AIT

Q 1) Explain what is Event Handling in Node.js with simple suitable example

In Node.js, event handling refers to the process of responding to events that occur within an application. Events can be triggered by various actions, such as user input, network requests, or system events. Node.js provides an event-driven architecture that allows developers to define and handle events using the EventEmitter module.

The EventEmitter module is a built-in module in Node.js that provides an implementation of the observer pattern. It allows you to create custom events and attach event listeners to them. Here's a simple example to illustrate event handling in Node.js:

```
const EventEmitter = require('events');

// Create an instance of the EventEmitter
const eventEmitter = new EventEmitter();

// Define an event handler function
const eventHandler = (message) => {
  console.log('Event handled:', message);
};

// Attach the event handler to the custom event
eventEmitter.on('customEvent', eventHandler);

// Emit the custom event
eventEmitter.emit('customEvent', 'Hello, World!');
```

In the example above, we first import the EventEmitter module from the 'events' core module in Node.js. We create an instance of EventEmitter using the new keyword.

Next, we define an event handler function called eventHandler, which takes a message parameter and logs a message to the console.

We attach the event handler to a custom event named 'customEvent' using the on method provided by EventEmitter. The on method takes the event name and the event handler function as arguments.

Finally, we emit the custom event using the emit method of EventEmitter, passing the event name and any necessary data. In this case, we emit the 'customEvent' and pass the message 'Hello, World!'.

When the custom event is emitted, the event handler function is executed, and the message 'Hello, World!' is logged to the console.

Q 2) Write a simple program to display string in uppercase and lower case using anguler js filters

To display a string in uppercase and lowercase using AngularJS filters, you can create a simple AngularJS application and utilize the `uppercase` and `lowercase` filters provided by AngularJS. Here's an example:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>String Case Display</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
</head>
<body>

<div ng-controller="myController">
  <label>Enter a string:</label>
  <input type="text" ng-model="inputString">

  <p>Uppercase: {{ inputString | uppercase }}</p>
  <p>Lowercase: {{ inputString | lowercase }}</p>
</div>

<script>
angular.module('myApp', [])
  .controller('myController', function($scope) {
    $scope.inputString = '';
  });
</script>

</body>
</html>
```

In the example above, we start by including AngularJS library using the <script> tag. We then define an AngularJS module called myApp and a controller called myController.

Inside the controller, we define a scope variable $scope.inputString to hold the user's input.

In the HTML, we bind the inputString variable to an <input> element using the ng-model directive. This allows us to capture the user's input.

We then use the uppercase and lowercase filters in the {{ }} curly braces notation to display the uppercase and lowercase versions of the input string, respectively.

When the user enters a string in the input field, AngularJS automatically updates the uppercase and lowercase values displayed on the page.

Q 3) Explain ng-app,ng-model and hg-bind with simple suitable example

Certainly! Here's an explanation of `ng-app`, `ng-model`, and `ng-bind` in AngularJS with simple examples:

1. `ng-app`: The `ng-app` directive is used to define the root element of an AngularJS application. It initializes the AngularJS framework and sets the scope for the application. It is typically placed on the `<html>` or `<body>` tag.

Example:

```
<!DOCTYPE html>
<html ng-app="myApp">
...
</html>
```

In this example, ng-app="myApp" sets the root element of the AngularJS application to be the <html> tag. The application module named myApp is used to bootstrap the AngularJS application.

2. ng-model:

The ng-model directive is used to bind the value of an HTML input element to a property in the AngularJS scope. It creates a two-way data binding between the input element and the model, allowing changes in one to reflect in the other.

Example:

```
<input type="text" ng-model="name">
<p>Hello, {{ name }}!</p>
```

In this example, the ng-model="name" binds the value of the <input> element to the name property in the AngularJS scope. Any changes made to the input field will automatically update the value of name, and it will be displayed in the <p> element using {{ name }}.

3. ng-bind:

The ng-bind directive is used to bind the value of an expression to a specific HTML element. It allows you to dynamically insert the value of an expression into the element's content.

Example:

