

MST MID 1 Important Questions

UNIT – 1: HTML

HTML

1.Demonstrate DOCTYPE, Metadata Element, Division and Span elements in HTML with example program. (BTL 3 – Apply, CO1)

DOCTYPE

All HTML documents must start with a `<!DOCTYPE>` declaration.

The declaration is not an HTML tag. It is an "information" to the browser about what document type to expect.

```
<!DOCTYPE html>
```

Metadata Elements

The `<meta>` tag defines metadata about an HTML document. Metadata is data (information) about data.

Metadata will not be displayed on the page, but is machine parsable.

EX: `<meta name="description" content="Free Web tutorials">`

HTML `<div>` Tag

The `<div>` tag defines a division or a section in an HTML document.

The `<div>` tag is used as a container for HTML elements - which is then styled with CSS or manipulated with JavaScript.

The `<div>` tag is easily styled by using the class or id attribute.

Any sort of content can be put inside the `<div>` tag!

HTML `` Tag

The `` tag is an inline container used to mark up a part of a text, or a part of a document.

The `` tag is easily styled by CSS or manipulated with JavaScript using the class or id attribute.

The `` tag is much like the `<div>` element, but `<div>` is a block-level element and `` is an inline element.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta name="author" content="John Doe">
  <title>Document</title>
</head>
<body>
  <div>
    Hello my name is <span>Sandeep</span>
  </div>
</body>
</html>
```

2.Demonstrate about Sectioning Elements (aside, header, footer, main, section, article, nav) in HTML with example program. Or Using HTML 5 tags design a simple web page. (BTL 3 – Apply, CO1)

HTML <aside> Tag

The `<aside>` tag defines some content aside from the content it is placed in.

The aside content should be indirectly related to the surrounding content.

HTML <header> Tag

The `<header>` element represents a container for introductory content or a set of navigational links.

It may contain some heading elements but also a logo, a search form, an author name, and other elements.

HTML <footer> Tag

A `<footer>` typically contains information about the author of the section, copyright data or links to related documents.

HTML <main> Tag

The `<main>` tag specifies the main content of a document.

The content inside the `<main>` element should be unique to the document. It should not contain any content that is repeated across documents such as sidebars, navigation links, copyright information, site logos, and search forms.

HTML <section> Tag

The <section> [HTML](#) element represents a generic standalone section of a document, which doesn't have a more specific semantic element to represent it. Sections should always have a heading, with very few exceptions.

<article>

The <article> tag specifies independent, self-contained content.

An article should make sense on its own and it should be possible to distribute it independently from the rest of the site.

HTML <nav> Tag

The <nav> tag defines a set of navigation links.

Notice that NOT all links of a document should be inside a <nav> element. The <nav> element is intended only for major blocks of navigation links.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta name="author" content="John Doe">
  <title>Document</title>
</head>
<body>
  <header>This is header tag</header>
  <nav>This is nav tag element</nav>
  <main>
    <section>
      This is section
      <article>
        this is some artical <aside>this is aside tag</aside>
      </article>
    </section>
  </main>
  <footer>This is footer</footer>
</body>
</html>
```

3.Determine different types of lists with an example program. (BTL 3 – Apply, CO1)

HTML Lists

Unordered HTML List

An unordered list starts with the `` tag. Each list item starts with the `` tag.

The list items will be marked with bullets (small black circles) by default

Ordered HTML List

An ordered list starts with the `` tag. Each list item starts with the `` tag.

The list items will be marked with numbers by default

HTML Description Lists

HTML also supports description lists.

A description list is a list of terms, with a description of each term.

