GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

Department of Computer Science and Engineering

HANDOUT

on

NODE AND ANGULAR JS

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

Program Educational Objectives

- Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.
- Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations
- Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON NODE AND ANGULAR JS

Class & Sem. : III B.Tech – II Semester	Year : 2019-20
Branch : CSE	Credits: 3

==

1. **Brief History and Scope of the Subject**

   NodeJS: Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. MERN has four components that breaths it into life; MongoDB, Express, React, and Node.js.

   AngularJS: AngularJS is a JavaScript-based open-source front-end web framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–viewmodel (MVVM) architectures, along with components commonly used in rich Internet applications. AngularJS is the frontend part of the MEAN stack, consisting of MongoDB database, Express.js web application server framework, Angular.js itself, and Node.js server runtime environment.

2. **Pre-Requisites:**
   - Should have knowledge on HTML, CSS, JavaScript, DOM and XML.
   - Need to be aware of various programming language constructs.

3. **Course Objectives:**

   1. To familiarize with defining own custom AngularJS directives that extend the HTML language.

2. To introduce the concepts of client-side services that can interact with the Node.js web server.
3. To understand the best practices for server side JavaScript.

4. Course Outcomes:

Upon successful completion of the course, the students will be able to

- develop single page applications that reduces app's time to market without plugins.
- identify the services, modules and directives to subdivide application logic into modules and share code across apps
- explain the routing process in angular for managing URL's.
- interpret command line applications in Node.js that allows developers a more maintainable code
- develop code with use of Node.js and JSON services for web applications.
- examine how error events affect piped streams and handling events in Node.js

5. Program Outcomes:

Computer Science and Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools

including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

6. Mapping of Course Outcomes with Program Outcomes:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
CO1	H	M	L	M	H							M
CO2	M	M	M	H	H							L
CO3	M	H	L	L	M					L		L
CO4	L	L	H	M	L			L				M
CO5	M	H	M	H	L	L		M	L		L	H
CO6	H	M	L	M	M			L	L	M	L	M

7. Prescribed Text Books

a. Agus Kurniawan, “Nodejs Programming By Example”, PE Press.
b. Andrew Grant, “Beginning AngularJS”, Apress Publishers.

8. Reference Text Books

a. David Herron, “Node.js Web Development”, 4th edition, Packt Publishing Ltd.
b. Marc Wandschneider, “Learning Node.js: A Hands-On Guide to Building Web Applications in JavaScript”, Addison Wesley.
c. Ken Williamson,”Learning AngularJS: A Guide to AngularJS Development”, O’Relly Media.
d. Matt Frisbie, “AngularJS Web Application Development Cookbook”, Packt Publishing Ltd.

9. URLs and Other E-Learning Resources

- https://www.techiedelight.com/json-introduction/
- https://docs.angularjs.org/tutorial
- https://docs.angularjs.org/tutorial
- https://www.tutorialspoint.com/angularjs/index.htm

10. Digital Learning Materials

- https://freevideolectures.com/course/3982/udemy-understand-nodejs

11. Lecture Schedule / Lesson Plan

TOPIC	No. of Periods	
	Theory	Tutorial
UNIT - I: Introduction to Node.js and JSON		
Introduction	1	0
Operators	1	
Decision statements	1	
Iterative statements	1	
Node.js collections: create array object	1	
Insert, access, update and remove data	2	
JSON : Create JSON object	1	
JSON : Display, access and edit data	1	

JSON Array : Creation	1	
JSON Array : Display, access and edit data	1	
JSON Array: Check JSON attribute	1	
	12	
UNIT - II: Node.js Files, Functions and Strings		
File modules	1	0
Reading text	1	
Creating file	1	
Functions: creating function	1	
Types of functions	1	
Callback function	1	
Strings: operations	2	
String to numeric and vice-versa	1	
String parser	1	
	10	
UNIT - III: **Node.js Modules, Error Handling & Logging and Events**		
Create simple module	1	0
module class	1	
Error handling and logging	2	
Events: Events module	2	
once event listener	2	
Remove events.	1	
	9	
UNIT - IV: Introduction to Angular		
Introduction to TypeScript (TS)	2	0
Node Package Manager	1	
Introduction to Angular	1	
Create angular application using TS and angular CLI	1	
Web pack	1	
Gulp introduction	1	
	7	
UNIT - V: **Elements in Angular**		
Angular components	1	0
Controllers	1	
Modules	2	
Dependency injection	1	
Angular service	1	
providers and directives	2	

Pipes and filters	2	
Angular forms-Reactive	1	
Lifecycle hooks	1	
	12	
UNIT - VI: **Routing in Angular**		
Routing-module	1	0
Component	1	
lazy loading of components	1	
apply route guards security	2	
Angular material design.	1	
	6	
Total No. of Periods:	**56**	**0**

12. Seminar Topics:

- Differences between NodeJS and Javascript
- Overview of AngularJS

Unit – I
Introduction to Node.js and JSON

Syllabus:
Introduction, operators, decision and iterative statements, Node.js collections: create array object, insert, access, update and remove data. JSON: Create JSON object, display, access and edit data. JSON Array: Creation, display, access and edit data. Check JSON attribute.

1.1. Introduction

What is Node.js?

- Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.
- Node.js is server-side scripting based on Google's V8 JavaScript engine.
- It is used to build scalable programs, especially web applications that are computationally simple but are frequently accessed.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.
- Node.js provides the complete solution for server-side applications, such as web platforms.
- It can communicate with other systems, like database, LDAP, and any legacy application.
- Following are the areas where it's perfect to use Node.js.
  - I/O bound Applications
  - Data Streaming Applications
  - Data Intensive Real-time Applications (DIRT)
  - JSON APIs based Applications
  - Single Page Applications

Features of NodeJS

- **Asynchronous event driven IO helps concurrent request handling:**
  - All APIs of Node.js are asynchronous.
  - This feature means that if a Node receives a request for some Input/Output operation, it will execute that operation in the background and continue with the processing of other requests.
  - Thus it will not wait for the response from the previous requests.

- **Fast in Code execution:**
  - Node.js uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome.
  - Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node.js also become faster.
- **Single Threaded but Highly Scalable**:
  - Node.js uses a single thread model for event looping. The response from these events may or may not reach the server immediately.
  - However, this does not block other operations. Thus making Node.js highly scalable.
  - Traditional servers create limited threads to handle requests while Node.js creates a single thread that provides service to much larger numbers of such requests.
- **Node.js library uses JavaScript**:
  - This is another important aspect of Node.js from the developer's point of view.
  - The majority of developers are already well-versed in JavaScript. Hence, development in Node.js becomes easier for a developer who knows JavaScript.
- **There is an Active and vibrant community for the Node.js framework:**
  - The active community always keeps the framework updated with the latest trends in the web development.
- **No Buffering:**
  - Node.js applications never buffer any data. They simply output the data in chunks.

How Does Node.Js Work?

- A Node.js application creates a single thread on its invocation.
- Whenever Node.js receives a request, it first completes its processing before moving on to the next request.
- Node.js works asynchronously by using the event loop and callback functions, to handle multiple requests coming in parallel.
- An Event Loop is a functionality which handles and processes all your external events and just converts them to a callback function.

- It invokes all the event handlers at a proper time. Thus, lots of work is done on the back-end, while processing a single request, so that the new incoming request doesn't have to wait if the processing is not complete.
- While processing a request, Node.js attaches a callback function to it and moves it to the back-end.
- Now, whenever its response is ready, an event is called which triggers the associated callback function to send this response.

When Should We Use Node.Js?

- It's ideal to use Node.js for developing streaming or event-based real-time applications that require less CPU usage such as.
  - Chat applications.
  - Game servers.
- Node.js is good for fast and high-performance servers that face the need to handle thousands of user requests simultaneously.
- **Good For A Collaborative Environment:**
  - It is suitable for environments where multiple people work together.
  - For example, they post their documents, modify them by doing check-out and check-in of these documents.
  - Node.js supports such situations by creating an event loop for every change made to the document.
  - The "Event loop" feature of Node.js enables it to handle multiple events simultaneously without getting blocked.
- **Advertisement Servers:**
  - Here again, we have servers that handle thousands of request for downloading advertisements from a central host. And Node.js is an ideal solution to handle such tasks.
- **Streaming Servers:**
  - Another ideal scenario to use Node.js is for multimedia streaming servers where clients fire request's towards the server to download different multimedia contents from it.
  - To summarize, it's good to use Node.js, when you need high levels of concurrency but less amount of dedicated CPU time.

- Since Node.js uses JavaScript internally, so it fits best for building client-side applications that also use JavaScript.

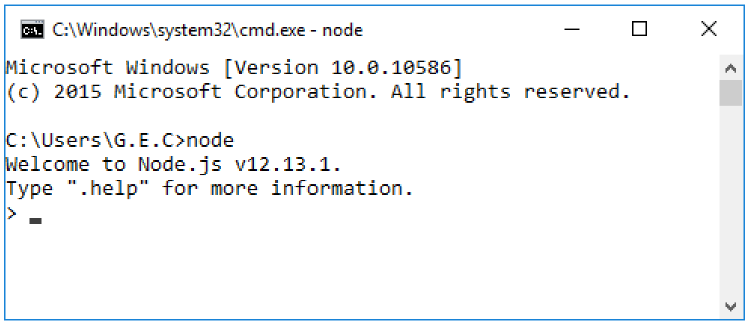
When To Not Use Node.Js?

- However, we can use Node.js for a variety of applications.
- But it is a single threaded framework, so we should not use it for cases where the application requires long processing time.
- If the server is doing some calculation, it won't be able to process any other requests. Hence, Node.js is best when processing needs less dedicated CPU time.

Installation

- Node.js can run on Windows, Linux, and Mac. It provides 32-bit and 64-bit platforms.
- download it from the Node.js website, http://nodejs.org
- You can run the Node.js console manually from Windows Command Prompt (CMD).
- Launch it and type the following:

  node



```
C:\Windows\system32\cmd.exe - node
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\G.E.C>node
Welcome to Node.js v12.13.1.
Type ".help" for more information.
> _
```

Development Tools

- You can use any text editor to write Node.js code.
- If you want to get more development experience, you could use the code editor with rich features such as WebStorm JetBrains, Eclipse, and Visual Studio.
- Some code editors may provide a debugging feature.

Start writing code

- First, run your code editor and write the following:

```
console.log ("Your Text");
```

Save this code to a file named *demo.js*.

- Then open CMD or Terminal (Linux) and execute this file:

```
node demo.js
```

1.2. Operators

- JavaScript has both *binary* and *unary* operators, and one special ternary operator, the conditional operator.
- A binary operator requires two operands, one before the operator and one after the operator.

1.2.1. Arithmetic Operators

Node.js supports basic arithmetic operations: addition, subtraction, multiplication, division and finding remainder.

Example:

```
var a, b;
a = 10;
b = 5.4;
// Addition
var c = a + b;
console.log(c);
// Subtraction
var c = a - b;
console.log(c);
// Multiplication
var c = a * b;
console.log(c);
// Division
var c = a / b;
console.log(c);
```

1.2.2. Comparison Operators

- Node.js adopts C language for comparison operator syntax. The following is the list of comparison operators
- Comparison operator compares its operands and returns a logical value based on whether the comparison is true.
- The operands can be numerical, string, logical, or object values. Strings are compared based on standard lexicographical ordering, using Unicode values.
- In most cases, if the two operands are not of the same type, JavaScript attempts to convert them to an appropriate type for the comparison.

Operator	**Description**	**Examples returning true**
Equal (==)	Returns true if the operands are equal.	3 == var1 "3" == var1 3 == '3'
Not equal (!=)	Returns true if the operands are not equal.	var1 != 4 var2 != "3"
Strict equal (===)	Returns true if the operands are equal and of the same type	3 === var1
Strict not equal (!==)	Returns true if the operands are of the same type but not equal, or are of different type.	var1 !== "3" 3 !== '3'
Greater than (>)	Returns true if the left operand is greater than the right operand.	var2 > var1 "12" > 2
Greater than or equal (>=)	Returns true if the left operand is greater than or equal to the right operand.	var2 >= var1 var1 >= 3
Less than (<)	Returns true if the left operand is less than the right operand.	var1 < var2 "2" < 12
Less than or equal (<=)	Returns true if the left operand is less than or equal to the right operand.	var1 <= var2 var2 <= 5

Example:

```
var a, b;
a = 10;
b = 20;
c = “20”;
console.log (a > b);    // false
console.log (a < b);    // true
console.log (a >= b);   // false
console.log (a <= b);   // true
console.log (a != b);   // true
console.log (a == b);   // false
console.log (b == c);   // true
console.log (b === c);  // false
```

1.2.3. Logical Operators

- Logical operators are typically used with Boolean (logical) values; when they are, they return a Boolean value.

- However, the && and || operators actually return the value of one of the specified operands, so if these operators are used with non-Boolean values, they may return a non-Boolean value.

