MODULE 1

1

NAME

Unit Structure

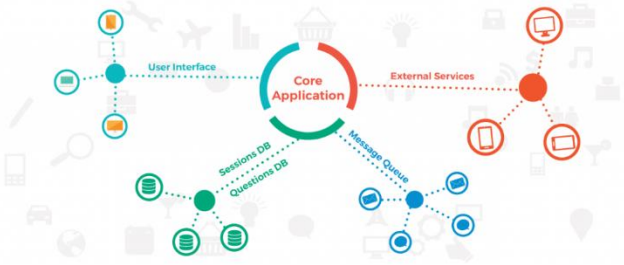
AIM:
What is node js
Objective:
Explain node JS
Theory:

It is a free, open-source, cross-platform runtime environment that runs on JavaScript. Meant mainly for the server side, or client side, of a mobile application, it is a full-stack development environment that divides up tasks into fully separate "nodes."

When people ask, "what are the advantages of Node.js," its structure is the reason we give them. It is, after all, the primary reason why it was created, and a whole ecosystem of dedicated, open-source modules was created around it.

Whether you're developing apps for the iPad, iPhone, or Android, nodes are what makes this environment so useful to programmers. Because each task is separated into independent, separate node paths, tasks can run simultaneously and seamlessly together, without bogging down the server or using up too much capacity.

A Node refers to this as a "microservice" pattern, with each a self-contained pathway to fulfill a particular service. It's an innovative way of breaking an app down into its smallest bits. But it's a very efficient way to handle mobile applications, which need speed, accessibility, and accuracy above all.



This chart represents how a typical application might be organized. See how everything is organized as a spoke off the core app programming? Functions like customer email, or shopping carts, or portal requests are

given their own “node.” And those self-contained areas can run without interfering with each other, or the core application itself.

Advantage of Node JS:

The ability to scale up quickly: Each of the nodes in Node.js is based around an “event.” A customer buys an in-app purchase, or sends an email to customer service, for instance. The amount of nodes you can add to your core programming function are nearly limitless. This means you can scale vertically, adding new capability paths that lead back to your core application code. Or, you can scale horizontally, adding new resources to existing programming. Either way, scalability gives your application room to grow, and that’s one of the key benefits of using Node.js.

Speed and Performance: Its non-blocking, input-output operations make the environment one of the speediest options available. Code runs quickly, and that enhances the entire run-time environment. This is largely due to its sectioned-off system. But it also has to do with the fact that it runs on Google’s V8 JavaScript engine. Its apps are more likely to be programmed end-to-end in Javascript, and that plug and play interoperability contributes to speed and performance.

Flexibility: In a discussion of Node.js pros and cons programming flexibility is perhaps the biggest benefit of all. When you make a change in Node.js, only that node is affected. Where other run time environments or frameworks may require you to make changes all the way back to the core programming, it doesn’t require anything more than a change to the node. And that is great not just for your initial build, but for your ongoing maintenance as well. And best of all, when you combine JSON with Node.js, you can easily exchange information between client servers and the webserver. Programmers can also use APIs to build TCP, HTTP, DNS, etc. into the server.

Efficient caching: In a debate over the pros and cons of Node.js, caching always comes up as a key Node.js benefit. It has a powerful ability to cache data. When requests are made to the app, they are cached in-app memory. Consequently, when requests cycle through execution and re-execution, the nodes are still able to run efficiently and not be bogged down by historical data.

Fast-to-market-development: Node’s basis in JavaScript brings many benefits to the table, especially the ease at which developers can add more features and predesigned tools and templates.

Efficient Queueing of Requests: A critical benefit of using Node.js is its ability to handle multiple requests at once. How does it do this? By offering users an optional nob-block I/O system. The system works by giving priority to those requests that take the lowest response time. This

prioritization helps speed up the overall running of your app, especially when comparing it to other languages like Python or Ruby on Rails.

Node.js Process Model:

The Node.js process model differs from traditional web servers in that Node.js runs in a single process with requests being processed on a single thread. One advantage of this is that Node.js requires far fewer resources. When a request comes in, it will be placed in an event queue. Node.js uses an event loop to listen for events to be raised for an asynchronous job. The event loop continuously runs, receiving requests from the event queue.

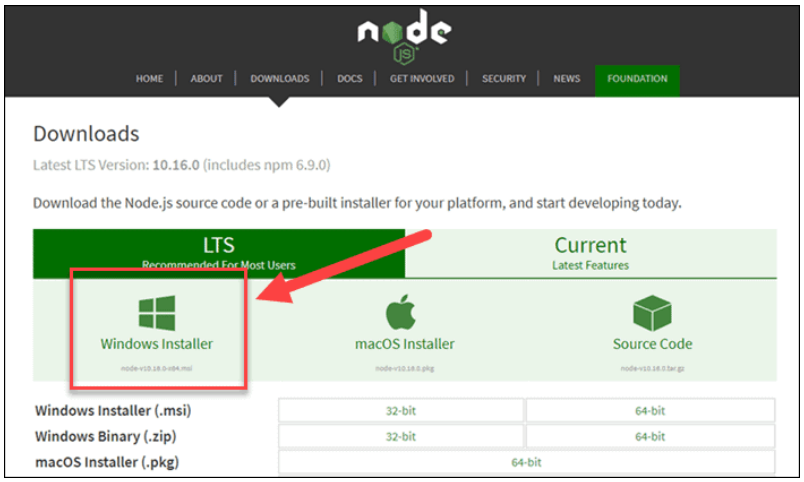
There are two scenarios that will occur depending on the nature of the request. If the request is non-blocking, it does not involve any long-running processes or data requests, the response will be immediately prepared and then sent back to the client. In the event the request is blocking, requiring I/O operations, the request will be sent to a worker thread pool. The request will have an associated call-back function that will fire when the request is finished and the worker thread can send the request to the event loop to be sent back to the client. In this way, when the single thread receives a blocking request, it hands it off so that the thread can process other requests in the meantime. In this way Node.js is inherently asynchronous.

Traditional Web Server Model:

The traditional web server model consists of a pool of threads which may process requests. Each time a new request comes in, it is assigned to a different thread in the pool. In the event a request is received and a thread is not available, the request will have to wait until a previous request finishes, a response is returned, and the thread is returned to the thread pool. In this way, the web server model is synchronous, or blocking.

How to Install Node.js and NPM on Windows:

In a web browser, navigate to https://nodejs.org/en/download/. Click the **Windows Installer** button to download the latest default version. At the time this article was written, version 10.16.0-x64 was the latest version. The Node.js installer includes the NPM package manager.



Step 2: Install Node.js and NPM from Browser

1. Once the installer finishes downloading, launch it. Open the **downloads** link in your browser and click the file. Or, browse to the location where you have saved the file and double-click it to launch.
2. The system will ask if you want to run the software – click **Run**.
3. You will be welcomed to the Node.js Setup Wizard – click **Next**.
4. On the next screen, review the license agreement. Click **Next** if you agree to the terms and install the software.
5. The installer will prompt you for the installation location. Leave the default location, unless you have a specific need to install it somewhere else – then click **Next**.
6. The wizard will let you select components to include or remove from the installation. Again, unless you have a specific need, accept the defaults by clicking **Next**.
7. Finally, click the **Install** button to run the installer. When it finishes, click **Finish**.

Step 3: Verify Installation:

Open a command prompt (or PowerShell), and enter the following:
node -v

The system should display the Node.js version installed on your system. You can do the same for NPM:
npm -v

Node.js - REPL Terminal:

REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode. Node.js or **Node** comes bundled with a REPL environment. It performs the following tasks −

- **Read** − Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
- **Eval** − Takes and evaluates the data structure.
- **Print** − Prints the result.
- **Loop** − Loops the above command until the user presses **ctrl- c** twice.

The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.

Starting REPL:

REPL can be started by simply running **node** on shell/console without any arguments as follows.

```
$ node
```

You will see the REPL Command prompt > where you can type any Node.js command −

```
$ node
>
```

Simple Expression

Let's try a simple mathematics at the Node.js REPL command prompt −

```
$ node
> 1 + 3
4
> 1 + ( 2 * 3 ) - 4
3
>
```

Node.js – Console:

Node.js **console** is a global object and is used to print different levels of messages to stdout and stderr. There are built-in methods to be used for printing informational, warning, and error messages.

It is used in synchronous way when the destination is a file or a terminal and in asynchronous way when the destination is a pipe.

Console Methods:

Following is a list of methods available with the console global object.

Sr.No.	Method & Description
1	**console.log([data][, ...])** Prints to stdout with newline. This function can take multiple arguments in a printf()-like way.
2	**console.info([data][, ...])** Prints to stdout with newline. This function can take multiple arguments in a printf()-like way.
3	**console.error([data][, ...])** Prints to stderr with newline. This function can take multiple arguments in a printf()-like way.
4	**console.warn([data][, ...])**

	Prints to stderr with newline. This function can take multiple arguments in a printf()-like way
5	**console.dir(obj[, options])** Uses util.inspect on obj and prints resulting string to stdout.
6	**console.time(label)** Mark a time.
7	**console.timeEnd(label)** Finish timer, record output.
8	**console.trace(message[, ...])** Print to stderr 'Trace :', followed by the formatted message and stack trace to the current position.
9	**console.assert(value[, message][, ...])** Similar to assert.ok(), but the error message is formatted as util.format(message...).

*****

Module II

AIM:

To demonstrate the use of Standard callback pattern

OBJECTIVE

Node js callback pattern function callback

Theory:
Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed.

Explanation:
Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.
Create a text file named **input.txt** with the following content :

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

Create a js file named **main.js** with the following code :

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');

console.log(data.toString());
console.log("Program Ended");
```

Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Tutorials Point is giving self learning content
```

```
to teach the world in simple and easy way!!!!!
Program Ended
```

AIM:

To demonstrate the event emitter pattern

Objective:

Explanation of event emitter pattern with programme.

Theory:

The EventEmitter is a module that facilitates communication/interaction between objects in Node. EventEmitter is at the core of Node asynchronous event-driven architecture. Many of Node's built-in modules inherit from EventEmitter including prominent frameworks like Express.js.

The concept is quite simple: emitter objects emit named events that cause previously registered listeners to be called. So, an emitter object basically has two main features:

- Emitting name events.
- Registering and unregistering listener functions.

It's kind of like a pub/sub or observer design pattern (though not exactly).

Explanation:

The $http.POST() service is used to deliver data to a certain URL and expects the resource at that URL to handle the request. In other words, the POST method is used to insert new data based on a specified URL and is one of the $http service's shortcut methods.

Create an event emitter instance and register a couple of callbacks

```
const myEmitter = new EventEmitter();

function c1() {
  console.log('an event occurred!');
}

function c2() {
  console.log('yet another event occurred!');
}

myEmitter.on('eventOne', c1); // Register for eventOne
myEmitter.on('eventOne', c2); // Register for eventOne
```

When the event 'eventOne' is emitted, both the above callbacks should be invoked.

```
myEmitter.emit('eventOne');
```

The output in the console will be as follows:

```
an event occurred!
yet another event occurred!
```

Events:

Sr.No.	Events & Description
1	**newListener** • **event** − String: the event name • **listener** − Function: the event handler function This event is emitted any time a listener is added. When this event is triggered, the listener may not yet have been added to the array of listeners for the event.
2	**removeListener** • **event** − String The event name • **listener** − Function The event handler function This event is emitted any time someone removes a listener. When this event is triggered, the listener may not yet have been removed from the array of listeners for the event.

emitter.removeListener(event, listener)

Remove a listener from the listener array for the specified event. **Caution**: changes array indices in the listener array behind the listener.

```
var callback = function(stream) {
  console.log('someone connected!');
};
server.on('connection', callback);
server.removeListener('connection', callback);
```

emitter.removeAllListeners([event])

Removes all listeners, or those of the specified event.

AIM:

To demonstrate the use of defer execution of a function

Objective

Implement defer execution in node JS function

Theory:

One occasionally needs to defer the execution of a function.

Traditional JavaScript uses timers for this purpose, with the well-known setTimeout and setInterval functions.

Node introduces another perspective on defers, primarily as means of controlling the order in which a callback executes in relation to I/O events, as well as timer events properly.

Two types of deferred event sources that give a developer the ability to schedule callback executions to occur either before, or after, the processing of queued I/O events are process.nextTick and setImmediate.

process.nextTick

The primary use of nextTick in a function is to postpone the broadcast of result events to listeners on the current execution stack until the caller has had an opportunity to register event listeners, giving the currently executing program a chance to bind callbacks to EventEmitter.emit events.

Explanation

```
// get the reference of EventEmitter class of events module
var events = require('events');

//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
   console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

In the above example, we first import the 'events' module and then create an object of EventEmitter class. We then specify event handler function using on() function. The on() method requires name of the event to handle and callback function which is called when an event is raised.

The emit() function raises the specified event. First parameter is name of the event as a string and then arguments. An event can be emitted with zero or more arguments. You can specify any name for a custom event in the emit() function.

AIM:

To demonstrate the use stop execution of a function

Objective:

Using process.exit() method stop function execution

Theory:

Using a return is the correct way to stop a function executing. You are correct in that process.exit() would kill the whole node process, rather than just stopping that individual function. Even if you are using a callback function, you'd want to return it to stop the function execution.

ASIDE: The standard callback is a function where the first argument is an error, or null if there was no error, so if you were using a callback the above would look like:

Explanation:

```
var thisIsTrue = false;

exports.test = function(request, response, cb){

   if (thisIsTrue) {
      response.send('All is good!');
      cb(null, response)
   } else {
      response.send('ERROR! ERROR!');
      return cb("THIS ISN'T TRUE!");
   }

   console.log('I do not want this to happen. If there is an error.');

}
```

We should use return, which will help you respond to what happened. Here's a bit cleaner version, basically first validate whatever you want to validate, rather than encapsulating everything in if{}else{} statements

Another way would be to use throw

a) Throw an error, which will also help you debug the app (wouldn't completely stop the app, if the test() function was wrapper in try{}catch(e){}):

```
throw new Error('Something went wrong')
```

b) Stop script execution (works with Node.js):

```
process.exit()
```

AIM:

To demonstrate the use Schedule and repetitive execution

Objective:

Using setTimeout & setInterval Schedule and repetitive execution

Theory:
We may decide to execute a function not right now, but at a certain time later. That's called "scheduling a call".

There are two methods for it:

- setTimeout allows us to run a function once after the interval of time.
- setInterval allows us to run a function repeatedly, starting after the interval of time, then repeating continuously at that interval.

These methods are not a part of JavaScript specification. But most environments have the internal scheduler and provide these methods. In particular, they are supported in all browsers and Node.js.

Explanation:
```
let timerId = setTimeout(func|code, [delay], [arg1], [arg2], ...)
```

Parameters:

func|code

Function or a string of code to execute. Usually, that's a function. For historical reasons, a string of code can be passed, but that's not recommended.

Delay

The delay before run, in milliseconds (1000 ms = 1 second), by default 0.

arg1, arg2…
Arguments for the function (not supported in IE9-)
For instance, this code calls sayHi() after one second:
```
function sayHi() {
  alert('Hello');
}

setTimeout(sayHi, 1000);
```
With arguments:

Canceling with clearTimeout

A call to setTimeout returns a "timer identifier" timerId that we can use to cancel the execution.

The syntax to cancel:
```
let timerId = setTimeout(...);
clearTimeout(timerId);
```

In the code below, we schedule the function and then cancel it (changed our mind). As a result, nothing happens:

```
let timerId = setTimeout(() => alert("never happens"), 1000);
alert(timerId); // timer identifier

clearTimeout(timerId);
alert(timerId); // same identifier (doesn't become null after canceling)
```

As we can see from alert output, in a browser the timer identifier is a number. In other environments, this can be something else. For instance, Node.js returns a timer object with additional methods.

Again, there is no universal specification for these methods, so that's fine.

For browsers, timers are described in the timers section of HTML5 standard.

setInterval

The setInterval method has the same syntax as setTimeout:

```
let timerId = setInterval(func|code, [delay], [arg1], [arg2], ...)
```

All arguments have the same meaning. But unlike setTimeout it runs the function not only once, but regularly after the given interval of time.

To stop further calls, we should call clearInterval(timerId).

The following example will show the message every 2 seconds. After 5 seconds, the output is stopped:

```
// repeat with the interval of 2 seconds
let timerId = setInterval(() => alert('tick'), 2000);

// after 5 seconds stop
setTimeout(() => { clearInterval(timerId); alert('stop'); }, 5000);
```

Nested setTimeout

There are two ways of running something regularly.

One is setInterval. The other one is a nested setTimeout, like this:

```
/** instead of:
let timerId = setInterval(() => alert('tick'), 2000);
*/

let timerId = setTimeout(function tick() {
```

```
  alert('tick');
  timerId = setTimeout(tick, 2000); // (*)
}, 2000);
```

The setTimeout above schedules the next call right at the end of the current one (*).

The nested setTimeout is a more flexible method than setInterval. This way the next call may be scheduled differently, depending on the results of the current one.

For instance, we need to write a service that sends a request to the server every 5 seconds asking for data, but in case the server is overloaded, it should increase the interval to 10, 20, 40 seconds…

AIM:
To demonstrate the use Block escape event loop

Objective
Using block loop method run node js function

Theory
Now that we have a healthy refresh on how threads work, we can finally tackle the **Node.js event loop logic**. By reading this, you will understand the reason behind the previous explanation, and every piece will go at the right spot by itself.

Whenever we run a Node program, a thread is automatically created. This thread is the only place where our entire codebase is going to be executed. Inside of it, something called the **event loop** is generated. The role of this loop is to schedule which operations our only thread should be performing at any given point in time.

Please note that the event loop doesn't get generated instantly as soon as we run our program. In fact, it only runs once the whole program has been executed.

Explanation:

```
const fs = require('fs');
function someAsyncOperation(callback) {
// Assume this takes 95ms to complete
fs.readFile('/path/to/file', callback);
}

const timeoutScheduled = Date.now();
```

For example, say you schedule a timeout to execute after a 100 ms threshold, then your script starts asynchronously reading a file which takes 95 ms:

When the event loop enters the **poll** phase, it has an empty queue (`fs.readFile()` has not completed), so it will wait for the number of ms remaining until the soonest timer's threshold is reached. While it is waiting 95 ms pass, `fs.readFile()` finishes reading the file and its callback which takes 10 ms to complete is added to the **poll** queue and executed. When the callback finishes, there are no more callbacks in the queue, so the event loop will see that the threshold of the soonest timer has been reached then wrap back to the **timers** phase to execute the timer's callback. In this example, you will see that the total delay between the timer being scheduled and its callback being executed will be 105ms.

```
setTimeout(() => {
  const delay = Date.now() - timeoutScheduled;

  console.log(`${delay}ms have passed since I was scheduled`);
}, 100);

// do someAsyncOperation which takes 95 ms to complete
someAsyncOperation(() => {
  const startCallback = Date.now();

  // do something that will take 10ms...
  while (Date.now() - startCallback < 10) {
    // do nothing
  }
});
```

*****

Module III

AIM:

To demonstrate the Fs module file paths

Objective

Explantion of Fs module file paths

Theory:

n Node.js, file handling is handled by fs module. You can read more about it here. We can check the path for file or directory in Node.js in both Synchronous and Asynchronous way.

Note: Asynchronous version is usually preferable if you care about application performance.

Synchronous method: Synchronous operations are great for performing one-time file/directory operations before returning a module. To check the path in synchronous mode in fs module, we can use statSync() method. The fs.statSync(path) method returns the instance of fs.Stats which is assigned to variable stats. A fs.Stats object provides information about a file. The stats.isFile() method returns true if the file path is File, otherwise returns false. The stats.isDirectory() method returns true if file path is Directory, otherwise returns false.

Explanation:

Example:

```
// Require the given module
var fs = require('fs');
// Use statSync() method to store the returned
// instance into variable named stats
var stats =
fs.statSync("/Users/divyarani/Documents/geekforgeeks/geeks.js");
// Use isFile() method to log the result to screen
console.log('is file ? ' + stats.isFile());
var stats = fs.statSync("/Users/divyarani/Documents/geekforgeeks/geek");
// Use isDirectory() method to log the result to screen
console.log('is directory ? ' + stats.isDirectory());
```

Output

```
is file ? true
is directory ? true
```

AIM:
To demonstrate the how to read, write, & close file

Objective:
Explantion of the how to read, write, & close file in node.js

Theory
Reading From Files:
Being able to read from files on your local file system can be hugely useful and there are a number of different things you can build on top of this. A log reader, importing information from spreadsheets and xml files or whatever you can think of, being able to read from files is hugely useful the file path is File, otherwise returns false. The stats.isDirectory() method returns true if file path is Directory, otherwise returns false

Explanation:

app.js

```
var fs = require("fs");
fs.readFile("temp.txt", function(err, buf) {
  console.log(buf.toString());
});
```

Create a temp.txt within the same directory and write in it anything you'd like. Run your script using node app.js and you should see in the console the contents of your file.

Understanding the Code
We'll step through this with comments.

```
var fs = require("fs");
```

This line does the job of importing the fs package and allowing us to utilize it within our own code.

```
fs.readFile("temp.txt", function(err, buf) {
console.log(buf);
});
```

AIM:
Demonstrate how to read data in SQL using node js

Objective:
Explanation how to read data in SQL using node js

Theory:

NoSQL databases are rather popular among Node developers, with MongoDB (the “M” in the MEAN stack) leading the pack. When starting a new Node project, however, you shouldn’t just accept Mongo as the default choice. Rather, the type of database you choose should depend on your project’s requirements. If, for example, you need dynamic table creation, or real-time inserts, then a NoSQL solution is the way to go. If your project deals with complex queries and transactions, on the other hand, an SQL database makes much more sense.

Explanation:
The steps for querying data in the MySQL database from a node.js application are as follows:

1. Establish a connection to the MySQL database server.
2. Execute a SELECT statement and process the result set.
3. Close the database connection.

Executing a simple query:

The following select.js program selects all data from the todos table of the todoapp database:

```
let mysql = require('mysql');
let config = require('./config.js');

let connection = mysql.createConnection(config);

let sql = `SELECT * FROM todos`;
connection.query(sql, (error, results, fields) => {
  if (error) {
    return console.error(error.message);
  }
  console.log(results);
});
connection.end();
```

Let’s run the select.js program.

```
>node select.js
```

```
[ RowDataPacket { id: 1, title: 'Learn how to insert a new row', completed: 1 },
  RowDataPacket { id: 2, title: 'Insert a new row with placeholders', completed:0 },
  RowDataPacket { id: 3, title: 'Insert multiple rows at a time', completed: 0 },
  RowDataPacket { id: 4, title: 'It should work perfectly', completed: 1 } ]
```

It returned 4 rows as expected.

Passing data to the query

The following select2.js program selects only completed todo:

```
let mysql = require('mysql');
let config = require('./config.js');

let connection = mysql.createConnection(config);

let sql = `SELECT * FROM todos WHERE completed=?`;
connection.query(sql, [true], (error, results, fields) => {
  if (error) {
    return console.error(error.message);
  }
  console.log(results);
});

connection.end();
```

In this example, we used the question mark (?) as the placeholder value of the completed field.

When we called the query() method, we passed an array as the second argument. The placeholders will be substituted by the values of the array in sequence.