The `<dl>` tag defines the description list, the `<dt>` tag defines the term (name), and the `<dd>` tag describes each term

```
<!DOCTYPE html>
<html lang="en">
<body>
  <ul>
    <li>Coffee</li>
    <li>Tea</li>
  </ul>
  <ol>
    <li>Coffee</li>
    <li>Tea</li>
  </ol>
  <dl>
    <dt>Coffee</dt>
    <dd>Black hot drink</dd>
    <dt>Tea</dt>
    <dd>Also a black hot drink</dd>
  </dl>
</body>
</html>
```

-
- Coffee
 - Tea
 - Milk

1. Coffee
2. Tea
3. Milk

Coffee

Black hot drink

Tea

Also a black hot drink

Milk

White cold drink

4. Illustrate links, images with an example program. (BTL 3 – Apply, CO1)

HTML Links

HTML links are hyperlinks.

You can click on a link and jump to another document.

The `<a>` [HTML](#) element (or *anchor* element), with [its href attribute](#), creates a hyperlink to web pages, files, email addresses, locations in the same page, or anything else a URL can address.

Syntax: `link text`

The `target` attribute specifies where to open the linked document.

HTML Images

The HTML `` tag is used to embed an image in a web page.

Images are not technically inserted into a web page; images are linked to web pages. The `` tag creates a holding space for the referenced image.

The `` tag is empty, it contains attributes only, and does not have a closing tag.

The `` tag has two required attributes:

- `src` - Specifies the path to the image
- `alt` - Specifies an alternate text for the image

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta name="author" content="John Doe">
  <title>Document</title>
</head>
<body>
  <a href="www.google.com">Google</a>
  
</body>
</html>
```



5. Apply form elements (text field, password field, select, checkbox, radio button, submit and reset buttons) to create a registration form. (BTL 3 – Apply, CO1)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta name="author" content="John Doe">
  <title>Document</title>
</head>
<body>
  <form action="">
    UserID: <input type="text"> <br>
    Password: <input type="password"> <br>
    Gender: <input type="radio" name="gender" value="male"> Male
```

```

        <input type="radio" name="gender" value="female"> Female <br>
Place:
<select>
    <option value="">Delhi</option>
    <option value="">Banglore</option>
    <option value="">Mumbai</option>
</select>
<input type="checkbox"> C++
<input type="checkbox"> Java <br>
<input type="submit" value="Login">

</form>
</body>
</html>

```

UserID:
 Password:
 Gender: ☐ Male ☐ Female
 Place: ☐ C++ ☐ Java

6. Using Table Elements (table, th, tr, td) and attributes (Colspan/Rowspan, border, cellpadding and cellspacing) design a static web page that displays an employee information table with three rows and three columns as shown below: (BTL 3 – Apply, CO1)

EID	ENAME	MOBILE
1	ABC	0123456789 9876543210
2	XYZ	9876543210 0123456789

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta name="author" content="John Doe">
    <title>Document</title>
</head>
<body>
    <table>
        <tr>
            <td>EID</td>
            <td>ENAME</td>
            <td>MOBILE</td>
        </tr>
    </table>

```

```

        <tr>
            <td>1</td>
            <td>ABC</td>
            <td>0123456789 9876543210</td>
        </tr>
        <tr>
            <td>2</td>
            <td>XYZ</td>
            <td>9876543210 0123456789</td>
        </tr>
    </tr>
</table>
</body>
</html>

```

EID ENAME MOBILE

1	ABC	0123456789 9876543210
2	XYZ	9876543210 0123456789

7. Apply audio and video elements to insert media elements into a web page. (BTL 3 – Apply, CO1)

The HTML5 <audio> and <video> tags make it simple to add media to a website. You need to set **src** attribute to identify the media source and include a controls attribute so the user can play and pause the media.

Embedding Video

Here is the simplest form of embedding a video file in your webpage –

```

<video src = "foo.mp4" width = "300" height = "200" controls>
    Your browser does not support the <video> element.
</video>

```

Embedding Audio

HTML5 supports <audio> tag which is used to embed sound content in an HTML or XHTML document as follows.