Operator	**Usage**	**Description**
Logical AND (&&)	expr1 && expr2	Returns expr1 if it can be converted to false; otherwise, returns expr2. Thus, when used with Boolean values, && returns true if both operands are true; otherwise, returns false.
Logical OR (||)	expr1 || expr2	Returns expr1 if it can be converted to true; otherwise, returns expr2. Thus, when used with Boolean values, || returns true if either operand is true; if both are false, returns false.
Logical NOT (!)	!expr	Returns false if its single operand that can be converted to true; otherwise, returns true.

Example:

```
var a, b;
a = 10;
b = 20;
c = -30;
console.log(a > b && a != b);       //false
console.log(a > b > c );            //true
console.log(!(a >= b));             //true
console.log(a == b || a > b);       //false
```

1.2.4. Conditional (ternary) operator

- The conditional operator is the only JavaScript operator that takes three operands.
- The operator can have one of two values based on a condition.
- The syntax is:

```
Condition ? value1 : value2
```

- If condition is true, it returns value1. Otherwise it returns value2.
- Example:

```
var person = (age >=18) ? ‘adult’  : ‘minor’;
```

1.2.5. Increment and Decrement

- The increment and decrement operators in JavaScript will add one (+1) or subtract one (-1), respectively, to their operand, and then return a value.
- Syntax

  Consider x to be the operand:
  - Increment: x++ or ++x
  - Decrement : x-- or --x

Post Increment / Post Decrement:

- When you use the increment/decrement operator after the operand, the value will be returned before the operand is increased / decreased.

Pre Increment / Pre Decrement:

- Using ++ or -- prior to our variable, the operation executes and adds/subtracts 1 prior to returning.

1.2.6. Comma Operator

- The comma operator (,) simply evaluates both of its operands and returns the value of the last operand.
- This operator is primarily used inside a for loop, to allow multiple variables to be updated each time through the loop.

1.2.7. Typeof operator

- The typeof operator is used in either of the following ways:

  typeof operand

  typeof (operand)
- The typeof operator returns a string indicating the type of the unevaluated operand.
- Operand is the string, variable, keyword, or object for which the type is to be returned. The parentheses are optional.
- Example:

```
var st = "node";
var today = new Date();

typeof st;        // returns "string"
typeof today;     // returns "object"
```

1.3. Decision statements

- A computer can execute instructions that are clear and precise. The computer will always do exactly what you say.
- In Node.js, we have two options to implement a decision program:
    - if...then
    - switch case

1.3.1. If...then

The following is syntax for decision in Node.js:

```
if(condition){
      do_something_a;
}else {
      do_something_b;
}
```

1.3.2. Switch case

- We can use switch...case syntax to implement decision behaviour in our program.
- The following is a syntax model for switch case.

```
switch (option) {
case option1:
            // do option1 job
            break;
case option2:
            // do option2 job
            break;
}
```

1.4. Iterative statements

- One of the most powerful concepts in any programming language is that of iterations.

1.4.1. For

- The for statement provides this mechanism, letting you specify the initializer, condition, and increment/decrement in one compact line.
- The following is node.js syntax of the iteration for.

```
for (initialize; condition; increment / decrement)
{
// do something
}
```

1.4.2. While

- Node.js has the simple form of while syntax.
- while evaluates the condition and executes the statement if that condition is true. Then it repeats that operation until the condition evaluates as false.

```
while (condition)
{
// do something
}
```

1.5. Node.js collections

- Node.js provides an Array object for collection manipulation.
- In general, a Node.js Array object has the same behavior with a JavaScript Array object.

1.5.1. Create array object

- There are three ways to create an Array object. The first is simply by typing the array object.

```
var array = [ ];
```

- Option two is to create an Array object by instantiating the Array object.

```
var array = new Array( );
```

- The last option is to create an Array object by inserting collection data.

```
var arr_name = [elemenet1, element2, element3,..., elementN];

Example:
      var odd = [1,3,5,7,9];
```

1.5.2. Insert data

- After creating an Array object, we can insert data. Use [] with index if you want to assign the value.

```
arr_name[0] = value;
arr_name[1] = value;
arr_name[2] = value;
arr_name[3] = value;
      ....
arr_name[size-1] = value;
```

- You can also use the push() function to insert data.

```
array.push(new_element);
arr_name.push(new_element);
```

1.5.3. Access data

- To access array data, you can use [] with data index parameter.

```
// show data
console.log(arr_name);

// show data using a loop
for(var i=0;i<arr_name.length;i++){
      console.log(arr_name[i]);
}
```

1.5.4. Update data

- To update an item of array data, you can use [] with data index and thus assign a new value.

```
// edit
arr_name[index] = updated_value;

console.log(arr_name);
```

1.5.5. Remove data

- You can use the pop() function to remove data from the Array.
- If you want to remove data by specific index then you can use the splice() function.

```
// remove data
arr_name.pop ();

// remove data by index
var index = 1;
arr_name.splice (index, 1);

//display array after removal
console.log (arr_name);
```

1.6. JSON

What is JSON?

- JSON (JavaScript Object Notation) is a lightweight, human readable, data-interchange format.
- It is easy for humans to read and write. It is easy for machines to parse and generate.
- JSON is a text format that is completely language independent.
- JSON uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.
- JSON is useful for serializing objects, and arrays for transmitting over the network.
- JSON is very easy to parse and generate and doesn’t use a full mark-up structure like a XML.
- JSON became a popular alternative of XML format for its fast asynchronous client–server communication.
- All JSON files have the extension .json

Programming Languages Support:

- Originally, JSON was intended to be a subset of the JavaScript languages.
- Now almost all major programming languages support JSON due to its language-independent data format.

1.6.1. Create JSON object

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- A JSON object in Node.js uses { } syntax to declare the JSON data type.

- JSON syntax rules:
    - A JSON object is surrounded by curly braces {}
    - The name-value pairs are grouped by a colon (:) and separated by a comma (,).
    - An array begins with a left bracket and ends with a right bracket [ ].
    - The trailing commas and leading zeros in a number are prohibited.
    - The octal and hexadecimal formats are not permitted.
    - Each key within the JSON should be unique and should be enclosed within the double quotes.
    - The Boolean type matches only two special values : true and false and NULL values are represented by the null literal (without quotes).

```
var  json_object_name = {
      attribute1 : value,
      attribute2 : value,
      attribute3 : {
            sub_attribute1 : value,
            sub_attribute2 : value,
      },
      attribute4 : value,
            .
            .
      attributeN : value
}
```

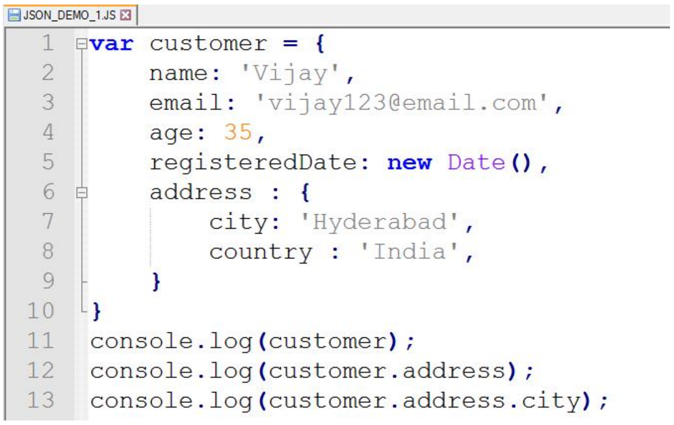
1.6.2. Display

- After you have created the JSON object, you use console.log ( ) to see your object.
- console.log() can display all JSON attributes.

```
console.log (json_object_name);
```

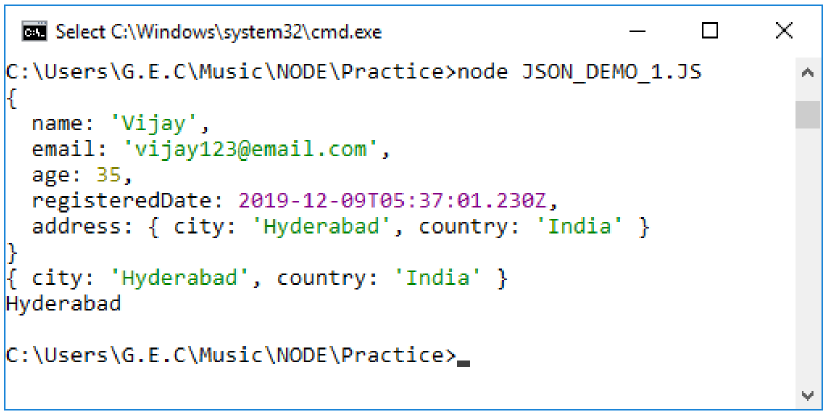
1.6.3. Access and edit data

- If you want to get a specific attribute of a JSON object, then you can call the attribute name directly.



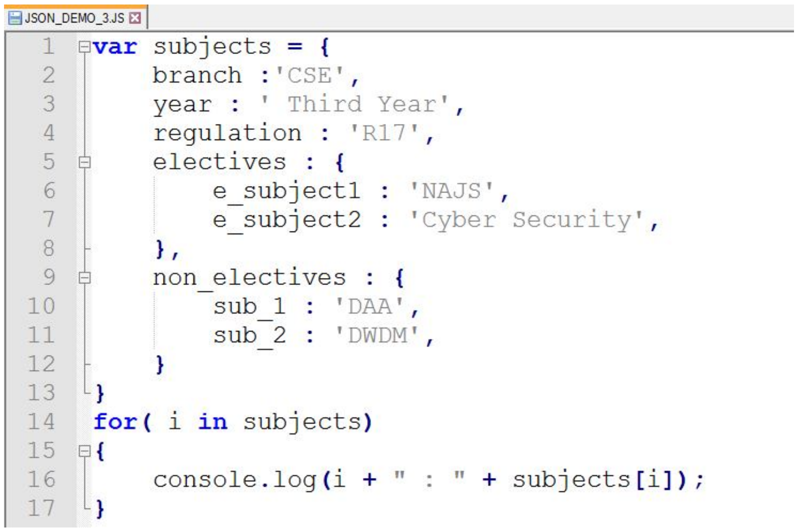
```
var customer = {
    name: 'Vijay',
    email: 'vijay123@email.com',
    age: 35,
    registeredDate: new Date(),
    address : {
        city: 'Hyderabad',
        country : 'India',
    }
}
console.log(customer);
console.log(customer.address);
console.log(customer.address.city);
```

Now, run the above file “JSON_DEMO_1.JS” on command prompt as below to see the results.



```
C:\Users\G.E.C\Music\NODE\Practice>node JSON_DEMO_1.JS
{
  name: 'Vijay',
  email: 'vijay123@email.com',
  age: 35,
  registeredDate: 2019-12-09T05:37:01.230Z,
  address: { city: 'Hyderabad', country: 'India' }
}
{ city: 'Hyderabad', country: 'India' }
Hyderabad

C:\Users\G.E.C\Music\NODE\Practice>
```

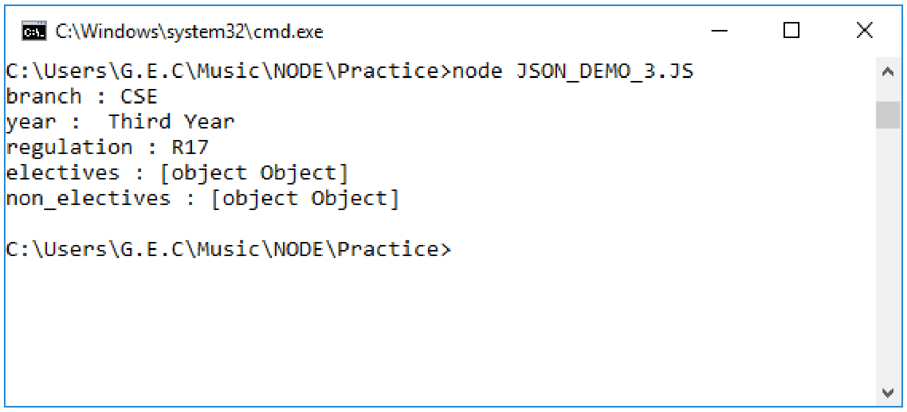
- You can use an iteration operation to get JSON object attributes.



```
var subjects = {
    branch :'CSE',
    year : ' Third Year',
    regulation : 'R17',
    electives : {
        e_subject1 : 'NAJS',
        e_subject2 : 'Cyber Security',
    },
    non_electives : {
        sub_1 : 'DAA',
        sub_2 : 'DWDM',
    }
}
for( i in subjects)
{
    console.log(i + " : " + subjects[i]);
}
```

Execute the above file using the command,

```
node  filename.js
```



```
C:\Users\G.E.C\Music\NODE\Practice>node JSON_DEMO_3.JS
branch : CSE
year :  Third Year
regulation : R17
electives : [object Object]
non_electives : [object Object]

C:\Users\G.E.C\Music\NODE\Practice>
```

1.7. JSON Array

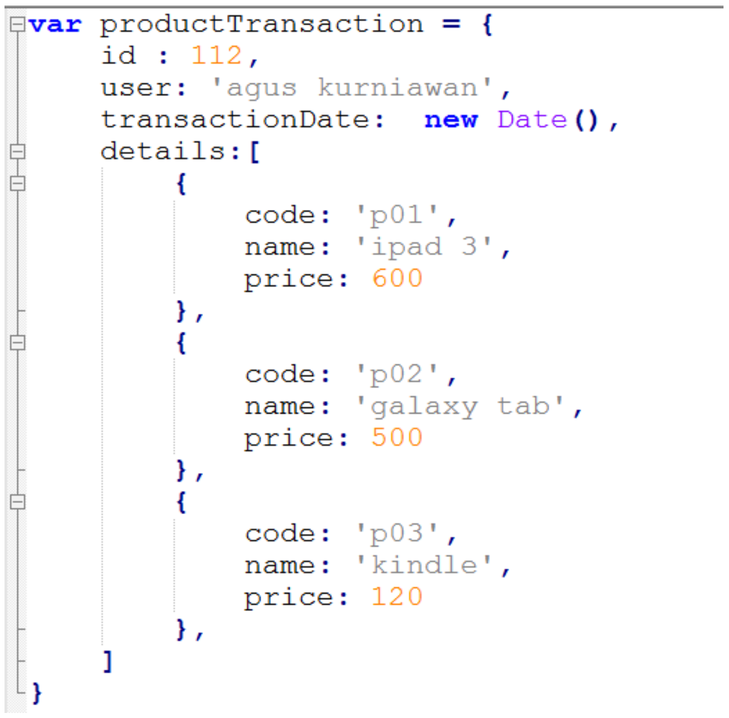
We want to combine JSON and collection. This means we create a collection of JSON objects.