```
<p ng-bind="message"></p>
```

In this example, the ng-bind="message" sets the content of the <p> element to be the value of the message property in the AngularJS scope. The value of message will be evaluated and displayed within the <p> element.

Q 4) write a simple program for multipilication of 2 numbers using event handlinng in node.js call multiplication function as an event call

Certainly! Here's an example of a Node.js program that multiplies two numbers using event handling and the EventEmitter module:

```
const EventEmitter = require('events');

// Create an instance of the EventEmitter
const eventEmitter = new EventEmitter();

// Define the multiplication function
const multiply = (num1, num2) => {
  const result = num1 * num2;
  // Emit the 'multiplication' event with the result
  eventEmitter.emit('multiplication', result);
};

// Attach an event listener to the 'multiplication' event
eventEmitter.on('multiplication', (result) => {
  console.log('Multiplication Result:', result);
});

// Call the multiply function as an event call
multiply(5, 7);
```

In the example above, we start by importing the EventEmitter module from the 'events' core module in Node.js. We create an instance of EventEmitter using the new keyword.

Next, we define the multiply function that takes two numbers, num1 and num2, as parameters. Inside the function, we calculate the multiplication result and emit the 'multiplication' event using the eventEmitter.emit() method, passing the result as an argument.

Then, we attach an event listener to the 'multiplication' event using the eventEmitter.on() method. The listener function receives the result as a parameter and logs it to the console.

Finally, we call the multiply function as an event call, passing the numbers 5 and 7 as arguments. This triggers the 'multiplication' event, and the event listener is executed, printing the multiplication result to the console.

When you run this Node.js program, it will output:

```
Multiplication Result: 35
```

Q 5) what is node.js ? Expalin its working and features?

Node.js is an open-source, server-side JavaScript runtime environment built on Chrome's V8 JavaScript engine. It allows you to run JavaScript code outside of a browser, enabling you to build scalable, high-performance network applications. Node.js follows an event-driven, non-blocking I/O model, making it well-suited for handling concurrent requests and building real-time applications.

Here are some key aspects of Node.js and its features:

1. **JavaScript on the server-side:** Node.js allows you to use JavaScript both on the client-side and server-side, providing a unified language for full-stack development.
2. **Event-driven architecture:** Node.js is built around an event-driven architecture, where events trigger asynchronous callbacks. This approach allows the server to efficiently handle concurrent requests without blocking the execution of other operations.
3. **Non-blocking I/O operations:** Node.js uses non-blocking, asynchronous I/O operations, which means that it doesn't wait for file I/O, network requests, or database queries to complete before moving on to the next task. This asynchronous nature makes Node.js highly scalable and efficient.
4. **NPM (Node Package Manager):** Node.js has a vast ecosystem of open-source packages available through NPM. NPM allows developers to easily manage dependencies, reuse code, and share their own packages with others.
5. **Scalability:** Node.js is designed to handle a large number of concurrent connections with minimal overhead. It is well-suited for building scalable applications that require high performance, such as real-time applications, chat servers, or streaming platforms.
6. **Single-threaded, event loop:** Node.js runs on a single thread but uses an event loop to handle multiple concurrent operations efficiently. The event loop manages I/O operations and callbacks, ensuring that the server remains responsive while processing multiple requests.
7. **Extensive library support:** Node.js has a rich set of built-in modules and libraries for various tasks, such as networking, file system operations, encryption, web servers, and more. These modules, along with the vast number of third-party libraries, make it easy to develop a wide range of applications.
8. **Microservices and API development:** Node.js is often used for building microservices and RESTful APIs due to its lightweight nature, fast I/O operations, and ability to handle concurrent requests. It integrates well with modern web technologies and frameworks like Express.js to build robust and scalable APIs.
9. **Real-time communication:** With the support of libraries like Socket.IO, Node.js is widely used for real-time communication applications, such as chat applications, multiplayer games, collaborative tools, and live streaming platforms.

Overall, Node.js is a powerful and popular platform for server-side development, offering high performance, scalability, and a vast ecosystem of libraries and tools. Its event-driven, non-blocking architecture makes it an excellent choice for building real-time, data-intensive applications.

Q 6) create a login form, both username and password fields are mandatory, after entering the values transfer user control to next web page showing message as "You have login successfully ". use ng-href and ng-required

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>Login Form</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js">
</script>
</head>
<body>

<div ng-controller="loginController">
  <h2>Login Form</h2>

  <form ng-submit="login()" novalidate>
    <div>
      <label>Username:</label>
      <input type="text" ng-model="username" ng-required="true">
    </div>

    <div>
      <label>Password:</label>
      <input type="password" ng-model="password" ng-required="true">
    </div>

    <div>
      <button type="submit" ng-disabled="loginForm.$invalid">Login</button>
    </div>
  </form>

  <div ng-if="showSuccessMessage">
    <p>You have logged in successfully!</p>
    <p>Redirecting to the next page...</p>
  </div>
</div>

<script>
angular.module('myApp', [])
  .controller('loginController', function($scope, $window) {
    $scope.login = function() {
      // Perform login validation
      if ($scope.username && $scope.password) {
        // Redirect to the next page on successful login
        $scope.showSuccessMessage = true;
        $window.setTimeout(function() {
          $window.location.href = 'nextpage.html';
        }, 3000);
      }
    };
  });
</script>
```

