1. What is the importance of controller in MVC pattern?

Ans:

1) A Controller is defined by a JavaScript constructor function which is used to augment with Angular Scope.
2) And Controller Constructor function is the place where we will be writing our business logics.
3) "ng-controller" is the directive that is to be used for instantiating new controller object.
4) If the controller has been attached using the "controller as" syntax then the controller instance will be assigned to a property on the new scope.

2. What are scopes in AngularJS?

Ans: Unlike the other MVC frameworks, AngularJS doesn't have specific classes or functions to create model objects. So, for creating model objects AngularJS has extended the raw javascript objects with custom methods and properties and these are known as "Scopes".

Scope communicates with both the views and other parts of angular (directives, controllers, and services) in an angular app.

When angular app is auto bootstrapped a rootscope will get created and from this all the other scopes get inherited.

3. What are the different types of scopes in AngularJS?

Ans:

1) 1. Shared Scope (scope: false) - Directive uses the parent scope.
2) 2. Inherited Scope (scope: true) - Directive gets new scope.
3) 3. Isolated Scope (scope: {}) - Directive gets a new Isolated Scope.

4. What are the different ways of injecting $scope dependency in an angular application?

Ans: As a developer, we really don't care about how AngularJS injects dependencies. But here are the three possible ways of passing dependencies to angular application.

1) Passing a dependency as Function Arguments
2) Passing a dependency as Array Arguments
3) Passing a dependency using the $inject service

5. What are the disadvantages of using ng-controller?

Ans: There are few reasons why ng-controller is problematic.

1) **Inheritance** - The ng-controller directive creates a controller, and gives it a scope that prototypally inherits from the parent scope. This prototypal inheritance tends to trips where we can really get rid of ng-controller completely because we have better ways like "Isolate Scope" to pass data along with the children.

2) **Semi-Global Data** - When you access some piece of data on a scope, how do you know where that data comes from? When you have even a couple of nested controllers, this isn't at all clear. The data might come from any controller that has access to some scope in the inheritance chain. You can easily lose track of where things are defined. Angular 1.2+ "controller as" syntax does a lot to help with this particular problem.
3) **Poor View Organization** - there's no one-to-one correspondence between HTML files and controller JavaScript files.

6. What are Scope less controllers?

Ans: If an application does not depend upon communication with other controllers, or does not require any inheritance feature, you can use Scopeless Controllers. Scopeless Controllers are useful for small applications, to avoid complexities. Scopeless Controlles will be using "this" keyword instead $scope.

7. Difference between "this" and $scope?

Ans:

$scope

1) $scope is a core concept of angular framework and dual data-binding functionalities.
2) It can share its content with templates, directives.
3) Every controller has an associated $scope object.
4) A controller (constructor) function is responsible for setting model properties and functions/behaviour on its associated $scope
5) Only methods defined on this $scope object and parent scope objects, if prototypical inheritance is in play are accessible from the HTML/view. E.g., from ng-click, filters, etc.

this

1) this keyword refers the javascript object referring to your controller. When the controller constructor function is called, this is the controller.
2) When a function defined on a $scope object is called, this is the "scope in effect when the function was called".
3) This may or may not be the $scope that the function is defined on. So, inside the function, this and $scope may not be the same.

8. Explain about $scope life cycle?

Ans: In a normal execution process of browser, when a javascript call back is completed then the browser immediately re-renders the DOM and it returns to wait for further more javascript events. But when we are using angular it is unaware of the model modifications in a normal execution process.

So, to properly process model modifications the execution has to enter the Angular execution context using the $apply method. Only model modifications which execute inside the $apply method will be properly accounted for by Angular.

After evaluating the expression, the $apply method performs a $digest. In the $digest phase the scope examines all of the $watch expressions and compares them with the previous value. This dirty checking is done asynchronously. This means that assignment such as $scope.username="angular" will not immediately cause a $watch to be notified, instead the $watch notification is delayed until the $digest phase. This delay is desirable, since it coalesces multiple model updates into one $watch notification as well as guarantees that during the $watch notification no other $watches are running. If a $watch changes the value of the model, it will force additional $digest cycle.

These are the steps in $scope's life cycle process.

1. **Creation**

The root scope is created during the application bootstrap by the $injector. During template linking, some directives create new child scopes.

2. **Watcher registration**

During template linking, directives register watches on the scope. These watches will be used to propagate model values to the DOM.

3. **Model mutation**

For mutations to be properly observed, you should make them only within the **scope.$apply().** Angular APIs do this implicitly, so no extra $apply call is needed when doing synchronous work in controllers, or asynchronous work with **$http, $timeout or $interval services**.

4. Mutation observation

At the end of $apply, Angular performs a $digest cycle on the root scope, which then propagates throughout all child scopes. During the $digest cycle, all $watched expressions or functions are checked for model mutation and if a mutation is detected, the $watch listener is called.

5. Scope destruction

When child scopes are no longer needed, it is the responsibility of the child scope creator to destroy them via scope. $destroy() API. This will stop propagation of $digest calls into the child scope and allow for memory used by the child scope models to be reclaimed by the garbage collector.