1.7.1. Creation

Syntax to Create :

```
var  json_object_name = {
            name1 : value1,
            name2 : value2,
            arr_name : [
                 {
                      name_1 : value,
                      name_2 : value,
                           ....
                      name_N : value,
                 },
                 {
                      name_1 : value,
                      name_2 : value,
                           ....
                      name_N : value,
                 }
            ]
}
```

Example:



```
var productTransaction = {
    id : 112,
    user: 'agus kurniawan',
    transactionDate:  new Date(),
    details:[
        {
            code: 'p01',
            name: 'ipad 3',
            price: 600
        },
        {
            code: 'p02',
            name: 'galaxy tab',
            price: 500
        },
        {
            code: 'p03',
            name: 'kindle',
            price: 120
        },
    ]
}
```

1.7.2. Display

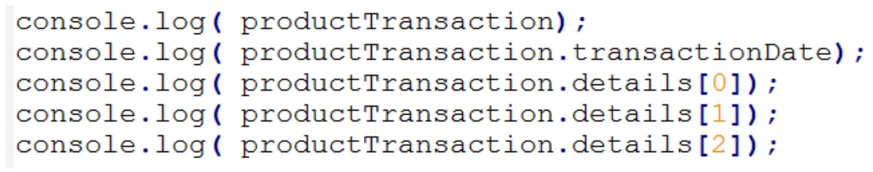
- We can display entire json object or a particular member of json object or the array member of json object using array index.

```
console.log( json_object_name);
console.log(json_object_name.attribute_name);
console.log(json_object_name.array_name[index]);
```

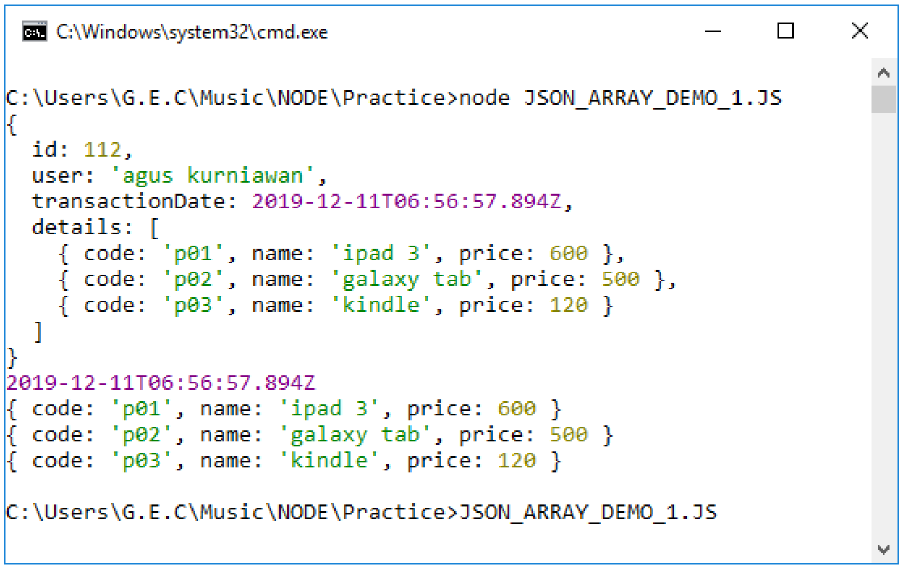
1.7.3. Access

- If you want to get a specific element of an array of a JSON object, then access it using

```
json_object_name.array_name[index].member
```



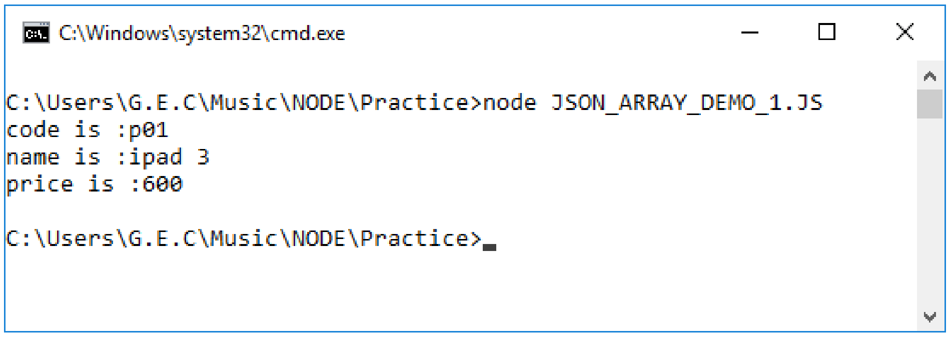
```
console.log( productTransaction);
console.log( productTransaction.transactionDate);
console.log( productTransaction.details[0]);
console.log( productTransaction.details[1]);
console.log( productTransaction.details[2]);
```



```
C:\Windows\system32\cmd.exe

C:\Users\G.E.C\Music\NODE\Practice>node JSON_ARRAY_DEMO_1.JS
{
  id: 112,
  user: 'agus kurniawan',
  transactionDate: 2019-12-11T06:56:57.894Z,
  details: [
    { code: 'p01', name: 'ipad 3', price: 600 },
    { code: 'p02', name: 'galaxy tab', price: 500 },
    { code: 'p03', name: 'kindle', price: 120 }
  ]
}
2019-12-11T06:56:57.894Z
{ code: 'p01', name: 'ipad 3', price: 600 }
{ code: 'p02', name: 'galaxy tab', price: 500 }
{ code: 'p03', name: 'kindle', price: 120 }

C:\Users\G.E.C\Music\NODE\Practice>JSON_ARRAY_DEMO_1.JS
```

Access the elements of array of JSON as follows:

```
console.log( "code is :"+productTransaction.details[0].code);
console.log( "name is :"+productTransaction.details[0].name);
console.log( "price is :"+productTransaction.details[0].price);
```

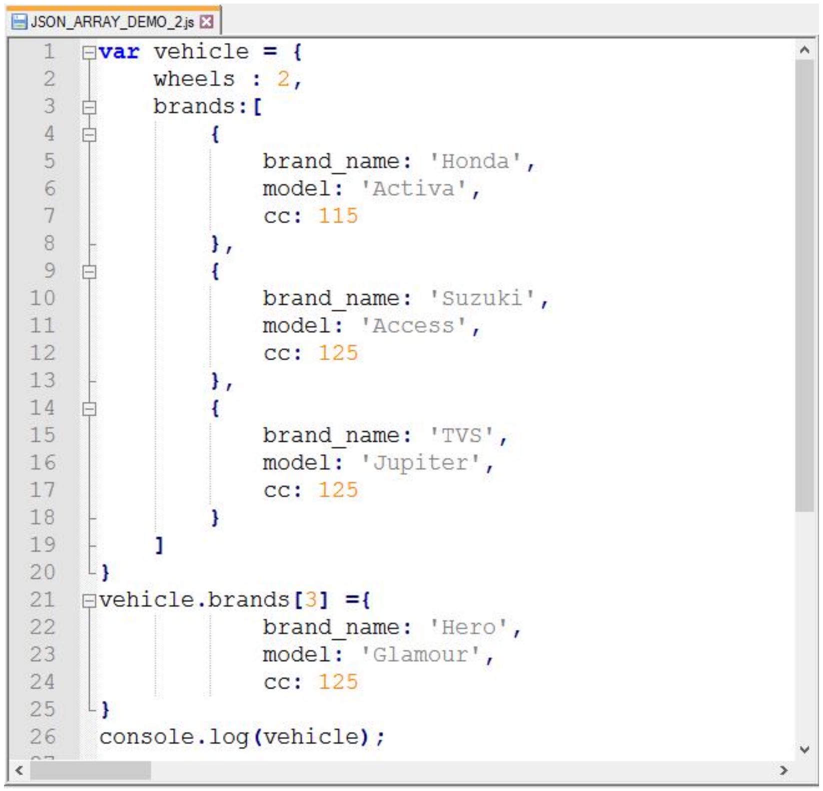


```
C:\Windows\system32\cmd.exe

C:\Users\G.E.C\Music\NODE\Practice>node JSON_ARRAY_DEMO_1.JS
code is :p01
name is :ipad 3
price is :600

C:\Users\G.E.C\Music\NODE\Practice>_
```

1.7.4. Edit data.

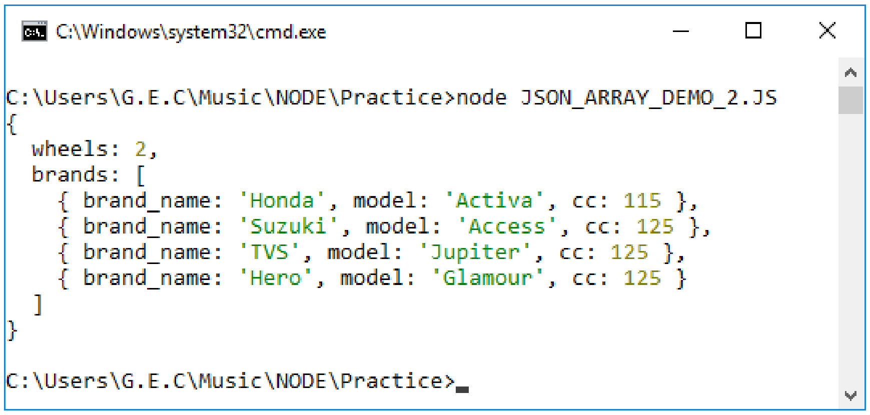
- Data of JSON array can be edited.
- A new array element can be inserted.
- A value of any existing attribute can be updated.

Example:



```
var vehicle = {
        wheels : 2,
        brands:[
            {
                brand_name: 'Honda',
                model: 'Activa',
                cc: 115
            },
            {
                brand_name: 'Suzuki',
                model: 'Access',
                cc: 125
            },
            {
                brand_name: 'TVS',
                model: 'Jupiter',
                cc: 125
            }
        ]
}
vehicle.brands[3] ={
                brand_name: 'Hero',
                model: 'Glamour',
                cc: 125
}
console.log(vehicle);
```

The result of the above code when we run it on command prompt using the command *node Filename.js* is below.



```
C:\Users\G.E.C\Music\NODE\Practice>node JSON_ARRAY_DEMO_2.JS
{
  wheels: 2,
  brands: [
    { brand_name: 'Honda', model: 'Activa', cc: 115 },
    { brand_name: 'Suzuki', model: 'Access', cc: 125 },
    { brand_name: 'TVS', model: 'Jupiter', cc: 125 },
    { brand_name: 'Hero', model: 'Glamour', cc: 125 }
  ]
}

C:\Users\G.E.C\Music\NODE\Practice>_
```

1.8. Check JSON attribute

- We can check if the attribute name exists in our JSON object. To do this, you can use the hasOwnProperty() function.
- The method hasOwnPropert() returns true if the attribute name exists, otherwise it returns false.