```
>node select2.js
[ RowDataPacket { id: 1, title: 'Learn how to insert a new row', completed:
1 },
  RowDataPacket { id: 4, title: 'It should work perfectly', completed: 1 } ]
```

Code language: JavaScript (javascript)

The select2.js program returns two rows with the completed column is 1, which means true in Node.js

*****

WEB TECHNOLOGIES
MODULE 5
EXPERIMENT 1

AIM:

Write a simple program for multiplication

OBJECTIVE

The objective is to create a simple multiplier application which will multiply two numbers and display the result. User will enter two numbers in two separate textboxes and the result will be displayed immediately. This programme is to make the users understand the concepts of Templates, Directives and Expressions

THEORY:

Before understanding AngularJS, user should have a basic understanding of

- HTML
- CSS
- JavaScript

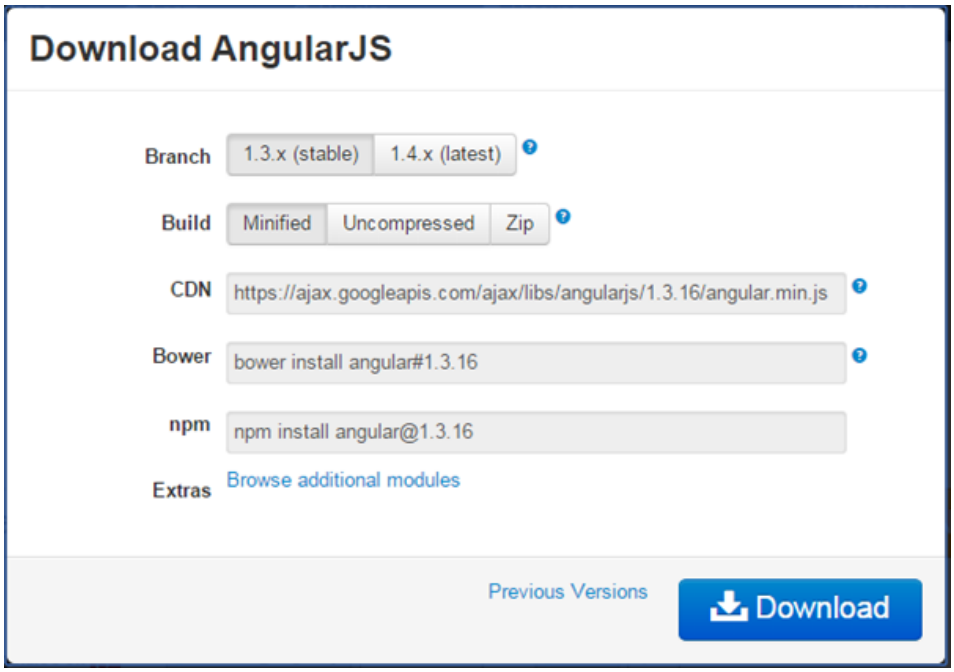
SETUP ANGULARJS DEVELOPMENT ENVIRONMENT:

The following tools arre needed to setup a development environment for AngularJS:

- AngularJS Library
- Editor/IDE
- Browser
- Web server

AngularJS Library:

To download AngularJS library, go to angularjs.org -> click download button, which will open the following popup.



Download AngularJS Library

Select the required version from the popup and click on download button in the popup.

CDN: AngularJS library can be included from CDN url - https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js

Editor:

AngularJS is eventually HTML and JavaScript code. So install any good editor/IDE as per your choice.

The following editors are recommended:

- Sublime Text
- Aptana Studio 3
- Ultra Edit
- Eclipse
- Visual Studio

Online Editor:

The following online editors can be used for learning purpose.

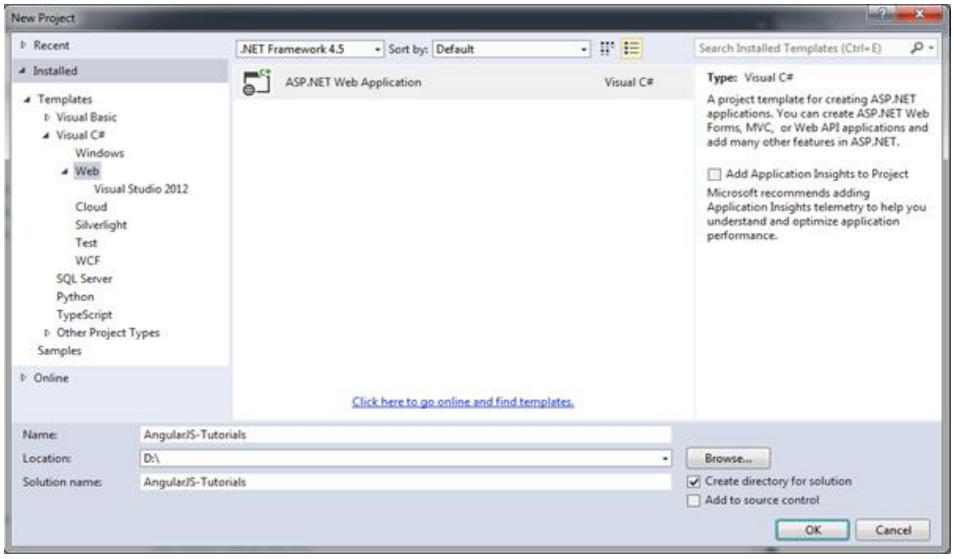
- plnkr.co
- jsbin.com

Web server:
Use any web server such as IIS, apache etc., locally for development purpose

Browser:
Install any browser of your choice as AngularJS supports cross-browser compatibility. However, it is recommended to use Google Chrome while developing an application.

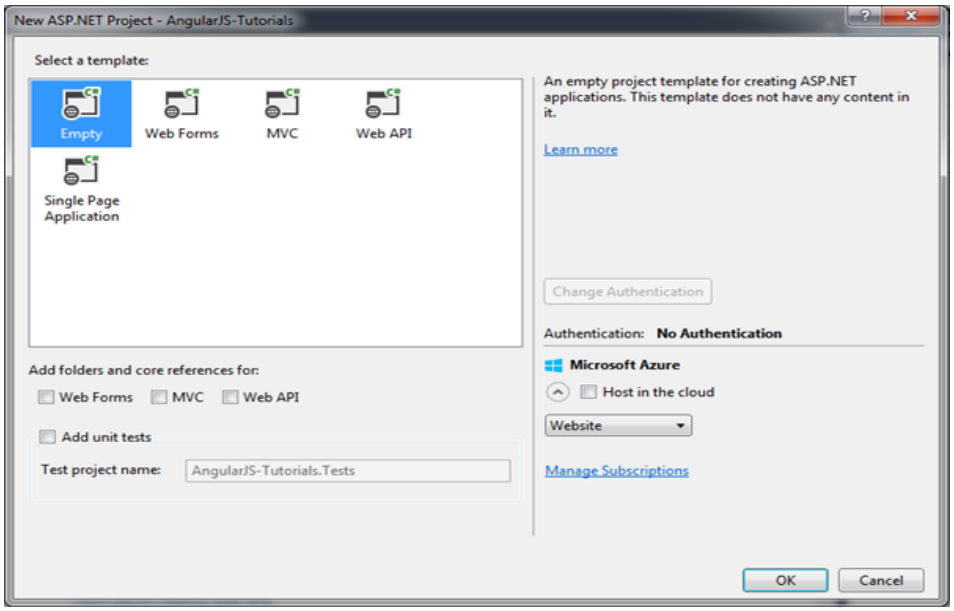
Angular Seed:
Use Angular seed project to quickly get started on AngularJS application. The Angular-seed is an application skeleton for a typical AngularJS web application. It can be used to quickly bootstrap your angular webapp projects and development environment for your project. Download angular-seed from GitHub

setup Angular project in Visual Studio 2013 for web:
Create AngularJS application in any version of Visual Studio. Here, Visual Studio 2013 for web is used. First, create new project by clicking on New Project link on start page. This will open New Project dialog box, as shown below.



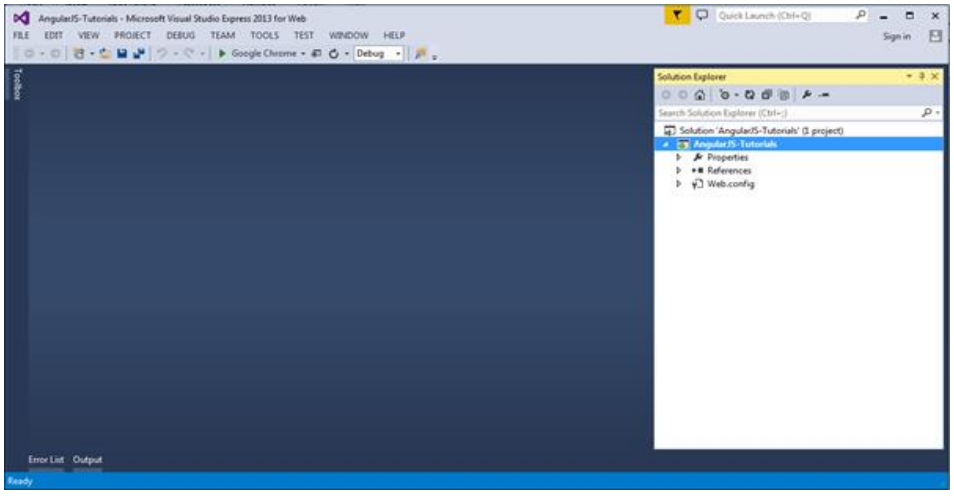
AngularJS in Visual Studio:

Select Web in the left pane and ASP.NET Web Application in the middle pane and then click OK. In the New ASP.NET Project dialog box, select Empty template and then click OK.



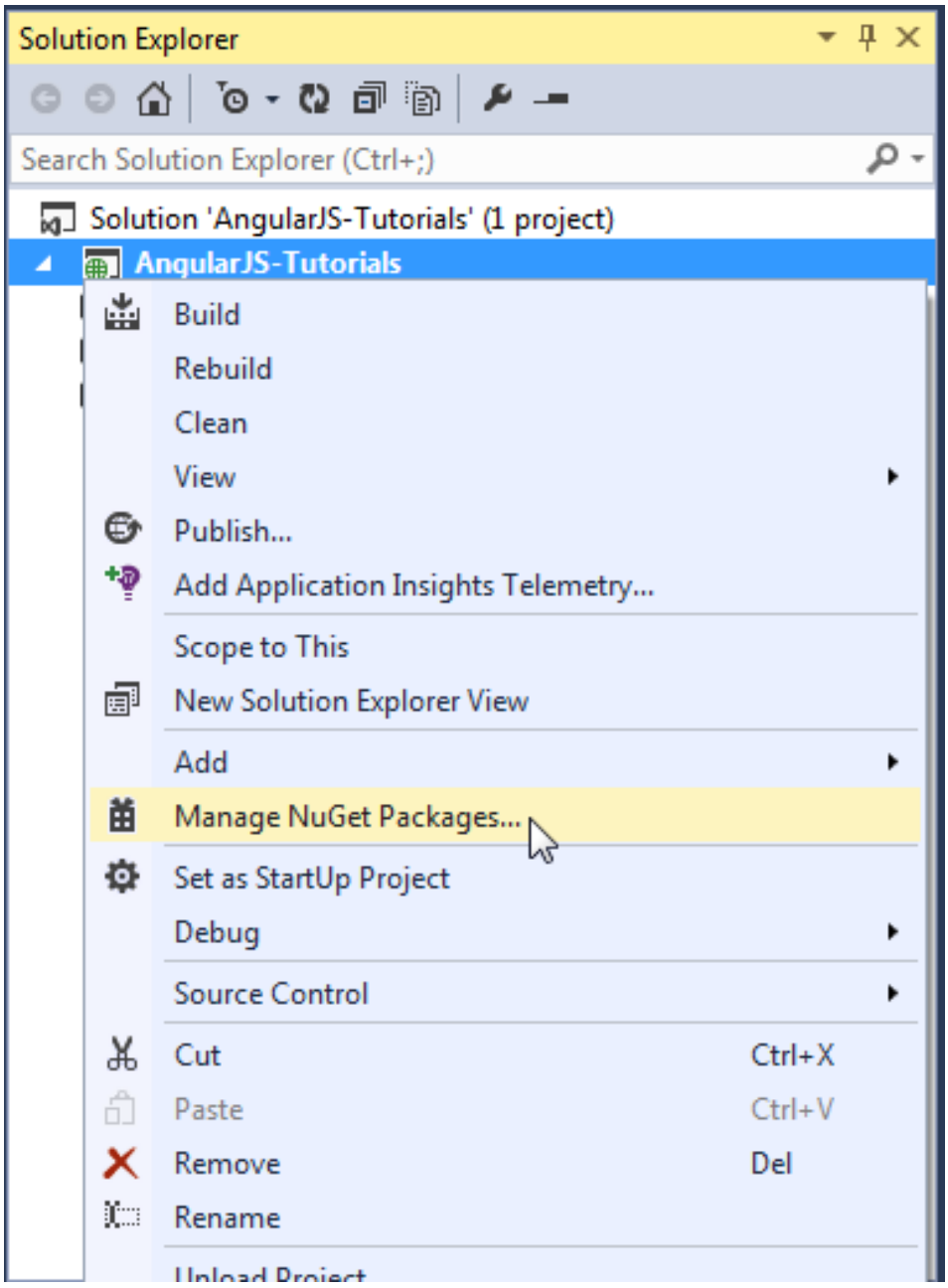
AngularJS in Visual Studio

This will create an empty website project in Visual Studio.



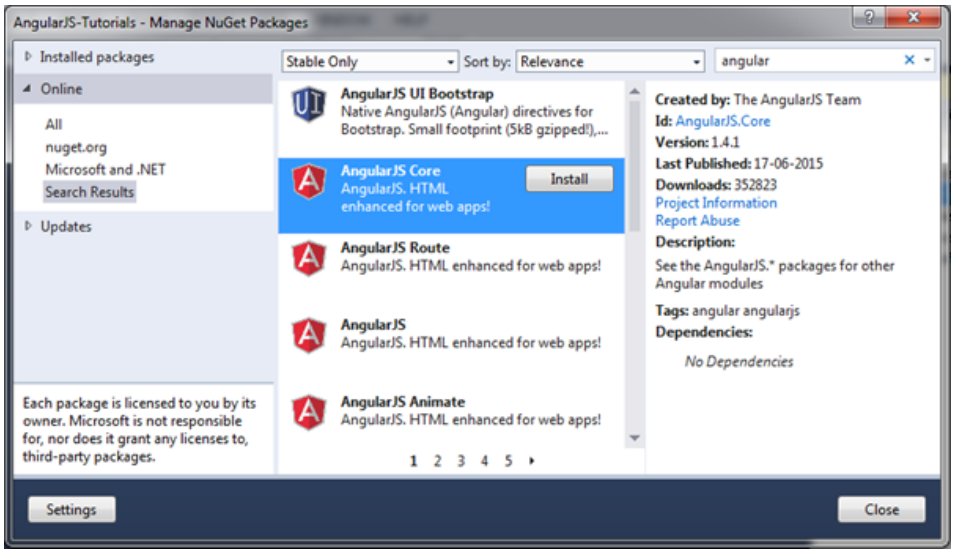
AngularJS in Visual Studio

Now, install AngularJS library from NuGet package manager. Right click on the project in Solution Explorer and select Manage NuGet Packages.



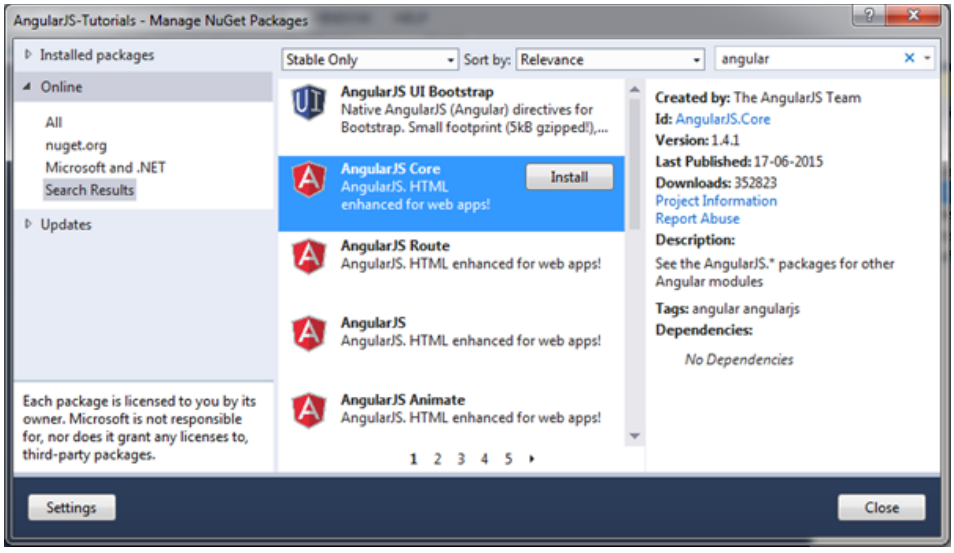
AngularJS in Visual Studio

Search for "angular" in the Manage NuGet Packages dialog box and install AngularJS Core.



AngularJS in Visual Studio

This will add AngularJS files into Scripts folder such as angular.js, angular.min.js, and angular-mocks.js, as shown below.



AngularJS in Visual Studio

Now, start writing AngularJS web application.

ALGORITHM:

STEP 1:
Create a HTML document with <head> and <body> elements, as shown below.

```
<!DOCTYPE html>

<html>
```

```
<head>

</head>
<body>

</body>
</html>
```

STEP 2:

Include angular.js file in the head section

```
<!DOCTYPE html>
<html>
<head>
 <title>First AngularJS Application</title>
 <script src= "~/Scripts/angular.js"></script>
</head>
<body>

</body>
</html>
```

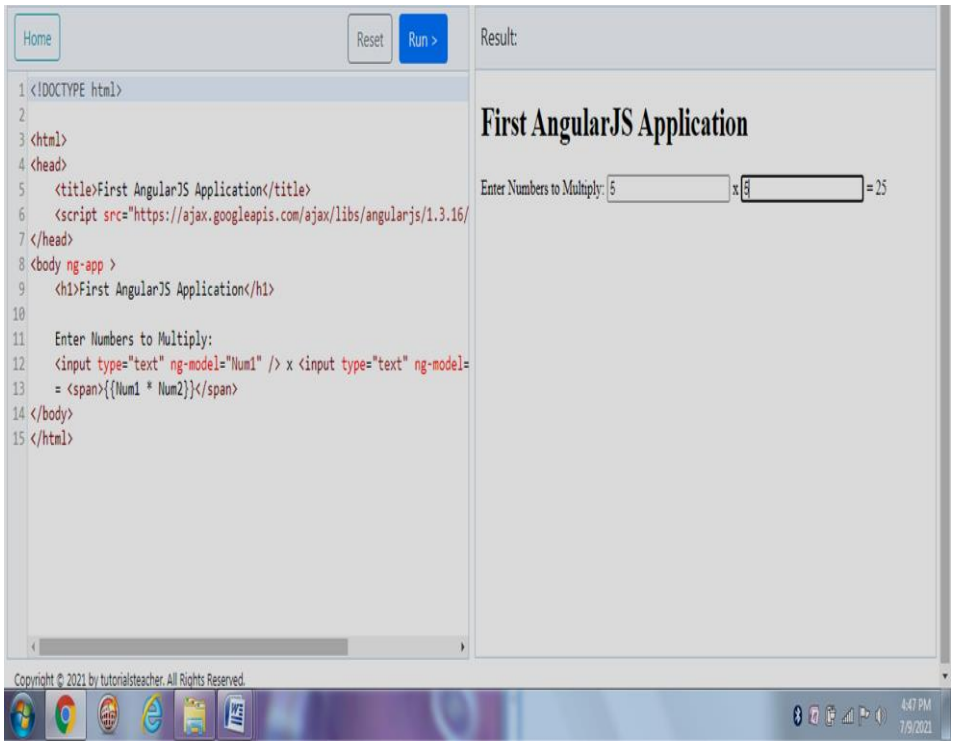
STEP 3:

Create a simple multiplier application which will multiply two numbers and display the result. Enter two numbers in two separate textboxes and the result will be displayed immediately, as shown below.

PROGRAM

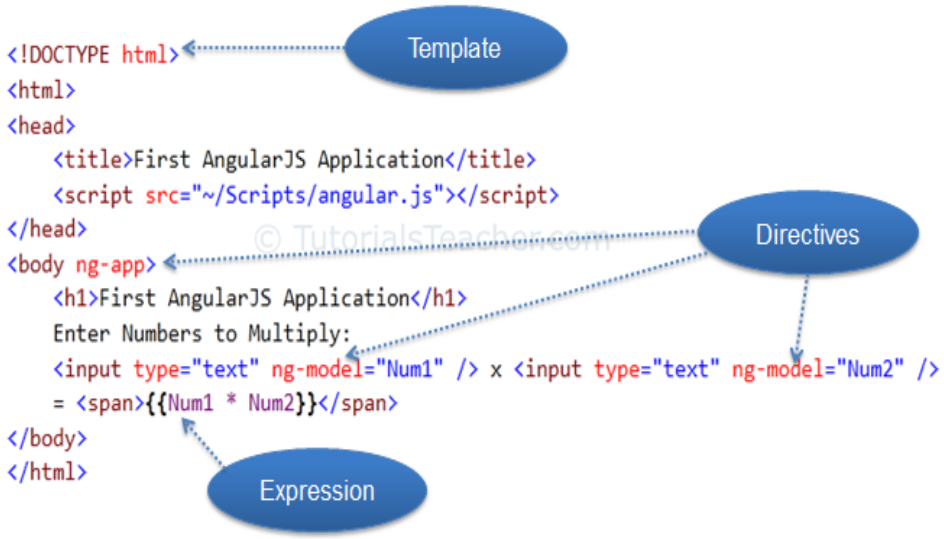
```
<!DOCTYPE html>

<html>
<head>
  <title>First AngularJS Application</title>
<script src= "~/Scripts/angular.js"></script>
</head>
<body ng-app >
<h1>First AngularJS Application</h1>

 Enter Numbers to Multiply:
        <input type="text" ng-model="Num1" /> x <input
type="text" ng-model="Num2" />
= <span>{{Num1 * Num2}}</span>
```

</body>
</html>
Try it



The above example looks like HTML code with some strange attributes and braces such as ng-app, ng-model, and {{ }}. These built-in attributes in AngularJS are called directives.

The following figure illustrates the AngularJS building blocks in the above example



First AngularJS Application

Template:
In AngularJS, a template is HTML with additional markups. AngularJS compiles templates and renders the resultant HTML.

Directive:
Directives are markers (attributes) on a DOM element that tell AngularJS to attach a specific behavior to that DOM element or even transform the DOM element and its children. Most of the directives in AngularJS are starting with **ng**. It stands for Angular.

ng-app: The ng-app directive is a starting point. If AngularJS framework finds ng-app directive anywhere in the HTML document then it bootstraps (initializes) itself and compiles the HTML template.

ng-model: The ng-model directive binds HTML element to a property on the $scope object.
In the above example, ng-model directive is added to both the textboxes with different names Num1 and Num2. AngularJS framework will create two properties called Num1 and Num2 in the scope and will assign a value that are typed into textboxes.

Expression:
An expression is like JavaScript code which is usually wrapped inside double curly braces such as {{ expression }}. AngularJS framework evaluates the expression and produces a result. In the above example, {{ Num1 * Num2}} will simply display the product of Num1 and Num2.

QUESTIONS

1. What are the major components of angularjs?
2. What is an expression?
3. What are directives?
4. What is {{ num1 * num2}}

a) Expression b) template c) directive d) none

5. Mention few editors used in angularjs?
6. What is angular seed?

*****

EXPERIMENT 2

AIM:

Write a program to display your name with welcome note :HELLO

OBJECTIVE

- Understand basic AngularJS components such as Modules, Directives, Expressions, Controllers, Services and Filters
- Understand the basic design of AngularJS
- Build AngularJS forms and bind data to objects

THEORY:

Before creating actual *Hello World !* application using AngularJS, let us see the parts of a AngularJS application. An AngularJS application consists of following three important parts −

- **ng-app:** This directive defines and links an AngularJS application to HTML.
- **ng-model:** This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind:** This directive binds the AngularJS Application data to HTML tags.

How AngularJS Integrates with HTML:

- The ng-app directive indicates the start of AngularJS application.
- The ng-model directive creates a model variable named name, which can be used with the HTML page and within the div having ng-app directive.
- The ng-bind then uses the name model to be displayed in the HTML tag whenever user enters input in the text box.
- Closing tag indicates the end of AngularJS application.

Very basic HTML page is extended with AngularJS **directives** by loading it from a CDN. AngularJS starts automatically when the web page has loaded.

- The **ng-app** directive tells AngularJS that the element is the owner of the application.
- The **ng-init** directive initializes AngularJS application variable message.
- AngularJS outputs data stored in the variable message where the **expression** is written. In this case <h1></h1>.

A CDN (Content Delivery Network) is a collection of global servers that caches and delivers content such as images, videos and Javascript files. By pulling AngularJS from a remote server we avoid having to store it loacally

on our computers. This technique is good to use during the development process but advised against for production code.

Interacting with a user:
See if AngularJS application is able to interact with some user generated data. How about letting the user add a name and have the “Hello”-message customized with that name?

ALGORITHM:

Creating AngularJS Application

Step 1 : Load framework
Being a pure JavaScript framework, it can be added using <Script> tag.

```
<script
  src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
```

Step 2: Define AngularJS application using ng-app directive

```
<div ng-app = "">
  ...
</div>
```

Step 3: Define a model name using ng-model directive

```
<p>Enter your Name: <input type = "text" ng-model = "name"></p>
```

Step 4: Bind the value of above model defined using ng-bind directive

```
<p>Hello <span ng-bind = "name"></span>!</p>
```

Step 5:
Executing AngularJS Application
Use the above-mentioned three steps in an HTML page. testAngularJS.htm

Step 6: Output
Open the file *testAngularJS.htm* in a web browser. Enter your name and see the result.

PROGRAM

testAngularJS.htm

```
<html>
  <head>
    <title>AngularJS First Application</title>
  </head>

  <body>
    <h1>Sample Application</h1>
```

```
    <div ng-app = "">
      <p>Enter your Name: <input type = "text" ng-model = "name"></p>
      <p>Hello <span ng-bind = "name"></span>!</p>
    </div>

    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
    </script>

  </body>
</html>
```

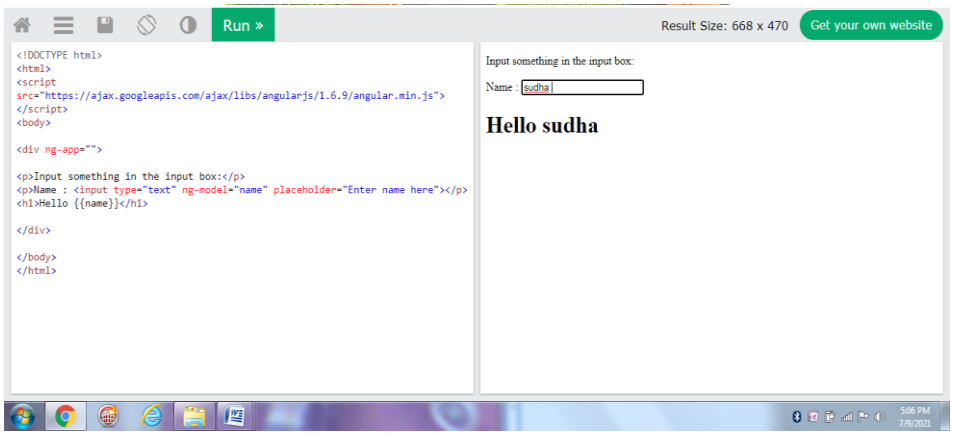
index.html

```
    <hr><!doctype html>
<html ng-app>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js">
</script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName" placeholder="Enter a name here">
      <h1>Hello {{yourName}}!</h1>
    </div>
  </body>
</html>
```

OUTPUT

Name: sudha

Hello sudha!



QUESTIONS

1. How angularjs integrates with HTML?
2. What are the parts of a AngularJS application?
3. Mention the tag used for loading the framework?
4. Define CDN?

*****

EXPERIMENT 3

AIM:
To create a real AngularJS Application for shopping Cart

OBJECTIVE

Use some of the AngularJS features to make a shopping list, where user can add or remove items:

Shopping List
- Milk×
- Bread×
- Cheese×

Add

THEORY:

A shopping cart is similar to original grocery shopping cart; it means that on a website that sells products or services online, the shopping cart is a common metaphor that acts as online store's catalog and ordering process. It is a graphical representation of a supermarket on a vendor's website that keeps a list of items a customer has picked up from the online store.

Shopping cart is an infrastructure that allows customers to review what they have selected, make necessary modifications such as adding or deleting products and purchase the merchandise. Customer checks off the products that are being ordered and when finished ordering, that proceeds to a page where the total order is confirmed and placed. Also, customers will enter their shipping tax information at the checkout. Shopping cart allows a website to build a catalog of products and integrate it into the website pages.

Shopping cart is important infrastructure to have smooth ecommerce transition. The shopping cart describes specialized content management system that includes,

- website wizards
- provides portal for catalog, order and customer management
- renders product data, product categories
- merchant tools
- shopping features
- payment options
- shipping and tax information
- passes transactional data to payment gateway
- statistics and security

ALGORITHM

Step 1.

Getting Started:
Start by making an application called myShoppingList, and add a controller named myCtrl to it.The controller adds an array named products to the current $scope. In the HTML, use the ng-repeat directive to display a list using the items in the array.

Example
Make a HTML list based on the items of an array:

```
<script>

var app = angular.module("myShoppingList", []);
app.controller("myCtrl", function($scope) {
  $scope.products = ["Milk", "Bread", "Cheese"];
});
</script>

<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x in products">{{x}}</li>
  </ul>
</div>
```

Step 2.

Adding Items:
In the HTML, add a text field, and bind it to the application with the ng-model directive. In the controller, make a function named addItem, and use the value of the addMe input field to add an item to the products array. Add a button, and give it an ng-click directive that will run the addItem function when the button is clicked.

Example
Now add items to our shopping list:

```
<script>
var app=angular.module("myShoppingList",[]);
app.controller("myCtrl", function($scope){
  $scope.products =["Milk", "Bread", "Cheese"];
  $scope.addItem = function (){
   $scope.products.push($scope.addMe);
  }
});
```