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

In the example above, we define an AngularJS module called myApp and a controller called loginController. The controller handles the login logic and redirection.

In the HTML, we use the ng-submit directive to bind the login() function to the form's submit event. The novalidate attribute is added to disable HTML5 form validation.

The username and password fields are required, indicated by the ng-required="true" attribute. This ensures that the user must fill in both fields before submitting the form.

The submit button is initially disabled using the ng-disabled directive, which checks if the form is invalid (loginForm.$invalid).

When the user clicks the Login button and the form is valid, the login() function is called. In this example, it performs a simple validation by checking if both the username and password are filled. If they are, it sets the $scope.showSuccessMessage variable to true, displaying the success message.

After a delay of 3 seconds, the $window.location.href is set to 'nextpage.html', redirecting the user to the next web page.

Please make sure to replace 'nextpage.html' with the actual URL or filename of the desired next page.

Q 7) Expain routing in anguler js with suitable and simple example

Routing in AngularJS allows you to navigate between different views or pages within a single-page application (SPA) without a full-page refresh. It enables you to define different routes for different views and map them to corresponding controllers and templates.

Here's a simple example to explain routing in AngularJS:

1. Include the AngularJS library in your HTML file:

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular-route.min.js"></script>
```

2. Define your AngularJS module and configure routing:

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

app.config(function($routeProvider) {
 $routeProvider
  .when('/', {
   templateUrl: 'home.html',
   controller: 'HomeController'
    })
    .when('/about', {
     templateUrl: 'about.html',
          controller: 'AboutController'
        })
        .when('/contact', {
          templateUrl: 'contact.html',
          controller: 'ContactController'
        })
        .otherwise({
          redirectTo: '/'
        });
});
```

In the code above, we have defined an AngularJS module named 'myApp' and injected the 'ngRoute' module as a dependency.

Inside the module's configuration function, we use the $routeProvider service to define different routes and their corresponding templates and controllers.

For example, the route '/' is associated with the home view, defined in 'home.html', and the 'HomeController' is the controller for this view.

Create the HTML templates and controllers:

Create 'home.html', 'about.html', and 'contact.html' files, and define their respective HTML templates.

For example, 'home.html':

```
<h2>Welcome to the Home Page</h2>
<p>This is the home page content.</p>
```

Create controllers for each view:

```
app.controller('HomeController', function($scope) {
  // Controller logic for the home view
});

app.controller('AboutController', function($scope) {
  // Controller logic for the about view
});

app.controller('ContactController', function($scope) {
  // Controller logic for the contact view
});
```

4. Add the `ng-view` directive to the main HTML file:

```
<div ng-view></div>
```

This is where the content of the different views will be dynamically inserted based on the current route.

