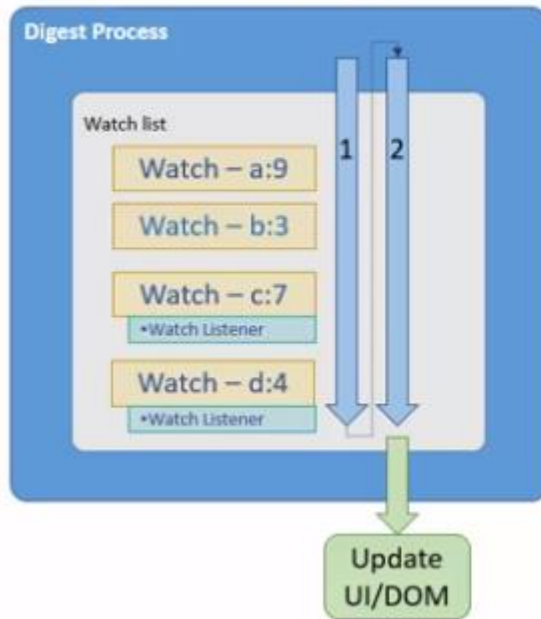    ...
}, true);
```

Digest Process

❖ Digest Process (in simple words):
  › Responsible to walk-through entire watch list for modifications
    » Also called "dirty-checking" (the process of checking the current values of scope variables with their previous values)
  › Exists modifications? Execute Watch Listeners, if any.
  › Keeps note of all modifications and notifies Angular JS framework to update DOM

❖ In general, DOM gets updated after Digest Process.

❖ Digest Process runs as part of Angular Context
  › **Angular Context**: run-time env. of AngularJS Framework

Digest Cycle

Angular Context