# PHP

### Concatenation

```php
<?php
$name = "Aman";
echo "Name  is " . $name . "<br>";
echo "Name  is $name.";
echo "<h2>PHP is Fun!</h2>";
echo "This ", "string ", "multiple parameters.";
?>
```

`var_dump($x)` - returns data type and size, ex `var_dump("Hi") :: string(2)`

### Strings

### Concatenate

```php
$x = "Hello";
$y = "World";
$z = $x . $y;
echo $z;
```

### Replace string

```php
$x = "Hello World!";
echo str_replace("World", "Dolly", $x);
```

### Remove whitespace

```php
$x = " Hello World! ";
echo trim($x);
```

### Str to array

```php
$x = "Hello World!";
$y = explode(" ", $x);

//Use the print_r() function to display the result:
print_r($y);
```

```
/*
Result:
Array ( [0] => Hello [1] => World! )
*/
```

### Slicing Strings

```
$x = "Hello World!";
echo substr($x, 6, 5);
```

---

### For loop

```
for ($x = 0; $x <= 10; $x++) {
  echo "The number is: $x <br>";
}

for ($x = 0; $x <= 10; $x++) {
  if ($x == 3) continue;
  echo "The number is: $x <br>";
}
```

### Foreach loop

```
$colors = array("red", "green", "blue", "yellow");
// For every loop iteration, the value of the current
// array element is assigned to the variabe
//$x. The iteration continues until
// it reaches the last array element.
foreach ($colors as $x) {
  echo "$x <br>";
}


// Can do the same with key and values
$members = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach ($members as $x => $y) {
  echo "$x : $y <br>";
}


// Either syntax is acceptable
$y = [
    "val" => 1
];

$y = array("val" => 1);
```

### For each by ref

```

/*
Oringal changes are made to array, now that its done by refernce. Vefoire it wasnt
changed.
*/
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as &$x) {
  if ($x == "blue") $x = "pink";
}

var_dump($colors);
```

---

### Function syntax

```

function familyName($fname, $year) {
  echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");

// returning a value
function sum($x, $y) {
  $z = $x + $y;
  return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";


// pass by refernce
function add_five(&$value) {
  $value += 5;
}

$num = 2;
add_five($num);
echo $num;
```

#### function, variable args

```
// variadic variable is last
function myFamily($lastname, ...$firstname) {
  txt = "";
  $len = count($firstname);
  for($i = 0; $i < $len; $i++) {
    $txt = $txt."Hi, $firstname[$i] $lastname.<br>";
  }
  return $txt;
}

$a = myFamily("Doe", "Jane", "John", "Joey");
echo $a;
```

#### Updating arrays

```
$cars = array("Volvo", "BMW", "Toyota");
$cars[1] = "Ford";

$cars = array("brand" => "Ford", "model" => "Mustang", "year" => 1964);
$cars["year"] = 2024;

// Update each item in array.
// must unset x so that last item
$cars = array("Volvo", "BMW", "Toyota");
foreach ($cars as &$x) {
  $x = "Ford";
}
unset($x);
var_dump($cars);
```

#### Adding array items

```
$fruits = array("Apple", "Banana", "Cherry");
$fruits[] = "Orange";

$cars = array("brand" => "Ford", "model" => "Mustang");
$cars["color"] = "Red";

// adds three items
$fruits = array("Apple", "Banana", "Cherry");
array_push($fruits, "Orange", "Kiwi", "Lemon");
```

```
// adds multiple items
$cars = array("brand" => "Ford", "model" => "Mustang");
$cars += ["color" => "red", "year" => 1964];
```

#### Removing array items

```


$cars = array("Volvo", "BMW", "Toyota");
unset($cars[1]);

$cars = array("Volvo", "BMW", "Toyota");
array_splice($cars, 1, 2);


$cars = array("Volvo", "BMW", "Toyota");
unset($cars[0], $cars[1]);

$cars = array("brand" => "Ford", "model" => "Mustang", "year" => 1964);
unset($cars["model"]);
```

#### Sorting an array

```
// alphabetticall
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);

// numerical
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);

// sort ascending based on value
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);

// sort assending based on key
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
```

#### Super GLobals

In other words, $_REQUEST is an array containing data from $_GET, $_POST, and $_COOKIE.

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = htmlspecialchars($_REQUEST['fname']);
  if (empty($name)) {
    echo "Name is empty";
  } else {
    echo $name;
  }
}
?>

</body>
</html>
```

**Ways to fetch data from php**

### Two Approaches to Handling a GET Request:

1. **Using a Simple HTML Form with PHP:**

   - When you submit a form using the standard `GET` or `POST` method, the page reloads, and the data is sent to the server.
   - This triggers the PHP script to handle the request, and the browser will load a new page (or the same page) with the server's response.

   **Example of a Basic HTML Form (GET request):**

   ```
   <form action="process.php" method="GET">
       <input type="text" name="query" placeholder="Enter your search">
       <input type="submit" value="Submit">
   </form>
   ```

   - When the user submits the form, the browser navigates to `process.php` and appends the form data as a query string (e.g., `process.php?query=value`).
   - The browser **reloads** the page to show the response from the server.

2. **Using JavaScript to Make an AJAX GET Request to PHP (Without Page Reload):**

- Instead of submitting the form directly and causing a page reload, you can handle the form submission with JavaScript (specifically, using **AJAX**).
- This allows you to make a **GET request** to a PHP script and retrieve data from the server **without refreshing the page**.

**Example: Using JavaScript to Send a GET Request to PHP:**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX GET Request Example</title>
    <script>
        function fetchData() {
            // Get the input value
            var query = document.getElementById('queryInput').value;

            // Create a new XMLHttpRequest object
            var xhr = new XMLHttpRequest();

            // Define what happens when the request is successful
            xhr.onreadystatechange = function() {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    // Update the content of the result div with the response from PHP
                    document.getElementById('result').innerHTML = xhr.responseText;
                }
            };

            // Open a GET request to the PHP script
            xhr.open('GET', 'process.php?query=' + encodeURIComponent(query), true);

            // Send the GET request
            xhr.send();
        }
    </script>
</head>
<body>
    <form onsubmit="event.preventDefault(); fetchData();">
        <input type="text" id="queryInput" placeholder="Enter your search">
        <input type="button" value="Submit" onclick="fetchData()">
    </form>

    <!-- Div to display the result -->
    <div id="result"></div>
</body>
</html>
```