5. Define navigation links:

```
<ul>
```

```
    <li><a href="#/">Home</a></li>
    <li><a href="#/about">About</a></li>
    <li><a href="#/contact">Contact</a></li>
</ul>
```

In the navigation links, the href attributes are set to the corresponding routes defined in the $routeProvider.

When a link is clicked, AngularJS will update the URL fragment (#/) and dynamically load the associated template and controller specified in the route configuration.

By following this example, you can create a basic routing system in AngularJS, allowing navigation between different views based on the defined routes.

Note: This example uses AngularJS version 1.8.2 and the ngRoute module. Make sure to include the correct version of AngularJS and the angular-route.min.js file.

Q 8) write a program to show current date and time using user defined module in node.js

1. Create a new file named `datetime.js` and add the following code:

```
exports.getCurrentDateTime = function() {

  const currentDate = new Date();
  return currentDate.toLocaleString();
};
```

In this module, we export a function called getCurrentDateTime using the exports object. Inside the function, we create a new Date object and use the toLocaleString method to convert it into a localized date and time string.

2. Create another file named `app.js` and add the following code:

```
const datetime = require('./datetime');

const currentDateTime = datetime.getCurrentDateTime();
console.log('Current Date and Time:', currentDateTime);
```

In this file, we import the `getCurrentDateTime` function from the `datetime.js` module using the `require` function. We assign the imported function to a variable named `datetime`.

We then call the `getCurrentDateTime` function to get the current date and time, and store the result in the `currentDateTime` variable.

Finally, we log the current date and time to the console

Q 9) explain video and svg tag in html5 with suitable example

In HTML5, the `<video>` and `<svg>` tags are used for displaying video content and scalable vector graphics, respectively. Let's explore each tag and provide suitable examples for better understanding.

`<video>` Tag: The `<video>` tag is used to embed video content in an HTML document. It supports various video formats such as MP4, WebM, and Ogg.

Here's an example of how to use the `<video>` tag:

```
<video src="video.mp4" controls>
  Your browser does not support the video tag.
</video>
```

In the example above, the `src` attribute specifies the URL or file path of the video file. The `controls` attribute adds playback controls (play, pause, volume, etc.) to the video player. If the browser does not support the `<video>` tag or the specified video format, the fallback content within the `<video>` tags will be displayed.

2. `<svg>` Tag: The `<svg>` tag is used to embed scalable vector graphics in an HTML document. SVG is an XML-based format for describing two-dimensional vector graphics. It allows you to create shapes, paths, text, and more.

Here's an example of how to use the `<svg>` tag:

```
<svg width="200" height="200">
  <circle cx="100" cy="100" r="80" fill="blue" />
  <text x="50%" y="50%" text-anchor="middle" fill="white" font-
size="20px">Hello SVG</text>
</svg>
```

In the example above, we create an SVG element with a width and height of 200 pixels. Inside the SVG element, we define a circle using the `<circle>` tag, specifying its center (`cx`, `cy`) and radius (`r`), as well as its fill color (`fill`).

We also add a text element using the `<text>` tag, positioning it at the middle of the SVG using `x="50%"` and `y="50%"`. The `text-anchor` attribute is set to `"middle"` to horizontally center the text. We also specify the text color (`fill`) and font size (`font-size`).

When the HTML page is rendered, the SVG code will be interpreted by the browser, resulting in the display of a blue circle with the text "Hello SVG" centered inside it.

Both the `<video>` and `<svg>` tags provide powerful capabilities for displaying multimedia content and vector graphics, respectively, enhancing the visual experience on web pages.