```

<audio src = "foo.wav" controls autoplay>
    Your browser does not support the <audio> element.
</audio>

```


CSS

1.Explain different ways of applying styles to a HTML web page. (K3 – Apply, CO1)

Adding Styles to HTML Elements

Style information can either be attached as a separate document or embedded in the HTML document itself. These are the three methods of implementing styling information to an HTML document.

- **Inline styles** — Using the `style` attribute in the HTML start tag.
- **Embedded style** — Using the `<style>` element in the head section of the document.
- **External style sheet** — Using the `<link>` element, pointing to an external CSS files.

Inline Styles

Inline styles are used to apply the unique style rules to an element, by putting the CSS rules directly into the start tag. It can be attached to an element using the `style` attribute.

```
<h1 style="color:red; font-size:30px;">This is a heading</h1>
```

Embedded Style Sheets

Embedded or internal style sheets only affect the document they are embedded in.

Embedded style sheets are defined in the `<head>` section of an HTML document using the `<style>` tag. You can define any number of `<style>` elements inside the `<head>` section.

```
<style>
    body { background-color: YellowGreen; }
    h1 { color: blue; }
    p { color: red; }
</style>
```

External Style Sheets

An external style sheet is ideal when the style is applied to many pages.

An external style sheet holds all the style rules in a separate document that you can link from any HTML document on your site. External style sheets are the most flexible because with an external style

```
body {
    color: blue;
    font-size: 14px;
}
```

Unit II : Javascript

Javascript

1. Discuss how to embed Javascript in a web page with example program. (K2 – Understand, CO2)

What is JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document

The `<script>` tag is used to embed a client-side script (JavaScript).

The `<script>` element either contains scripting statements, or it points to an external script file through the src attribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content

```
<!DOCTYPE html>
<html>
<body>
<h1>The script element</h1>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
</body>
</html>
```

2. Explain about I/O statements with example programs. (K2 – Understand, CO2)

The `prompt()` method displays a dialog box that prompts the user for input.

The `prompt()` method returns the input value if the user clicks "OK", otherwise it returns `null`.

The `<input>` tag specifies an input field where the user can enter data.

The `<input>` element is the most important form element.

The `<input>` element can be displayed in several ways, depending on the type attribute.

Output:

JavaScript can "display" data in different ways:

- Writing into the HTML output using `document.write()`.
- The `write()` method writes directly to an open (HTML) document stream.
- Writing into an alert box, using `window.alert()`.
- The `alert()` method displays an alert box with a message and an OK button.
- The `alert()` method is used when you want information to come through to the user.
- Writing into the browser console, using `console.log()`.
- The `log()` method writes (logs) a message to the console.
- The `log()` method is useful for testing purposes.

```
<!DOCTYPE html>
<html lang="en">
  <head>
  </head>
  <body>
    <script>
      x = window.prompt("Enter your name");
      document.write("Hello " + x);
    </script>
  </body>
</html>
```

3. Illustrate control statements (conditional and loops) with syntax and examples. (K3 – Apply, CO2)

1. Conditional Statements

This is where the flow of the execution in a program is decided. Conditional Statements decide the next step based on the result. Conditional statement results in either True or False. Whatever the condition is passed, if that is true, then the program moves to the next step and if the condition is False, then the program moves to another step

- IF

when you want to check for a specific condition. With the IF condition, the inner code block is executed if the condition provided is satisfied.

Syntax:

```
if (condition) {  
  
    //code block to be executed if condition is satisfied  
  
}
```

- IF-ELSE

an extended version of IF. When you want to check a specific condition and two

Syntax:

```
if (condition)  
{  
  
    // code to be executed of condition is true  
  
}  
  
else {  
  
    // code to be executed of condition is false  
  
}
```

As you can see, when the condition is satisfied in IF-ELSE, the first block of code will be executed and if the condition isn't satisfied, the second block of code will be executed.

- SWITCH

A switch statement is similar to IF and is of use when you need to execute one code out of the multiple code block execution possibilities, based on the result of the expression passed. Switch statements carry an expression, which is compared with values of the following cases and once a match is found, code associated with that case executes.

Syntax:

```
switch (expression) {  
  
  case a:  
  
    //code block to be executed  
  
    Break;  
  
  case b:  
  
    //code block to be executed  
  
    Break;  
  
  case n:  
  
    //code block to be executed  
  
    Break;  
  
  default:  
  
    //default code to be executed if none of the above case is executed  
  
}
```

2. Iterative Statement

Looping, for any programming language, is a powerful tool in order to execute a set of instructions, repeatedly, while the expression passed is satisfied. A very basic example can be, to print "Hello World" for 10 times.

- **WHILE**

one of the control flow statement, which executes a code block when the condition is satisfied. But unlike IF, while keeps repeating itself until the condition is satisfied. Difference between IF and while can be, IF executes code 'if' the condition is satisfied while the while keeps repeating itself until the condition is satisfied.

Syntax:

```
while (condition)
{
    //code block to be executed when condition is satisfied
}
```

- **DO-WHILE**

Similar to a while loop, with a twist that keeps a condition at the end of the loop. Also known as Exit Control Loop, DO-WHILE executes the code and then checks for the condition.

Syntax:

```
while
{
    //code block to be executed when condition is satisfied
} (condition)
```

If the condition at the end is satisfied, the loop will repeat.

- **FOR**

a for loop will execute a code block for a number of times. Compared to other loops, FOR is shorter and easy to debug as it contains initialization, condition and increment or decrement in a single line.

Syntax:

```
for (initialize; condition; increment/decrement)
{
    //code block to be executed
}
```

a. Program to find whether a given number is Prime or not

```
<!DOCTYPE html>
<html lang="en">
<head>
</head>
<body>
  <script>
    function primeNumber(num) {
      for (var i = 2; i < num; i++) {
        if (num % i === 0) {
          return false;
        }
      }
      return num > 1;
    }
    x = 7;
    document.write(primeNumber(x));
  </script>
</body>
</html>
```

4. Explain different types of functions (predefined, user-defined, non-parameterized, parameterized, return value) with example program. (K2 – Understand, CO2)

5. Discuss the advantage of Arrow function with example program. (K2 – Understand, CO2)

- This arrow function reduces lots of code and makes the mode more readable.
- Arrow function syntax automatically binds "this" to the surrounding code's context.
- Writing the arrow => is more flexible as compared with the writing **function** keyword.

Advantages of Arrow Function

1

Reduces code size

2

Return Statement is optional for single line function

3

Lexically bind the context

4

Functional braces are optional for single line Statement

6.Explain Objects (Predefined and User Defined) with example programs. (K2 – Understand, CO2)

Predefined:

Array – concat, push, pop, slice, splice, reverse etc

String – length, charAt, concat, indexOf, split, substr, substring, toLowercase, toUppercase

Math – max, min, pow, sqrt, ceil, round, floor, random, parseInt

Regexp – match, search etc

Userdefined:

class, object, properties, function/method, constructor

8.Discuss about Array creation, destructuring and accessing arrays with syntax and example programs.
(K2 – Understand, CO2)

An array is an object that can store multiple values at once. For example,

```
const words = ['hello', 'world', 'welcome'];
```

Create an Array

You can create an array using two ways:

1. Using an array literal

The easiest way to create an array is by using an array literal `[]`. For example,

```
const array1 = ["eat", "sleep"];
```

2. Using the new keyword

You can also create an array using JavaScript's `new` keyword.