Syntax is:

```
            json_object.hasOwnProperty()
```

Example:
Let us consider the vehicle example and use hasOwnProperty() method to check if the vehicle.brand[] array has own properties called brand_name, model, cc.

```
console.log(vehicle.brands[0].hasOwnProperty('model'));     // true
console.log(vehicle.brands[0].hasOwnProperty('cc'));        // true
```

If we check the price value which is not available in vehicle.brands[], then it returns false.

```
console.log(vehicle.brands[0].hasOwnProperty('price'));     // false
```

UNIT-I

Assignment-Cum-Tutorial Questions

A. Objective Questions:

1. To make sure Node.js was installed, type ______ in the command window. [ ]
   a) http://localhost:8080
   b) http://127.0.0.1:8080/
   c) eval
   d) node –version
2. The way you run Node.js is the shell is called *REPL.* REPL stands for [ ]
   a) Read-Eval-Print-Loop
   b) Respond- Encode -Pool-Layout
   c) Request- encode **-** Port- loop
   d) Read- Edge -Print-Layout
3. How Node.js based web servers do differ from traditional web servers? [ ]
   a) Node based server uses a single threaded model and can serve much larger number of requests compared to any traditional server like Apache HTTP Server.
   b) Node based server process request much faster than traditional server.
   c) There is no much difference between the two.
   d) All of the above
4. Which of the following are True? [ ]
   i. If you ever see three dots (...) in the Node REPL, that means it is expecting more input from you to complete the current expression, statement, or function.
   ii. If you do not understand why REPL is giving you the ellipsis i.e (...), you can just type .break to get out of it.
   iii. One or more servers on your machine listens on a port 80 for HTTP.
   iv. When a request is received, web server forks a new process or a thread to begin processing and responding to the query.

   a) i,ii,iii b) i,ii ,iv c) ii,iii,iv d) i, ii,iii,iv,

5. Which of the following are True? [ ]
   i. Web server work involves communicating with external services, such as a database, memory cache, external computing server, or even just the file system.
   ii. When all Web server work is finally finished, the thread or process is returned to the pool of "available" servers, and more requests can be handled.
   iii. To help with debug issue, Node.js you just add the debug flag before the name of your program: node debug debugging.js
   iv. The types null and undefined are special kinds of objects and are treated specially in JavaScript.

   a) i,ii,iii b) i,iii,iv c) ii,iii,iv d) i, ii,iii, iv
6. The push and pop functions help us add and remove items to the end of an array, respectively in Node.JS. [True/False]
7. All numbers in JavaScript are 64-bit IEEE 754 double-precision floating-point numbers. [True/False]
8. A JavaScript Engine is a program that converts code written in JavaScript to something that computer processor understands. [True/False]

9. What operation is used to insert data into array in Node.js. [ ]

   a) insert() b) push() c)place() d) put()
10. Which of the following is the valid command to execute nodejs file using command prompt? [ ]

    a) node filename c) node filename.js
    b) nodejs filename d) nodejs filename.js
11. Match the following.

I.	JSON	P) collection manipulation
II.	NodeJS Array	Q) to combine JSON and collection
III.	JSON Array	R) an open-source, cross-platform, JavaScript runtime environment
IV.	NodeJS	S) is an open-standard file format

12. Which of the following function is used to check for the existence of attribute name a JSON object? [ ]
    a) isOwnProperty( )
    b) findOwnProperty( )

c) hasOwnProperty( )

d) ownProperty( )

13. Identify the features of nodejs from the following options. [ ]

I. Everything is asynchronous

II. It yields great concurrency

III. single-threaded

IV. multi-threaded

a) I, II only b) I, II, IV only c) I, II, IV only c) I, II, III only

14. What is the expected output of the following code when executed on nodejs environment? [ ]

```
var a= '2019';
var b=2019;
console.log(a+b);
console.log(a==b);
console.log(a===b);
console.log(a-b);
console.log(a/b);
```

a)	4038	true	false	0	undefined
b)	4038	true	false	0	1
c)	'20192019'	false	false	0	1
d)	'20192019'	true	false	0	undefined
e)	'20192019'	true	false	0	1

15. What is the output of the following code in nodejs? [ ]

```
var  year = "2019";
console.log((year++)== 2020);
console.log((year++)== 2020);
```

a)	true	true
b)	true	false
c)	false	true
d)	false	false

16. Node uses ________ engine in core. [ ]

a) Microsoft Spartan

b) SpiderMonkey

c) Chrome V8

d) Node En 12

17. Which of the following line numbers display true? [ ]

```
Line 1:     console.log("222" === 222+"");
Line 2:     console.log("222" === 221+"1");
Line 3:     console.log("222" !== 220+2);
Line 4:     console.log("222" === 222+null);
```

a) Line 1, Line 2

b) Line 2, Line 3

c) Line 1, Line 3

d) Line 3, Line 4

18. What is the output of the following nodejs code?

```
var cars = ["Swift", "Grand i10", "Brezza", "Honda Jazz","Honda
City", "Kia Seltos"];
console.log(cars.splice(1,4).sort());
console.log(cars);
console.log(cars.splice(1,4).sort());
console.log(cars);
```

a) ['Brezza', 'Grand i10', 'Honda City', 'Honda Jazz']
['Swift', 'Kia Seltos']
['Kia Seltos']
['Swift']

b) ['Brezza', 'Grand i10', 'Honda City', 'Honda Jazz']
["Swift", "Grand i10", "Brezza", "Honda Jazz","Honda City", "Kia Seltos"]
['Brezza', 'Grand i10', 'Honda City', 'Honda Jazz']
["Swift", "Grand i10", "Brezza", "Honda Jazz","Honda City", "Kia Seltos"]

c) ["Grand i10","Honda City", "Honda Jazz", "Kia Seltos"]
['Brezza', 'Swift']
["Grand i10","Honda City", "Honda Jazz", "Kia Seltos"]
['Brezza', 'Swift']

d) ["Grand i10","Honda City", "Honda Jazz", "Kia Seltos"]
['Brezza', 'Swift']
["Grand i10","Honda City", "Honda Jazz", "Kia Seltos"]
['Brezza']

19. Consider the following JSON and choose the correct option to display NAJS and DAA from the given options.

```
var subjects = {
      branch :'CSE',
      year : ' Third Year',
      regulation : 'R17',
      electives : {
            e_subject1 : 'NAJS',
            e_subject2 : 'Cyber Security',
      },
      non_electives : {
            sub_1 : 'DAA',
            sub_2 : 'DWDM',
      }
}
```

a) console.log(subjects.e_subject1);
   console.log(subjects.e_subject1);

b) console.log(subjects.electives.e_subject1);
   console.log(subjects. non_electives.e_subject1);

c) console.log(subjects.electives[0].e_subject1);
   console.log(subjects. non_electives[0].e_subject1);

d) console.log(subjects.electives[0]);
   console.log(subjects. non_electives[0]);

B. Descriptive Questions:

1. Discuss main features of Nodejs?
2. Discuss different operators, decision and iterative statements in Nodejs.
3. Identify the differences between JavaScript and Node JS.
4. Explain how Node.Js works.
5. Develop a program to create an array object, insert, access and update and remove data from it in Nodejs.
6. Develop a program to create JSON object, display, access and edit data from JSON object.
7. Develop a program for Creation, display, access and edit data from JSON Array.
8. Develop a program to differentiate slice and splice functions of array object in Nodejs.
9. Explain how do we decide, when to use Node.Js and when not to use it?

Unit – II
Node.js Files, Functions and Strings

Syllabus:

File modules, reading text, creating file. Functions: creating function, types of functions, callback function.
Strings: operations, string to numeric and vice-versa, string parser.

2.1. File modules

- Node FS Module provides an API to interact with File System and to perform some IO Operations like create file, read File, delete file, update file etc.
- Like some Node modules for example "npm", "http" etc, Node JS "fs" also comes with basic Node JS Platform. We don't need to do anything to setup Node JS FS module.

Node FS Module import

- We just need to import node fs module into our code and start writing IO operations code.
- To import a node fs module;

  ```
  var fs = require("fs");
  ```
- require() call loads specified Node FS module into cache and assign that to an object named as fs.
- This `require()` call imports Node JS "fs" module into cache and creates an object of type Node FS module. Once it's done, we can perform any IO Operation using node fs object.
- IO or Streams are two types:

  Write Stream – To write data to a Stream.

  Read Stream – To read data from a Stream.
- Every method in the fs module has synchronous as well as asynchronous forms.
- Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error.
- It is better to use an asynchronous method instead of a synchronous method, as the asynchronous method never blocks a program during its execution, whereas synchronous method does.

2.2. Reading text

- One of the most common things you'll want to do with just about any programming language is open and read a file.
- Node.js, as you probably know, is much different than your typical JavaScript in the browser. It has its own set of libraries meant for handling OS and filesystem tasks, like opening and reading files.
- We will use Node FS API to open and read an existing file content and write that content to the console.

Buffering Contents with fs.readFile:

- This is the most common way to read a file with Node.js
  Example:

```
var fs = require('fs');
fs.readFile('my-file.txt', 'utf8', function(err, data)
{
      if (err) throw err;
      console.log(data);
});
```

- The data argument to the callback contains the full contents of the file represented as a string in utf8 format.
- If you omit the utf8 argument completely, then the method will just return the raw contents in a Buffer object.

2.3. Creating file

- Create file using one the following methods.

Using `writeFile()` function

```
fs.writeFile('<fileName>',<contenet>,callbackFunction)
```

- A new file is created with the specified name.
- After writing to the file is completed (could be with or without error), callback function is called with error if there is an error reading file.
- If a file already exists with the name, the file gets overwritten with a new file.
- Care has to be taken while using this function, as it overwrites existing file if any.
- Example:

```
// include node fs module
var fs = require('fs');

// writeFile function with filename, content and
callback function
fs.writeFile('newfile.txt', 'Learn Node FS
module', function (err) {
  if (err) throw err;
  console.log('File is created successfully.');
});
```

Using appendFile() function

```
fs.appendFile(‘<fileName>’,<contenet>,callbackFunction)
```

- If the file specified in the appendFile() function does not exist, a new file is created with the content passed to the function.
- Example:

```
// include node fs module
var fs = require('fs');

// appendFile function with filename, content and
callback function
fs.appendFile('newfile_2.txt', 'Learn Node FS
module', function (err) {
  if (err) throw err;
  console.log('File is created successfully.');
});
```

Using open() function

```
fs.open(‘<fileName>’,<file_open_mode>,callbackFunction)
```

- If the specified file is not found, a new file is created with the specified name and mode and sent to the callback function.
- Example:

```
// include node fs module
var fs = require('fs');

// open function with filename, file opening mode and
callback function
fs.open('newfile_3.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('File is opened in write mode.');
});
```

2.4. Functions

2.4.1. Creating function

Syntax:

```
function function_name(parameters)
{
   // code
}
```

- The keyword “function” is used to create a function in nodejs.
- function_name represents the name of the function.
- Parameters are the input values supplies to the function.

```
function display(name, roll)  // function definition
{
   console.log('Student name is '+ name);
   console.log('Student roll number is '+ roll);
}
display('Raj', '101');        // function call
```

Immediately Executing Function

- We can execute a function immediately after you define it. Simply wrap the function in parentheses () and invoke it.

```
(function myData() {
   console.log('myData was executed!');
})();
```

Anonymous Function:

- A function without a name is called an anonymous function. In JavaScript, you can assign a function to a variable. If you are going to use a function as a variable, you don't need to name the function.
- Example 1:

```
var myData = 123;
if (true) {
    (function () { // create a new scope
        var myData = 456;
console.log(myData); // displays 456;
     })();
}
console.log(myData); // displays 123;
```

- Example 2:

```
var myData = 123;
if (true) {
    (function () {
        myData = 456;
console.log(myData); // displays 456;
     })();
}
console.log(myData); // displays 456;
```

2.4.2. Types of functions

Functions can be of four types.

i. Function with parameters and with return type
ii. Function with parameters and without return type
iii. Function without parameters and with return type
iv. Function without parameters and without return type.

Example:

```
function add(a,b){
    return a+b;
}
var result = add(10,15); console.log(result);
```

2.4.3. Callback function

- A callback is a function that is to be executed after another function has finished executing — hence the name 'call back'.
- *Functions that do this are called higher-order functions. Any function that is passed as an argument is called a callback function.*
- A callback function is a function that is called through a function pointer.
- If you pass the pointer (address) of a function as an argument to another and that pointer is used to call the function it points to, it is said that a callback is made.
- A Callback is simply a function passed as an argument to another function which will then use it (call it back).

Why do we need Callbacks?

- For one very important reason — JavaScript is an event driven language.
- This means that instead of waiting for a response before moving on, JavaScript will keep executing while listening for other events.

Example 1:

```
function doWork(subject,callback)
{
    console.log('Starting my '+subject + ' work.');
      callback();
}

doWork('nodejs', fun2);

function fun2(){
```

```
            console.log("Finished my work");
      }
```

Execute the above code on nodejs command prompt.
It results,

```
Starting my nodejs subject work.
Finished my work.
```

Example 2:

```
function perform(a,b,callback){
     var c = a*b + a;
     callback(c);
}
perform(10,5,function(result){
    console.log(result);
 })
```

- Values 10 and 5 are function parameters. We also pass a function with a parameter result. This parameter is used to get a return value from the callback function.
- The above code prints 60 as output.
- You can define a callback function with many parameters, for example, check this code:

```
function perform(a,b,callback){     // do processing
     var errorCode = 102;
     callback(errorCode,'Internal server error');
}
perform(10,5,function(errCode,msg){
if(errCode){
 console.log(msg);
}
})
```

- First check the errCode value. If it has a value, then the code will write the message msg in the console.
- This function is very useful when you want to implement a long process. The caller will be notified if the process was done.