Q 10) write a php script to demonstrate multidiamensional arrays

Certainly! Here's an example of a PHP script that demonstrates multidimensional arrays:

```php
<?php
// Creating a multidimensional array
$students = array(
    array("John", 18, "Male"),
    array("Sarah", 20, "Female"),
    array("Michael", 19, "Male"),
    array("Emily", 21, "Female")
);

// Accessing elements in a multidimensional array

echo "Name: " . $students[0][0] . ", Age: " . $students[0][1] . ", Gender: " . $students[0][2] . "<br>";
echo "Name: " . $students[1][0] . ", Age: " . $students[1][1] . ", Gender: " . $students[1][2] . "<br>";
echo "Name: " . $students[2][0] . ", Age: " . $students[2][1] . ", Gender: " . $students[2][2] . "<br>";
echo "Name: " . $students[3][0] . ", Age: " . $students[3][1] . ", Gender: " . $students[3][2] . "<br>";

// Adding elements to a multidimensional array
$students[] = array("David", 22, "Male");

// Accessing the newly added element

echo "Name: " . $students[4][0] . ", Age: " . $students[4][1] . ", Gender: " . $students[4][2] . "<br>";

// Iterating through a multidimensional array
foreach ($students as $student) {
    echo "Name: " . $student[0] . ", Age: " . $student[1] . ", Gender: " . $student[2] . "<br>";
}
?>
```

Q 11) what is cookie and session in php with example simple

In PHP, cookies and sessions are used to manage and store data related to a user's interaction with a website. They both serve the purpose of maintaining state and preserving data across multiple page requests. However, they differ in how the data is stored and accessed. Let's explain cookies and sessions with simple examples:

1. Cookies: Cookies are small text files stored on the user's computer by the web browser. They allow the server to store data that can be accessed and manipulated by the server and client.

Example:

```
// Setting a cookie
setcookie("username", "John Doe", time() + 3600, "/");

// Accessing a cookie
if (isset($_COOKIE["username"])) {
echo "Welcome back, " . $_COOKIE["username"];
} else {
echo "Welcome, guest";
}
```

In the example above, the setcookie() function is used to set a cookie named "username" with the value "John Doe". The cookie will expire in 1 hour (time() + 3600), and the cookie is accessible in the root directory ("/") of the website.

To access the cookie, we use the $_COOKIE superglobal array. We check if the "username" cookie is set using isset(), and if it is, we display a personalized welcome message with the cookie value. Otherwise, we display a generic welcome message for a guest.

2. Sessions: Sessions are a server-side mechanism for storing and managing user-specific data. A session assigns a unique identifier (session ID) to each user, and the associated data is stored on the server.

Example:

```
// Starting a session
session_start();

// Storing data in a session
$_SESSION["username"] = "John Doe";

// Accessing session data
if (isset($_SESSION["username"])) {
    echo "Welcome back, " . $_SESSION["username"];
} else {
    echo "Welcome, guest";
}
```

In the example above, we start a session using session_start(), which initializes or resumes an existing session. We store the value "John Doe" in the session variable $_SESSION["username"].

To access the session data, we use the $_SESSION superglobal array. We check if the "username" session variable is set, and if it is, we display a personalized welcome message with the session value. Otherwise, we display a generic welcome message for a guest.

Sessions are typically more secure than cookies since the data is stored on the server and the session ID is sent to the client as a cookie. The client's browser then sends the session ID back to the server with each request, allowing the server to retrieve the corresponding session data.

Both cookies and sessions have their use cases and advantages. Cookies are suitable for storing small amounts of data on the client side, while sessions are more appropriate for storing larger amounts of data securely on the server side.

Q 13) what is html 5 explian is fetures and advntages

HTML5 is the fifth revision of the Hypertext Markup Language (HTML), which is the standard markup language for creating web pages and applications. HTML5 introduces several new features and enhancements compared to its predecessor, HTML4. Here are some key features and advantages of HTML5:

1. Semantics: HTML5 introduces new semantic elements that provide a more meaningful structure to web content. These elements include `<header>`, `<nav>`, `<section>`, `<article>`, `<footer>`, and more. Using these semantic elements improves accessibility, search engine optimization, and the overall understanding of the page structure.
2. Multimedia Support: HTML5 includes native support for audio and video playback without the need for third-party plugins like Flash. The `<audio>` and `<video>` tags allow embedding media files directly into web pages. This simplifies multimedia integration and enhances cross-platform compatibility.
3. Canvas: The `<canvas>` element provides a powerful and flexible drawing surface for 2D graphics and animations. It allows developers to programmatically draw shapes, images, and apply transformations using JavaScript. The canvas element is widely used for creating games, data visualizations, and interactive graphics.
4. Geolocation: HTML5 introduces the Geolocation API, which allows websites to access the user's geographical location. This feature enables location-based services, such as finding nearby stores, mapping applications, and personalized content based on the user's location.
5. Offline Web Applications: HTML5 includes the Application Cache and Local Storage APIs, which enable web applications to work offline or in low-connectivity environments. Developers can

cache application files and data on the user's device, allowing the application to run even without an internet connection.

6. Improved Forms: HTML5 introduces several new form input types, such as date, time, email, number, and range, making it easier to capture and validate user input. Additionally, HTML5 provides native form validation, reducing the need for custom JavaScript validation.
7. Web Storage: HTML5 provides the Web Storage API, which allows web applications to store data locally on the user's device. The localStorage and sessionStorage objects enable persistent storage of key-value pairs, providing a more efficient and secure alternative to using cookies.
8. Enhanced Accessibility: HTML5 includes new accessibility features, such as the `<mark>` element for highlighting text, the `aria-*` attributes for defining accessible roles and properties, and the `<figure>` and `<figcaption>` elements for better image and caption associations. These enhancements help developers create more accessible web content.
9. Mobile-Friendly: HTML5 is designed with mobile devices in mind. It supports responsive design techniques, allowing developers to create websites that adapt and adjust their layout based on the screen size and orientation. The mobile-friendly nature of HTML5 helps deliver a consistent user experience across different devices.
10. Backward Compatibility: HTML5 is backward compatible with older versions of HTML, ensuring that existing HTML documents will still render correctly in modern browsers. This makes it easier for developers to transition to HTML5 gradually without breaking existing web content.

Overall, HTML5 brings a wide range of features and advantages that improve the capabilities and user experience of web applications. It offers better multimedia support, enhanced semantics, improved offline functionality, and increased accessibility, making it an essential technology for modern web development.

Q 14) wrtie a php script to create a databasse using mysql

To create a database using MySQL in PHP, you can use the MySQLi extension, which provides an object-oriented interface for interacting with the MySQL database. Here's an example script that demonstrates how to create a database:

```php
<?php
// Database connection details
$servername = "localhost";
$username = "your_username";
$password = "your_password";

// Create a connection
$conn = new mysqli($servername, $username, $password);

// Check the connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Create a new database
$databaseName = "mydatabase";
$sql = "CREATE DATABASE $databaseName";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn->error;
}

// Close the connection
$conn->close();
?>
```

Q 15) what is define function in php

In PHP, the `define()` function is used to define a constant. A constant is a value that cannot be changed during the execution of a script. Once defined, its value remains the same throughout the script's execution. Constants are useful for storing values that should not be modified, such as configuration settings or fixed values.

The syntax of the `define()` function is as follows:

```
define(name, value, case_insensitive);
```

- `name`: Specifies the name of the constant.
- `value`: Specifies the value assigned to the constant.
- `case_insensitive` (optional): Specifies whether the constant name is case-insensitive. By default, it is set to `false`, meaning the constant name is case-sensitive.

Here's an example that demonstrates the usage of the `define()` function:

```
define("PI", 3.14159);
define("GREETING", "Hello, World!", true); // Case-insensitive constant

echo PI; // Output: 3.14159
echo GREETING; // Output: Hello, World!echo greeting; // Output: Hello,
World! (case-insensitive)
```

Once defined, we can access the constant using its name (without the dollar sign `$`). Constants are not preceded by the dollar sign as variables are.It's important to note that constants are globally accessible within the script, meaning they can be accessed from any part of the code, regardless of the scope. Additionally, constants cannot be redefined or unset once they are defined.Using constants helps make your code more readable, maintainable, and avoids accidental changes to important values during the execution of your PHP scripts.