```
</script>

<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x      in        products">{{x}}</li>
  </ul>
  <input ng-model="addMe">
  <button ng-click="addItem()">Add</button>
</div>
```

Step 3.

Removing Items:
We also want to be able to remove items from the shopping list.

In the controller, make a function named removeItem, which takes the index of the item you want to remove, as a parameter. In the HTML, make a element for each item, and give them an ng-click directive which calls the removeItem function with the current $index.

Example
Now we can remove items from our shopping list:

```
<script>
var app=angular.module("myShoppingList",[]);
app.controller("myCtrl", function($scope){
  $scope.products =["Milk", "Bread", "Cheese"];
  $scope.addItem = function (){
   $scope.products.push($scope.addMe);
  }
 $scope.removeItem = function (x){
    $scope.products.splice(x, 1);
 }
});

</script>

<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x      in        products">
      {{x}}<span ng-click="removeItem($index)">&times;</span>
    </li>
  </ul>
  <input ng-model="addMe">
  <button ng-click="addItem()">Add</button>
</div>
```

Step 4.

Error Handling:
The application has some errors, like if you try to add the same item twice, the application crashes. Also, it should not be allowed to add empty items. We will fix that by checking the value before adding new items. In the HTML, we will add a container for error messages, and write an error message when someone tries to add an existing item.

Example
A shopping list, with the possibility to write error messages:

```
<script>
var app=angular.module("myShoppingList",[]);
app.controller("myCtrl", function($scope){
 $scope.products =["Milk", "Bread", "Cheese"];
  $scope.addItem = function (){
    $scope.errortext = "";
    if (!$scope.addMe){return;}
    if ($scope.products.indexOf($scope.addMe) ==-1){
      $scope.products.push($scope.addMe);
    } else {
      $scope.errortext = "The item is already in your shopping list.";
   }
  }
 $scope.removeItem = function (x){
    $scope.errortext = "";
    $scope.products.splice(x, 1);
 }
});

</script>

<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x      in        products">
      {{x}}<span ng-click="removeItem($index)">&times;</span>
    </li>
  </ul>
  <input ng-model="addMe">
  <button ng-click="addItem()">Add</button>
  <p>{{errortext}}</p>
</div>
```

Step 5.

Design:
The application works, but could use a better design. Use the W3.CSS stylesheet to style the application. Add the W3.CSS stylesheet, and include the proper classes throughout the application, and the result will be the same as the shopping list at the top of this page.

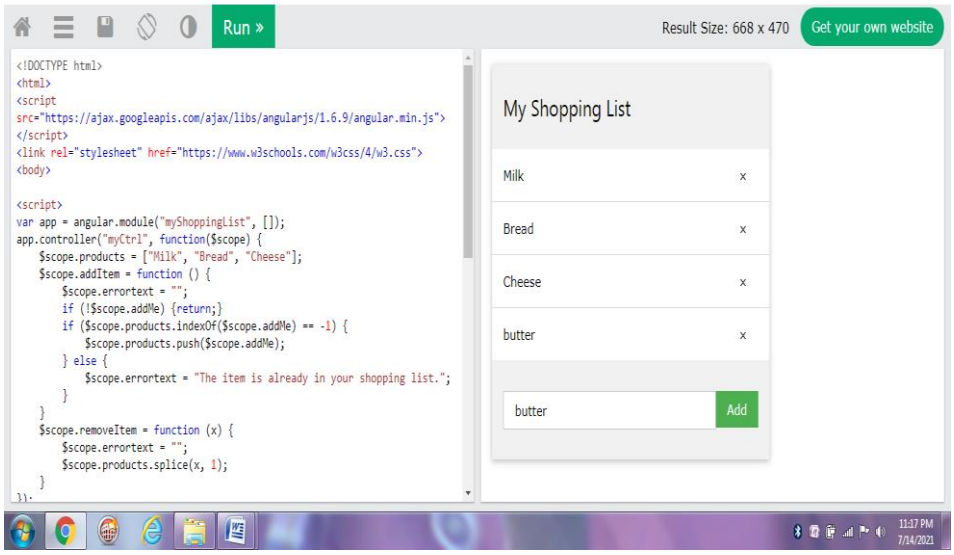
Example
Style your application using the W3.CSS stylesheet:

```
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css"
>
```

PROGRAM:

```
<script>
var app=angular.module("myShoppingList",[]);
app.controller("myCtrl", function($scope){
 $scope.products =["Milk", "Bread", "Cheese"];
  $scope.addItem = function (){
    $scope.errortext = "";
    if (!$scope.addMe){return;}
    if ($scope.products.indexOf($scope.addMe) ==-1){
      $scope.products.push($scope.addMe);
    } else {
      $scope.errortext = "The item is already in your shopping list.";
   }
  }
 $scope.removeItem = function (x){
    $scope.errortext = "";
    $scope.products.splice(x, 1);
 }
});

</script>

<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x      in        products">
      {{x}}<span ng-click="removeItem($index)">&times;</span>
    </li>
  </ul>
  <input ng-model="addMe">
  <button ng-click="addItem()">Add</button>
  <p>{{errortext}}</p>
</div>
```

OUTPUT:



QUESTIONS

1. What do you mean by shopping cart?
2. How do you apply styles to your application?
3. What is error handling?

*****

EXPERIMENT 4

AIM:

Show the use of md-dialog directive and mdDialog service and also the use of angular dialog boxes.

OBJECTIVE:

The **md-dialog**, an Angular Directive, is a container element and is used to display a dialog box. Its element, the **md-dialog-content**, contains the content of the dialog and the **md-dialog-actions** is responsible for the dialog actions. The **mdDialog**, an Angular Service, opens a dialog over the application to inform the users about the information and help them make decisions.

THEORY:

$mdDialog opens a dialog over the app to inform users about critical information or require them to make decisions. There are two approaches for setup: a simple promise API and regular object syntax.

Restrictions

- The dialog is always given an isolate scope.
- The dialog's template must have an outer <md-dialog> element. Inside, use an <md-dialog-content> element for the dialog's content, and use an <md-dialog-actions> element for the dialog's actions.
- Dialogs must cover the entire application to keep interactions inside of them. Use the parent option to change where dialogs are appended.

PROGRAM

am_dialog.htm

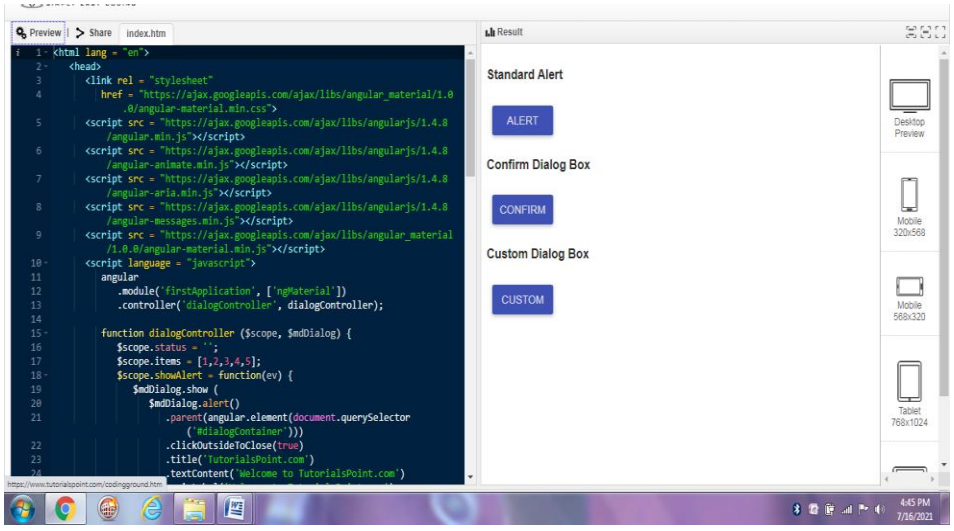
```
<html lang = "en">
  <head>
    <link rel = "stylesheet"

Href="https://ajax.googleapis.com/ajax/libs/angular_material/1.0.0/angular-material.min.css">
    <script      src
        ="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
    <script      src
        ="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-animate.min.js"></script>
    <script      src
        ="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-aria.min.js"></script>
```

```
    <script      src      =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-
messages.min.js"></script>
    <script      src      =
"https://ajax.googleapis.com/ajax/libs/angular_material/1.0.0/angular-
material.min.js"></script>
    <script language = "javascript">
      angular
        .module('firstApplication', ['ngMaterial'])
        .controller('dialogController', dialogController);

      function dialogController ($scope, $mdDialog) {
        $scope.status = '';
        $scope.items = [1,2,3,4,5];
        $scope.showAlert = function(ev) {
          $mdDialog.show (
            $mdDialog.alert()

.parent(angular.element(document.querySelector('#dialogContainer')))
              .clickOutsideToClose(true)
              .title('TutorialsPoint.com')
              .textContent('Welcome to TutorialsPoint.com')
              .ariaLabel('Welcome to TutorialsPoint.com')
              .ok('Ok!')
              .targetEvent(ev)
          );
        };

        $scope.showConfirm = function(event) {
          var confirm = $mdDialog.confirm()
            .title('Are you sure to delete the record?')
            .textContent('Record will be deleted permanently.')
            .ariaLabel('TutorialsPoint.com')
            .targetEvent(event)
            .ok('Yes')
            .cancel('No');
          $mdDialog.show(confirm).then(function() {
            $scope.status = 'Record deleted successfully!';
          }, function() {
            $scope.status = 'You decided to keep your record.';
          });
        };

        $scope.showCustom = function(event) {
          $mdDialog.show ({
            clickOutsideToClose: true,
            scope: $scope,
            preserveScope: true,
            template: '<md-dialog>' +
```

```
            ' <md-dialog-content>' +
            ' Welcome to TutorialsPoint.com' +
            ' </md-dialog-content>' +
            ' </md-dialog>',
          controller: function DialogController($scope, $mdDialog) {
            $scope.closeDialog = function() {
              $mdDialog.hide();
            }
          }
        });
      };
    }
  </script>
</head>

<body ng-app = "firstApplication">
  <div id = "dialogContainer" ng-controller = "dialogController as ctrl"
    layout = "row" ng-cloak>
    <md-content>
      <h4>Standard Alert</h4>
      <md-button class = "md-primary md-raised"
        ng-click = "showAlert($event)" flex = "100" flex-gt-md =
"auto">
        Alert
      </md-button>

      <h4>Confirm Dialog Box</h4>
      <md-button class = "md-primary md-raised" ng-click =
"showConfirm($event)"
        flex = "100"  flex-gt-md = "auto">
        Confirm
      </md-button>

      <h4>Custom Dialog Box</h4>
      <md-button class = "md-primary md-raised" ng-click =
"showCustom($event)"
        flex = "100"  flex-gt-md = "auto">
        Custom
      </md-button>

      <div ng-if = "status">
        <br/>
        <b layout = "row" layout-align = "center center" class = "md-
padding">
          {{status}}
        </b>
      </div>

    </md-content>
```

```
    </div>
  </body>
</html>
```

OUTPUT



QUESTIONS

1. What is **md-dialog** and its elements?
2. What are the restrictions of **md-dialog** directive?
3. What are the two basic approaches for setup of **md-dialog?**

*****

MODULE 5
EXPERIMENT 5

AIM:
To Develop a Single Page Application Using AngularJS

OBJECTIVE:
Create single page applicaiton in which all the code (JS, HTML, CSS) is loaded when application loads for the first time. Loading of the dynamic contents and the navigation between pages is done without refreshing the page.

THEORY:
Traditionally, applications were Multi-Page Application (MPA) where with every click a new page would be loaded from the server. This was not only time consuming but also increased the server load and made the website slower. AngularJS is a JavaScript-based front-end web framework based on bidirectional UI data binding and is used to design Single Page Applications. Single Page Applications are web applications that load a single HTML page and only a part of the page instead of the entire page gets updated with every click of the mouse. The page does not reload or transfer control to another page during the process. This ensures high performance and loading pages faster. Most modern applications use the concept of SPA. In the SPA, the whole data is sent to the client from the server at the beginning. As the client clicks certain parts on the webpage, only the required part of the information is fetched from the server and the page is rewritten dynamically. This results in a lesser load on the server and is cost-efficient. SPAs use AJAX and HTML5 to create a fluid and responsive Web applications and most of the work happens on the client-side. Popular applications such as Facebook, Gmail, Twitter, Google Drive, Netflix, and many more are examples of SPA.

SPAs are good when the volume of data is small and the website that needs a dynamic platform. It is also a good option for mobile applications. But businesses that depend largely on search engine optimizations such as e-commerce applications must avoid single-page applications and opt for MPAs.

ALGORITHM

Step 1:
Create a Module
AngularJS follows MVC architecture, hence every AngularJS application contains a module comprising of controllers, services, etc.
var app = angular.module('myApp', []);

Step 2:

Define a Simple Controller

```
app.controller('FirstController', function($scope) {
$scope.message = 'Hello from FirstController';
});
```

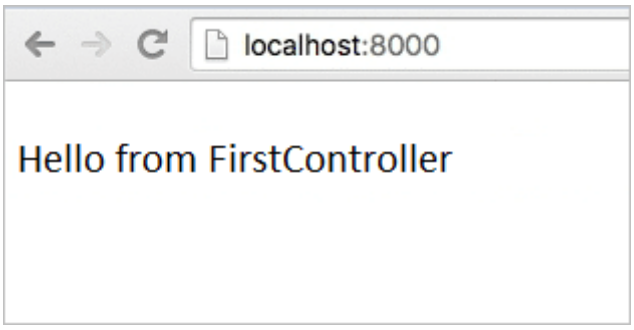
Step 3:

Include AngularJS script in HTML code

Specify the module (created in step 1) in **ng-app** attribute and the controller (defined in step 2) in **the ng-controller** attribute.

```
<!doctype html>
<html ng-app="myApp">
<head>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular.min.js"
></script>
</head>
<body ng-controller="FirstController">
<h1>{{message}}</h1>
<script src="app.js"></script>
</body>
</html>
```

Once the code is run on localhost, the **output** will be as shown below.



It is now confirmed that our module and controller are set up, and AngularJS is working properly.

Step 4:

Use AngularJS's routing capabilities to add different views to our SPA

Include **angular-route** script after the main angular script.

```
<script src="https://cdnjs. cloudflare.com/ajax/libs/angular.js /1.4.7
/angular.min.js"> </script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular-
route.min.js"></script>
```

Use the **ngRoute** directive to enable routing.

```
var app = angular.module('myApp', ['ngRoute']);
```

Step 5:

Create an HTML layout for the website

Once an HTML layout for the website is created, use the **ng-view** directive to specify the place where the HTML of each page will be placed in our layout.

```
<!doctype html>
<html ng-app="myApp">
<head>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular.min.js"
></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular-
route.min.js"></script>
</head>
<body>
<div ng-view></div>
<script src="app.js"></script>
</body>
</html>
```

Step 6:

Use $routeProvider service from ngRoute module to configure the navigation to different views

It is necessary to specify a templateUrl and a controller for each route that we wish to add.

Exception handling has to be accommodated when a user tries to navigate to a route that doesn't exist. For simplicity, we can write an "otherwise" function to redirect the user to the "/" route.

```
var app = angular.module('myApp', ['ngRoute']);

app.config(function($routeProvider) {
$routeProvider
 .when('/', {
templateUrl : 'pages/first.html',
controller : 'FirstController'
})
 .when('/blog', {
templateUrl : 'pages/second.html',
controller : 'SecondController'
})
.when('/about', {
templateUrl : 'pages/third.html',
controller : 'ThirdController'
})
 .otherwise({redirectTo: '/'});
});
```

Step 7:

Build controllers for every route specified in $routeProvider

```
app.controller('FirstController', function($scope) {
$scope.message = 'Hello from FirstController';
});
app.controller('SecondController', function($scope) {
$scope.message = 'Hello from SecondController';
});

app.controller('ThirdController', function($scope) {
$scope.message = 'Hello from ThirdController';
});
```

Step 8:
Configure the pages

first.html

```
<h1>First</h1>
<h3>{{message}}</h3>
```

second.html

```
<h1>Second</h1>
<h3>{{message}}</h3>
```

third.html

```
<h1>Third</h1>
<h3>{{message}}</h3>
```

Step 9:
Add links to the HTML that will help in navigating to the configured pages

```
<!doctype html>
<html ng-app="myApp">
<head>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular.min.js"
></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular-
route.min.js"></script>
</head>
<body>
<a href="#/">First</a>
<a href="#/second">Second</a>
<a href="#/third">Third</a>
<div ng-view></div>
<script src="app.js"></script>
</body>
</html>
```

Step 10:

Include the HTML of routing pages to index.html file using script tag

```
<!doctype html>
<html ng-app="myApp">
<head>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular.min.js"
></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular-
route.min.js"></script>
</head>
<body>
<script type="text/ng-template" id="pages/first.html">
<h1>First</h1>
<h3>{{message}}</h3>
</script>
<script type="text/ng-template" id="pages/second.html">
<h1>Second</h1>
<h3>{{message}}</h3>
</script>
<script type="text/ng-template" id="pages/third.html">
<h1>Third</h1>
<h3>{{message}}</h3>
</script>
<a href="#/">First</a>
<a href="#/second">Second</a>
<a href="#/third">Third</a>
<div ng-view></div>
<script src="app.js"></script>
</body>
</html>
```

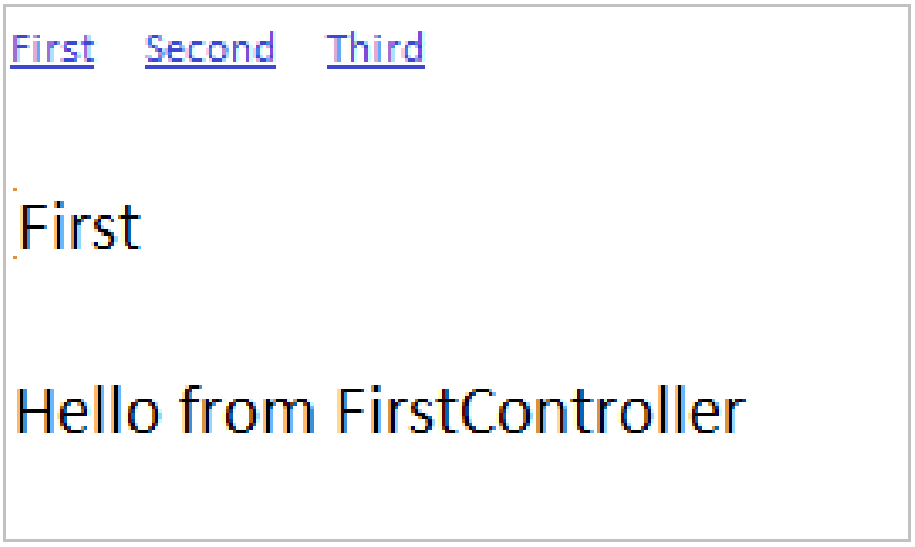
(When Angular detects the templates defined by the ng-template directives, it will load its content to the template cache and will not perform Ajax request to get their content.)

PROGRAM:

```
<!doctype html>
<html ng-app="myApp">
<head>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular.min.js"
></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular-
route.min.js"></script>
</head>
<body>
```

```
<script type="text/ng-template" id="pages/first.html">
<h1>First</h1>
<h3>{{message}}</h3>
</script>
<script type="text/ng-template" id="pages/second.html">
<h1>Second</h1>
<h3>{{message}}</h3>
</script>
<script type="text/ng-template" id="pages/third.html">
<h1>Third</h1>
<h3>{{message}}</h3>
</script>
<a href="#/">First</a>
<a href="#/second">Second</a>
<a href="#/third">Third</a>
<div ng-view></div>
<script src="app.js"></script>
</body>
</html>
```

Once the HTML is run on localhost, the following page is displayed.



Observe that the hyperlinks **First, Second, Third** on the page are routers and when you click on them, navigation to the corresponding web pages occur, without refreshing.

QUESTIONS

1. List few applications of SPA?
2. Mention the steps to define a Simple Controller?
3. How to Add links to the HTML that will help in navigating to the configured pages?

*****

EXPERIMENT 6

AIM:
Create Simple User Registration Form in AngularJS

OBJECTIVE

User Registration is very basic and common feature in modern web application. It is one of the basic and most important features for a web application that is used to authenticate or restrict unauthorized access to member only areas and features in a web application. The user is required to register an account using the registration form provided on the application so the username and password is created in order to login in the application.

THEORY:
Most of the websites features a Login and Registration link. This is done to authenticate a user and to provide some extra features/privileges to the user. The user needs to register himself first so that the username and password get created in order to enable the login functionality. This article, leads the user through a step by step process to understand the User Registration in an Angular platform.

Following are the salient features of this Registration example.

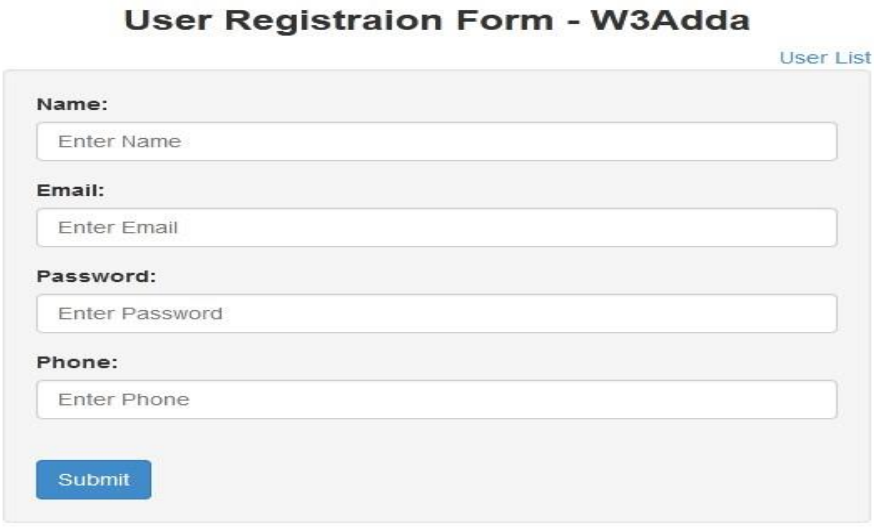
- User Registration form has following basic fields.
  1. Name
  2. Email
  3. Password
  4. Phone

- The form will have a hyperlink (top-right corner) to see a **list of users** which will list all the registered users.
- Below the form, a list of registered users with **Edit/Delete** features is enabled.
- When the user goes to see the list of users, he can simply click on the **Back button** to come back to the registration form.
- Initially, when the application is loaded, the form gets displayed. To add a new user, enter all the details as follows and click on the Submit button.
- To update a user, use the Edit action which will display the current details in form fields. Change the values and click on Submit button to save the latest details. The updated details are displayed in the registered user's table.

- To remove a user from the application, use delete action which will erase all the details pertaining to that user and the user row will be deleted from the table.
- To see all the users, click on the hyperlink 'List of Users' on the top-right corner of the page. It will take you to the next page where the user details are displayed.

ALGORITHM:

Step 1:
Create registration form
Create a registration form that allows users to create a new account by filling out a web form. User Registration form will have **Name, Email, Password and Phone** as basic registration fields. Registration form will look like as following –



Step 2
Create **index.html** and **app.js**

Create **index.html** in project's root directory and put the following code in it

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>User Registraion Form - W3Adda</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></
script>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.min.js"
></script>
```

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
<script src="app.js"></script>
</head>
<body>
<div ng-app = "myApp" class = "container" style="width:550px">
<div style="text-align:center;">
<h3><b>User Registraion Form - W3Adda</b></h3>
</div>
<div ng-controller = "RegisterController">
<div align="right">
<a href="#" ng-click="searchUser()">{{title}}</a>
</div>
<form role = "form" class="well" ng-hide="ifSearchUser">
<div class = "form-group">
<label for = "name"> Name: </label>
<input type = "text" id = "name" class = "form-control" placeholder = "Enter Name " ng-model = "newuser.name">
</div>
<div class = "form-group">
<label for = "email"> Email: </label>
<input type = "email" id = "email" class = "form-control" placeholder = "Enter Email " ng-model = "newuser.email">
</div>
<div class = "form-group">
<label for = "password"> Password: </label>
<input type = "password" id = "password" class = "form-control" placeholder = "Enter Password " ng-model = "newuser.password">
</div>
<div class = "form-group">
<label for = "phone"> Phone: </label>
<input type = "text" id = "phone" class = "form-control" placeholder = "Enter Phone " ng-model = "newuser.phone">
</div>
<br>
<input type="hidden" ng-model="newuser.id">
<input type="button" class="btn btn-primary" ng-click="saveUser()" class="btn btn-primary" value = "Submit">
</form>
<div><h4><b>Registered Users</b></h4>
<table ng-if="users.length" class = "table table-striped table-bordered table-hover">
<thead>
<tr class = "info">
<th>Name</th>
<th>Email</th>
```

```
<th>Phone</th>
<th ng-if="!ifSearchUser">Action</th>
</tr>
</thead>
<tbody>
<tr ng-repeat = "user in users">
<td>{{ user.name }}</td>
<td>{{ user.email }}</td>
<td>{{ user.phone }}</td>
<td ng-if="!ifSearchUser">
<a href="#" ng-click="edit(user.id)" role = "button" class = "btn btn-info">edit</a>  
<a href="#" ng-click="delete(user.id)" role = "button" class = "btn btn-danger">delete</a>
</td>
</tr>
</tbody>
</table>
<div ng-hide="users.length > 0">No Users Found</div>
</div>
</div>
</div>
</body>
</html>
```

On top of the the form **User List** link is available to list all the registered users. Right below the form, list of registered users along with **Edit/Delete** button is displayed.

Step 3:
Create AngularJS Application
In this step, create a javascript file **app.js** in project's root directory and define a AngularJS application.

app.js

```
var myApp = angular.module("myApp", []);
```

Step 4:
Create Registration Service
Create a registration service that will be used to add, list, edit and delete users. Let's open **app.js** file and append the following code in it –

```
myApp.service("RegisterService" , function(){
var uid = 1;
var users = [{
'id' : 0,
'name' : 'John Doe',
'email' : 'johndoe@gmail.com',
'password': 'johndoe',
```

```
'phone' : '123-45-678-901'}];

// Save User
this.save = function(user)
{
if(user.id == null)
{
user.id = uid++;
users.push(user);
}
else
{
for(var i in users)
{
if(users[i].id == user.id)
{
users[i] = user;
}
}
}
};

// Search User
this.get = function(id)
{
for(var i in users )
{
if( users[i].id == id)
{
return users[i];
}
}
};

// Delete User
this.delete = function(id)
{
for(var i in users)
{
if(users[i].id == id)
{
users.splice(i,1);
}
}
};