2.5. Strings

The string type represents a sequence of zero or more Unicode characters. In Node.js, string type is defined in String.

Declaring String Type:

var obj1 = new String("hello world");

var obj2 = "hello world";

2.5.1. Operations

2.5.1.1. Concatenating String

- If you have a list of string data, you can concatenate it into one string.
- For instance, you have a list of string data as follows:

```
var str1 = 'hello ';
var str2 = 'world ';
var str3 = 'nodejs';
```

- Now you can concatenate all of the data into one variable with a string type. You can use the + operator.
- Here is a sample code:

```
console.log(str1 + str2 + str3);
```

It prints hello world nodejs

2.5.2. String to numeric and vice-versa

2.5.2.1. String to Numeric

- Sometimes you may want to do math operations, but the input data is String type.
- To convert String type data into numeric, you can use parseInt() for String to Int and parseFloat() for String to Float.

```
console.log(parseInt('123.45'));         // prints 123
console.log(parseInt('-123'));           // prints -123
console.log(parseInt('0.34'));           // prints 0
console.log(parseInt('123abc'));         // prints 123
console.log(parseInt('abc123'));         // prints NaN
console.log(parseInt('123 456'));        // prints 123
console.log(parseInt('1'+23));           // prints 123

console.log('-----parseFloat-----');
console.log(parseFloat('123'));     // prints 123
console.log(parseFloat('123.45'));  // prints 123.45
console.log(parseFloat('-123'));    // prints -123
console.log(parseFloat('0.34'));    // prints 0.34
console.log(parseFloat('12abc'));   // prints 12
```

2.5.2.2. Numeric to String

- It is easy to convert numeric data into string type data. You just add '' and get string type data automatically.

```
var a = 123;
var b = a + '';
```

- Another solution is the toString() method.

```
var num = 405;
var str = num.toString();
```

2.5.3. String parser

- If you have data, for instance, 'Berlin;Amsterdam;London;Jakarta', then you want to parse by using a character as a separator.
- The simple solution to parsing String uses split() with a parameter delimiter, for instance, ';'.

```
var data = 'Berlin;Amsterdam;London;Jakarta';
var strs = data.split(';');
for(var index in strs){
      console.log(strs[index]);
}
```

- split() method splits the given string into array of words.
- Each word can be accessed by index number of the array.

  The above code prints:

```
Berlin
Amsterdam
London
Jakarta
```

- Now, see another example.

```
var msg = "NodeJs-works-for:server-side--also";
var token = msg.split('-');
for(var i in token)
{
      if(token[i].length)
      console.log(token[i]);
}
```

The above code prints:

```
NodeJs
Works
for:server
side
also
```

UNIT-II
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions:

1. All file system operations have synchronous and asynchronous forms. The asynchronous form takes a ______ as its last argument. [ ]
   a) callback b) node c)process d) thread

2. The arguments passed to the completion callback depend on the method, but the first argument is always reserved for ___________. [ ]
   a) error b) node c)process d) node –-version

3. If file system operation was completed successfully, then the first argument will be null or undefined. [True/ False]

4. In the Node.js module system, each file is treated as a separate module. [True/ False]

5. Which is needed to import node fs module into our code and start writing IO operations code? [ ]
   a) request('fs') b) require('fs') c) require('files') d) request('files')

6. Which method takes the last parameter as the completion function callback and the first parameter of the callback function as error.
   a) callback b) asynchronous c) synchronous d) module

7. Asynchronous method blocks a program during its execution, whereas synchronous method does not. [True/False]

8. How does the following nodejs code work? [ ]

```
var fs = require("fs");
function readData(err, data) {
        console.log(data);
}

fs.readFile('ExistingFile.txt', 'utf8', readData);
```

   a) Reads the data from ExistingFile.txt and ready to print on the console.
   b) Reads the data from ExistingFile.txt and prints on the console.
   c) Reads the data from ExistingFile.txt and prints on the browser.
   d) Generates compile time error.

9. Which of the following options shows correct display output to the below NodeJs code? [ ]

```
var myData = 123.45;
if (true) {
    (function () {
        var myData = 'strange';
        console.log(myData);
    })();
}
console.log(myData);
```

a) strange b) 123.45 c) strange 123.45 d) compile time error

10. Which of the following options shows correct display output to the below NodeJs code? [ ]

```
var status = 'normal';
if (true) {
    (function () {
        status = 'strange';
console.log(status);
    })();
}
console.log(status);
```

a) normal strange
b) strange strange
c) normal normal
d) strange normal

11. Which of the following is FALSE about callback function? [ ]

a) a function that is to be executed after another function has finished executing
b) The function which stores its previous state and called by itself.
c) Any function that is passed as an argument is called a callback function.
d) A callback function is a function that is called through a function pointer.
e) Callbacks are a way to make sure certain code doesn't execute until other code has already finished execution.

12. What is the output of the following code?

```
console.log("first");
setTimeout(function() {
    console.log("second");
}, 0);
console.log("third");
```

a) first second
b) first second third
c) first third second
d) first third

13. Which of the following is/are optional in writing functions in nodejs? [ ]
    a) Function name, parameters, function keyword
    b) Function name, return statement, function keyword
    c) Return statement, parameters, function keyword
    d) Function name, parameters, return statement
14. What is the output of the following nodejs code? [ ]

```
        function calculate(a,b,callback){
                var c = a/b + a;
                callback(c); }
        calculate(111,3,function(result){
            console.log(result);
                 })
```

    a) Compile time error
    b) 138
    c) 148
    d) 111
    e) What is the output of the following nodejs code? [ ]
15. Which of the following options is correct output for the following nodejs code? [ ]

```
        function perform(a,b,callback)
        {
            var errorCode = 102;

            callback(errorCode,'Internal error');
        }
        perform(33,11,function(errCode,msg)
        {
                if(errCode){
                console.log(msg);
                }
        })
```

    a) errorCode   b) Internal error   c) Compile time Error   d) 102
16. Which of the following lines print 125 as output? [ ]

```
        Line1:      console.log(parseInt('125'));
        Line2:      console.log(parseInt('125.34'));
        Line3:      console.log(parseInt('-125'));
        Line4:      console.log(parseInt('0.125'));
        Line5:      console.log(parseInt('125abc'));
```

```
Line6:      console.log(parseInt('1'+25));
Line7:      console.log(parseInt('abc125'));
```

a) Line 1, Line 2, Line 5, Line 6 Only
b) Line 1, Line 2 Only
c) Line 1, Line 5, Line 6 Only
d) Line 1, Line 2, Line 5, Line 7 Only

17. What is the output of the following nodejs code?

```
var msg = "Today is a nice day";
var token = msg.split(' ');
for(var i in token)
        console.log(i);
```

Space for output:

18. Consider the following code and choose the correct option as output of the code. [ ]

```
var msg = "Today is a nice day";
var token = msg.split(' ');
for(var i in token)
{
        if(token[i++].length < token[1].length)
        console.log(token[i]);
}
```

a) is   b) a   c) nice   d) day   e) nothing is displayed

19. What is the output of the following nodejs code? [ ]

```
var msg = "Today is a nice day";
var token = msg.split(' ');
for(var i in token)
{
        if(token[i].length > token[i].length)
        console.log(token[i]);
}
```

a) Compile time error   b) nothing will be displayed
c) Today is a nice day   d) Today nice

20. Write the correct output for the given nodejs code when split() method is used to parse a string.

```
var msg = "NodeJs-works-for:server-side--also";
var token = msg.split('-');
for(var i in token)
{
        if(token[i].length)
        console.log(token[i]);
}
```

Space for output:

SECTION-B

Descriptive Questions:

1. Discuss file modules required for reading text in NodeJS.
2. Develop a program for reading text in NodeJS.
3. Develop a program for create fie with the message “Today is a nice day” in Node.JS.
4. Discuss different arguments in the following function:

   fs.read(fd, buffer, offset, length, position, callback)
5. Discuss different types of functions in NodeJS.
6. Illustrate the different string operations, string to numeric and vice-versa in Node.JS.
7. Create a function with parameters and a returning value in NodeJS.
8. Design a callback function in NodeJS.
9. Discuss string parser in NodeJS.

UNIT – 3

Learning Material

UNIT - III: Node.js Modules, Error Handling & Logging and Events:

3.1 Create simple module, module class.

3.2 Error handling

3.3 logging.

3.4 Events: Events module, once event listener, remove events.

https://www.w3schools.com/angular/default.asp

https://www.json.org/json-en.html

http://www.java2s.com/Tutorials/Javascript/Node.js_Tutorial/1430__Node.js_Stream.htm

3.1 Modules

We can create a simple function and then it will be exported as a module in Node.js. Use the **exports** keyword to make properties and methods available outside the module file.

The following example creates a module that returns a date and time object: Save the code in a file called "myfirstmodule.js"

```
exports.myDateTime = function () {
  return Date();
};
```

Now you can include and use the module in any of your Node.js files. Use the module "myfirstmodule" in a Node.js file:”**web2.js**”

```
var http = require('http');
var dt = require('./myfirstmodule');
http.createServer(function (req, res) {
 res.writeHead(200, {'Content-Type': 'text/html'});
 res.write("The date and time are currently: " + dt.myDateTime());
 res.end();
}).listen(8080);
```

Now, your computer works as a server! If anyone else tries to access your computer on port 8080, they will get a message "Hello World!" in return! Start your internet browser from your computer, and type in the address: http://localhost:8080

You can see that we can use exports to expose our functions. Save this code into a file, called MyModule.js.

```
1. var calculate = function(numA,numB){
2. return numA*numB + 10*numB;
3. }
4. exports.calculate = calculate;
```

To call this module from our code, we can use require.

```
var myModule = require('./MyModule.js');
var result = myModule.calculate(20,10);
console.log(result);
```

require function needs the full path of the module. './' means the module has the same location with the caller. Save this code into a file called **testModule.js**. Run it:

3.1.2. Module Class

As object-oriented programming, you may implement a class in Node.js. You can create a **class as a module** in Node.js. Here, we create a class, called Account. First, create a file called "**Account.js"**" and write this code:

```
// constructor
var Account = module.exports = function(){
console.log('constructor');
}
// method
Account.prototype.perform = function(){
console.log('perform');
}
// method
Account.prototype.foo = function(a,b){
console.log('foo - ' + a + '-' + b);
}
```

We expose our class using module.exports. Then we implement class methods using **prototype**.

```
 var Account = require('./Account.js');
var account = new Account();
account.perform();
```

You can see that we need to instantiate our object by calling new Account().Save this code into a file called **testAccount.js**. Run it:

node testAccount.js

3.2 Error handling:

In JavaScript (and Node.js especially), there's a difference between an error and an exception. An error is any instance of the Error class. Errors may be constructed and then passed directly to another function or thrown. When you throw an error, it becomes an exception.[2] Here's an example of using an error as an exception:

```
throw new Error('something bad happened');
```

but you can just as well create an Error without throwing it:

```
callback(new Error('something bad happened'));
```

Operational errors vs. programmer errors

It’s helpful to divide all errors into two broad categories:[3]

- **Operational errors** represent run-time problems experienced by correctly-written programs. These are not bugs in the program. In fact, these are usually problems with something else: the system itself (e.g., out of memory or too many open files), the system’s configuration (e.g., no route to a remote host), the network (e.g., socket hang-up), or a remote service (e.g., a 500 error, failure to connect, or the like). Examples include:
  - failed to connect to server
  - failed to resolve hostname
  - invalid user input
  - request timeout
  - server returned a 500 response
  - socket hang-up
  - system is out of memory
- **Programmer errors** are bugs in the program. These are things that can always be avoided by changing the code. They can never be handled properly (since by definition the code in question is broken).
  - tried to read property of “undefined”
  - called an asynchronous function without a callback
  - passed a “string” where an object was expected
  - passed an object where an IP address string was expected

People use the term “errors” to talk about both operational and programmer errors, but they’re really quite different. Operational errors are error conditions that all correct programs must deal with, and as long as they’re dealt with, they don’t necessarily indicate a bug or even a serious problem. “File not found” is an operational error, but it doesn't necessarily mean anything's wrong. It might just mean the program has to create the file it's looking for first.

By contrast, programmer errors are bugs. They're cases where you made a mistake, maybe by forgetting to validate user input, mistyping a variable name, or something like that. By definition there's no way to handle those. If there were, you would have just used the error handling code in place of the code that caused the error!

This distinction is very important: operational errors are part of the normal operation of a program. Programmer errors are bugs.

The error object is an implementation of a constructor function that uses a set of instructions (the arguments and constructor body itself) to create an object. That's it. The built-in error constructor is simply a unified way of creating an error object.

In order to handle our errors properly, we need to understand:

- Error object
- Try...catch
- Throw
- Call stack
- Effective function naming
- Asynchronous paradigms like promise.

Errors#

Applications running in Node.js will generally experience four categories of errors:

- Standard JavaScript errors such as <EvalError>, <SyntaxError>, <RangeError>, <ReferenceError>, <Type Error>, and <URIError>.
- System errors triggered by underlying operating system constraints such as attempting to open a file that does not exist or attempting to send data over a closed socket.
- User-specified errors triggered by application code.
- AssertionErrors are a special class of error that can be triggered when Node.js detects an exceptional logic violation that should never occur. These are raised typically by the assert module.