Q 16) what is pseudo classes in css explain with suitable example

In CSS, pseudo-classes are used to select and style elements based on their state or position within the document tree. Pseudo-classes are preceded by a colon (`:`) and allow you to target elements that cannot be selected with regular selectors alone. They provide additional control over the styling and behavior of elements.

Here are a few examples of commonly used pseudo-classes:

1. **`:hover`**: This pseudo-class is used to select an element when the user hovers over it with the mouse pointer. It is often used to apply special styling or effects to elements on mouseover.

   cssCopy code

   ```
   :hover   color
   ```

   In the example above, the anchor (`<a>`) elements will turn red when the mouse is hovered over them.

2. **`:focus`**: This pseudo-class is used to select an element that currently has focus, typically through user interaction, such as clicking or tabbing onto the element. It is commonly used to style form elements when they are active or selected.

   cssCopy code

   ```
         :focus   border-color
   ```

   In the example above, when an input element receives focus, such as when clicking on it, it will have a blue border.

3. **`:nth-child()`**: This pseudo-class is used to select elements based on their position within a parent container. It allows you to target specific child elements using a mathematical formula. For example, `:nth-child(2n)` selects every even child element.

   cssCopy code

   ```
     :nth-child          background-color
   ```

   In the example above, every odd `<li>` element within a `<ul>` will have a light gray background color.

4. **`:first-child`** and **`:last-child`**: These pseudo-classes target the first and last child elements of a parent container, respectively.

   cssCopy code

   ```
     :first          font-weight            :last-child    text-decoration
   ```

   In the example above, the first `<li>` element within a `<ul>` will have bold font weight, while the last `<li>` element will have an underline text decoration.

These are just a few examples of the many pseudo-classes available in CSS. Pseudo-classes allow you to apply specific styles to elements based on various conditions, states, and positions, adding flexibility and interactivity to your CSS styling.

Q 17) write a php script to design form for exam registration. insert 5 recoreds in database and dispaly all the inseted all the inserted records on new page

Certainly! Here's a PHP script that demonstrates how to design a form for exam registration, insert the records into a database, and display all the inserted records on a new page:

exam_registration_form.php:

```
<!DOCTYPE html>
<html>
<head>
    <title>Exam Registration Form</title>
</head>
<body>
    <h2>Exam Registration Form</h2>
    <form action="insert_records.php" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required><br><br>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br><br>

        <label for="exam">Exam:</label>
        <input type="text" id="exam" name="exam" required><br><br>

        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

insert_records.php:

```
<?php
// Database connection details
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database_name";

// Create a connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check the connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Get form data
```

```
$name = $_POST['name'];
$email = $_POST['email'];
$exam = $_POST['exam'];

// Insert record into the database
$sql = "INSERT INTO exam_records (name, email, exam) VALUES
('$name', '$email', '$exam')";
if ($conn->query($sql) === TRUE) {
    echo "Record inserted successfully";
} else {
    echo "Error inserting record: " . $conn->error;
}

// Close the connection
$conn->close();
?>
```

view_records.php:

```
<?php
// Database connection details
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database_name";

// Create a connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check the connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Retrieve all records from the database
$sql = "SELECT * FROM exam_records";
$result = $conn->query($sql);

// Display the records
if ($result->num_rows > 0) {
    echo "<h2>Registered Exam Records</h2>";
    while ($row = $result->fetch_assoc()) {
        echo "Name: " . $row['name'] . "<br>";
        echo "Email: " . $row['email'] . "<br>";
        echo "Exam: " . $row['exam'] . "<br><br>";
    }
} else {
    echo "No records found";
}