// List Users
this.list = function()
```

```
 {
 return users;
 };
});
```

Step 5:

Create Registration Controlle

Create a registration controller that will bind add, list, edit and delete action to registration service. Let's open **app.js** file and append the following code in it –

app.js

```
 Register Controller
myApp.controller("RegisterController", function($scope , RegisterService){
console.clear();
$scope.ifSearchUser = false;
$scope.title ="User List";
$scope.users = RegisterService.list();
$scope.saveUser = function()
{
 console.log($scope.newuser);
 if($scope.newuser == null || $scope.newuser == angular.undefined)
 return;
 RegisterService.save($scope.newuser);
 $scope.newuser = {};
};
$scope.delete = function(id)
{
 RegisterService.delete(id);
 if($scope.newuser != angular.undefined && $scope.newuser.id == id)
 {
 $scope.newuser = {};
 }
};
$scope.edit = function(id)
{
 $scope.newuser = angular.copy(RegisterService.get(id));
};
$scope.searchUser = function(){
 if($scope.title == "User List"){
 $scope.ifSearchUser=true;
 $scope.title = "Back";
 }
 else
 {
 $scope.ifSearchUser = false;
 $scope.title = "User List";
 }
```

```
};
});
```

Step 6:

Run AngularJS Application

Now start the application server and visit the application to view the output.

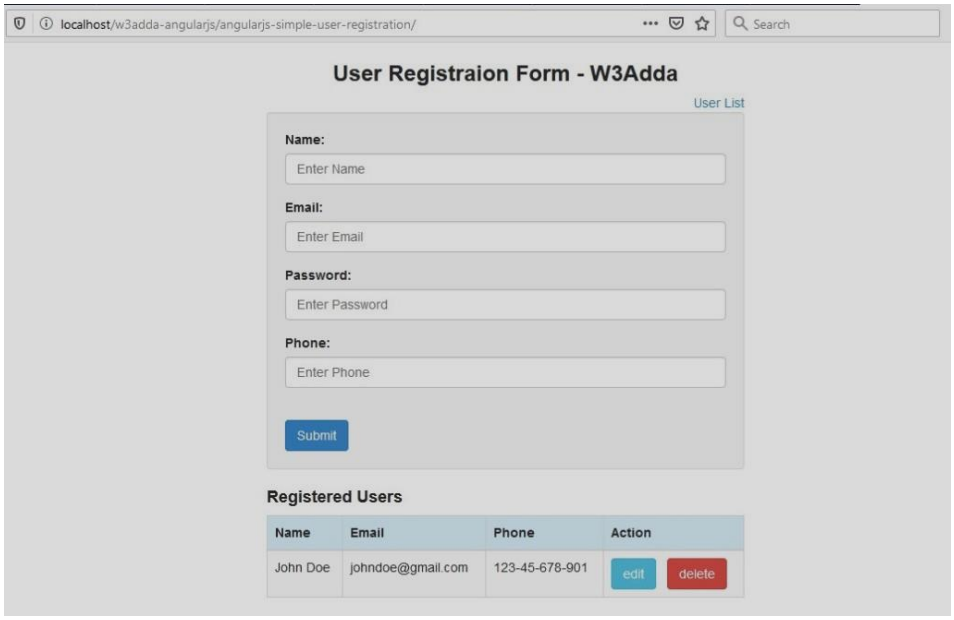
PROGRAM:

```
 var myApp = angular.module("myApp", []);
// Register Service
myApp.service("RegisterService" , function(){
var uid = 1;
var users = [{
'id' : 0,
'name' : 'John Doe',
'email' : 'johndoe@gmail.com',
'password': 'johndoe',
'phone' : '123-45-678-901'}];

 // Save User
 this.save = function(user)
 {
 if(user.id == null)
 {
 user.id = uid++;
 users.push(user);
 }
 else
 {
 for(var i in users)
 {
 if(users[i].id == user.id)
 {
 users[i] = user;
 }
 }
 }
 };

 // Search User
 this.get = function(id)
 {
 for(var i in users )
 { if( users[i].id == id)
 {
 return users[i];
```

```
 }
 }
 };
// Delete User
 this.delete = function(id)
 {
 for(var i in users)
 {
 if(users[i].id == id)
 {
 users.splice(i,1);
 }
 }
 };

 // List Users
 this.list = function()
 {
 return users;
 };
});

// Register Controller
myApp.controller("RegisterController" , function($scope , RegisterService){
console.clear();
$scope.ifSearchUser = false;
$scope.title ="User List";
$scope.users = RegisterService.list();
$scope.saveUser = function()
{
 console.log($scope.newuser);
 if($scope.newuser == null || $scope.newuser == angular.undefined)
 return;
 RegisterService.save($scope.newuser);
 $scope.newuser = {};
};
$scope.delete = function(id)
{
 RegisterService.delete(id);
 if($scope.newuser != angular.undefined && $scope.newuser.id == id)
 {
 $scope.newuser = {};
 }
};
$scope.edit = function(id)
```

```
{
 $scope.newuser = angular.copy(RegisterService.get(id));
};
$scope.searchUser = function(){
 if($scope.title == "User List"){
 $scope.ifSearchUser=true;
 $scope.title = "Back";
 }
 else
 {
 $scope.ifSearchUser = false;
 $scope.title = "User List";
 }
};
});
```

OUTPUT



QUESTIONS

1. What is the use of registration controller?
2. What is the difference between registration service and controller?
3. Write the steps to create angularjs application?

EXPERIMENT 7

AIM:
To implement form validation in angular js.

OBJECTIVE
Understand form and input field controls validations in angularjs, different validation properties of form and input fields in angularjs and different validation classes of form and input field controls in angularjs. since all forms come across sending data to server which is a validate input from the user. So that desired value can be submitted from the form.

THEORY:
AngularJS Form Validations:

A form is a collection of controls like input textbox, number text, email text and checkbox etc. In angularjs we can validate form and input field controls (text, select, textarea) in client side and notify users for invalid data while submitting forms with better user experience when compared to server side validations. In angularjs form input field controls arE validated by using HTML5 property **required** or angularjs property **ng-required**.

What is required or ng-required ? How to use it?

Generally by using html5 **required** and angularjs **ng-required** attributes required field can be validated in form. Both **required** and **ng-required** will do the same validation so either of them can be used based on the interest. Following is the way of defining **required** and **ng-required** properties in angularjs applications.

1. <input required>

2. <input ng-required="true"> are essentially the same thing.

Both of them can be used as per the preference but as per angularjs its preferred use **ng-required** attribute.
Syntax of AngularJS Form Validation with Input Fields
Following is the syntax of implementing validations to input fields on form in angularjs applications.

```
<form name="personForm">
<input type="text" name="firstname" ng-
model="person.fname" required />
<span ng-
show="personForm.firstname.$error.required">Required!</span>
</form>
```

Observe the above syntax where it is defined **required** to input control and showing **span** element based on validation of input control.

AngularJS Form Validations:

Following is the simple angularjs example to validate input field controls in form.

```
<!DOCTYPE html>
<html>
<head>
<title>
AngularJs Validate Controls in Form
</title>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<script type="text/javascript">
var app = angular.module('formApp', []);
app.controller('formCtrl', function ($scope) {
$scope.sendForm = function () {
$scope.msg = "Form Validated";
};
});
</script>
</head>
<body>
<div ng-app="formApp" ng-controller="formCtrl">
<form name="personForm" ng-submit="sendForm()">
First Name:<input type="text" name="firstname" ng-model="person.fname" required />
<span ng-show="personForm.firstname.$error.required"> Required! </span>
<br /><br />
Last Name:<input type="text" name="lastname" ng-model="person.lname" required />
<span ng-show="personForm.lastname.$error.required"> Required! </span>
<br /><br />
<button type="submit">Submit Form</button><br /><br />
<span>{{msg}}</span>
</form>
</div>
</body>
</html>
```

Observe above example where property “required” is defined to textbox controls and showing error messages in case if those controls are empty.

AngularJS Form and Input Field Validation States:

Angularjs have different properties available for form and input fields that helps to validate form controls. Following are the different properties available for **form** validation in angularjs and all these properties will return either **true** or **false** based on status.

Property	Description
$pristine	This property helps to know whether form elements has been modified or not. It will return true if no form elements has been modified yet
$dirty	It returns true if any form elements modified
$invalid	It tells whether form element invalid or not based on rules
$valid	It tells whether form element valid or not based on rules

Following are the different properties available for **input controls** for validation in angularjs and all these properties will return either **true** or **false**.

Property	Description
$pristine	This property helps to know whether form elements has been modified or not. It will return true if no form elements has been modified yet
$dirty	It returns true if any form elements modified
$invalid	It tells whether field invalid or not based on rules
$valid	It tells whether field valid or not based on rules
$touched	It returns true if field has touched or blurred
$untouched	It tells field has not been touched or blurred yet

Accessing AngularJS Properties of Form and Input Fields

Access properties of form and input fields for validationsas shown below

- Access the form properties like **<form name>.<angular property>**

- Access the Input field properties like **<form name>.<input name>.<angular property>**

AngularJS Form with Input Fields Validation:

Following uses different properties of form and input fields in angularjs application to show validation messages to user based on requirements.

```
<!DOCTYPE html>
<html>
<head>
<title>
AngularJs Form and Input Validation Controls
</title>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
</head>
<body>
<div ng-app="">
<form name="personForm">
First Name: <input type="text" name="firstname" ng-model="person.fname" required />
<span ng-show="personForm.firstname.$touched && personForm.firstname.$invalid "> Required! </span><br /><br />
Last Name:<input type="text" name="lastname" ng-model="person.lname" required />
<span ng-show="personForm.lastname.$dirty && personForm.lastname.$valid"> Thanks for Text </span><br /><br />
</form>
</div>
</body>
</html>
```

Observe above example for **first name** textbox which is showing error message when control was touched or blurred and if there is no input text. Same way for **last name** textbox shows whether textbox content modified or not using **$dirty** property and textbox containing text or

AngularJS CSS Classes for Form Validation:

In angularjs different classes added to forms for validation based on their status. Following are the classes add or removed from **forms** based on the statuses.

Class	Description
ng-pristine	No fields in form has been not modified yet
ng-dirty	One or more fields in form has been modified
ng-valid	It tells whether form content valid or not based on rules
ng-invalid	It tells whether form content invalid or not based on rules

AngularJS Applying Classes for Form Validation

Following code uses classes for form validation in angularjs applications.

```
<!DOCTYPE html>
```

```
<html>
<head>
<title>
AngularJs Applying classes for Form Validation
</title>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<style>
form.ng-invalid {
background-color:#f2672e;
}
form.ng-valid {
background-color:#2b8bc0;
}
</style>
</head>
<body>
<div ng-app="">
<form name="personForm">
<br />
First Name:<input type="text" name="firstname" ng-model="person.fname" required /><br /><br />
Last Name:<input type="text" name="lastname" ng-model="person.lname" required />
<br /><br />
</form>
</div>
</body>
</html>
```

In the above code classes are written classes form **ng-valid** and **ng-invalid** properties.

AngularJS CSS Classes for Input Field Controls Validation
In angularjs different classes added to input field controls for validation based on their status. Following are the classes add or removed from **input controls** based on the statuses.

Class	Description
ng-pristine	It returns **true** if field has not been modified yet
ng-dirty	It returns **true** if field has been modified
ng-valid	It tells whether field content valid or not based on rules
ng-invalid	It tells whether field content invalid or not based on rules
ng-touched	It tells field has been touched or blurred
ng-untouched	It tells field has been not touched or blurred yet

AngularJS Applying Classes for Input Field Validation:

Following code uses classes for input fields validation in angularjs applications.

```
<!DOCTYPE html>
<html>
<head>
<title>
AngularJs Applying classes for Input Fields Validation
</title>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<style>
input.ng-invalid {
background-color:#f2672e;
}
input.ng-valid {
background-color:#2b8bc0;
}
</style>
</head>
<body>
<div ng-app="">
<form name="personForm">
<br />
First Name:<input type="text" name="firstname" ng-model="person.fname" required /> <br /><br />
Last Name:<input type="text" name="lastname" ng-model="person.lname" required />
<br /><br />
</form>
</div>
</body>
</html>
```

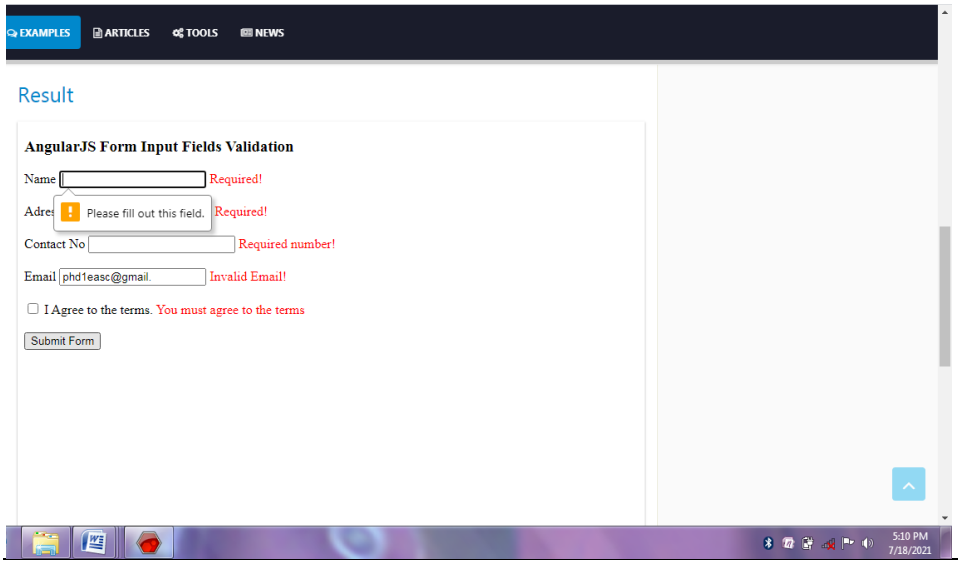
In the above code classes are written for input fields **ng-valid** and **ng-invalid** properties.

PROGRAM

```
<!DOCTYPE html>
<html>
<head>
<title>
AngularJs Form Input Fields Validation Example
</title>
```

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.m
in.js"></script>
<script>
var app = angular.module('formApp', []);
app.controller('formCtrl', function ($scope) {

$scope.sendForm = function () {
$scope.msg='Form Submited Successfully';
};

$scope.getClass = function (color) {
return color.toString();
}
});
</script>
<style>
.valid.false {
background: red;
}
.valid.true {
background: green;
}
.error {
color: red;
}
</style>
</head>
<body ng-app="formApp" ng-controller="formCtrl">
<h3>Form validation demo app in AngularJs</h3>
<form name="personForm" ng-submit="sendForm()">
<label for="name">Name</label>
<input id="name" name="name" type="text" ng-
model="person.name" required />
<span class="error" ng-
show="personForm.name.$error.required"> Required! </span>
<br /><br />
<label for="adress">Adress</label>
<input id="address" name="address" type="text" ng-
model="person.address" required />
<span class="error" ng-
show="personForm.address.$error.required"> Required! </span>
<br /><br />
<label for="contact">Contact No</label>
<input id="mobile" name="mobile" type="number" ng-
model="person.mobile" required />
<span class="error" ng-
show="personForm.mobile.$error.required">Required number!</span>
```

```
<span class="error" ng-show="personForm.mobile.$error.mobile">Invalid
mobile!</span>
<br /><br />
<label for="email">Email</label>
<input id="email" name="email" type="email" ng-
model="person.email" required />
<span class="error" ng-
show="personForm.email.$error.required">Required!</span>
<span class="error" ng-show="personForm.email.$error.email">Invalid
Email!</span>
<br /><br />
<input type="checkbox" ng-
model="terms" name="terms" id="terms" required />
<label for="terms">I Agree to the terms.</label>
<span class="error" ng-show="personForm.terms.$error.required">You
must agree to the terms</span>
<br /><br />
<button type="submit">Submit Form</button>
<br /><br />
<span>{{msg}}</span>
</form>
</body>
</html>
```



QUESTIONS

1. What is required or ng-required ? How to use it?

2. What is a form?
3. Write the syntax of implementing validations to input fields on form?

*****

MODULE 6

EXPERIMENT 1

AIM :
Create an application using Filters

OBJECTIVE:
A **Filter in AngularJS** helps to format the value of an expression to display to the user without changing the original format. For example, if you want your string in either lowercase or uppercase, you can do it using filters. There are built-in filters such as 'lowercase', 'uppercase', which can retrieve the lowercase and uppercase output accordingly. Similarly, for numbers, you can use other filters.

THEORY:
AngularJS filter is a tool, which we can use to format the data. With this filter, the user can see and modify according to the requirement. It is added in angular to format the data that is being displayed on the view part.

SYNTAX OF FILTER:
With pipe character (|), we can add filters in angularJS. Filters can club with the expression as well as a directive.

SYNTAX:

{{expression| name_of_filter}}
Example: {{name | uppercase}}

Here, a name is the expression and uppercase is the built-in filter provided by angular. It converts the name in uppercase in view part.

WHEN TO USE A FILTER IN ANGULARJS?

To display the data in the view part in a particular format AngularJS filter can be used. Data can be displayed in the uppercase format, lowercase format etc. In whatever format be the entered text it can easily get displayed in any format by angular as per the type of filter used. AngularJS Filters can change the way of displaying the output.

ANGULARJS FILTER IN VIEW TEMPLATE:

Filters are applied on expression to refrain the input and also AngualrJS Filters can be applied in a view template and on the result obtained by another filter. Chaining is the phenomenon when filters are applied to the already filtered content.

Syntax:

```
{{ expression | filter1 | filter2 | ... }}
```

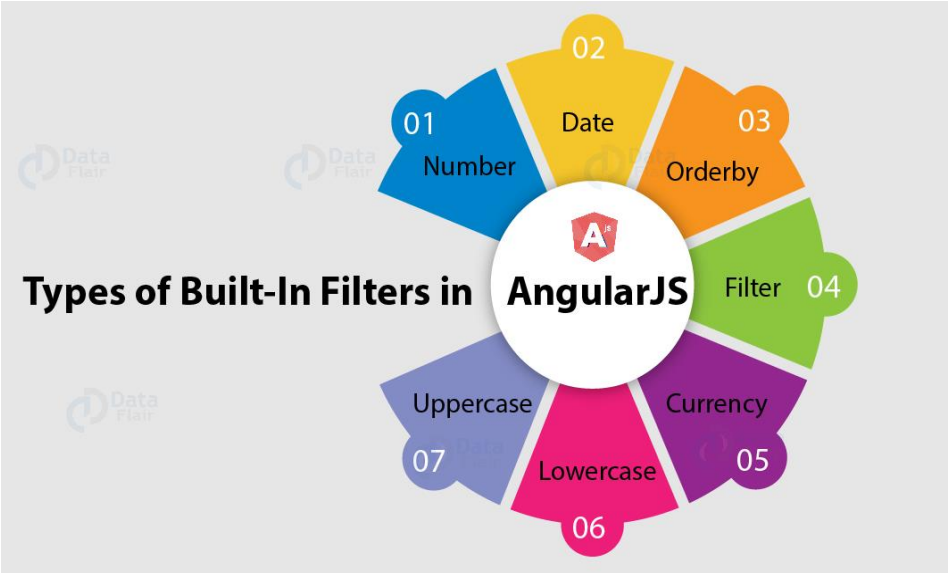
AngularJS Filters can have arguments.

Syntax:

```
{{ expression | filter:argument1:argument2:... }}
```

WHEN WILL FILTER GET EXECUTE:
Whenever the inputs have changed in the template, the filter gets to execute.

TYPES OF BUILT-IN FILTERS IN ANGULARJS:
These are the following built-in filters, let's discuss them one by one:



1. UPPERCASE:
Uppercase Filter use to convert the string into an uppercase string.

If Uppercase filter is used then the string that is taken as an input in any format is formatted and gets displayed to the user in an uppercase format as an output.

Syntax:

```
{{expression | uppercase}}
```

2. LOWERCASE:
Lowercase Filter in AngularJS uses to convert the string into a lowercase string.

If a lowercase filter is used then the string that is taken as an input in any format is formatted and gets displayed to the user in the lowercase format as an output.

Syntax:

```
{{expression | lowercase}}
```

3. CURRENCY:
Currency filter is used to change the convert the number in the specified currency. In case no symbol of currency is specified then by default the symbol for current locale is used.

Syntax:

```
{{expression | currency : 'currency_symbol' : 'fraction'}}
```

4. FILTER:
It is applied only on array elements. A particular element is selected from an array based on certain condition and a new array is created from the existing array.

Syntax:

```
{{ expression | filter : filter_criteria }}
```

5. ORDERBY:
It is used to sort the data either in ascending or in descending order.

Syntax:

```
{{ expression | orderBy : predicate_expression : reverse}}
```

6. DATE:
Date filter in AngularJS use to convert the date into a string according to the specified date format.

Syntax:

```
{{date_expression| date : 'format'}}
```

The various formats are:
YYYY,MM,DD,D etc

7. NUMBER:
It is used to format the data or a numerical value taken as an input. It can add a comma or specified fraction size in it. In case the number expression doesn't contain valid numeric data in that case number filter display an empty string.

Syntax:

```
{{ number_expression | number:fractionSize}}
```

CUSTOM FILTERS IN ANGULARJS:
Angular JS provides a way to create our own filter that is customized filter. User can create one by registering a filter factory function in a module. AngularJS Filter function should be a pure function. Pure function means the same result is produced as per the given input. Also, it should not affect an external state.

PROGRAM
testAngularJS.htm

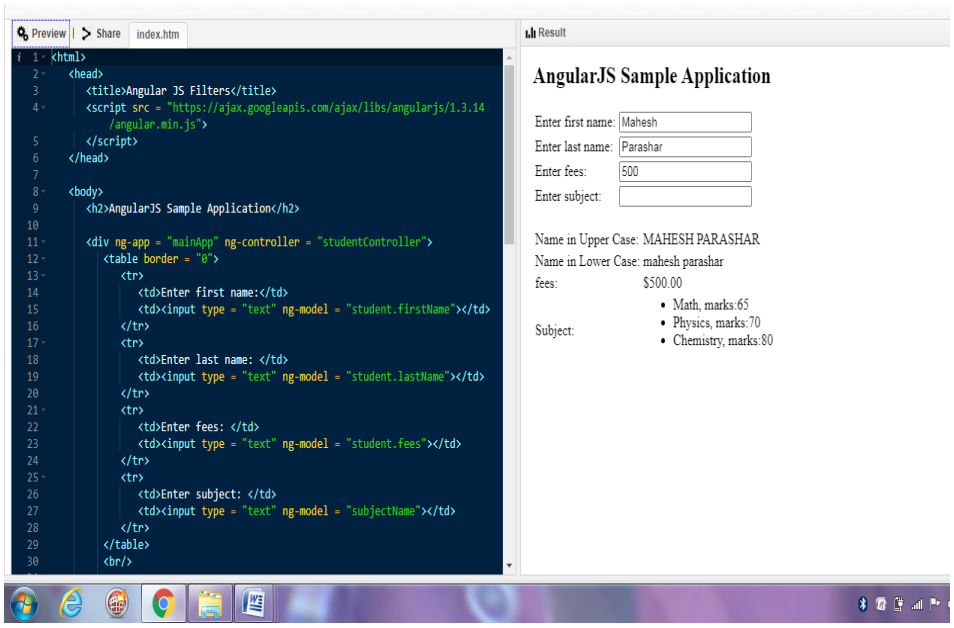
```
<html>
<head>
    <title>Angular JS Filters</title>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
    </script>
  </head>

  <body>
    <h2>AngularJS Sample Application</h2>

    <div ng-app = "mainApp" ng-controller = "studentController">
      <table border = "0">
        <tr>
          <td>Enter first name:</td>
          <td><input type = "text" ng-model = "student.firstName"></td>
        </tr>
        <tr>
          <td>Enter last name: </td>
          <td><input type = "text" ng-model = "student.lastName"></td>
        </tr>
        <tr>
          <td>Enter fees: </td>
```

```
            <td><input type = "text" ng-model = "student.fees"></td>
          </tr>
          <tr>
            <td>Enter subject: </td>
            <td><input type = "text" ng-model = "subjectName"></td>
          </tr>
        </table>
        <br/>

        <table border = "0">
          <tr>
            <td>Name in Upper Case: </td><td>{{student.fullName() |
uppercase}}</td>
          </tr>
          <tr>
            <td>Name in Lower Case: </td><td>{{student.fullName() |
lowercase}}</td>
          </tr>
          <tr>
            <td>fees: </td><td>{{student.fees | currency}}
            </td>
          </tr>
          <tr>
            <td>Subject:</td>
            <td>
              <ul>
                <li ng-repeat = "subject in student.subjects | filter:
subjectName |orderBy:'marks'">
                  {{ subject.name + ', marks:' + subject.marks }}
                </li>
              </ul>
            </td>
          </tr>
        </table>
      </div>

      <script>
        var mainApp = angular.module("mainApp", []);

        mainApp.controller('studentController', function($scope) {
          $scope.student = {
            firstName: "Mahesh",
            lastName: "Parashar",
            fees:500,

            subjects:[
```

```
            {name:'Physics',marks:70},
            {name:'Chemistry',marks:80},
            {name:'Math',marks:65}
         ],
         fullName: function() {
            var studentObject;
            studentObject = $scope.student;
            return studentObject.firstName + " " + studentObject.lastName;
         }
      };
   });
</script>

</body>
</html>
```

OUTPUT

Open the file testAngularJS.htm in a web browser. See the result.



QUESTIONS

1. Define filter?
2. When to use a filter in angularjs?
3. Write the syntax of any one filter type?
4. What is the use of Orderby filter?
5. List out the types of built in filters?

*****

EXPERIMENT 2

AIM:
Implement the use of number filter

OBJECTIVE:
Learn
What is angularjs number filter?
Use of number filter in angularjs
How to show number with decimal values
How to limit size of decimal values using angularjs number filter.

THEORY:
In angularjs, **number** filter is used to format the number and convert it as string or text. By using number filter in angularjs we can show number with decimal values and we define limit to show number of decimal values.

AngularJS Number Filter Syntax:
{{numberexpression | number}}

Suppose if the expression is given as **{{1000 | number}}** it will format the number and return result as **1,000**.

To show number with decimal values the syntax will be as shown below

{{numberexpression | number:fractionsize}}

If the expression is given as **{{1000 | number:2}}** it will format the number and return result as **1,000.00**

<**div**>Default Number filter :{{numberValue | number}}

This line will format the numberValue based on current system locale and return the formatted value in string.

<**div**>Number with 4 decimal fraction :{{numberValue | number : 4}}

This line will format the numberValue based on current system locale with decimal precision of 4 points and return the formatted value in string.

<**div**>Number with null value :{{null | number}}
This line show whenever Number Filter encounters the numberValue a null, it will return 0 value back.