All JavaScript and System errors raised by Node.js inherit from, or are instances of, the standard JavaScript <Error> class and are guaranteed to provide *at least* the properties available on that class.

Error Propagation and Interception#

Node.js supports several mechanisms for propagating and handling errors that occur while an application is running. How these errors are reported and handled depends entirely on the type of Error and the style of the API that is called.

All JavaScript errors are handled as exceptions that *immediately* generate and throw an error using the standard JavaScript throw mechanism. These are handled using the try...catch construct provided by the JavaScript language.

// Throws with a ReferenceError because z is not defined.

Error Handling

We cannot assure our code runs well. Because of this, we should think about how to handle errors. To do so, we can use try..catch syntax. Here is the error handling format in Node.js:

```
var n = 3;
var b = 0;
try{
var c = n/b;
if(c==Infinity)
throw new Error('this error is caused by invalid operation');
}catch (err){
console.log(err);
}
```

You can see the n/b operation returns an Infinity value. In this situation, we can throw our error so it will be caught.

If you run this code, you will get the program output shown

3.3 Logging

In the previous section, we used console.log() to print an error message in the console. Imagine there are many messages in the console and we want to identify which error message is occurring. The simple solution is to apply font color to the message to identify an error occurring.

Alternatively we can use log4js-node. To install this module, write this script:

For the Windows platform, you should run it under administrator level. You can do it using RunAs in the console:

```
var log4js = require('log4js');
var logger = log4js.getLogger('myapplication');
logger.info('Application is running');
logger.warn('Module cannot be loaded');
logger.error('Saved data was error');
logger.fatal('Server could not process');
logger.debug("Some debug messages");
```

The colours used are:

- TRACE - ‘blue’
- DEBUG - ‘cyan’
- INFO - ‘green’
- WARN - ‘yellow’
- ERROR - ‘red’
- FATAL - ‘magenta’

By default, log4js-node writes the message on the console. If you want to write all messages in a file, you can configure a logging file. To get started:

1. Activate file logging by calling loadAppender()

2. Configure the file logging configuration and category name.

For instance, we store the log file on c:\temp\myapplication.log with the category name **myapplication**. Here is a code illustration:

```
var log4js = require('log4js');
log4js.loadAppender('file');
log4js.addAppender(log4js.appenders.file('c:\temp\myapplication.log'),
'myapplication');
var logger = log4js.getLogger('myapplication');
logger.info('Application is running');
logger.warn('Module cannot be loaded.');
logger.error('Saved data was error');
logger.fatal('Server could not process');
logger.debug("Some debug messages");
```

3.4 Events: Events module, once event listener, remove events.

When a user interacts with an application in the form of a keyboard movement, a mouse click, or a mouseover, it generates an **event**. These events need to be handled to perform some kind of action. This is where **event binding** comes into picture.

Events enable an object to notify other objects when something of interest occurs. The object that sends (or raises) the event is called the publisher and the objects that receive (or handle) the event are called subscribers.
All event properties and methods are an instance of an EventEmitter object.
we prepare a function as a callback function and pass it into the on() function from the EventEmitter object.

```
var EventEmitter = require('events').EventEmitter;
var myEmitter = new EventEmitter;
var connection = function(id){
// do something
console.log('client id: ' + id);
};
myEmitter.on('connection', connection);
myEmitter.on('message', function(msg){
// do something
console.log('message: ' + msg);
});

myEmitter.emit('connection', 6);
myEmitter.emit('connection', 8);
myEmitter.emit('message', 'this is the first message');
myEmitter.emit('message', 'this is the second message');
myEmitter.emit('message', 'welcome to nodejs');
```

Explanation:

1. First, we load the events module.
2. Define the EventEmitter object and instantiate it.
3. We can define a function variable or put the function into the on() method directly.

4. To send the message, we can use the emit() method with the event name and data as parameters.

After writing the code, run it in the console. Here is program output of our appli

Once Event Listener

If you use the on() method, it means this event listener will listen for the event forever until the application closes. If you plan to listen for the event once, you can use the once() method.

```
var EventEmitter = require('events').EventEmitter;
var myEmitter = new EventEmitter;
myEmitter.once('message', function(msg){
// do something
console.log('message: ' + msg);
});
myEmitter.emit('message', 'this is the first message');
myEmitter.emit('message', 'this is the second message');
   myEmitter.emit('message', 'welcome to nodejs');
```

Remove Events

If you want to remove the event listener, call **removeListener**(). This function needs an event name and a function variable for parameters.

```
var EventEmitter = require('events').EventEmitter;
var myEmitter = new EventEmitter;
// functions
var connection = function(id){
// do something
console.log('client id: ' + id);
};
var message = function(msg){
// do something
console.log('message: ' + msg);
};
// waiting event
myEmitter.on('connection', connection);
myEmitter.on('message', message);
// send message
myEmitter.emit('connection', 6);
// remove event
myEmitter.removeListener('connection',connection);
```

```
// test to send message
myEmitter.emit('connection', 10);
myEmitter.emit('message', 'welcome to nodejs');
```

Above is a sample code for how to remove the 'connection' event.

UNIT-III

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. Most asynchronous methods that accept a _____ function will accept an Error object passed as the first argument to that function. [ ]

A. callback B. var clc = require('cli-color');

C. http://127.0.0.1:9000/ D. version

2. The primary methodology or innovation in Node.js is that it is built entirely around an event-driven nonblocking model of programming. [T/F]

3. Match the Following: [ ]

a) eventEmitter.once()	i) calls all listeners synchronously in the order
b) EventEmitter	ii) listener functions switch to asynchronous mode
c) setImmediate()	iii) if event is emitted, listener is unregistered

d) xhrCall()
.then(S1, E1) //P1
.then(S2, E2) //P2
.then(S3, E3) //P3
	iv) promise chain.

A) a-i, b-ii, c-iii, d-iv B) a-iii, b-i, c-ii d-iv

C) a-iv, b-iii c-ii d-i D) a-iii, b-iv, c-i, d-ii

4. In event handling we prepare a function as a callback function and pass it into _____ function from the EventEmitter object. [ ]

A. on() B. .catch() C. then() D.All of the above

5. Which of the following are TRUE? [ ]

i) If you use the on() method, it means this event listener will listen for the event forever until the application closes.

ii) If you plan to listen for the event once, you can use the once()

iii) all event properties and methods are an instance of an EventEmitter object.

A. i,ii,iii B. i, iii C. ii,iii D. i, ii

6. To send the message, we can use the emit() method with the event name and data as parameters. [T/F]

7. You could change the display text (**default**) by application category, just write the category on getLogger(), for instance, **myapplication**. [T/F]

8. The JavaScript try...catch mechanism **cannot** be used to intercept errors generated by asynchronous APIs. [T/F]

9. Which of the following are TRUE? [ ]

i) The callback function passed to fs.readFile() is called asynchronously

ii) Throwing an error inside the callback function passed to fs.readFile() **can crash the Node.js process** in most cases.

iii) If domains are enabled, or a handler has been registered with process.on ('uncaughtException'), errors in asynchronous calls can be intercepted.

iv) The JavaScript try...catch mechanism **cannot** be used to intercept errors generated by asynchronous APIs.

A. i,ii,iii B. i, iii C. ii,iii D.All

10. Match the following colors in error logging : [ ]

i) TRACE	a) 'blue'
ii) DEBUG	b) 'cyan'
iii) INFO	c) 'green'
iv) WARN	d) 'yellow'
v) ERROR	e) 'red'
vi) FATAL	f) 'magenta'

11. By default, log4js-node writes message on the console. If you want to write all messages in a file, you configure a logging file by _____. [ ]

```
A. log4js.addAppender(log4js.appenders.file('c:\myapl.log'),'myapplication');
```
B. `var logger = log4js.getLogger('myapplication');`
```
C. var logger = log4js.getLogger();
D. var EventEmitter = require('events').EventEmitter;
```

12. By default, all objects in JavaScript have a *prototype* object, which is the mechanism through which they inherit properties and methods.[ ]

A. prototype B. process C. Object D. Class

SECTION-B

SUBJECTIVE QUESTIONS

1) Develop a node.js program to illustrate once event listener.
2) Develop a node.js program for exporting module class.
3) Develop node.js program for events module and events class.
4) Develop a node.js program for error handling.
5) Apply the use of EventEmitter object to create events and prepare a function as a callback function and pass it into the on() function from the EventEmitter object.
6) Develop a node.js program for error logging.
7) Develop a node.js program for error file logging by calling loadAppender().

Node and Angular JS

UNIT - IV: Introduction to Angular:

4.1 Introduction to TypeScript (TS),

4.2 node package manager,

4.3 introduction to Angular4,

4.4 create angular application using TS and angular CLI,

4.5 webpack,

4.6 gulp introduction

https://www.typescriptlang.org/v2/docs/handbook/variable-declarations.html

https://www.tektutorialshub.com/typescript/variable-scope-in-typescript/

https://www.tutorialspoint.com/angularjs/angularjs_quick_guide.htm

https://codecraft.tv/courses/angular/angular-cli/overview/

https://www.tutorialspoint.com/angular4/index.htm

https://www.tutorialspoint.com/online_angularjs_editor.php

https://hackernoon.com/angular-4-by-examples-1dc9eb98be2

https://v2.angular.io/docs/ts/latest/guide/webpack.html

https://developers.google.com/web/ilt/pwa/introduction-to-gulp

4.1 Introduction to TypeScript (TS)

TypeScript is an open source programming language that is developed and maintained by Microsoft.

TypeScript is a superset of ECMAScript, supporting all of the syntax and semantics of JavaScript with some extra features on top, such as static typing and richer syntax.

TypeScript *is* a primary language for Angular application *development. It is a* superset *of* JavaScript *with* design-time support for *type* safety and tooling. Browsers can't execute *TypeScript* directly. *Typescript* must be "transpiled" into JavaScript *using* the tsc compiler, which requires some configuration.

- **TypeScript Code is converted into Plain JavaScript Code:** TypeScript code is not understandable by the browsers. That is why if the code is written in TypeScript then it is compiled and converted the code i.e. translate the code into JavaScript. The above process is known as **Trans-piled**. By the help of JavaScript code, browsers are able to read the code and display.
- **JavaScript is TypeScript**: Whatever code is written in JavaScript can be converted to TypeScript by changing the extension from **.js** to **.ts**.
- **Use TypeScript anywhere:** TypeScript code can be run on any browser, devices or in any operating system. TypeScipt is not specific to any Virtual-machine etc.
- **TypeScript supports JS libraries:** With TypeScript, developers can use existing JavaScript code, incorporate popular JavaScript libraries, and can be called from other JavaScript code.

Difference between TypeScript and JavaScript:

- TypesScript is known as Object oriented programming language whereas JavaScript is a scripting language.
- TypeScript has a feature known as Static typing but JavaScript does not have this feature.
- TypeScript gives support for modules whereas JavaScript does not support modules.
- TypeScript has Interface but JavaScript does not have Interface.
- TypeScript support optional parameter function but JavaScript does not support optional parameter function.

Advantages of using TypeScript over JavaScript

- TypeScript always point out the compilation errors at the time of development only. Because of this at the run-time the chance of getting errors are very less whereas JavaScript is an interpreted language.
- TypeScript has a feature which is strongly-typed or supports static typing. That means Static typing allows for checking type correctness at compile time. This is not available in JavaScript.
- TypeScript is nothing but JavaScript and some additional features i.e. ES6 features. It may not be supported in your target browser but TypeScript compiler can compile the **.ts** files into ES3,ES4 and ES5 also.

Disadvantages of using TypeScript over JavaScript

- Generally TypeScript takes time to compile the code.
- TypeScript does not support abstract classes.

Installing TypeScript:

There are two main ways to get the TypeScript tools:

- Via npm (the Node.js package manager)
- By installing TypeScript's Visual Studio plugins

Visual Studio 2017 and Visual Studio 2015 Update 3 include TypeScript by default. If you didn't install TypeScript with Visual Studio, you can still download it. For NPM users:

```
> npm install -g typescript
```

Program a **TypeScript** file **greeter.ts:**

```
function greeter(person) {
    return "Hello, " + person;
}

let user = "Jane User";
document.body.textContent = greeter(user);
```

Compiling TypeScript code:

At the command line, run the TypeScript compiler:

tsc greeter.ts

The result will be a file **greeter.js** which contains the same JavaScript that you fed in.

```
<!DOCTYPE html>
<html>
    <head><title>TypeScript Greeter</title></head>
    <body>
        <script src="greeter.js"></script>
    </body>
</html>
```

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page
- The <!DOCTYPE html> declaration defines this document to be HTML5
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the document
- The <title> element specifies a title for the document
- The <body> element contains the visible page content
- The <h1> element defines a large heading
- The <p> element defines a paragraph
- HTML tags normally come **in pairs** like <p> and </p>
- The first tag in a pair is the **start tag,** the second tag is the **end tag**
- The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

The start tag is also called the **opening tag**, and the end tag the **closing tag**.