```
const array2 = new Array("eat", "sleep");
```

Destructuring Assignment is a JavaScript expression that allows to **unpack values** from arrays, or properties from objects, into distinct variables data can be extracted from *arrays, objects, nested objects* and **assigning to variable**

```
var x, y;
```

```
[x, y] = [10, 20];
```



```
[a, b, ...rest] = [10, 20, 30, 40, 50];
```

9. Illustrate Asynchronous programming using callback, promises, async, await and fetch with syntax and example program. (K3 – Apply, CO2)

callback

In JavaScript, you can also pass a function as an argument to a function. This function that is passed as an argument inside of another function is called a callback function.

Promises:

A **Promise** is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a *promise* to supply the value at some point in the future

Async:

An async function is a function declared with the `async` keyword, and the `await` keyword is permitted within it. The `async` and `await` keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

Await:

The `await` expression causes async function execution to pause until a promise is settled (that is, fulfilled or rejected), and to resume execution of the async function after fulfillment. When resumed, the value of the `await` expression is that of the fulfilled promise.

The **`fetch()`** method in JavaScript is used to request to the server and load the information on the webpages. The request can be of any APIs that return the data of the format JSON or XML. This method returns a promise.

Syntax:

```
fetch('url')           //api for the get request
  .then(response => response.json())
  .then(data => console.log(data));
```



10. Demonstrate how to create and consume module in Javascript with an example program. (K3 – Apply, CO2)

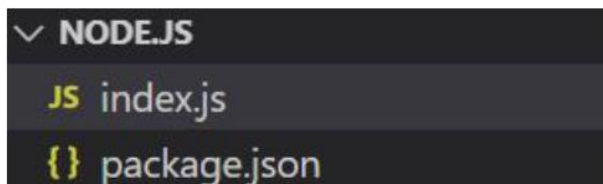
UNIT – III: Node.js

1. Discuss the process of creating a web server in node.js with an example program and explain how to start the web server, run a node application and restart a node application automatically whenever the node.js file is changed without any manual restart. (K2 – Understand, CO3)

Introduction: Node.js is an open-source and cross-platform runtime environment for executing [JavaScript](#) code outside a browser. You need to remember that **NodeJS is not a framework, and it's not a programming language**. Node.js is mostly used in server-side programming. In this article, we will discuss how to make a web server using node.js.

Using http module: HTTP and HTTPS, these two inbuilt modules are used to create a simple server. The HTTPS module provides the feature of the encryption of communication with the help of the secure layer feature of this module. Whereas the HTTP module doesn't provide the encryption of the data.

Project structure: It will look like this.



```
const http = require("http")

// Creating server
const server = http.createServer((req, res) => {
  // Sending the response
  res.write("This is the response from the server")
  res.end();
})

// Server listening to port 3000
server.listen(3000, () => {
  console.log("Server is Running");
})
```

Run **index.js** file using below command:

```
node index.js
```

Output

A screenshot of a web browser's address bar. It features navigation icons (back, forward, refresh) on the left and an information icon followed by the text 'localhost:3000' on the right.

This is the response from the server

Node.js Automatic restart Node.js server with nodemon

Nodemon is a package for handling this restart process automatically when changes occur in the project file.

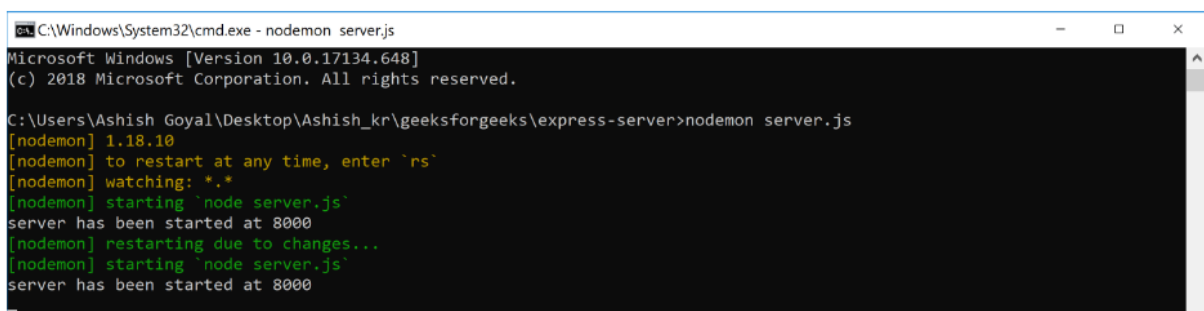
Installing nodemon: nodemon should be installed globally in our system:

```
Windows system: npm i nodemon -g
```

Starting node server with nodemon:

```
nodemon [Your node application]
```

Now, when we make changes to our nodejs application, the server automatically restarts by nodemon as shown in the below screenshot.



```
C:\Windows\System32\cmd.exe - nodemon server.js
Microsoft Windows [Version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Ashish Goyal\Desktop\Ashish_kr\geeksforgeeks\express-server>nodemon server.js
[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
server has been started at 8000
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
server has been started at 8000
```

2.Explain the importance of npm to install the modules. (K2 – Understand, CO3)

NPM - Node Package Manager

Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application. It is also an online repository for open-source Node.js packages. The node community around the world creates useful modules and publishes them as packages in this repository.

npm is the world's largest **Software Registry**.

The registry contains over 800,000 **code packages**.

Open-source developers use **npm** to **share** software.

Many organizations also use npm to manage private development.

npm includes a **CLI** (Command Line Client) that can be used to download and install software:

3.Demonstrate http, uri, fs modules with example programs. (K3 – Apply, CO3)

Node.js HTTP Module

To make HTTP requests in Node.js, there is a built-in module **HTTP** in Node.js to transfer data over the HTTP. To use the HTTP server in node, we need to require the HTTP module. The HTTP module creates an HTTP server that listens to server ports and gives a response back to the client.

Syntax:

```
var http = require('http');
```

```
var http = require('http');
http.createServer((request, response)=>{
    response.write('Hello World!');
    response.end();
})
.listen(3000);
```

The URL module provides utilities for URL resolution and parsing. It can be accessed using:

```
01. | var url = require('url');
```

Url module is one of the core modules that comes with node.js, which is used to parse the URL and its other properties.

MyApp.js

```
01. var http = require('http');
02. var url = require('url');
03.
04. http.createServer(function (req, res) {
05.
06.     var queryString = url.parse(req.url, true);
07.     console.log(queryString);
08.
09. }).listen(4200);
```

About Node.js file system: To handle file operations like creating, reading, deleting, etc., Node.js provides an inbuilt module called FS (File System). Node.js gives the functionality of file I/O by providing wrappers around the standard POSIX functions. All file system operations can have synchronous and asynchronous forms depending upon user requirements. To use this File System module, use the `require()` method:

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

```
var fs = require("fs");
fs.readFile('input.txt', function (err, data) {
  if (err) {
    return console.error(err);
  }
  console.log("Asynchronous read: " + data.toString());
});
```

- **Output:**

```
Asynchronous read: GeeksforGeeks: A computer science portal
```

4.Demonstrate the process of creating user-defined module and use in node.js application with an example program. (K3 – Apply, CO3)

How to Create Modules in Node.js ?

Modules are the collection of JavaScript codes in a separate logical file that can be used in external applications on the basis of their related functionality. Modules are popular as they are easy to use and are reusable.

To create a module in Node.js, you will need the **exports** keyword. This keyword tells Node.js that the function can be used outside the module.

Syntax:

```
exports.function_name = function(arg1, arg2, ....argN) {  
    // function body  
};
```

```
// File name: calc.js  
  
exports.add = function (x, y) {  
    return x + y;  
};
```

- Use the 'require' keyword to import the file

```
// File name: App.js  
var calculator = require('./calc');  
  
var x = 50, y = 20;  
  
console.log("Addition of 50 and 20 is "  
            + calculator.add(x, y));
```

- Output:

```
Addition of 50 and 20 is 70
```