// Close the connection
$conn->close();
?>
```

Q 18) what is objective of css architecture ?

The objective of CSS architecture is to establish a structured and maintainable approach for organizing CSS code in large-scale projects. It aims to improve code readability, reusability, modularity, and collaboration among developers working on the same project.

CSS architecture provides guidelines and best practices to manage the complexity of styling a website or application by breaking down the CSS code into smaller, more manageable components. It focuses on creating a scalable and maintainable system that allows for easier updates, reduces code duplication, and enhances code organization.

Key objectives of CSS architecture include:

1. **Modularity:** Breaking down CSS code into small, self-contained modules or components allows for easier management and reuse. Each module focuses on a specific aspect of the design, such as buttons, typography, or navigation, making it easier to understand and maintain.
2. **Scalability:** CSS architecture provides a scalable approach that accommodates the growth and evolution of a project. It ensures that as the project expands and new features are added, the CSS code remains organized and maintainable.
3. **Consistency:** By following a predefined CSS architecture, developers can ensure consistency in naming conventions, coding styles, and code structure. This consistency improves collaboration and makes it easier for developers to work on different parts of the project.
4. **Maintainability:** CSS architecture aims to make the codebase more maintainable by reducing code duplication, improving code organization, and providing clear separation of concerns. It allows for easier updates and modifications without introducing unexpected side effects.
5. **Performance:** Properly organized CSS code can improve website performance by reducing file size, optimizing selectors, and minimizing redundant or unused styles. CSS architecture helps in identifying and eliminating unused styles, resulting in faster page load times.
6. **Code Reusability:** CSS architecture encourages the creation of reusable CSS components and utility classes. This reusability reduces the need to write new CSS code for similar design elements, promoting code efficiency and consistency.
7. **Ease of Collaboration:** CSS architecture provides a common structure and set of guidelines that facilitate collaboration among developers working on the same project. It promotes better communication and understanding of the codebase, making it easier to work together effectively.

By following a well-defined CSS architecture, developers can create a robust and maintainable CSS codebase that enhances productivity, code quality, and collaboration, ultimately leading to better-designed and more maintainable web projects.

Q 19) write a php script to design a form to reschedule the journey dates of filghts and display the updated information on new web page

Certainly! Here's a PHP script that demonstrates how to design a form to reschedule the journey dates of flights and display the updated information on a new web page:

reschedule_form.php:

```
<!DOCTYPE html>
<html>
<head>
    <title>Flight Journey Rescheduling</title>
</head>
<body>
    <h2>Flight Journey Rescheduling Form</h2>
    <form action="update_flight.php" method="POST">
        <label for="flight_number">Flight Number:</label>
        <input type="text" id="flight_number" name="flight_number"
required><br><br>

        <label for="new_date">New Journey Date:</label>
        <input type="date" id="new_date" name="new_date"
required><br><br>

        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

update_flight.php:

```
<?php
// Database connection details
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database_name";

// Create a connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check the connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Get form data
$flightNumber = $_POST['flight_number'];
$newDate = $_POST['new_date'];

// Update flight information
$sql = "UPDATE flights SET journey_date = '$newDate' WHERE
flight_number = '$flightNumber'";
if ($conn->query($sql) === TRUE) {
    // Redirect to the updated flight information page
    header("Location: flight_details.php?flight_number=$flightNumber");
    exit();
} else {
    echo "Error updating flight information: " . $conn->error;
}

// Close the connection
$conn->close();
?>
```

flight_details.php:

```php
<?php
// Database connection details
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database_name";

// Create a connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check the connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Get the flight number from the URL parameter
$flightNumber = $_GET['flight_number'];

// Retrieve the updated flight information
$sql = "SELECT * FROM flights WHERE flight_number =
'$flightNumber'";
$result = $conn->query($sql);

// Display the updated flight information
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "<h2>Updated Flight Information</h2>";
        echo "Flight Number: " . $row['flight_number'] . "<br>";
        echo "Journey Date: " . $row['journey_date'] . "<br>";
        // Display other flight details as needed
    }
} else {
    echo "Flight information not found";
}

// Close the connection
$conn->close();
?>
```