<**div**>Number with undefined value :{{undefined | number}}
This line show whenever Number Filter encounters the numberValue as undefined, it will return " " Empty String value back.

<**div**>Number is NaN :{{ NanNumber | number}}

This line show whenever Number Filter encounters an Nan numberValue, it will return " " Empty String value back.
var formatterNumber =$filter('number')(value, 2);

Points to keep a Note:

- If input is Nan(Not A Number), then Empty String **""** is returned
- If input is null, then **0** gets returned
- If input is undefined, then **""** Empty String is returned
- If input is having infinite value, then after formatting Infinity symbol ∞ is returned
- Number formatting happens based on current locale (system time) for eg- en_IN local the decimal separator will be "." and group separator will be ","

PROGRAM

Index.html

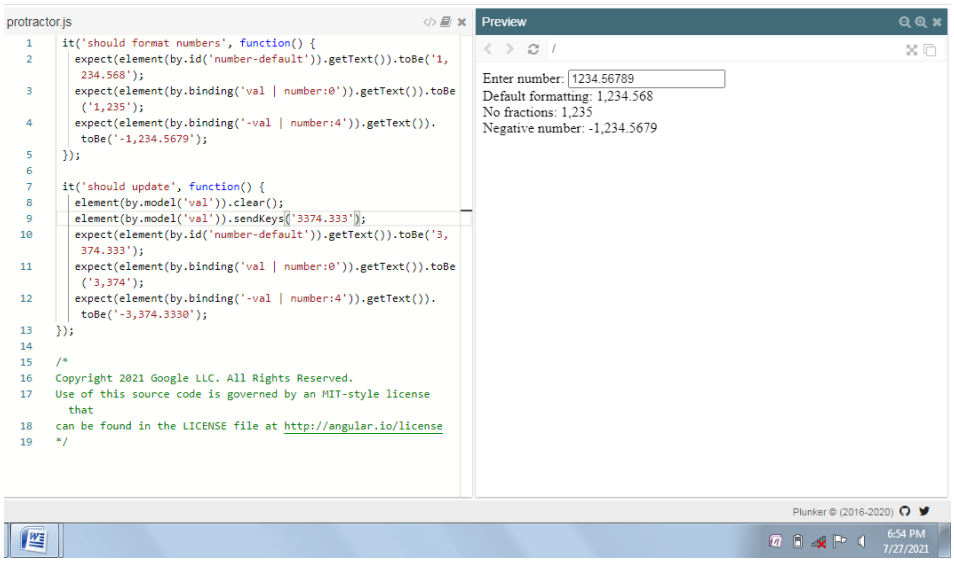
```
<!doctype html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Example - example-number-filter-production</title>
  <script src="//code.angularjs.org/snapshot/angular.min.js"></script>
</head>
<body ng-app="numberFilterExample">
 <script>
 angular.module('numberFilterExample', [])
  .controller('ExampleController', ['$scope', function($scope) {
   $scope.val = 1234.56789;
  }]);
</script>
<div ng-controller="ExampleController">
 <label>Enter number: <input ng-model='val'></label><br>
 Default formatting: <span id='number-
default'>{{val | number}}</span><br>
 No fractions: <span>{{val | number:0}}</span><br>
 Negative number: <span>{{-val | number:4}}</span>
</div>
</body>
</html>
```

Protractor.js

```
it('should format numbers', function() {
  expect(element(by.id('number-default')).getText()).toBe('1,234.568');
```

```
  expect(element(by.binding('val | number:0')).getText()).toBe('1,235');
  expect(element(by.binding('-val | number:4')).getText()).toBe('-
1,234.5679');
 });

 it('should update', function() {
  element(by.model('val')).clear();
  element(by.model('val')).sendKeys('3374.333');
  expect(element(by.id('number-default')).getText()).toBe('3,374.333');
  expect(element(by.binding('val | number:0')).getText()).toBe('3,374');
  expect(element(by.binding('-val | number:4')).getText()).toBe('-
3,374.3330');
});
```

OUTPUT



QUESTIONS

1. How to show number with decimal values
2. How to limit size of decimal values using angularjs number filter.
3. Write the syntax of number filter?

*****

EXPERIMENT 3

AIM:
Implement date filter in angularjs applications

OBJECTIVE:
At times there is a need to check the date for any particular reason. What happens is that sometimes it is in desired format and sometimes it is not. AngularJS date filter is used to convert a date into a specified format.

THEORY:
In angularjs, **date** filter is used to convert date to string based on the given format. In angularjs, date can be converted to multiple formats using date filter.

AngularJS Date Filter Syntax:
Generally the syntax of date filter
{{dateexpression | date : format}}

In angularjs different type of date formats are available with date filter. Following table shows different type of date formats available with date filter in angularjs.

Format	Expression	Result
yyyy (Year)	{{sampledate|date:"yyyy" }}	2016
yy (Year)	{{sampledate | date:"yy" }}	16
y (Year)	{{sampledate | date:"y" }}	2016
MMMM (Month)	{{sampledat | date:"MMMM" }}	February
MMM (Month)	{{sampledate | date:"MMM" }}	Feb
MM (Month)	{{sampledate | date:"MMM" }}	02
M (Month)	{{sampledate | date:"M" }}	2
dd (Date)	{{sampledate | date:"dd" }}	28
d (Date)	{{sampledate | date:"d" }}	28
EEEE (Day)	{{sampledate | date:"EEEE" }}	Sunday
EEE (Day)	{{sampledate | date:"EEE" }}	Sun
HH (24 Hours Format)	{{sampledate | date:"HH" }}	19

Format	Expression	Result
hh (12 Hours Format)	{{sampledate|date:"hh" }}	07
h (12 Hours Format)	{{sampledate | date:"h" }}	7
mm (Minute)	{{sampledate | date:"mm" }}	16
m (Minute)	{{sampledate | date:"m" }}	16
sss (Milli Seconds)	{{sampledate | date:"sss" }}	501

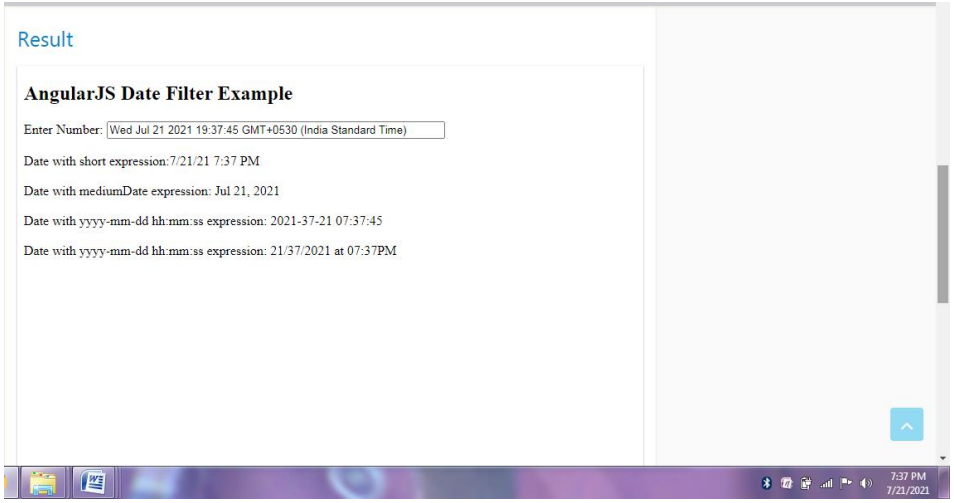
ss (Seconds)	{{sampledate | date:"ss" }}	45
s (Seconds)	{{sampledate | date:"s" }}	45
a (AM/PM)	{{sampledate | date:"a" }}	PM
Z (TimeZone)	{{sampledate | date:"a" }}	0530
ww (Week of year)	{{sampledate | date:"ww" }}	09
w (Week of year)	{{sampledate | date:"w" }}	9
medium	{{sampledate|date:"medium" }	Feb 28, 2016 7:32:55 PM
short	{{sampledate | date:"short" }}	2/28/16 7:33 PM
fullDate	{{sampledate | date:"fullDate" }}	Sunday, February 28, 2016
longDate	{{sampledate | date:"longDate" }}	February 28, 2016
mediumDate	{{sampledate | date:"medium" }}	Feb 28, 2016
shortDate	{{sampledate | date:"shortDate" }}	2/28/16
mediumTime	{{sampledate|date:"mediumTime" }}	7:37:34 PM
shortTime	{{sampledate | date:"shortTime" }}	7:37 PM

PROGRAM

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>
AngularJs Date filter Example
</title>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.m
in.js"></script>
<script>
var app = angular.module("AngulardateApp", []);
app.controller("datectrl", function ($scope) {
$scope.sampledate = new Date();
});
</script>
</head>
<body ng-app="AngulardateApp">
<div ng-controller="datectrl">
Enter                                    Number: <input type="text" ng-
model="sampledate" style="width:400px" /><br /><br />
Date with short expression:{{sampledate | date:"short" }}<br /><br />
```

```
Date with mediumDate expression: {{sampledate | date : "mediumDate"}} <br /><br />
Date with yyyy-mm-dd hh:mm:ss expression: {{sampledate | date : "yyyy-mm-dd hh:mm:ss" : 0}} <br /><br />
Date with yyyy-mm-dd hh:mm:ss expression: {{sampledate | date : "dd/mm/yyyy 'at' hh:mma" : 0}}
</div>
</body>
</html>
```

OUTPUT:



QUESTIONS

1. Give out the syntax of number filter?
2. Specify any four type of date formats are available with date filter ?
3. What is the use angular date filter?

*****

EXPERIMENT 4

AIM:
Create an application to demonstrate the use of Custom Filters

OBJECTIVE:
AngularJS provides many in-built directives like search. If required, AngularJS also allows creating custom filters.AngularJS gives a simple API to create a custom filter. Remember that app.controller() is used to create controllers and app.module() is used to create modules. In exactly the same way, AngularJS has given the angular.filter API to create a custom filter in AngularJS.

THEORY:
Sometimes the built-in filters in Angular cannot meet the needs or requirements for filtering output. In such a case, an AngularJS custom filter can be created which can pass the output in the required manner.

How to Create Custom Filter:
In the below custom filter AngularJS example, a string is passed to the view from the controller via the scope object, but don't want that string to be displayed as it is. Whenever the string is displayed, a custom filter is passed in AngularJS which will append another string and display the completed string to the user.

Few Example of Custom Filter:
Some example requirement that can be solved by creating custom filters could be:

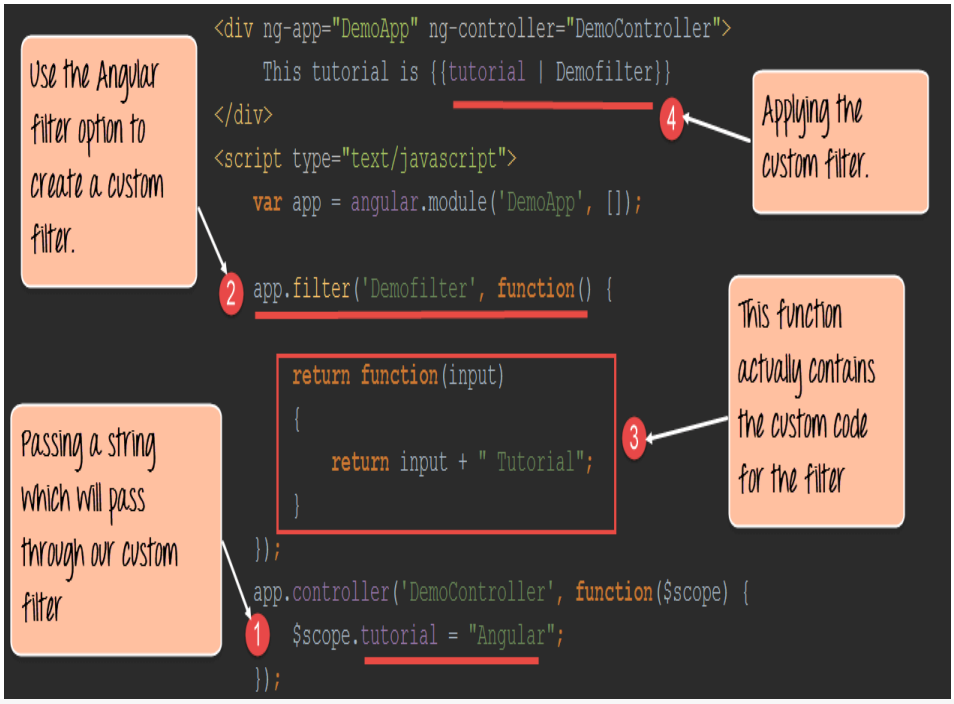
- Prefix or suffix any String to the input.
- Filter even/odd numbers from an array of.
- Filter based on any custom logic e. multiple of five etc.
- Reverse a string or.

Steps to Create a Custom Filter:

- Create an AngularJS application.
- Use (dot) .filter API provide by AngularJS framework to add.
- Pss a filter name and filter function to custom.
- Write logic for transforming the input to output in return.

Custom filter syntax:

```
MyApp.filter('filtername', function () {

    return function (input,optionalparam1,optionalparam2)
    {
        var output;

        //custom filter code goes here

        return output;
    }
})
```



PROGRAM

Index.html

```
<!DOCTYPE html>
<html>


  <head>
    <link data-require="bootstrap-css@3.1.1" data-semver="3.1.1" rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css" />
    <script data-require="angular.js@*" data-semver="1.3.0-beta.5" src="https://code.angularjs.org/1.3.0-beta.5/angular.js"></script>
    <script data-require="lodash.js@*" data-semver="2.4.1" src="http://cdnjs.cloudflare.com/ajax/libs/lodash.js/2.4.1/lodash.js"></script>
```

```
  <script data-require="jquery@*" data-
semver="2.0.3" src="http://code.jquery.com/jquery-
2.0.3.min.js"></script>
   <script data-require="bootstrap@*" data-
semver="3.1.1" src="//netdna.bootstrapcdn.com/bootstrap/3.1.1/js/bootstra
p.min.js"></script>
   <link rel="stylesheet" href="style.css"/>
   <script src="script.js"></script>
 </head>

 <body ng-app="app" class="container">
   <div ng-controller="mainController as main">

     <button class="btn btn-default" ng-
click="main.clearTitle()">Clear Title</button>
     <select ng-model="main.selectedTitle" ng-
options="title for title in main.titles" class="form-control"></select>

     <button class="btn btn-default" ng-
click="main.clearGenre()">Clear Genre</button>
     <select ng-model="main.selectedGenre" ng-
options="genre for genre in main.genres" class="form-control"></select>

     <button class="btn btn-default" ng-
click="main.clearYear()">Clear Year</button>
     <select ng-model="main.selectedYear" ng-
options="year for year in main.years" class="form-control"></select>

     <table class="table">
       <thead>
         <tr>
           <th>Title</th>
           <th>Category</th>
           <th>Released</th>
         </tr>
       </thead>
       <tbody>
         <tr ng-
repeat="movie in main.movies | filter:{ genre: main.selectedGenre, release
d: main.selectedYear, title: main.selectedTitle}">
           <td>{{movie.title}}</td>
           <td>{{movie.genre}}</td>
           <td>{{movie.released}}</td>
         </tr>
       </tbody>
     </table>
   </div>
```

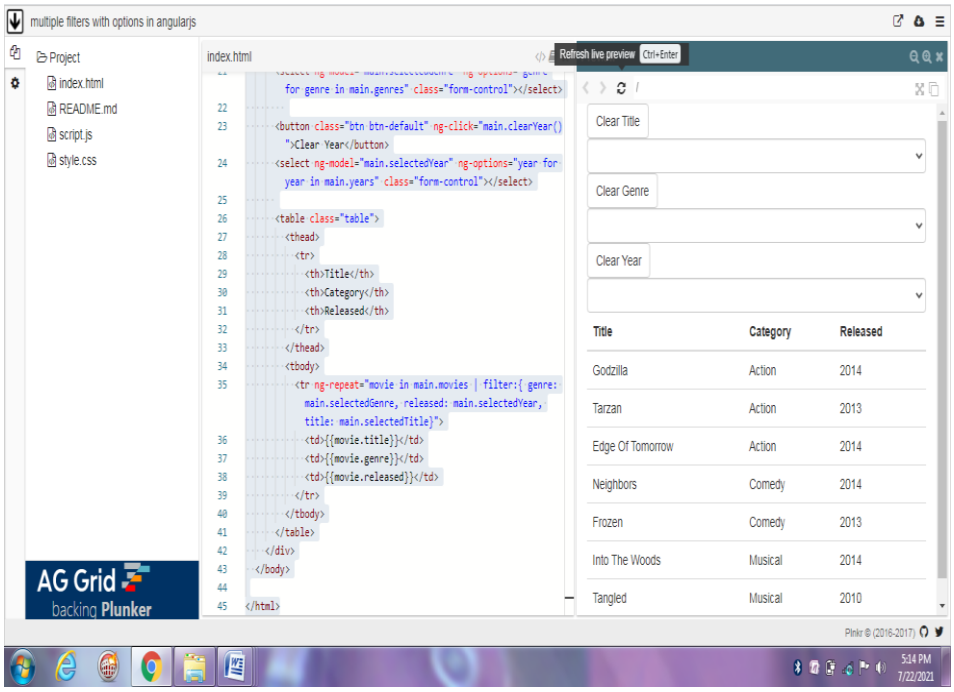
```
</body>
</html>
```

Script.js

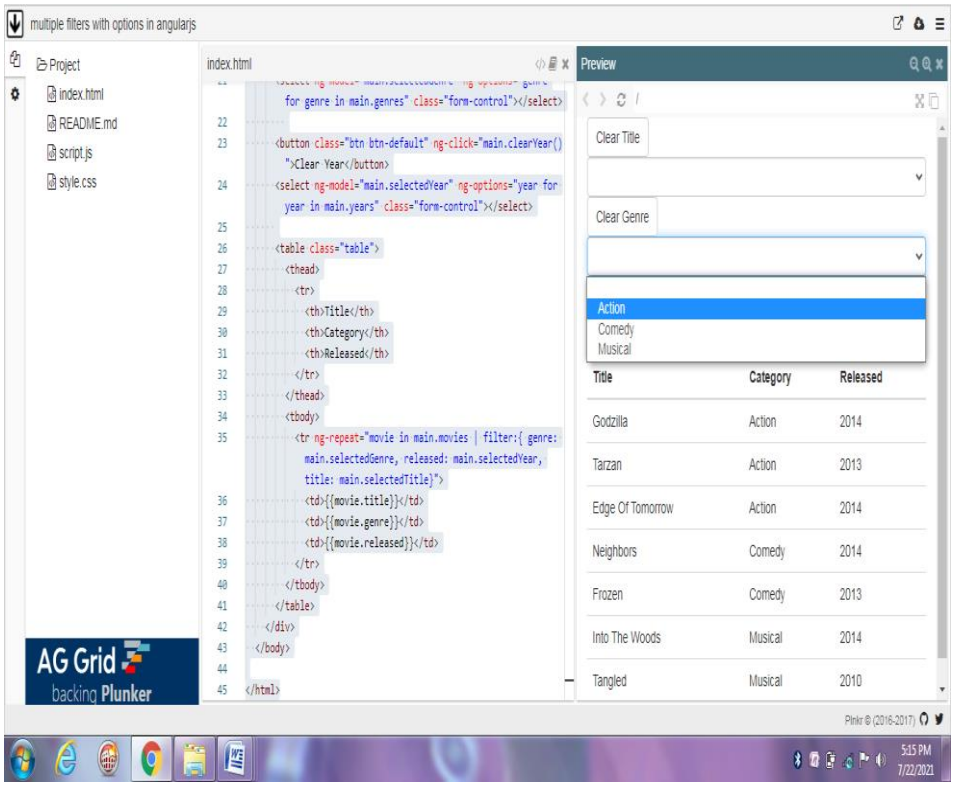
```
(function(){

   var app = angular.module("app", []);

   app.controller("mainController", function($scope, movieData) {

      var typeOf = this;
      typeOf.movies = [];

      typeOf.movies = movieData.getAll();

      typeOf.titles = _.chain(typeOf.movies).pluck("title").uniq().sortBy().
value();
      typeOf.genres = _.chain(typeOf.movies).pluck("genre").uniq().sortBy
().value();
      typeOf.years = _.chain(typeOf.movies).pluck("released").uniq().sort
By().value();

      typeOf.clearTitle = function(){
        typeOf.selectedTitle = undefined;
      };

      typeOf.clearGenre = function(){
         typeOf.selectedGenre = undefined;
      };

      typeOf.clearYear = function(){
         typeOf.selectedYear = undefined;
      };

   });

   app.factory("movieData", function(){

      var movies = [
         {
           title: "Godzilla",
           genre:"Action",
           released: 2014

         },
         {
           title: "Tarzan",
          genre: "Action",
```

```
            released: 2013

          },
          {
            title: "Edge Of Tomorrow",
            genre: "Action",
            released: 2014

          },
          {
            title: "Neighbors",
            genre: "Comedy",
            released: 2014

          },
          {
            title: "Frozen",
            genre: "Comedy",
            released: 2013

          },
          {
            title: "Into The Woods",
            genre: "Musical",
            released: 2014

          },
          {
            title: "Tangled",
            genre: "Musical",
            released: 2010

          }
        ];

        return {
          getAll: function(){
            return movies;
          }
        };
      });
}
()
);
```

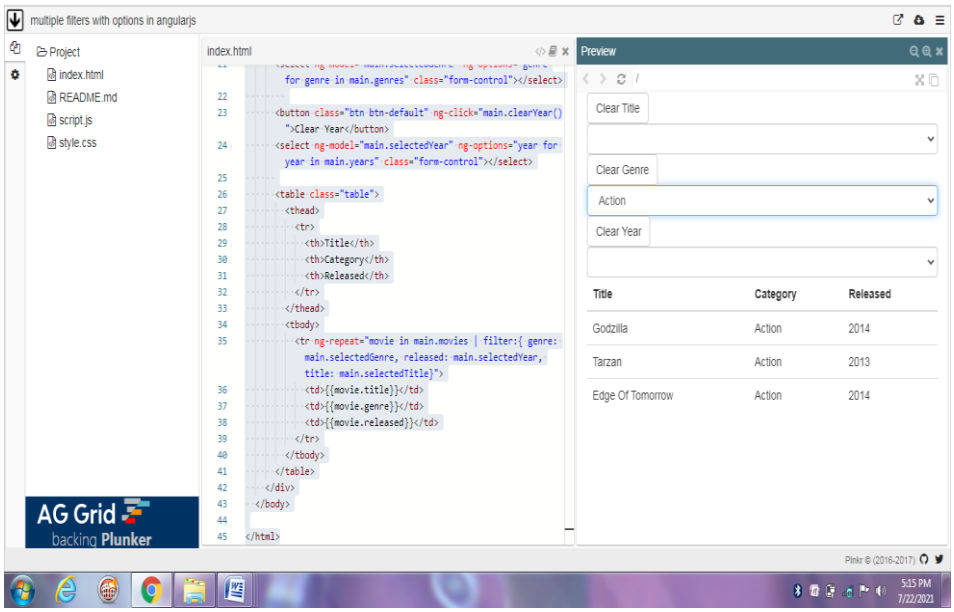
OUTPUT


multiple filters with options in angularjs
Project
index.html
README.md
script.js
style.css
index.html
Refresh live preview Ctrl+Enter
for genre in main.genres" class="form-control"></select>
<button class="btn btn-default" ng-click="main.clearYear()
">Clear Year</button>
<select ng-model="main.selectedYear" ng-options="year for
year in main.years" class="form-control"></select>
<table class="table">
<thead>
<tr>
<th>Title</th>
<th>Category</th>
<th>Released</th>
</tr>
</thead>
<tbody>
<tr ng-repeat="movie in main.movies | filter:{ genre:
main.selectedGenre, released: main.selectedYear,
title: main.selectedTitle}">
<td>{{movie.title}}</td>
<td>{{movie.genre}}</td>
<td>{{movie.released}}</td>
</tr>
</tbody>
</table>

</body>
</html>
AG Grid
backing Plunker
Clear Title
Clear Genre
Clear Year
Title
Category
Released
Godzilla
Action
2014
Tarzan
Action
2013
Edge Of Tomorrow
Action
2014
Neighbors
Comedy
2014
Frozen
Comedy
2013
Into The Woods
Musical
2014
Tangled
Musical
2010
Plnkr © (2016-2017)
5:14 PM
7/22/2021


multiple filters with options in angularjs
Project
index.html
README.md
script.js
style.css
index.html
Preview
for genre in main.genres" class="form-control"></select>
<button class="btn btn-default" ng-click="main.clearYear()
">Clear Year</button>
<select ng-model="main.selectedYear" ng-options="year for
year in main.years" class="form-control"></select>
<table class="table">
<thead>
<tr>
<th>Title</th>
<th>Category</th>
<th>Released</th>
</tr>
</thead>
<tbody>
<tr ng-repeat="movie in main.movies | filter:{ genre:
main.selectedGenre, released: main.selectedYear,
title: main.selectedTitle}">
<td>{{movie.title}}</td>
<td>{{movie.genre}}</td>
<td>{{movie.released}}</td>
</tr>
</tbody>
</table>

</body>
</html>
AG Grid
backing Plunker
Clear Title
Clear Genre
Action
Comedy
Musical
Title
Category
Released
Godzilla
Action
2014
Tarzan
Action
2013
Edge Of Tomorrow
Action
2014
Neighbors
Comedy
2014
Frozen
Comedy
2013
Into The Woods
Musical
2014
Tangled
Musical
2010
Plnkr © (2016-2017)
5:15 PM
7/22/2021



QUESTIONS

1. How to Create Custom Filter
2. Write the steps to create custom filter
3. List out few example of custom filter?

*****

EXPERIMENT 1

AIM :
Create an applicaiton to demonstrate the use of conditional directives.

OBJECTIVE:
When require some advance feature, to create an angular application (advanced) like a mouse click, keyboard presses, changes events, moves etc. The advance application focuses on handling DOM events in AngularJS. It provides a model to add an event listener in the HTML part.

THEORY:
AngularJS lets you extend HTML with new attributes called **Directives**. AngularJS has a set of built-in directives which offers functionality to your applications. AngularJS also lets you define your own directives.

AngularJS Directives:
AngularJS directives are extended HTML attributes with the prefix ng-. The ng-app directive initializes an AngularJS application. The ng-init directive initializes application data. The ng-model directive binds the value of HTML controls (input, select, textarea) to application data. The NgIf directive is used when you want to display or remove an element based on a condition.If the condition is false the element the directive is *attached to* will be *removed* from the DOM.

Important:
The difference between [hidden]='false' and *ngIf='false' is that the first method simply *hides* the element. The second method with ngIf *removes* the element completely from the DOM.

Definition and Usage:
The ng-if directive removes the HTML element if the expression evaluates to false. If the if statement evaluates to true, a copy of the Element is added in the DOM. The ng-if directive is different from the ng-hide, which hides the display of the element, where the ng-if directive completely removes the element from the DOM.

Syntax
<*element* ng-if="*expression*"></*element*>
Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression that will completely remove the element if it returns false. If it returns true, a copy of the element will be inserted instead.

PROGRAM

App.js