HTML Tags are element names surrounded by angle brackets:

<tagname>content goes here...</tagname>

The type any

All the types in TypeScript are subtypes of a type called any. We can declare variables belonging to the any type using the any keyword. Such variables can hold the value of any type:

```
let    foo: any;
foo = {};
foo = 'bar ';
foo += 42;
console.log(foo); // "bar 42"
```

Unlike variables declared with var, variables declared with let have a block-scope.

var and let are both used for variable declaration in javascript but the difference between them is that var is function scoped and let is block scoped.

Decorators:

With the introduction of Classes in TypeScript and ES6, there now exist certain scenarios that require additional features to support annotating or modifying classes and class members. Decorators provide a way to add both annotations and a meta-programming syntax for class declarations and members. Decorators are a stage 2 proposal for JavaScript and are available as an experimental feature of TypeScript.

Decorators are an experimental feature that may change in future releases.

To enable experimental support for decorators, you must enable the experimentalDecorators compiler option either on the command line or in your tsconfig.json:

Command Line:

```
tsc --target ES5 --experimentalDecorators
```

SASS (Syntactically Awesome Stylesheet) is a CSS pre-processor, which helps to reduce repetition with CSS and saves time. It is more stable and powerful CSS extension language that describes the style of a document cleanly and structurally.

Why to Use SASS?

- It is a pre-processing language which provides indented syntax (its own syntax) for CSS.
- It provides some features, which are used for creating stylesheets that allows writing code more efficiently and is easy to maintain.
- It is a super set of CSS, which means it contains all the features of CSS and is an open source pre-processor, coded in **Ruby**.
- It provides the document style in a good, structured format than flat CSS. It uses re-usable methods, logic statements and some of the built-in functions such as color manipulation, mathematics and parameter lists.

Features of SASS

- It is more stable, powerful, and compatible with versions of CSS.
- It is a super set of CSS and is based on JavaScript.

4.2 Node Package Manager (NPM):

NPM is a package manager for Node.js packages, or modules if you like. www.npmjs.com hosts thousands of free packages to download and use. The NPM program is installed on your computer when you install Node.js.

npm makes it easy for JavaScript developers to share and reuse code, and makes it easy to update the code that you're sharing, so you can build amazing things.

npm is distributed with **Node.js**- which means that when you download Node.js, you automatically get npm installed on your computer.

Check that you have node and npm installed

To check if you have Node.js installed, run this command in your terminal:

```
node -v
```

To confirm that you have npm installed you can run this command in your terminal:

```
npm -v
```

npm versions

npm is a separate project from Node.js, and tends to update more frequently. As a result, even if you've just downloaded Node.js (and therefore npm), you'll probably need to update your npm. Luckily, npm knows how to update itself! To update your npm, type this into your terminal:

```
npm install npm@latest -g
```

Node versions and Long Term Support: Node.js has lots of versions! To use Node.js, and therefore npm, effectively, you'll want to make sure that you are on a version that is supported by the Node.js team. In general, you should use the version of Node.js labelled "LTS".

Use a Node.js version manager

Software is always changing, and so it's often a good practice to use a version manager to help account for this change. For this reason we use a version manager for Node.js installation. There are many options:

- NVM, nodist, n, nave

4.3 Salient features of Angular 4:

AngularJS is based on the model view controller (MVC), whereas Angular 2 is based on the components structure. Angular 4 works on the same structure as Angular2 but is faster when compared to Angular2.

Angular4 uses TypeScript 2.2 version whereas Angular 2 uses TypeScript version 1.8. This brings a lot of difference in the performance.

AngularJS is the most popular JavaScript MVC framework. Angular is essentially an HTML5 compiler.

To install Angular 4 with Angular CLI eases the installation. You need to run through a few commands to install Angular 4. Go to this site https://cli.angular.io to install Angular CLI.
Angular CLI helps development to debugging, testing to deploying the Angular apps.
angular-cli - refers to Angular 2. @angular/cli - refers to Angular 4.
To install
npm install -g @angular/cli
The underlying language used is typescript which is translated
to javascript using babel.
The task manager (like gulp, grunt) used by Angular CLI is webpack.

4 pillars of Angular 4

1. Component—encapsulates the template (html), data (variables) but not source of data and the behaviour (functions) of a view.
2. Directives—bridges the gap between backend and front end. Used for DOM manipulation.
3. Routers—takes care of navigation between components.
4. Services—reusable tags primarily used for manipulating DOM elements

Adding a component

Major part of the development with Angular 4 is done in the components. Components are basically classes that interact with the .html file of the component, which gets displayed on the browser. We have seen the file structure in one of our previous chapters. The file structure has the app **component** and it consists of the following files −

- **app.component.css**
- **app.component.html**
- **app.component.spec.ts**
- **app.component.ts**
- **app.module.ts**

The above files were created by default when we created new project using the angular-cli command.

Ahead of Time compilation - View Engine

In AoT mode, Angular compiles your templates during the **build**, and generates JavaScript code (by opposition to the Just in Time mode, where this compilation is done at runtime, when the application starts).

AoT has several advantages: it errors if one of your templates is incorrect at build time instead of having to wait at runtime, and the application starts faster (as the code generation is already done).
AOT compilation, include the --aot option with the ng build or ng serve command:

```
ng build --aot
ng serve --aot
```

The ng build command with the --prod meta-flag (ng build --prod) compiles with AOT by default.
The AOT compiler does not support function expressions and arrow functions, also called *lambda* functions.

Installing the Angular Command Line Interface

Now that we have a package manager in place, let's start using it to install everything that we need. To install the Angular CLI, which is a command line tool that we can use to scaffold Angular applications, we can run the following command:

```
ng    -v
npm install -g @angular/cli      //command to install angular 4
ng new Angular 4-app // name of the project
cd my-dream-app
ng serve
npm install -g yarn
```

If you are on Windows, you can instead install the nvm-windows tool

```
yarn global add @angular/cli
```

At this stage, if everything went well we should have the Angular CLI available at the command line. If we run this command we should have:

```
>ng --version
```

4.4 Create angular application using TS and angular CLI

We successfully installed Angular CLI on our system and now we can set up a workspace for Angular projects in our system and create a new app. This can be done with the below command.

```
ng new awesome-project
```

Running Our App
Now, the app we created can be run using **ng serve**.

```
cd awesome-project

ng serve --open
```

This launches the server, watches our files, and rebuilds the app as we make changes to those files.
The –open (or just -o) option automatically opens our browser to the below address.

```
http://local
```

Component	ng g component new-component
Directive	ng g directive new-directive
Pipe	ng g pipe new-pipe
Service	ng g service new-service
Module	ng g module my-module

The app-root is a component that is defined by our Angular application. In Angular we can define our own HTML tags and give them custom functionality. The app-root tag will be the “entry point” for our application on the page.

The @NgModule decorator identifies AppModule as an NgModule class.

@NgModule takes a metadata object that tells Angular how to compile and launch the application.

The [] parameter in the module definition can be used to define dependent modules.

Without the [] parameter, you are not *creating* a new module, but *retrieving* an existing one.

There are 5 types of binding are supported.
1. Property binding [] - bind the ts component property with html template property.
2. Event binding () - binding the html template event to ts component.
3. Two way data binding [()] - for component to template and vice versa data flow.
4. Class binding—e.g.: [class.active]. Is used to add/remove a CSS class.
5. Style binding—used to set the CSS style rules

4.5 Web pack:

Webpack is a popular module bundler, a tool for bundling application source code in convenient *chunks* and for loading that code from a server into a browser. **webpack** is a *static module bundler* for modern JavaScript applications. When webpack processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more *bundles*.

webpack.dev.js is the development configuration file.

To get started you only need to understand its **Core Concepts**:

1) Entry
2) Output
3) Loaders
4) Plugins
5) Mode
6) Browser Compatibility

Entries and outputs: You supply Webpack with one or more *entry* files and let it find and incorporate the dependencies that radiate from those entries. The one entry point file in this example is the application's root file, src/main.ts:

webpack.config.js (single entry)

```
entry: {
  'app': './src/main.ts'
},
```

Webpack inspects that file and traverses its import dependencies recursively.

4.6 Gulp

Tools like Gulp are often referred to as "build tools" because they are tools for running the tasks for building a website. The two most popular build tools out there right now are Gulp and Grunt.

Create a file named **gulpfile.js** in the folder which you would like to use Gulp and add the following:

```
//  Require Gulp into file and define the variable
var gulp = require('gulp');

//  Run the example task, if installed correctly and "gulp talktome" is ran,
// "Hello From Zestcode" should be printed in the logs
gulp.task('talktome', function() {
console.log('Hello From Zestcode');
});
```

Install Gulp: To install Gulp, open the terminal in the same directory you created the gulpfile.js file and run npm i gulp --save-dev, once it has finished running type gulp talktome in to the command line. Hello From Zestcode should appear in the terminal. If it does, congratulations! You have successfully installed NPM & Gulp.

Gulp tasks are defined in the **gulpfile.js** file using gulp.task. A simple task looks like this:

gulpfile.js

```
gulp.task('hello', function() {
  console.log('Hello, World!');
});
```

This code defines a hello task that can be executed by running the following from the command line: gulp hello

A common pattern for gulp tasks is the following:

1. Read some source files using gulp.src
2. Process these files with one or more functions using Node's pipe functionality
3. Write the modified files to a destination directory (creating the directory if doesn't exist) with gulp.dest

```
gulp.task('task-name', function() {
  gulp.src('source-files') // 1
  .pipe(gulpPluginFunction()) // 2
  .pipe(gulp.dest('destination')); // 3
});
```

gulp.watch: Even with default tasks, running tasks each time a file is updated during development can become tedious. gulp.watch watches files and automatically runs tasks when the corresponding files change. For example, the following code in **gulpfile.js** watches CSS files and executes the processCSS task any time the files are updated:

gulpfile.js

```
gulp.task('watch', function() {
  gulp.watch('styles/**/*.css', ['processCSS']);
});
```

Running the following in the command line starts the watch:

```
gulp watch
```

UNIT-4
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. Which of the following are True? [ ]

i) app-root is a component that is defined by our Angular application.

ii) In Angular we define our own HTML tags and give them custom functionality.

iii) The app-root tag will be the "entry point" for our application on the page.
iv) HTML <link> element refers to an external CSS file
A. i and ii B. i, iii and iv C. i and iv D. I, ii, iii and iv

2. Which of the following are TRUE? [ ]
i) AngularJS is the most popular JavaScript MVC.
ii) Angular is essentially an HTML5 compiler.
iii) The [] parameter in the module definition can be used to define dependent modules.
iv) Without the [] parameter, you are not *creating* a new module, but *retrieving* an existing one.
A. i and ii B. i, iii and iv C. i and iv D. I, ii, iii and iv

3. Which of the following are True?
i) Angular CLI is based on Webpack tool which helps process & bundle our various TypeScript, JavaScript, CSS, HTML, and image files.
ii) Angular CLI is not a requirement for using Angular.
iii) Angular CLI is a wrapper around Webpack that makes it easy to get started

iv) Some developers are of the opinion that AngularJS follows MVVM pattern instead of MVC, that replaces the Controller with a View-Model.

A. i,ii,iii, iv B. i,iii C. ii,iii D. i,ii

4. Which of the following are True? [ ]
i) The @NgModule decorator identifies AppModule as an NgModule class.
ii) The @NgModule takes a metadata object that tells Angular how to compile and launch the application.
iii) The $timeout service is AngularJS' version of the window.setTimeout function.
iv) The $location service has methods which return information about the location of the current web page:
A. i,ii,iii,iv B. i,iii,iv C. ii, iv D. ii,iii
5. The TypeScript compiler will give an error if we use variables before declaring them using let, whereas it won't give an error when using variables before declaring them using var. [T/F]
6. Whenever a new module, a component, or a service is created, the reference of the same is updated in the parent module __________ [ ]
A. app.module.ts B. app.component.spec.ts

C. app.component.ts   D. app.component.css

7. var and let are both used for variable declaration in javascript but the difference between them is that var is function scoped and let is block scoped. [T/F]
8. *Typescript* must be "transpiled" into JavaScript *using* the tsc compiler. [T/F]
9. 4200 is default port used when a new project is created in Angular.[T /F]
10. Match the following commands: [ ]

.a) ng build	i)	To compile Angular projects
b) ng serve.	ii)	To run Angular example
c) ng e2e.	iii)	run endto-end tests

A. a-i, b-ii, c-iii   B. a-ii, b-i, c -iii   C. a-ii, b-iii, c-i   D. a-iii, b-i,c-ii

SECTION-B

SUBJECTIVE QUESTIONS

1. Illustrate TypeScript configuration and the TypeScript environment that are important to Angular developers including details about the tsconfig.json configuration.