```
// Create AngularJS application
var app = angular.module('demoLearningTurn',[]);

// Create Controller with name mainCtrl

app.controller('mainCtrl', function($scope){

  $scope.txtOptions = {
    '1': 'Text Paragraph First',
    '2': 'Text Paragraph Second',
    '3': 'Text Paragraph Third',
    '4': 'Text Paragraph Default'
  };
});
```

Index.html

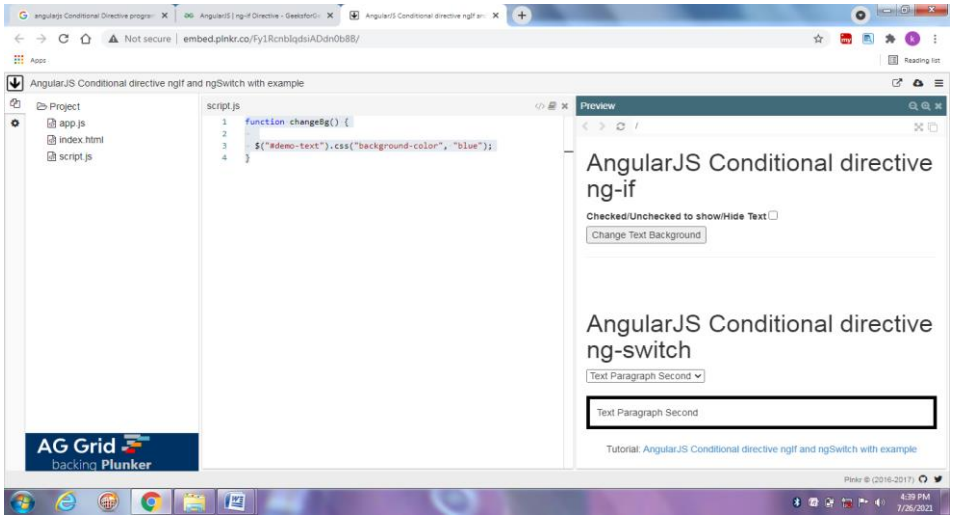
```
<!DOCTYPE html>
<html>
<head>
  <title>AngularJS Conditional directive ng-if and ng-
switch with example</title>
  <link href="http://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstra
p.min.css" rel="stylesheet">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.m
in.js"></script>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular
.min.js"></script>
  <script src="app.js"></script>
  <script src="script.js"></script>
  <style>hr{padding:30px 0;}</style>
</head>
<body>

<!-- Main Content -->
  <div class="container" ng-app="demoLearningTurn">
```

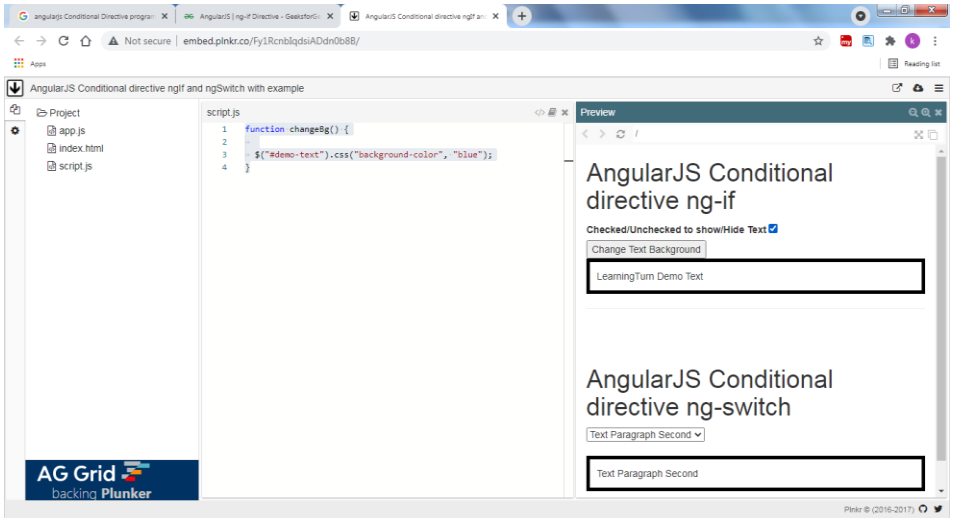
```
   <div class="content" ng-controller="mainCtrl">
   <h1>AngularJS Conditional directive ng-if</h1>
   <label>Checked/Unchecked to show/Hide Text</label>
   <input type="checkbox" ng-model="txtStatus"></br>
   <input type="button" onclick="changeBg();" value="Change Text Back
ground"></br>

   <p ng-if="txtStatus" id="demo-
text" style="border: 5px solid #000; padding:10px;">LearningTurn Demo
Text</p>
   <hr/>
   <h1>AngularJS Conditional directive ng-switch</h1>

   <select ng-model='txtOption' ng-
options="key as label for (key, label) in txtOptions">

   </select>
   <br/><br/>
     <div ng-switch on="txtOption">
       <p ng-switch-
when="1" style="border: 5px solid #000; padding:10px;">Text Paragraph
First</p>
       <p ng-switch-
when="2" style="border: 5px solid #000; padding:10px;">Text Paragraph
Second</p>
       <p ng-switch-
when="3" style="border: 5px solid #000; padding:10px;">Text Paragraph
Third</p>
       <p ng-switch-
default style="border: 5px solid #000; padding:10px;">Text Paragraph De
fault</p>
     </div>
   </div>
  </div>
  <p><center>Tutorial: <a href="http://learningturn.com/angular-
js/angularjs-conditional-directive-ng-if-and-ng-switch-with-
example/">AngularJS Conditional directive ngIf and ngSwitch with exam
ple</center></p>
</body>
</html>
```

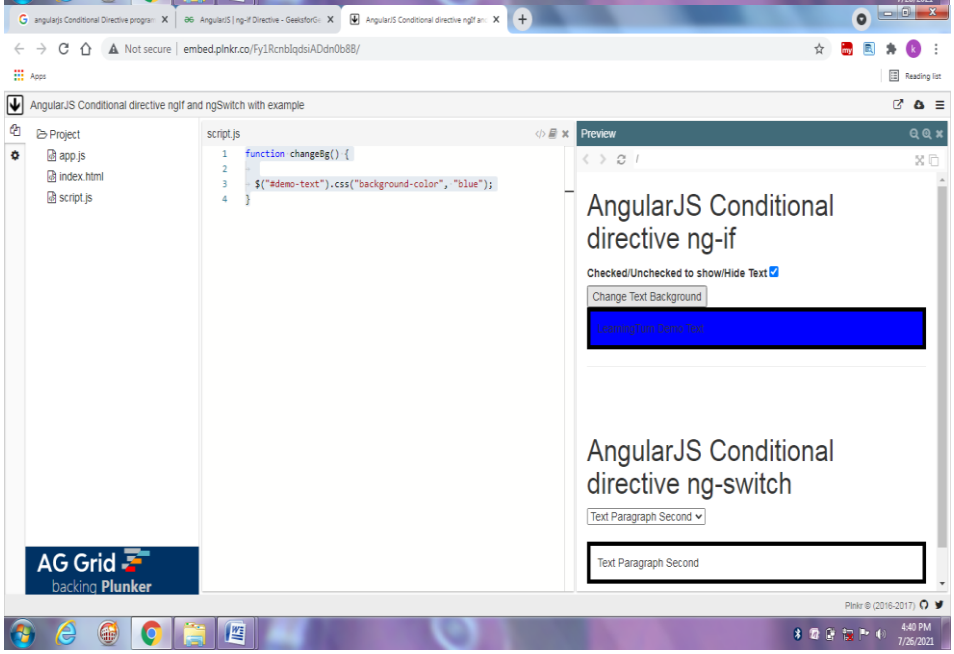
Script.js

```
function changeBg() {

  $("#demo-text").css("background-color", "blue");
}
```

OUTPUT


AngularJS Conditional directive ngIf and ngSwitch with example
Project
app.js
index.html
script.js
function changeBg() {
$("#demo-text").css("background-color", "blue");
}
Preview
AngularJS Conditional directive ng-if
Checked/Unchecked to show/Hide Text
Change Text Background
AngularJS Conditional directive ng-switch
Text Paragraph Second
Text Paragraph Second
Tutorial: AngularJS Conditional directive ngIf and ngSwitch with example
AG Grid
backing Plunker


AngularJS Conditional directive ngIf and ngSwitch with example
Project
app.js
index.html
script.js
function changeBg() {
$("#demo-text").css("background-color", "blue");
}
Preview
AngularJS Conditional directive ng-if
Checked/Unchecked to show/Hide Text
Change Text Background
LearningTurn Demo Text
AngularJS Conditional directive ng-switch
Text Paragraph Second
Text Paragraph Second
AG Grid
backing Plunker


AngularJS Conditional directive ngIf and ngSwitch with example
Project
app.js
index.html
script.js
function changeBg() {
$("#demo-text").css("background-color", "blue");
}
Preview
AngularJS Conditional directive ng-if
Checked/Unchecked to show/Hide Text
Change Text Background
AngularJS Conditional directive ng-switch
Text Paragraph Second
Text Paragraph Second
AG Grid
backing Plunker

QUESTIONS

1. List out the parameter values of ng-if directive
2. What is the difference between [hidden]='false' and *ngIf='false' ?
3. Write the syntax of ng-if directive?

*****

EXPERIMENT 2

AIM:
Create an application to demonstrate the use of style directive

OBJECTIVE:
The **ng-style Directive** in AngularJS is used to apply custom CSS styles on an HTML element. The expression inside the ng-style directive must be an object. It is supported by all the HTML elements.

THEORY:

Definition and Usage:
The ng-style directive specifies the style attribute for the HTML element. The value of the ng-style attribute must be an object, or an expression returning an object. The object consists of CSS properties and values, in key value pairs.

Syntax
<*element* ng-style="*expression*"></*element*> Supported by all HTML elements.

Parameter Values

Value	Description
expression	An expression which returns an object where the keys are CSS properties, and the values are CSS values.

Overview:
The ngStyle directive allows you to set CSS style on an HTML element conditionally.

Known Issues:
Do not use interpolations in the value of the style attribute, when using the ngStyle directive on the same element.

Directive Info:
This directive executes at priority level 0.

Usage
as attribute:

```
<ANY
  ng-style="expression">
...
</ANY>
```

as CSS class:

```
<ANY class="ng-style: expression;"> ... </ANY>
```

Arguments

Param	Type	Details
ngStyle	**expression**	Expression which evals to an object whose keys are CSS style names and values are corresponding values for those CSS keys. Since some CSS style names are not valid keys for an object, they must be quoted. See the 'background-color' style in the example below.

PROGRAM

Index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Example - example-ng-style-production</title>
  <link href="style.css" rel="stylesheet" type="text/css">
    <script src="//code.angularjs.org/snapshot/angular.min.js"></script>
  </head>
<body ng-app="">
  <input type="button" value="set color" ng-
click="myStyle={color:'red'}">
<input type="button" value="set background" ng-
click="myStyle={'background-color':'blue'}">
<input type="button" value="clear" ng-click="myStyle={}">
<br/>
<span ng-style="myStyle">Sample Text</span>
<pre>myStyle={{myStyle}}</pre>
</body>
</html>
```

Protractor.js

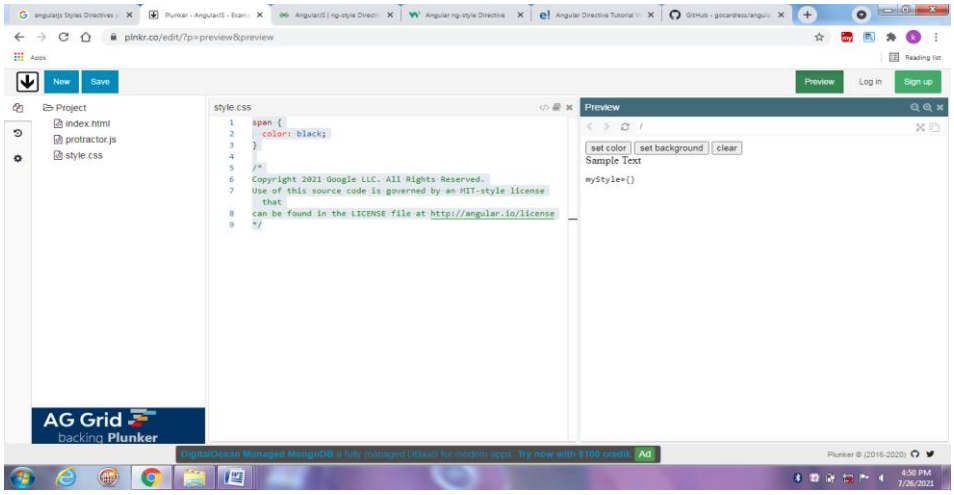
```
var colorSpan = element(by.css('span'));

it('should check ng-style', function() {
  expect(colorSpan.getCssValue('color')).toMatch(/rgba\(0, 0, 0, 1\)|rgb\(0,
 0, 0\)/);
  element(by.css('input[value=\'set color\']')).click();
  expect(colorSpan.getCssValue('color')).toMatch(/rgba\(255, 0, 0, 1\)|rgb\
(255, 0, 0\)/);
  element(by.css('input[value=clear]')).click();
  expect(colorSpan.getCssValue('color')).toMatch(/rgba\(0, 0, 0, 1\)|rgb\(0,
 0, 0\)/);
```

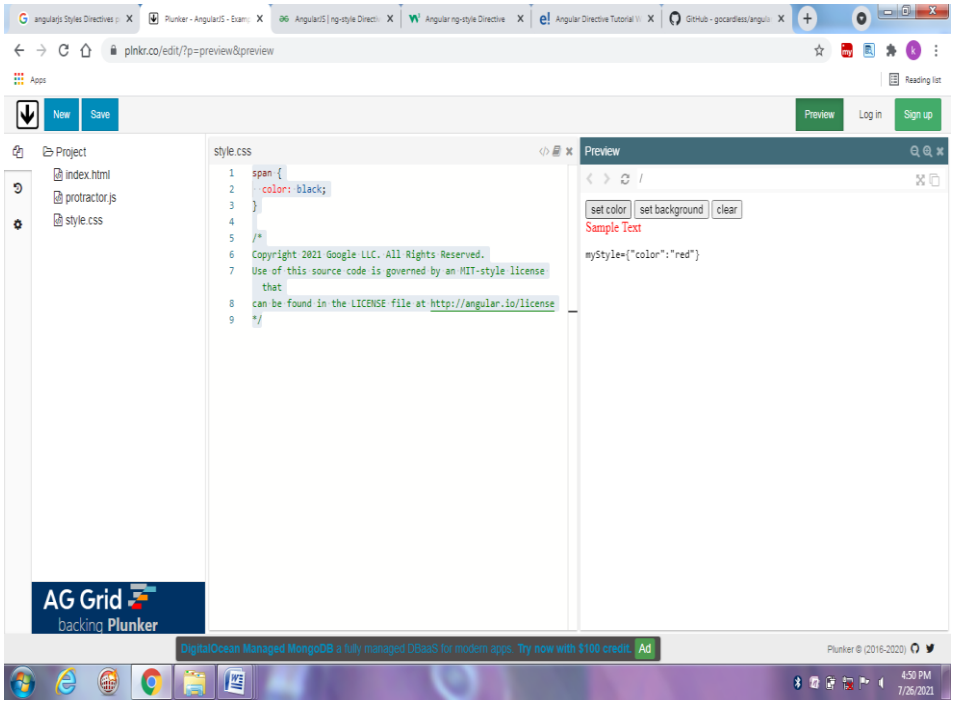
});

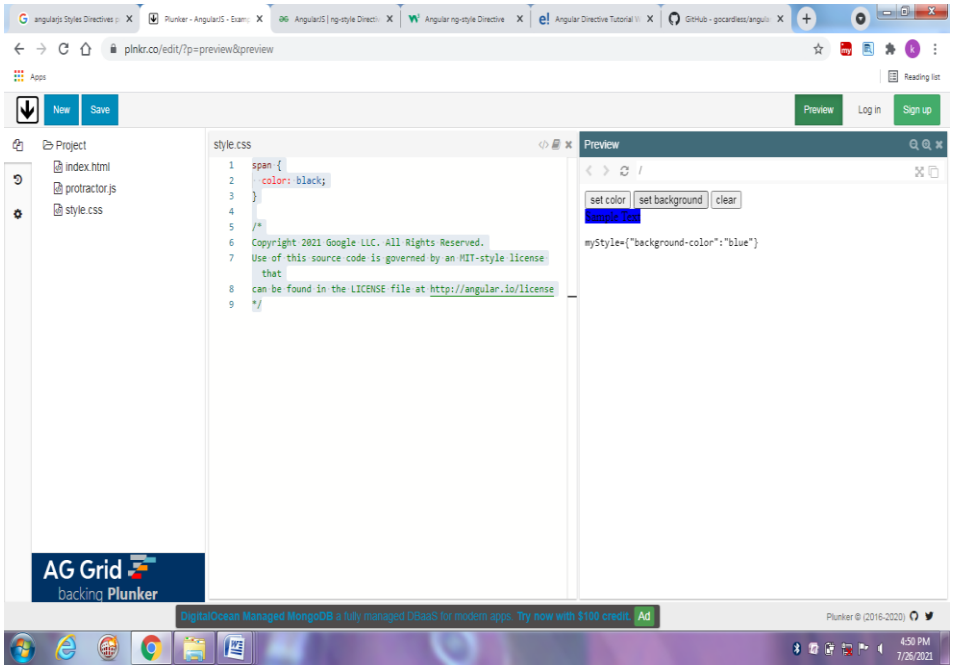
Style.css

```
span {
  color: black;
}
```

OUTPUT







QUESTIONS

1. Describe the parameter values of ng-style directive?
2. At what priority level does ng-style directive work?
3. What are the two usage of ng-style directive?

*****

EXPERIMENT 3

AIM:
Create an application to demonstrate mouse and keyboard events directives

OBJECTIVE:
Generally while developing applications different type of html DOM events like mouse clicks, key press, change events, etc can be used likewise angularjs is having its own event directives for DOM interactions.In angularjs different type of DOM event listener directives are available and which can attach those events to html elements.

THEORY:
AngularJS includes certain directives which can be used to provide custom behavior on various DOM events, such as click, dblclick, mouseenter etc. The following table lists AngularJS event directives.

Event Directive
ng-blur
ng-change
ng-click
ng-dblclick
ng-focus
ng-keydown
ng-keyup
ng-keypress
ng-mousedown
ng-mouseenter
ng-mouseleave
ng-mousemove
ng-mouseover
ng-mouseup

ng-click : The ng-click directive is used to provide event handler for click event.

ng-dblclick : The directive ng-dblclick in AngularJS invokes whenever an element with which ng-dblclick is attached is double-clicked. Angular JS will not override the element's original.

ng-focus : This directive will execute the particular code when the user focuses on the element with which the ng-focus directive is attached.

ng-blur : This directive will execute the particular code when a user loses focuses from the element with which ng-blur directive attach.

mouse events : It occurs when the control of cursor moves over an element or an element is clicked by mouse event. The order of mouse event when the cursor moves over an element is:

- ng-mouseover
- ng-mouseenter
- ng-mousemove
- ng-mouseleave

The order of mouse event when the mouse clicks on an element

- ng-mousedown
- ng-mouseup
- ng-click

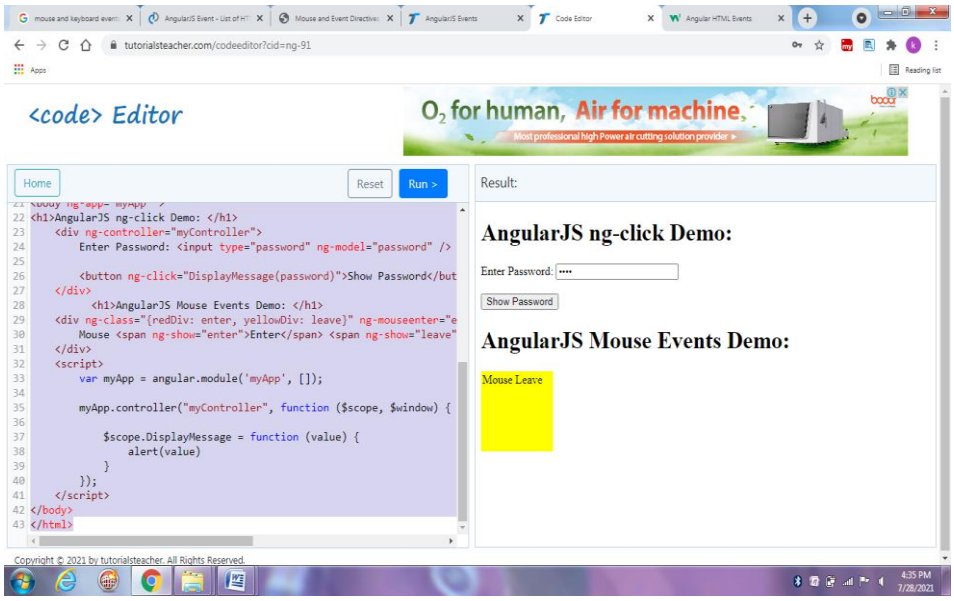
$event Object : passed as an argument, when calling a function. The $event object contains the Browser's event.

PROGRAM

```
<!DOCTYPE html>
<html>
<head>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"
></script>
 <style>
       .redDiv {
          width: 100px;
          height: 100px;
          background-color: red;
          padding:2px 2px 2px 2px;
       }

       .yellowDiv {
          width: 100px;
          height: 100px;
          background-color: yellow;
          padding:2px 2px 2px 2px;
       }
   </style>
  </head>
<body ng-app="myApp" >
<h1>AngularJS ng-click Demo: </h1>
   <div ng-controller="myController">
      Enter Password: <input type="password" ng-model="password" />
<br /><br />
<button ng-click="DisplayMessage(password)">Show Password</button
</div>
```

```
        <h1>AngularJS Mouse Events Demo: </h1>
    <div ng-class="{redDiv: enter, yellowDiv: leave}" ng-mouseenter="enter=true;leave=false;" ng-mouseleave="leave=true;enter=false">
        Mouse <span ng-show="enter">Enter</span> <span ng-show="leave">Leave</span>
    </div>
    <script>
        var myApp = angular.module('myApp', []);
          myApp.controller("myController", function ($scope, $window) {
                    $scope.DisplayMessage = function (value) {
                                        alert(value)
          }
        });
    </script>
</body>
</html>
```

OUTPUT:



In the above example, ng-click directive is used to call a DisplayMessage() function with the 'password' parameter when a user clicks a button. A 'password' is a model property defined using ng-model directive in the input box. The DisplayMessage() function is attached to a $scope object in myController, so it will be accessible from button click as button comes under myController. The $window service is used to display an alert.

Mouse Events:

In the above example, the ng-class directive includes map of CSS classes, so redDiv will be applied if enter=true and yellowDiv will be

applied if leave=true. The ng-mouseenter directive sets 'enter' to true, which will apply redDiv class to the element. In the same way, ng-mouseleave will set leave to true, which will apply yellowDiv class.

QUESTIONS

1. List out any four event directives?
2. What is the use of $event Object?
3. Differentiate between ng-focus and ng-blur directive?

*****

EXPERIMENT 4

AIM:
Create an application to demonstrate builtin directives

OBJECTIVE:
Angular provides a number of built-in *directives,* which are attributes, added to the HTML elements that give us dynamic behavior.

THEORY:
Directives are markers on a DOM element that tell AngularJS to attach a specified behavior to that DOM element or even transform the DOM element and its children. In short, it extends the HTML.

Most of the directives in AngularJS are starting with ng- where ng stands for Angular. AngularJS includes various built-in directives. In addition to this, custom directives can also be created for the application.

The following table lists the important built-in AngularJS directives.

Directive	Description
ng-app	Auto bootstrap AngularJS application.
ng-init	Initializes AngularJS variables
ng-model	Binds HTML control's value to a property on the $scope object.
ng-controller	Attaches the controller of MVC to the view.
ng-bind	Replaces the value of HTML control with the value of specified AngularJS expression.
ng-repeat	Repeats HTML template once per each item in the specified collection.
ng-show	Display HTML element based on the value of the specified expression.
ng-readonly	Makes HTML element read-only based on the value of the specified expression.
ng-disabled	Sets the disable attribute on the HTML element if specified expression evaluates to true.
ng-if	Removes or recreates HTML element based on an expression.
ng-click	Specifies custom behavior when an element is clicked.

AngularJS directives are used to extend HTML. They are special attributes starting with **ng**-prefix.

- **ng-app** − This directive starts an AngularJS Application.
- **ng-init** − This directive initializes application data.
- **ng-model** − This directive defines the model that is variable to be used in AngularJS.

- **ng-repeat** − This directive repeats HTML elements for each item in a collection.

ng-ap :
The ng-app directive starts AngularJS and makes the specified element a root element of the application.

ng-init:
The ng-init directive can be used to initialize variables in AngularJS application.

ng-model:
The ng-model directive is used for two-way data binding in AngularJS. It binds <input>, <select> or <textarea> elements to a specified property on the $scope object. So, the value of the element will be the value of a property and vica-versa.

ng-bind:
The ng-bind directive binds the model property declared via $scope or ng-model directive or the result of an expression to the HTML element. It also updates an element if the value of an expression changes.

ng-repeat:
The ng-repeat directive repeats HTML once per each item in the specified array collection.

ng-if:
The ng-if directive creates or removes an HTML element based on the Boolean value returned from the specified expression. If an expression returns true then it recreates an element otherwise removes an element from the HTML document.

ng-readonly:

The ng-readonly directive makes an HTML element read-only, based on the Boolean value returned from the specified expression. If an expression returns true then the element will become read-only, otherwise not.

The ng-disabled directive disables an HTML element, based on the Boolean value returned from the specified expression. If an expression returns true the element will be disabled, otherwise not.