2. Create angular application using TS and angular CLI. (5M)
3. Discuss two groups of packages, organized in package.json in Angular.
4) Illustrate defining the variables in Typescript using let, var and const.
5) Discuss 4 pillars of Angular 4.
6. Discuss Variable Scope in TypeScript : Global, Local & function scope.
7) Develop Angulur JS program using angular CLI.
8) Develop Angulur JS program using angular webpack.
9) Summarize 5 types of binding Angular 4.
10) Develop Angulur JS program using gulp.

UNIT - V: Elements in Angular: Angular components, controllers, modules, dependency injection, angular service, providers and directives, pipes and filters, Angular forms-Reactive, lifecycle hooks.

https://github.com/codecraft-tv/angular-course/tree/current/9.forms/5.reactive-model-form/code/
https://www.tutorialspoint.com/angular2/angular2_services.htm
https://v2.angular.io/docs/ts/latest/guide/reactive-forms.html

5.1 Angular components,
5.2 controllers,
5.3 modules,
5.4 dependency injection,
5.5. angular service,
5.6. providers and directives,
5.7 pipes and filters,
5.8 Angular forms-Reactive,
5.9 lifecycle hooks.

5. Elements in Angular

Four pillars of Angular 4

1. Component—encapsulates the template (html), data (variables) but not source of data and the behaviour (functions) of a view.
2. Directives—bridges the gap between backend and front end. Used for DOM manipulation.
3. Routers—takes care of navigation between components.
4. Services—reusable tags primarily used for manipulating DOM elements

5.1 Angular components:

Components are the fundamental building blocks of Angular applications. They display data on the screen, listen for user input, and take action based on that input.

A basic Component has two parts:

1. A Component decorator

2. A component definition class

Let's look at the component code and then take these one at a time. Open up our first TypeScript file:

src/app/hello-world/hello-world.component.ts.

create a mycomp.component.ts file

```
import {Component} from '@angular/core'

@Component({
selector:'my-component',
template:'welcome to my custom component'
})

export class MyComponent{


}
```

Components :

Make changes to the application

Open the project in your favorite editor or IDE and navigate to src/app folder to make some changes to the starter app.

The implementation of AppComponent is distributed over three files:

1. app.component.ts— the component class code, written in TypeScript.
2. app.component.html— the component template, written in HTML.
3. app.component.css— the component's private CSS styles.

In AngularJS, a Component is a special kind of directive that uses a simpler configuration which is suitable for a component-based application structure.

Advantages of Components:

- simpler configuration than plain directives
- promote sane defaults and best practices
- optimized for component-based architecture
- writing component directives will make it easier to upgrade to Angular

When not to use Components:

for directives that need to perform actions in compile and pre-link functions, because they aren't available
when you need advanced directive definition options like priority, terminal, multi-element
when you want a directive that is triggered by an attribute or CSS class, rather than an element

5.2 AngularJS Controllers

AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is defined using ng-controller directive. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application or module that controller is to control.

AngularJS controllers **control the data** of AngularJS applications.
AngularJS controllers are regular **JavaScript Objects**, created by a standard JavaScript **object constructor**.

The **ng-controller** directive defines the application controller.

```
<div ng-app="myApp" ng-controller="myCtrl">
First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
  $scope.firstName = "John";
  $scope.lastName = "Doe";
});
</script>
```

5.3 Creating a Module

A module is created by using the AngularJS function **angular.module**.
An AngularJS module defines an application. The module is a container for the different parts of an application. The module is a container for the application controllers. Controllers always belong to a module.

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

var app = angular.module("myApp", []);
</script>

The "myApp" parameter refers to HTML element in which the application will run. Now you can add controllers, directives, filters, and more, to your AngularJS application.

The [] parameter in the module definition can be used to define dependent modules.

Without the [] parameter, you are not *creating* a new module, but *retrieving* an existing one.

5.4. Angular dependency injection:

What is a dependency?

When module A in an application needs module B to run, then module B is a *dependency* of module A.

The Angular dependency injection allows dependencies to be injected into the component or class.

Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies.

Dependency Injection (DI) is a technique in which we provide an instance of an object to another object, which depends on it. This is technique is also known as "Inversion of Control" (IoC)

The AngularJS injector subsystem is in charge of creating components, resolving their dependencies, and providing them to other components as requested.

We built a ProductService in the Angular Services. The AppComponent depends on the ProductService to provide the list of Products to display.

In short, the AppComponent has a **Dependency** on ProductService.

Parts of Angular Dependency Injection Framework

There are five main parts of the Angular Dependency injection Framework.

Consumer

The Component that needs the Dependency. In the above example, the AppComponent is the Consumer

Dependency:

The Service that is being injected. In the above example the ProductService is the Dependency

DI Token

The DI Token uniquely identifies a Dependency. We use DI Token when we register dependency

Provider

The Providers Maintains the list of Dependencies along with their *Tokens*. The *DI Token* is used to identify the Dependency.

Injector:

Injector holds the *Providers* and is responsible for resolving the dependencies and injecting the instance of the Dependency to the Consumer

5.5. What is a Angular Service

An Angular service is simply a Javascript function as a piece of reusable code with a focused purpose. A code that you will use it in many components across your application. All we need to do is to create a class and add methods & properties. We can then create an instance of this class in our component and call its methods.

One of the best uses of services is to get the data from the data source. Let us create a simple service, which gets the product data and passes it to our component.

What services are used for

1. Features that are independent of components such a logging services
2. Share logic or data across components
3. Encapsulate external interactions like data access

How to create a Service in Angular: The following key steps need to be carried out when creating a service.

Step 1 – Create a separate class which has the **injectable** decorator. The injectable decorator allows the functionality of this class to be injected and used in any Angular JS module.

```
@Injectable()

   export class classname {

}
```

Step 2 – Next in your appComponent module or the module in which you want to use the service, you need to define it as a provider in the @Component decorator.

```
@Component ({

   providers : [classname]

})
```

Letus look at an example on how to achieve this. Following are the steps involved.

Step 1 – Create a **ts** file for the service called app.service.ts.

5.6. Providers and Directives

A provider is an instruction to the Dependency Injection system on how to obtain a value for a dependency. Most of the time, these dependencies are services that you create and provide.

For the final sample app using the provider that this page describes, see the live example / download example.

Providing a service

If you already have an app that was created with the Angular CLI, you can create a service using the ng generate CLI command in the root project directory. Replace *User* with the name of your service.

ng generate service User

This command creates the following UserService skeleton:

src/app/user.service.ts

Directives in Angular is a **js** class, which is declared as **@directive**. We have 3 directives in Angular.

A **directive** is a custom HTML element that is used to extend the power of HTML.

Directives are components *without* a view. They are components without a template. Or to put it another way, components are directives *with* a view.

Everything you can do with a directive you can also do with a component. But not everything you can do with a component you can do with a directive.

We typically *associate* directives to existing elements by use *attribute* selectors, like so:

<elemenent aDirective></element>

The directives are listed below −

Component Directives

These form the main class having details of how the component should be processed, instantiated and used at runtime.

Structural Directives

A Structural directive basically deals with manipulating the dom elements. Structural directives have a * sign before the directive. ex ***ngIf** and ***ngFor**.

Attribute Directives

Attribute directives deal with changing the look and behavior of the dom element. You can create your own directives as shown below.

Angular 2 has following directives that get called as part of the BrowserModule module.

- ngif
- ngFor

If you view the app.module.ts file, you will see the following code and the BrowserModule module defined. By defining this module, you will have access to the 2 directives.

```
import { NgModule }      from '@angular/core';

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

import { AppComponent }  from './app.component';


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

5.7 pipes and filters

Angular 2 comes with several pipes, like following,

- CurrencyPipe: This pipe is used for formatting currency data. As an argument, it accepts the abbreviation of the currency type (that is, “EUR”, “USD”, and so on).
- DatePipe: This pipe is used for the transformation of dates.
- DecimalPipe: This pipe is used for transformation of decimal numbers. The argument it accepts is of the following form: “{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}”.
- JsonPipe: This transforms a JavaScript object into a JSON string.
- LowerCasePipe: This transforms a string to lowercase.
- UpperCasePipe: This transforms a string to uppercase.
- PercentPipe: This transforms a number into a percentage.
- SlicePipe: This returns a slice of an array. The pipe accepts the start and the end indexes of the slice.
- AsyncPipe: This is a stateful pipe that accepts an observable or a promise.

```
<!DOCTYPE html>
<html>
```

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

<div ng-app="myApp" ng-controller="myCtrl">
<p>The url of this page is:</p>
<h3>{{myUrl}}</h3>
</div>

<p>This example uses the built-in $location service to get the absolute url of
the page.</p>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $location) {
    $scope.myUrl = $location.absUrl();
});
</script>

</body>
</html>
```

5.9 Angular forms-Reactive,

Angular offers two form-building technologies: *reactive* forms and *template-driven* forms. The two technologies belong to the @angular/forms library and share a common set of form control classes.

But they diverge markedly in philosophy, programming style, and technique. They even have their own modules: the ReactiveFormsModule and the FormsModule.

"Reactive" forms (also known as model-driven), we'll be *avoiding* directives such as **ngModel**, **required** and friends. The idea is that instead of declaring that we want Angular to power things for us, we can actually use the underlying APIs to do them for us. In a sense, instead of binding Object models to directives like template-driven forms, we in fact boot up our own instances inside a component class and construct our own JavaScript models.

This has much more power and is extremely productive to work with as it allows us to write expressive code, that is very testable and keeps all logic in the same place, instead of scattering it around different form templates.

Reactive forms are synchronous. Template-driven forms are asynchronous.

5.10. Lifecycle hooks.

Following is a description of each lifecycle hook.

- **ngOnChanges** − When the value of a data bound property changes, then this method is called.
- **ngOnInit** − This is called whenever the initialization of the directive/component after Angular first displays the data-bound properties happens.
- **ngDoCheck** − This is for the detection and to act on changes that Angular can't or won't detect on its own.
- **ngAfterContentInit** − This is called in response after Angular projects external content into the component's view.

- **ngAfterContentChecked** – This is called in response after Angular checks the content projected into the component.
- **ngAfterViewInit** – This is called in response after Angular initializes the component's views and child views.
- **ngAfterViewChecked** – This is called in response after Angular checks the component's views and child views.
- **ngOnDestroy** – This is the cleanup phase just before Angular destroys the directive/component.

Following is an example of implementing one lifecycle hook. In the **app.component.ts** file, place the following code.

```
import {
   Component
} from '@angular/core';

@Component ({
   selector: 'my-app',
   template: '<div> {{values}} </div> '
})

export class AppComponent {
   values = '';
   ngOnInit() {
      this.values = "Hello";
   }
}
```

In the above program, we are calling the **ngOnInit** lifecycle hook to specifically mention that the value of the **this.values** parameter should be set to “Hello”.

UNIT-5

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. Import **ReactiveFormsModule** for reactive forms, and **FormsModule** for template-driven forms. [ ]

A. http://localhost:8080 B. var clc = require('cli-color');

C. http://127.0.0.1:9000/ D. **node --version**

2. Instead of using **FormControl** directly, we can use a API underneath that does it all for us with **FormBuilder.** [T/F]

3. Match the Following life cycle hooks : [ ]

a) ngAfterViewInit i) Invoked when the component's view has been fully initialized.

b) constructor ii) This is invoked when Angular creates a component or directive by calling new on the class.

c) ngAfterViewInit iii) Invoked when the component's view has been fully initialized.

d) ngOnDestroy iv) This method will be invoked before Angular destroys component.

A. a-i, b-ii, c-iii, d-iv B. a-ii, b-i, c-iii d-iv

C. a-ii, b-iii, c-i, d-iv D. a-iii, b-i,c-ii, d-iv

4.) In __________forms, we'll be avoiding directives such as **ngModel**, **required** and friends. [ ]

A. "reactive" C. template

B. model-driven. D. A and B

5. ng-model directive binds the values of AngularJS application data to HTML input controls. [T/F]

6. Reactive forms are synchronous and Template-driven forms are asynchronous. [T/F]

7. Which of the following are TRUE?

i) Angular offers two form-building technologies: reactive forms and template-driven forms.

ii) The two technologies belong to the @angular/forms library and share a common set of form control classes.

iii) Two form-building technologies have their own modules: the ReactiveFormsModule and the FormsModule.

A. i, ii, B. ii, iii

C. ii, iii, iv D. i, ii, iii, iv

8. Match the following in DI: [ ]

a) Consumer i) The Component that needs the Dependency

b) Dependency ii) The Service that is being injected

c) Token iii) identifies that we want injected, *dependency* of our code.

d) Injector iv) function which when passed a *token* returns a *dependency*

A. a-i, b-ii, c-iii, d-iv B. a-ii, b-i, c-iii d-iv

C. a-ii, b-iii, c-i, d-iv D. a-iii, b-i,c-ii, d-iv

SECTION-B

SUBJECTIVE QUESTIONS

1) Develop Angulur JS program using pipes and filters and lifecycle hooks.

2) Illustrate controllers, modules, angular service, providers and directives.

3) Develop Angulur JS program using Angular components for Angular forms-Reactive and lifecycle hooks.

4) Discuss dependency injection in angular service.

5. Illustrate controllers, modules, angular service, providers and directives