Example: Directives syntax variation

```
<!DOCTYPE html>
<html >
<head>
   <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app>
```

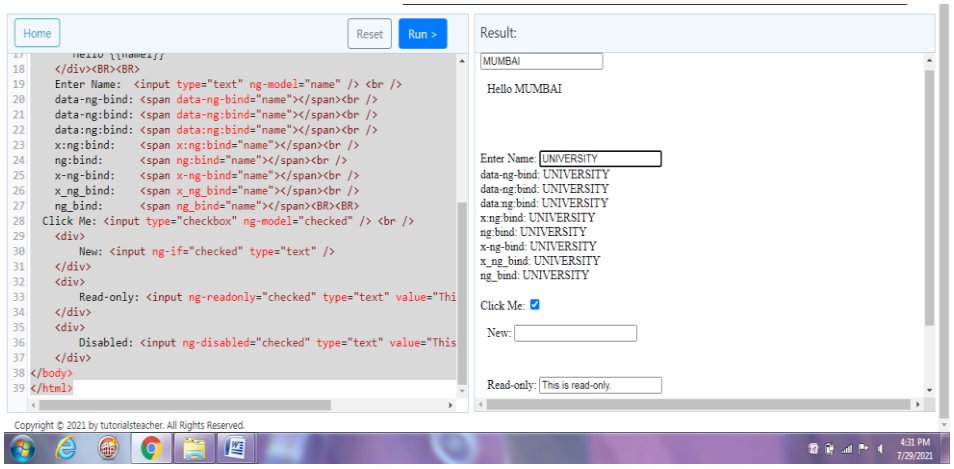
```
   Enter Name:  <input type="text" ng-model="name" /> <br />
   data-ng-bind: <span data-ng-bind="name"></span><br />
   data-ng:bind: <span data-ng:bind="name"></span><br />
   data:ng:bind: <span data:ng:bind="name"></span><br />
   x:ng:bind:    <span x:ng:bind="name"></span><br />
   ng:bind:      <span ng:bind="name"></span><br />
   x-ng-bind:    <span x-ng-bind="name"></span><br />
   x_ng_bind:    <span x_ng_bind="name"></span><br />
   ng_bind:      <span ng_bind="name"></span>
</body>
</html>
```

PROGRAM

```
<!DOCTYPE html>
<html >
<head>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.16/angular.min.js"
></script>
<style>
       div {
          width: 100%;
          height: 50px;
          display: block;
          margin: 15px 0 0 10px;
       }
   </style>
 </head>
<body ng-app ng-init="checked=true">
  <input type="text" ng-model="name1" />
   <div>
     Hello {{name1}}
   </div><BR><BR>
   Enter Name:  <input type="text" ng-model="name" /> <br />
   data-ng-bind: <span data-ng-bind="name"></span><br />
   data-ng:bind: <span data-ng:bind="name"></span><br />
   data:ng:bind: <span data:ng:bind="name"></span><br />
   x:ng:bind:    <span x:ng:bind="name"></span><br />
   ng:bind:      <span ng:bind="name"></span><br />
   x-ng-bind:    <span x-ng-bind="name"></span><br />
   x_ng_bind:    <span x_ng_bind="name"></span><br />
   ng_bind:      <span ng_bind="name"></span><BR><BR>
  Click Me: <input type="checkbox" ng-model="checked" /> <br />
   <div>
     New: <input ng-if="checked" type="text" />
   </div>
```

```
    <div>
        Read-only: <input ng-readonly="checked" type="text" value="This is read-only." />
    </div>
    <div>
        Disabled: <input ng-disabled="checked" type="text" value="This is disabled." />
    </div>
</body>
</html>
```

OUTPUT:



QUESTIONS

1. List out any two built in directives?
2. Difference between ng-app and ng-init?
3. What is the use of ng-readonly directive?

*****

MODULE 7
EXPERIMENT NO. 1

AIM: Demonstrate controllers in Angular.js through an application
a) Programming Controllers & $scope object

OBJECTIVE

Create an application that needs to set up the initial state for the AngularJS $scope. set up the initial state of a scope by attaching properties to the $scope object.

THEORY:
The controller in AngularJS is a JavaScript function that maintains the application data and behavior using $scope object.

This can attach properties and methods to the $scope object inside a controller function, which in turn will add or update the data and attach behaviours to HTML elements. The $scope object is a glue between the controller and HTML.

The ng-controller directive is used to specify a controller in an HTML element, which will add behavior or maintain the data in that HTML element and its child elements.

The following example demonstrates attaching properties to the $scope object inside a controller and then displaying property value in HTML.

Example: AngularJS Controller

```
<!DOCTYPE html>
<html >
<head>
        <title>AngualrJS Controller</title>
        <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="myNgApp">
        <div ng-controller="myController">
    {{message}}
        </div>
        <script>
    var ngApp = angular.module('myNgApp', []);

    ngApp.controller('myController', function ($scope) {
        $scope.message = "Hello World!";
    });
        </script>
</body>
```

```
</html>

Result:
Hello World!
```

In the above example, ng-controller="myController" directive is applied to the element where "myController" is the name of the controller. Inside the div element, we have specified the {{message}} expression.

The $ sign is used as a prefix in all the built-in objects in AngularJS, so that can differentiate AngularJS built-in objects and other objects.

Now, to create "myController", we need to create an application module. The module defines an application and keeps its parts like controllers, services etc. out of global scope. After creating a module, we add a controller function using the controller() method where the first parameter should be the name of the controller and the second parameter should be a function for the controller. The controller function includes $scope parameter which will be injected by AngularJS framework.

Note: AngularJS framework injects $scope object to each controller function. It also injects other services if included as a parameter of controller function.

ALGORITHM:
Steps to create an AngularJS Controller

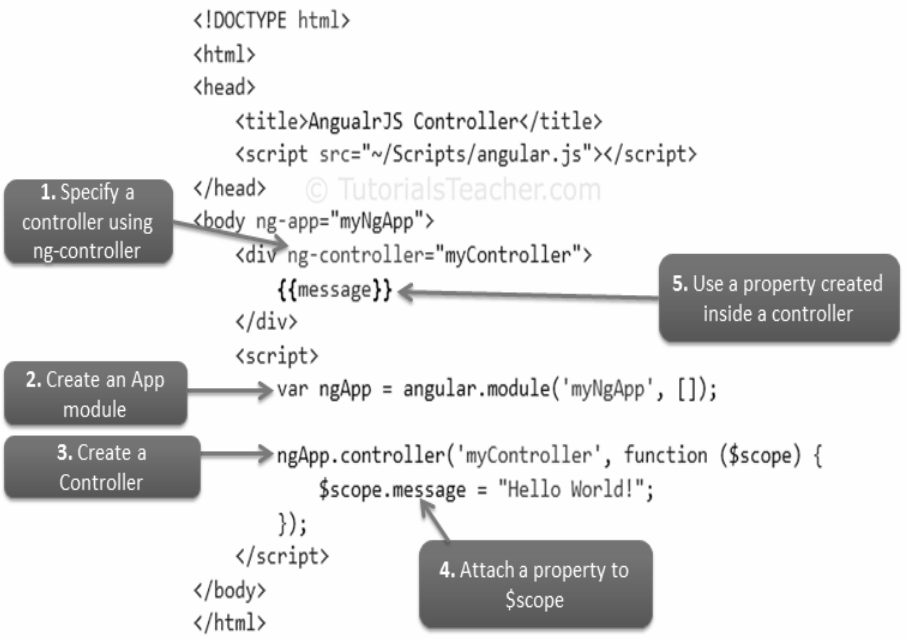
Step 1: Specify the controller using ng-controller.

Step 2: Create an app module.
var myApp = angular.module('spicyApp2', []);

Step 3: Create a controller
myApp.controller('SpicyController', ['$scope', function($scope) {

Step 4: Attach property to scope
$scope.customSpice = 'wasabi';
$scope.spice = 'very';

Step 5: Use a property created inside a controller
{{message}}



PROGRAM:

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Example - example-controller-spicy-2-production</title>
  <script src="//code.angularjs.org/snapshot/angular.min.js"></script>
  <script src="app.js"></script>
</head>

<body ng-app="spicyApp2">
  <div ng-controller="SpicyController">
 <input ng-model="customSpice">
 <button ng-click="spicy('chili')">Chili</button>
 <button ng-click="spicy(customSpice)">Custom spice</button>
 <p>The food is {{spice}} spicy!</p>
</div>
</body>
</html>
```

app.js

```
(function(angular) {
  'use strict';
var myApp = angular.module('spicyApp2', []);

myApp.controller('SpicyController', ['$scope', function($scope) {
    $scope.customSpice = 'wasabi';
    $scope.spice = 'very';
```

```
  $scope.spicy = function(spice) {
    $scope.spice = spice;
  };
}]);
})(window.angular);
```

OUTPUT:

EXPERIMENT NO. 2

AIM:
Demonstrate controllers in Angular.js through an application
b) Adding Behavior to a Scope Object

OBJECTIVE

Create an application that needs to set up the initial state for the AngularJS to add behavior to the scope by attaching methods to the $scope object.

THEORY:
In order to react to events or execute computation in the view we must provide behavior to the scope. We add behavior to the scope by attaching methods to the $scope object. These methods are then available to be called from the template/view.

The following example uses a Controller to add a method, which doubles a number, to the scope:

```
var myApp = angular.module('myApp',[]);

myApp.controller('DoubleController', ['$scope', function($scope)
{
        $scope.double = function(value) { return value * 2; };
}]);
```

Once the Controller has been attached to the DOM, the double method can be invoked in an AngularJS expression in the template:

```
<div ng-controller="DoubleController">
        Two times <input ng-model="num"> equals {{ double(num) }}
</div>
```

An object (or primitives) assigned to the scope become model properties. Any methods assigned to the scope are available in the template/view, and can be invoked via AngularJS expressions and ng event handler directives (e.g. ngClick).

ALGORITHM:

Steps to create an AngularJS Controller by adding method to scope

Step 1: Specify the controller using ng-controller.

```
<div id="div1" ng-controller="myController">
```

Step 2: Create an app module.

```
var ngApp = angular.module('myNgApp', []);
```

Step 3: Create a controller

ngApp.controller('myController', function ($scope)

Step 4: Attach property to scope

$scope.message = "This is myController";

Step 5: Use a property created inside a controller

{{message}}

The "message" property is defined inside myController, so it will only be available to div1 and div2 but not div3 and div4. The same way, message property defined inside anotherController will only be available to div4. The div3 element does not come under any controller, so "message" property will be null or undefined.

PROGRAM:

```
<!DOCTYPE html>
<html>
<head>
        <title>AngualrJS Controller</title>
        <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="myNgApp">
        <div id="div1" ng-controller="myController">
    Message: {{message}} <br />
    <div id="div2">
       Message: {{message}}
    </div>
       </div>
       <div id="div3">
    Message: {{message}}
       </div>
       <div id="div4" ng-controller="anotherController">
    Message: {{message}}
       </div>
       <script>
    var ngApp = angular.module('myNgApp', []);

    ngApp.controller('myController', function ($scope) {
       $scope.message = "This is myController";
    });

    ngApp.controller('anotherController', function ($scope) {
       $scope.message = "This is anotherController";
    });
       </script>
</body>
</html>
```

OUTPUT:

Message: This is myController
Message: This is myController
Message:
Message: This is anotherController

*****

EXPERIMENT No. 3

AIM:
Demonstrate controllers in Angular.js through an application
c) Passing Parameters to the Methods

OBJECTIVE:
Create an application that needs to pass parameters to the methods by using controllers in Angular.js.

THEORY:
To pass arguments to the controller method, we need to define them with those number of arguments and while calling them, we need to call them with that number of arguments.

In the above code snippet, we have declared two functions, Preference and Preference1. The Preference function accepts only one parameter (lang). The Preference1 function accepts two parameters (lang and lang1).

On clicking on the first three buttons, we call the Preference method with only one parameter (language name - English, Hindi etc.) however on clicking on the 4th button, we call the Preference method and pass two parameters 'French' and ' and Hindi language also'.

ALGORITHM:

Steps to create an AngularJS Controller by adding method to scope

Step 1: Create an app module.

```
var app = angular.module('app', []);
```

Step 2: Create a controller & pass parameters

```
app.controller('myController', ['$scope', function($scope)
```

Step 3: Attach property to scope

```
$scope.Language = 'Hindi';
```

Step 4: Specify the controller using ng-controller.

```
<div ng-app="app" ng-controller="myController">
```

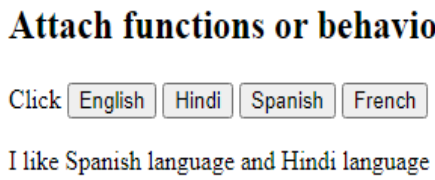
Step 5: Adding button using ng-click

```
<button ng-click="Preference('English')">English</button>
```

PROGRAM:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Demo</title>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.
js"></script>

        <script>
      var app = angular.module('app', []);
      app.controller('myController', ['$scope', function($scope) {
                $scope.Language = 'Hindi';
        $scope.Preference = function(lang) {
          $scope.Language = lang;
        };

        $scope.Preference1 = function(lang, lang1) {
          $scope.Language = lang;
                $scope.Language1 = lang1;
        };
      }]);
  </script>
        </head>
<body>
          <h2>Attach functions or behavior - AngularJS</h2>
  <div ng-app="app" ng-controller="myController">
        Click
        <button ng-click="Preference('English')">English</button>
        <button ng-click="Preference('Hindi')">Hindi</button>
        <button ng-click="Preference('Spanish')">Spanish</button>
        <button ng-click="Preference1('French', ' and Hindi language
also.')">French</button>
        <p>I like {{ Language}} language {{ Language1 }} </p>
  </div>
</body>
</html>
```

OUTPUT:



*****

EXPERIMENT No. 4

AIM:
Demonstrate controllers in Angular.js through an application
d) Listing from array in AngularJS

OBJECTIVE:
Create an application that demonstrates listing from arrays in AngularJS controllers.

THEORY:
In this case also we shall use ng-repeat directive to list the data from the array. However, here will see how to list the data from an array that is declared in JavaScript.

In the code snippet, we have declared the usernames array in JavaScript.

We have created a module named "app" and then a controller. In the controller, we have set the Usernames property of the $scope to the usernames array declared.

In the HTML code, the ng-app, ng-controller both have been declared in the same element.

ALGORITHM:

Steps to create an AngularJS Controller by listing arrays

Step 1: Create and define an array.
var usernames = ['Sheo', 'Narayan', 'Dutta'];

Step 2: Create an app module.
var app = angular.module("app", []);

Step 3: Create a controller & pass $scope to function.
app.controller("ArrayController", ["$scope", function ($scope)

Step 4: Attach property to scope
$scope.Usernames = usernames;

Step 5: Define the ng-controller

Step 6: Use a ng-repeat to read value from arrays
<li ng-repeat="username in Usernames">{{username}}

PROGRAM:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Demo</title>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.m
in.js"></script>

        </head>
<body>
<h4>Looping array</h4>
<script>
   var usernames = ['Sheo', 'Narayan', 'Dutta'];

   var app = angular.module("app", []);
   app.controller("ArrayController", ["$scope", function ($scope) {
      $scope.Usernames = usernames;
   }]);

</script>

<div ng-app="app" ng-controller="ArrayController">
   <ul>
      <li ng-repeat="username in Usernames">{{username}}</li>
   </ul>
</div>
</body>
</html>
```

OUTPUT:

Looping array

- Sheo
- Narayan
- Dutta

*****

EXPERIMENT NO. 4

AIM:
Program to implement Nested Controllers using angularJS

THEORY:
Angular allows nested controllers. The following example demonstrates multiple controllers.

In this controller's example, a child controller can access properties and methods attached in the parent controller function, whereas a parent controller cannot access properties and methods attached in the child controller.

ALGORITHM:

Steps to create an AngularJS Nested Controller

Step 1:
Add the script
<script src="~/Scripts/angular.js"></script>

Step 2:
Define the ng-app directive
<body ng-app="myNgApp">

Step 3:
Define the ng-controller directive for parent

Step 4:
Define the ng-controller directive for child

Step 5:
Create an app module.
var ngApp = angular.module('myNgApp', []);

Step 6:
Create a ngApp controller for parent & pass $scope to function.

```
ngApp.controller('parentController', function ($scope) {
        $scope.message1 = "This is parentController";
    });
```

Step 7:

Create a ngApp controller for child & pass $scope to function.

```
ngApp.controller('childController', function ($scope) {
        $scope.message2 = "This is childController";
    });
```

PROGRAM:

```
<!DOCTYPE html>
<html>
<head>
        <title>AngualrJS Controller</title>
        <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="myNgApp">
        <div ng-controller="parentController">
    Message: {{message1}}
    <div ng-controller="childController">
        Parent Message: {{message1}}  </br>
        Child Message: {{message2}}
    </div>
    Child Message: {{message2}}
        </div>
        <script>
     var ngApp = angular.module('myNgApp', []);

    ngApp.controller('parentController', function ($scope) {
        $scope.message1 = "This is parentController";
    });

    ngApp.controller('childController', function ($scope) {
        $scope.message2 = "This is childController";
    });
        </script>
</body>
</html>
```

OUTPUT:

Message: This is parentController

Parent Message: This is parentController

Child Message: This is childController

Child Message:

MODULE 7 QUESTIONS:

1. What are Controllers? Explain with an example?
2. What is a $Scope object?
3. How to add behaviour to the scope object?
4. How to Test Controller?
5. How to pass parameters to the controller?
6. What is meant by Array as members in Controller Scope?

*****

MODULE 8

EXPERIMENT NO. 1

AIM:
Demonstrate features of Simple Angular.js forms with a program

OBJECTIVE:
Create Simple Angular Forms using different input controls & events.

THEORY:
AngularJS facilitates you to create a form enriched with data binding and validation of input controls.

Input controls are ways for a user to enter data. A form is a collection of controls for the purpose of grouping related controls together.

Following are the input controls used in AngularJS forms:

- input elements
- select elements
- button elements
- textarea elements

AngularJS provides multiple events that can be associated with the HTML controls. These events are associated with the different HTML input elements.

Following is a list of events supported in AngularJS:

- ng-click
- ng-dbl-click
- ng-mousedown
- ng-mouseup
- ng-mouseenter
- ng-mouseleave
- ng-mousemove
- ng-mouseover
- ng-keydown
- ng-keyup
- ng-keypress
- ng-change

ALGORITHM

Step 1:.Specify a controller using ng-controller.

Step 2: Create a form with novalidate

<form name = "studentForm" novalidate>

Step 3: Add the the input controls in AngularJS forms using ng-model

<input name = "firstname" type = "text" ng-model = "firstName" required>

Step 4: Add CSS property to table

Step 5: Create a controller

mainApp.controller('studentController', function($scope)

Step 6: Add events to the submit button using ng-click.

PROGRAM

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular JS Forms</title>
    <script src = "http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>

    <style>
      table, th , td {
        border: 1px solid grey;
        border-collapse: collapse;
        padding: 5px;
      }

      table tr:nth-child(odd) {
        background-color: lightpink;
      }

      table tr:nth-child(even) {
        background-color: lightyellow;
      }
    </style>

  </head>
```

```
  <body>

    <h2>AngularJS Sample Application</h2>
    <div ng-app = "mainApp" ng-controller = "studentController">

      <form name = "studentForm" novalidate>
        <table border = "0">
          <tr>
            <td>Enter first name:</td>
            <td><input name = "firstname" type = "text" ng-model
= "firstName" required>
              <span style = "color:red" ng-show =
"studentForm.firstname.$dirty &&
studentForm.firstname.$invalid">
                <span ng-show =
"studentForm.firstname.$error.required">First Name is
required.</span>
              </span>
            </td>
          </tr>

          <tr>
            <td>Enter last name: </td>
            <td><input name = "lastname"  type = "text" ng-model
= "lastName" required>
              <span style = "color:red" ng-show =
"studentForm.lastname.$dirty &&
studentForm.lastname.$invalid">
                <span ng-show =
"studentForm.lastname.$error.required">Last Name is
required.</span>
              </span>
            </td>
          </tr>

          <tr>
            <td>Email: </td><td><input name = "email" type =
"email" ng-model = "email" length = "100" required>
              <span style = "color:red" ng-show =
"studentForm.email.$dirty && studentForm.email.$invalid">
                <span ng-show =
"studentForm.email.$error.required">Email is required.</span>
                <span ng-show =
"studentForm.email.$error.email">Invalid email address.</span>
              </span>
            </td>
          </tr>
          <tr>
            <td>
```

```
            <button ng-click = "reset()">Reset</button>
          </td>
          <td>
            <button ng-disabled = "studentForm.firstname.$dirty &&
              studentForm.firstname.$invalid || studentForm.lastname.$dirty &&
              studentForm.lastname.$invalid || studentForm.email.$dirty &&
              studentForm.email.$invalid" ng-click="submit()">Submit</button>
          </td>
        </tr>

      </table>
    </form>
  </div>

  <script>
    var mainApp = angular.module("mainApp", []);

    mainApp.controller('studentController', function($scope) {
      $scope.reset = function(){
        $scope.firstName = "idol";
        $scope.lastName = "mumbai university";
        $scope.email = "mca@mu.ac.in";
      }

      $scope.reset();
    });
  </script>

</body>
</html>
```

OUTPUT:

AngularJS Sample Application

Enter first name:		First Name is required.
Enter last name:		Last Name is required.
Email:		Email is required. Invalid email address.
Reset	Submit	

EXPERIMENT NO. 2

AIM:
Demonstrate Select & Option features of Angular.js forms with a program

OBJECTIVE:
Create Simple Angular Forms that demonstrate drop-down list using select and option directive.

THEORY:
In AngularJS, we can create a dropdown list (select box) based on items in an array, or an object.

Using ng-options:
We should use the ng-option directive to create a dropdown list, based on an object or an array in AngularJS.
We can also use the ng-repeat directive to make the same dropdown list as ng-options.

ng-options vs. ng-repeat:
Although, both can be used for dropdown list, but ng-repeat directive repeats a block of HTML code for each item in an array, it can be used to create options in a dropdown list, while the ng-options directive was made especially for filling a dropdown list with options, and has at least one important advantage:

Dropdowns made with ng-options allow the selected value to be an object, while dropdowns made from ng-repeat have to be a string.

Consider that you have an array of objects:

```
$scope.cars = [
   {model : "Ford Mustang", color : "red"},
   {model : "Fiat 500", color : "white"},
   {model : "Volvo XC90", color : "black"}
];
```

Limitation of ng-repeat:
The ng-repeat directive has a limitation that the selected value must be a string:
While using the ng-options directive, you can select an object value:

Use data source as an object:
We can also use a data source as an object.
Consider that you have an object with following key-value pairs:

```
$scope.cars = {
    car01 : "Ford",
      car02 : "Honda",
      car03 : "Volvo",
      car03 : "Hundai",
```

};

ALGORITHM:

Step 1:.Specify a controller using ng-controller.

Step 2: Select ng-model directive
<select ng-model="selectedCar">

Step 3: Add the ng-repeat directive in the option.
<optionng-repeat="x in cars" value="{{x.model}}">{{x.model}}</option>

Step 4: Create an app module.
var app = angular.module('myApp', []);

Step 5: Create a controller
app.controller('myCtrl', function($scope)

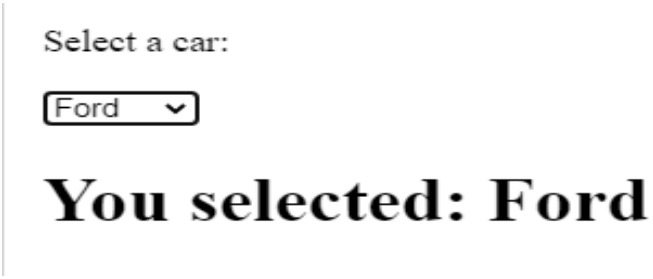
Step 6: Attach property to scope
$scope.cars

PROGRAM:

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angula
r.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
<p>Select a car:</p>
<select ng-model="selectedCar">
<option ng-repeat="x in cars"
value="{{x.model}}">{{x.model}}</option>
</select>
<h1>You selected: {{selectedCar}}</h1>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.cars = [
      {model : "Ford", color : "red"},
      {model : "Honda", color : "white"},
      {model : "Volvo", color : "black"},
      {model : "Hundai", color : "gray"}
```

```
    ];
});
</script>
</body>
</html>
```

OUTPUT:



*****

EXPERIMENT NO. 3

AIM:
Demonstrate Input Validation features of Angular.js forms with a program

OBJECTIVE:
Create Simple Angular Forms that demonstrate input validation to name, address, contact number and email address fields.

THEORY
AngularJS provides client-side form validation. It checks the state of the form and input fields (input, textarea, select), and lets its notify the user about the current state. It also holds the information about whether the input fields have been touched, or modified, or not.

Following directives are generally used to track errors in an AngularJS form:
$dirty - states that value has been changed.
$invalid - states that value entered is invalid.
$error - states the exact error.

AngularJS offers client-side form validation. AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state. AngularJS also holds information about whether they have been touched, or modified, or not. We can use standard HTML5 attributes to validate input, or we can make our own validation functions. Client-side validation cannot alone secure user input. Server side validation is also necessary.

Required:
Adding the required HTML5 tag to an input field validates whether the field is empty or not. This ensures that the field should have some value. The following syntax is used for required input.
<input type="text" required />

AngularJS provides a **ng-required** directive to do the same thing. Using this directive you have the flexibility to set the input field should have a value or not.

The following syntax is used to make sure that the input field should not be empty. We can set it to false if you do not want to restrict this.
<input type="text" ng-required="true" />

Minimum Length:
The directive ng-minlength is used to validate the minimum length of the input value. This will make sure that the length of the entered value is not less than the value set using ng-minlength directive.
<input type="text" ng-minlength=10 />
Maximum Length:

The directive ng-maxlength is used to validate the maximum length of the input value. This will make sure that the length of the entered value is not more than the value set using ng-max length directive.

<input type="text" ng-maxlength=20 />

Pattern:

The ng-pattern directive is used to ensure that an input matches a regex pattern, the following syntax is used. The ng-pattern directive returns true if input text is validated as per expression otherwise it returns false.

<input type="text" ng-pattern="[a-zA-Z]" />

Email:

We can set the input type to email to ensure that the input field is a valid email id.

<input type="email" name="email" ng model="user.email" />

Number:

We can set the input type to number to ensure that the input field is a number.

<input type="number" name="age" ng-model="user.age" />

URL:

We can set the input type to url to ensure that the input field is a url.
<input type="url" name="homepage" ng-model="user.url" />

Input State:

AngularJS is constantly updating the state of both the form and the input fields.

Input fields have the following states:

- $untouched The field has not been touched yet
- $touched The field has been touched
- $pristine The field has not been modified yet
- $dirty The field has been modified
- $invalid The field content is not valid
- $valid The field content is valid

They are all properties of the input field, and are either true or false.

Form State:

Forms have the following states:

- $pristine No fields have been modified yet
- $dirty One or more have been modified
- $invalid The form content is not valid
- $valid The form content is valid
- $submitted The form is submitted

They are all properties of the form, and are either true or false.

ALGORITHM:

Step 1:.Specify a controller using ng-controller.
<body ng-app="formApp" ng-controller="formCtrl"

Step 2: Create an app module.
var app = angular.module('formApp', []);

Step 3: Create a controller
app.controller('formCtrl', function ($scope)

Step 4: Attach property to scope
$scope.sendForm = function ()

Step 5: Add CSS

Step 6: Add the validation to the input state field.

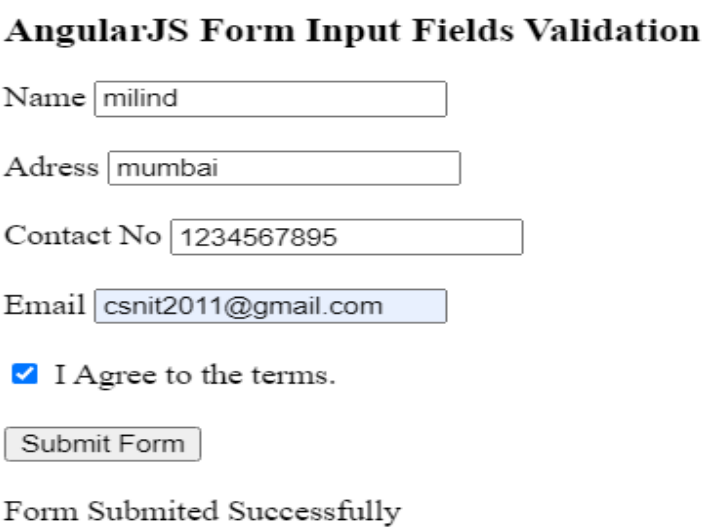
PROGRAM:

```
<!DOCTYPE html>
<html>
<head>
<title>
AngularJs Form Input Fields Validations Example
</title>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"
></script>
<script>
var app = angular.module('formApp', []);
app.controller('formCtrl', function ($scope) {

$scope.sendForm = function () {
$scope.msg='Form Submited Successfully';
};

$scope.getClass = function (color) {
return color.toString();
}
});
```

```
</script>
<style>
.valid.false {
background: red;
}
.valid.true {
background: green;
}
.error {
color: red;
}
</style>
</head>
<body ng-app="formApp" ng-controller="formCtrl">
<h3>AngularJS Form Input Fields Validation</h3>
<form name="personForm" ng-submit="sendForm()">
<label for="name">Name</label>
<input id="name" name="name" type="text" ng-model="person.name"
required />
<span class="error" ng-show="personForm.name.$error.required">
Required! </span>
<br /><br />
<label for="adress">Adress</label>
<input id="address" name="address" type="text" ng-
model="person.address" required />
<span class="error" ng-show="personForm.address.$error.required">
Required! </span>
<br /><br />
<label for="contact">Contact No</label>
<input id="mobile" name="mobile" type="number" ng-
model="person.mobile" required />
<span class="error" ng-
show="personForm.mobile.$error.required">Required number!</span>
<span class="error" ng-
show="personForm.mobile.$error.mobile">Invalid mobile!</span>
<br /><br />
<label for="email">Email</label>
<input id="email" name="email" type="email" ng-model="person.email"
required />
<span class="error" ng-
show="personForm.email.$error.required">Required!</span>
<span class="error" ng-show="personForm.email.$error.email">Invalid
Email!</span>
<br /><br />
<input type="checkbox" ng-model="terms" name="terms" id="terms"
required />
<label for="terms">I Agree to the terms.</label>
<span class="error" ng-show="personForm.terms.$error.required">You
must agree to the terms</span>
```

```
<br /><br />
<button type="submit">Submit Form</button>
<br /><br />
<span>{{msg}}</span>
</form>
</body>
</html>
```

OUTPUT:



*****

EXPERIMENT NO. 4

AIM:
Demonstrate features of Angular.js forms with a program using CSS classes.

OBJECTIVE:
Create Simple Angular Forms that demonstrate input validation using CSS classes.

THEORY:
AngularJS adds CSS classes to forms and input fields depending on their states.

The following classes are added to, or removed from, input fields:

ng-untouched The field has not been touched yet

ng-touched The field has been touched

ng-pristine The field has not been modified yet

ng-dirty The field has been modified

ng-valid The field content is valid

ng-invalid The field content is not valid

ng-valid-key One key for each validation. Example: ng-valid-required, useful when there are more than one thing that must be validated

ng-invalid-key Example: ng-invalid-required

The following classes are added to, or removed from, forms:

ng-pristine No fields has not been modified yet

ng-dirty One or more fields has been modified

ng-valid The form content is valid

ng-invalid The form content is not valid

ng-valid-key One key for each validation. Example: ng-valid-required, useful when there are more than one thing that must be validated

ng-invalid-key Example: ng-invalid-required

The classes are removed if the value they represent is false.
Add styles for these classes to give your application a better and more intuitive user interface.

ALGORITHM:

Step 1:.Specify a controller using ng-controller.

Step 2: Specify the ng-model in input fields of forms.

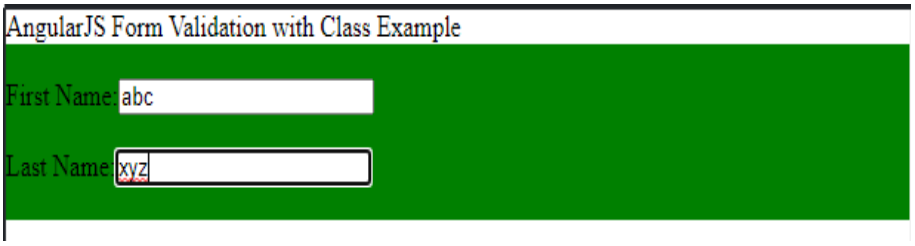
<input type="text" name="firstname" ng-model="person.fname" required />

Step 3: Add the CSS property to form using form.ng-valid and form.ng-invalid.

PROGRAM:

```
<!DOCTYPE html>
<html>
<head>
<title>
AngularJs Form Validation with Classes Example
</title>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js
"></script>
<style>
form.ng-invalid {
background-color:red;
}
form.ng-valid {
background-color:green;
}
</style>
</head>
<body>
<div ng-app="">
<h2>AngularJS Form Validation with Class Example</h2>
<form name="personForm">
<br />
First Name:<input type="text" name="firstname" ng-
model="person.fname" required /><br /><br />
Last Name:<input type="text" name="lastname" ng-
model="person.lname" required />
<br /><br />
</form>
</div>
</body>
</html>
```

OUTPUT:



QUESTION

1. Explain the different input controls and events used in SPA?
2. Explain the difference between ng-option & ng-repeat?
3. What are the limitations of ng-repeat?
4. What are the different CSS classes added to SPA?

*****

MODULE 8
EXPERIMENT NO. 1

AIM:
Write a Angular.js program to implement the concept of Single page application.

OBJECTIVE:
Create a single page application that loads a single HTML page and only a part of the page instead of the entire page gets updated with every click of the mouse.

THEORY:
Single page applications or (SPAs) are web applications that load a single HTML page and dynamically update the page based on the user interaction with the web application.

Single page application (SPA) is a web application that fits on a single page. All your code (JS, HTML, CSS) is retrieved with a single page load. And navigation between pages performed without refreshing the whole page.

The page does not reload or transfer control to another page during the process. This ensures high performance and loading pages faster. Most modern applications use the concept of SPA. In the SPA, the whole data is sent to the client from the server at the beginning. As the client clicks certain parts on the webpage, only the required part of the information is fetched from the server and the page is rewritten dynamically. This results in a lesser load on the server and is cost-efficient. SPAs use AJAX and HTML5 to create fluid and responsive Web applications and most of the work happens on the client-side.

Popular applications such as Facebook, Gmail, Twitter, Google Drive, Netflix, and many more are examples of SPA.

Advantages:

1. Team collaboration:
Single-page applications are excellent when more than one developer is working on the same project. It allows backend developers to focus on the API, while the frontend developers can focus on creating the user interface based on the backend API.

2. Caching:
The application sends a single request to the server and stores all the received information in the cache. This proves beneficial when the client has poor network connectivity.

3. Fast and responsive:
As only parts of the pages are loaded dynamically, it improves the website's speed.

- Debugging is easier

Debugging single page applications with chrome is easier since such applications are developed using AngularJS Batarang and React developer tools.

- Linear user experience

Browsing or navigating through the website is easy.

Disadvantages:

1. SEO optimization:
SPAs provide poor SEO optimization. This is because single-page applications operate on JavaScript and load data at once server. The URL does not change and different pages do not have a unique URL. Hence it is hard for the search engines to index the SPA website as opposed to traditional server-rendered pages.

2. Browser history:
A SPA does not save the users' transition of states within the website. A browser saves the previous pages only, not the state transition. Thus when users click the back button, they are not redirected to the previous state of the website. To solve this problem, developers can equip their SPA frameworks with the HTML5 History API.

3. Security issues:
Single-page apps are less immune to cross-site scripting (XSS) and since no new pages are loaded, hackers can easily gain access to the website and inject new scripts on the client-side.

4. Memory Consumption:
Since the SPA can run for a long time, sometimes hours at a time, one needs to make sure the application does not consume more memory than it needs. Else, users with low memory devices may face serious performance issues.

5.Disabled Javascript:
Developers need to chalk out ideas for users to access the information on the website for browsers that have Javascript disabled.

ALGORITHM:

Step 1: Create a Module

We all know that AngularJS follows MVC architecture. Hence, every AngularJS application contains a module consisting of controllers, services, etc.

```
var app = angular.module('myApp', []);
```

Step 2: Define a Simple Controller

```
app.controller('FirstController', function($scope) {
$scope.message = 'Hello from FirstController';
});
```

Step 3: Include AngularJS script in HTML code

Specify the module (created in step 1) in ng-app attribute and the controller (defined in step 2) in the ng-controller attribute.

Step 4: Use AngularJS's routing capabilities to add different views to our SPA

We need to include angular-route script after the main angular script.

```
<script
   src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular.min.
   js"></script>
<script
   src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular-
   route.min.js"></script>
```

We need to use the ngRoute directive to enable routing.

```
var app = angular.module('myApp', ['ngRoute']);
```

Step 5: Create an HTML layout for the website

Once an HTML layout for the website is created, we need to use the ng-view directive to specify the place where the HTML of each page will be placed in our layout.

Step 6: Use $routeProvider service from ngRoute module to configure the navigation to different views

It is necessary to specify a templateUrl and a controller for each route that we wish to add.

Exception handling has to be accommodated when a user tries to navigate to a route that doesn't exist. For simplicity, we can write an "otherwise" function to redirect the user to the "/" route.

Step 7: Build controllers for every route specified in $routeProvider

Step 8: Configure the pages

Step 9: Add links to the HTML that will help in navigating to the configured pages

Step 10: Include the HTML of routing pages to index.html file using script tag

PROGRAM:

```
<!doctype html>
<html ng-app="myApp">
<head>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular.min.js
"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.7/angular-
route.min.js"></script>
</head>
<body>
<script type="text/ng-template" id="pages/first.html">
<h1>First</h1>
<h3>{{message}}</h3>
</script>
<script type="text/ng-template" id="pages/second.html">
<h1>Second</h1>
<h3>{{message}}</h3>
</script>
<script type="text/ng-template" id="pages/third.html">
<h1>Third</h1>
<h3>{{message}}</h3>
</script>

<a href="#/">First</a>
<a href="#/second">Second</a>
<a href="#/third">Third</a>

<div ng-view></div>

<script src="app.js"></script>
</body>
</html>
```

OUTPUT:


First Second Third
First
Hello from FirstController


First Second Third
Second
Hello from SecondController


First Second Third
Third
Hello from ThirdController

*****

EXPERIMENT NO. 2

AIM:
Demonstrate route provider features of Angular.js forms with a program

OBJECTIVE:
Create a single page application that shows a routing provider demo with layout and template.

THEORY:
AngularJS Routing

We can build Single Page Application (SPA) with AngularJS. It is a web app that loads a single HTML page and dynamically updates that page as the user interacts with the web app.

AngularJS supports SPA using the routing module ngRoute. This routing module acts based on the url. When a user requests a specific url, the routing engine captures that url and renders the view based on the defined routing rules.

Implementation of simple routing in an AngularJS application.

Routing Example:
Let's build an application, which will display a login page when a user requests a base url - http://localhost/. Once the user logs in successfully, we will redirect it to the student page http://localhost/student/{username} where the username would be logged in user's name.

In our example, we will have one layout page - index.html, and two HTML templates - login.html and student.html.

1. Index.html - layout view
2. login.html - template
3. student.html - template

ALGORITHM:

Step 1:
To include angular.js, angular-route.js, and bootstrap.css in the index.html. The angular-route.js includes necessary functions for routing.

Step 2:
Apply ng-app directive.

Step 3:

Apply ng-view directive to or other elements where you want to inject another child view. AngularJS routing module uses ng-view directive to inject another child view where it is defined. Therefore, Angular will inject login.html or student.html inside this div element.

Step 4:

Now, create an application module and specify 'ngRoute' as a dependency module.

Step 5:

Now, we need to configure the routing rules that need to be compiled before any other module of an application. So, use the config() method to configure the routing rules using the $routingProvider object.

Step 6:

Use $routeProvider.when(path, route) method to configure routing rules, where the first parameter is request URL and the second parameter is an object which contains controller, template, or other properties. In the above example, we specified that if a user requests for "/" URL, meaning the base url then inject login.html and loginController. In the same way, if a user requests for "/student/:username" url then inject student.html and studentController. The :username would be the url parameter.

Step 7: Use otherwise() method to redirect to base url if user requests for the URL other than configured rules.

Step 8: Now, define loginController which attaches authenticate() function to the $scope. The authenticate() method redirects to "/student/username/" using the $location service.

Step 9: Define studentController which attaches username property to $scope, to display user name in the view. Notice that $routeParams is used to get the value of the url parameter supplied from the login view.

PROGRAM:

1. **Index.html - layout view**
2. **login.html - template**
3. **student.html - template**

Index.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
        <title></title>
        <script src="Scripts/angular.js"></script>
        <script src="Scripts/angular-route.js"></script>
        <link href="Content/bootstrap.css" rel="stylesheet" />
</head>
<body ng-app="ngRoutingDemo">
        <h1>Angular Routing Demo</h1>

        <div ng-view>

        </div>
        <script>
    var app = angular.module('ngRoutingDemo', ['ngRoute']);

    app.config(function ($routeProvider) {

        $routeProvider.when('/', {
                templateUrl: '/login.html',
                controller: 'loginController'
        }).when('/student/:username', {
                templateUrl: '/student.html',
                controller: 'studentController'
        }).otherwise({
                redirectTo: "/"
        });

    app.controller("loginController", function ($scope, $location) {

        $scope.authenticate = function (username) {
                // write authentication code here..

                $location.path('/student/' + username)
        };

    });

    app.controller("studentController", function ($scope, $routeParams) {
        $scope.username = $routeParams.username;
    });

        });
        </script>
</body>
</html>
```

Create login.html as shown below, which contains username and password input box with validation. Please note that we are using bootstrap.css.

login.html

```
<form class="form-horizontal" role="form" name="loginForm" novalidate>
        <div class="form-group" >
      <div class="col-sm-3">
      </div>
      <div class="col-sm-6">
        <input type="text" id="userName" name="userName" placeholder="User Name" class="form-control" ng-model="userName" required />
        <span class="help-block" ng-show="loginForm.userName.$touched && loginForm.userName.$invalid">Please enter User Name.</span>
      </div>
      <div class="col-sm-3">
      </div>

        </div>
        <div class="form-group" >
      <div class="col-sm-3">
      </div>
      <div class="col-sm-6">
        <input type="password" id="password" name="password" placeholder="Password" class="form-control" ng-model="password" required />
        <span ng-show="loginForm.password.$touched && loginForm.password.$error.required">Please enter Password.</span>
      </div>
      <div class="col-sm-3">
      </div>
        </div>

        <input type="submit" value="Login" class="btn btn-primary col-sm-offset-3" ng-click="authenticate(userName)" />
</form>
```

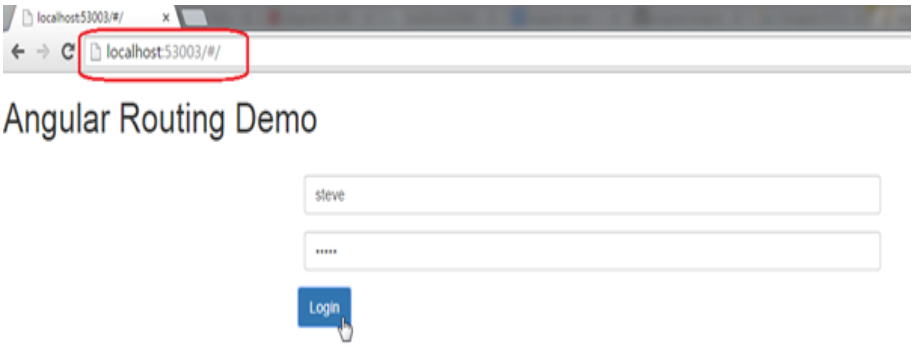
Create student.html with necessary fields as shown below. Visit the Bootstrap Form section to learn how to create bootstrap forms in AngularJS.

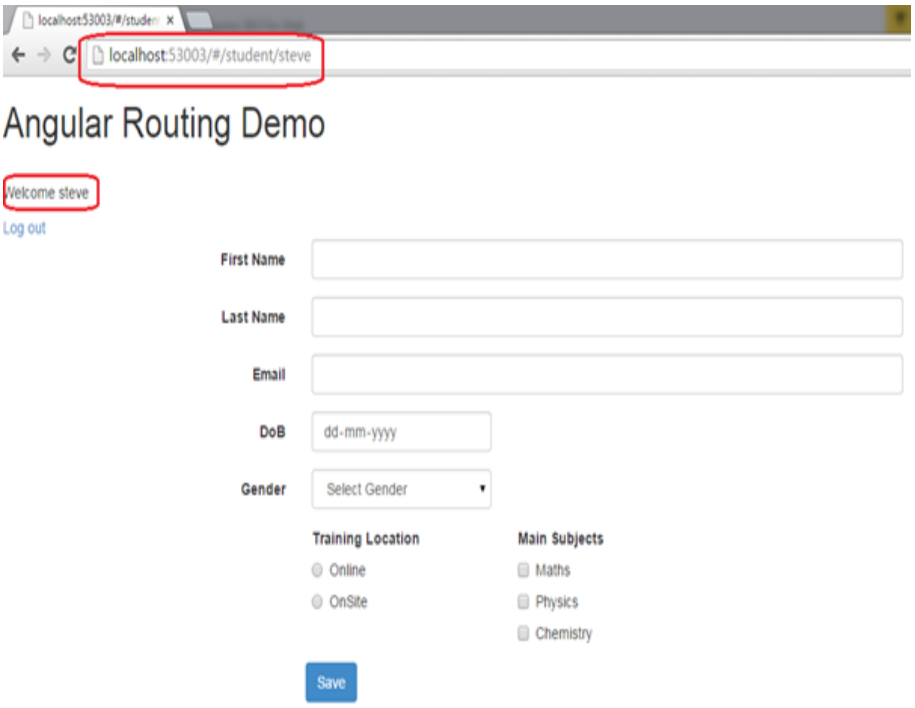
student.html

```
<div>
        <p>Welcome {{username}}</p>
        <a href="/">Log out</a>
</div>

<form class="form-horizontal" ng-submit="submitStudnetForm()" role="form">
        <div class="form-group">
      <label for="firstName" class="col-sm-3 control-label">First Name</label>
      <div class="col-sm-6">
        <input type="text" id="firstName" class="form-control" ng-model="student.firstName" />
      </div>
      <div class="col-sm-3"></div>

        </div>
        <div class="form-group">
      <label for="lastName" class="col-sm-3 control-label">Last Name</label>
      <div class="col-sm-6">
        <input type="text" id="lastName" class="form-control" ng-model="student.lastName" />
      </div>
      <div class="col-sm-3"></div>
        </div>

        <div class="form-group">
      <label for="dob" class="col-sm-3 control-label">DoB</label>
      <div class="col-sm-2">
        <input type="date" id="dob" class="form-control" ng-model="student.DoB" />
      </div>
      <div class="col-sm-7"></div>
        </div>

        <div class="form-group">
      <label for="gender" class="col-sm-3 control-label">Gender</label>
      <div class="col-sm-2">
        <select id="gender" class="form-control" ng-model="student.gender">
                <option value="male">Male</option>
                <option value="female">Female</option>
        </select>
      </div>
      <div class="col-sm-7"></div>
```

```
        </div>

        <div class="form-group">
      <div class="col-sm-3"></div>
      <div class="col-sm-2">
        <span><b>Training Location</b></span>
        <div class="radio">
                <label><input value="online" type="radio"
name="training" ng-model="student.trainingType"
/>Online</label>
        </div>
        <div class="radio">
                <label><input value="onsite" type="radio"
name="training" ng-model="student.trainingType"
/>OnSite</label>
        </div>
      </div>
      <div class="col-sm-7">
        <span><b>Main Subjects</b></span>
        <div class="checkbox">
                <label><input type="checkbox" ng-
model="student.maths" />Maths</label>
        </div>
        <div class="checkbox">
                <label><input type="checkbox" ng-
model="student.physics" />Physics</label>
        </div>
        <div class="checkbox">
                <label><input type="checkbox" ng-
model="student.chemistry" />Chemistry</label>
        </div>
      </div>
        </div>

        <input type="submit" value="Save" class="btn btn-
primary col-sm-offset-3" />
        <input type="reset" value="Reset" ng-
click="resetForm()"
</form>
```

OUTPUT:


localhost:53003/#/
localhost:53003/#/
Angular Routing Demo
steve
Login


localhost:53003/#/student/steve
Angular Routing Demo
Welcome steve
Log out
First Name
Last Name
Email
DoB
dd-mm-yyyy
Gender
Select Gender
Training Location
Online
OnSite
Main Subjects
Maths
Physics
Chemistry
Save

*****

EXPERIMENT NO. 3

AIM:
Demonstrate feature of Animations using angular.js

OBJECTIVE:
Create a simple animation by using ng-hide directive.

THEORY:
The framework offers a very interesting mechanism to hook specific style classes to each step of the life cycle of some of the most used directives such as ngRepeat, ngShow, ngHide, ngInclude, ngView, ngIf, ngClass, and ngSwitch. The first thing that we need to do in order to start is import the angular animation.js file to our application. After that, we just need to declare it in our module as follows:

app.js
```
var parking = angular.module("parking", ["ngAnimate"]);
```

The AngularJS animation uses CSS transitions in order to animate each kind of event such as when we add a new element the array that is being iterated by ngRepeat or when something is shown or hidden through the ngShow directive.

Based on this, it's time to check out the supported directives and their events:

Event	From	To	Directives
Enter	.ng-enter	.ng-enter-active	ngRepeat, ngInclude, ngIf, ngView
Leave	.ng-leave	.ng-leave-active	ngRepeat, ngInclude, ngIf, ngView
Hide	.ng-hide-add	.ng-hide-add-active	ngShow, ngHide
Show	.ng-hide-remove	.ng-hide-remove-active	ngShow, ngHide
Move	.ng-move	.ng-move-active	ngRepeat
addClass	.CLASS-add-class	.CLASS-add-class-active	ngClass
removeClass	.CLASS-remove-class	.CLASS-remove-class-active	ngClass

This means that every time a new element is rendered by an ngRepeat directive, the .ng-enter class is attached to the element and kept there until the transition is over. Right after this, the .ng-enter-active class is also attached, triggering the transition.

Animating ngHide:

To animate the ngHide directive, we need to follow the same previous steps, however, using the .ng-hide-add and .ng-hide-add-active classes:

app.css

```
.ng-hide-add
{
  -webkit-transition: all 5s linear;
  -moz-transition: all 5s linear;
  -ms-transition: all 5s linear;
  -o-transition: all 5s linear;
  transition: all 5s linear;
  opacity: 1;
}

.ng-hide-add-active
{
  display: block !important;
  opacity: 0;
}
```

In this case, the transition must flow in the opposite way. For the fade-out effect, we need to shift from the opacity 1 to 0. This display property is set to block, because the regular behavior of the ngHide directive is to change the display property to none. With that property in place, the element will vanish instantly, and our fade-out effect will not work as expected.

ALGORITHM:

Step 1:

Declare the CSS code for DIV

```
div {
  transition: all linear 0.5s;
  background-color: lightblue;
  height: 100px;
  width: 100%;
  position: relative;
  top: 0;
  left: 0;
}
```

Step 2:

Declare the CSS code for ng-hide directive

```
.ng-hide {
```

```
  height: 0;
  width: 0;
  background-color: transparent;
  top:-200px;
  left: 200px;
}
```

Step 3:

Add the source to script that shows angular animation

```
<script
    src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.j
    s"></script>
<script  src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-
    animate.js">
</script>
```

Step 4:

Add `<body ng-app="ngAnimate">`

Step 5:

Add the input control checkbox

```
<input type="checkbox" ng-model="myCheck"></h1>
```

Step 6:

Add the following

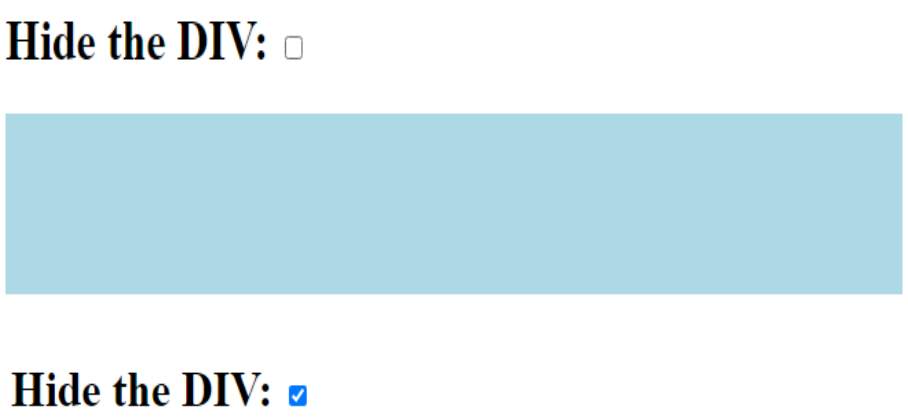
```
<div ng-hide="myCheck"></div>
```

PROGRAM:

```
<!DOCTYPE html>
<html>
<style>
div {
  transition: all linear 0.5s;
  background-color: lightblue;
  height: 100px;
  width: 100%;
  position: relative;
  top: 0;
  left: 0;
}

.ng-hide {
  height: 0;
  width: 0;
  background-color: transparent;
```

```
 top:-200px;
 left: 200px;
}
</style>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular-animate.js"></script>
<body ng-app="ngAnimate">
<h1>Hide the DIV: <input type="checkbox" ng-model="myCheck"></h1>
<div ng-hide="myCheck"></div>
</body>
</html>
```

OUTPUT:



QUESTIONS

1. What are the advantages & disadvantages of SPA?
2. What is ngRoute? Explain with the help of an example.
3. Explain the different directives and their events used during animation?

*****