- **Explanation:**
    - The `onsubmit="event.preventDefault(); fetchData();"` prevents the form from submitting normally and reloading the page.
    - `fetchData()` is a JavaScript function that creates an `XMLHttpRequest` object to send a **GET request** to the `process.php` script.
    - The query string is dynamically appended to the URL (`process.php?query=yourValue`), just like it would be in a form submission.
    - The server response is handled within the `onreadystatechange` function, where the response (from the PHP script) is displayed in the `<div id="result"></div>` element.

### Example of PHP (`process.php`) to Handle the AJAX GET Request:

```php
<?php
// Check if the 'query' parameter is set in the GET request
if (isset($_GET['query'])) {
    $query = htmlspecialchars($_GET['query']);  // Sanitize the input for security

    // Perform some operation based on the query (e.g., search a database, etc.)
    echo "You searched for: " . $query;
} else {
    echo "No query parameter provided.";
}
?>
```

### Key Differences Between These Approaches:

- **HTML Form Submission (Normal GET Request):**
    - Causes a full page reload.
    - Data is sent to the server via the URL query string (with `method="GET"`).
    - The browser navigates to a new page (or the same page) to display the server's response.
- **JavaScript AJAX Request (GET Request Using JavaScript):**
    - Does **not reload** the page.
    - Allows you to send a GET request to the server and dynamically retrieve information without leaving the current page.
    - The server's response can be updated in a specific section of the page (using `innerHTML`, for example).
    - Provides a more interactive, seamless user experience.

### Summary:

- If you submit a form with a normal HTML form submission, the browser will **reload** the page.
- If you use **JavaScript (AJAX)** to send a GET request to the PHP script, you can retrieve data from the server **without reloading the page**. This is useful for building interactive web applications.

Both methods work with PHP, but AJAX is more flexible and provides a smoother user experience when you want to dynamically update content without a full page reload.

---

**Sessions**

Should start a session at your index php:It’s best practice to **start the session at the very beginning** of your application, such as in your `index.php` or any other initial script that the user first visits. This ensures that session data is available throughout the user’s interaction with your website, no matter which page or PHP script they access.

### Here's why you should start the session in `index.php`:

1. **Consistency**: Starting the session at the very beginning (such as in `index.php`) ensures that session data is initialized and available for any subsequent PHP script, even if those scripts don’t directly handle session data.
2. **Access Across the Application**: Once a session is started, session variables can be accessed or modified from any PHP script, so initializing it early ensures the session is ready throughout the entire application.
3. **Prevents "Headers Already Sent" Errors**: If you wait until later in the application (like in a backend script), and output (HTML, headers, etc.) is already sent to the browser, trying to start a session later may result in errors. This happens because PHP sessions send headers to the browser, which must be done before any output is sent.

### Best Practice: Start Session at the Beginning

- **`index.php`** is often the entry point for most web applications, and it’s a common place to initialize things like sessions, configuration settings, and database connections.
- You should call `session_start()` at the top of your main PHP entry point file (`index.php` or similar), ensuring the session starts before any output is sent to the browser.

### Example: Starting a Session in `index.php`

```php
<?php
session_start();  // Start the session as soon as the user accesses the site
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My PHP Application</title>
</head>
<body>
    <h1>Welcome to My Application</h1>
    <?php
    // Initialize session variables if needed
    if (!isset($_SESSION['username'])) {
        $_SESSION['username'] = 'Guest';  // Default value
    }

    echo "Hello, " . $_SESSION['username'];
    ?>
```

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

### What if Multiple PHP Scripts Are Called?

If you have multiple PHP scripts (e.g., `backend.php`, `process.php`, etc.), you will need to start the session on every page where you want to access or modify session variables. **Every PHP script that interacts with session data must have `session_start()`** at the beginning of the script.

### Example: A Backend Script

For example, if `process.php` or any other PHP script is accessed via an AJAX request or a form submission and it needs to access session data, it also needs `session_start()`:

```php
<?php
session_start();  // Start the session in every script where you want to access
session data

// Check or modify session variables
if (isset($_SESSION['username'])) {
    echo "Hello, " . $_SESSION['username'];
} else {
    echo "Session not set!";
}
?>
```

### Summary:

- **Start the session in `index.php` or any initial script** that the user first reaches. This ensures session handling is initialized early in the application.
- **Include `session_start()` at the top of any other PHP script** that interacts with session variables.
- This ensures that session variables are available across all pages without encountering "headers already sent" errors or losing session data.

**Response and REponseTExt**

```javascript
var xhr = new XMLHttpRequest();
xhr.open('GET', 'process.php', true);
xhr.responseType = 'json';  // Set responseType to 'json'

xhr.onload = function() {
    if (xhr.status == 200) {
        console.log(xhr.response);  // No need to parse, already a JS object
    }
};

xhr.send();
```

Example: When to Use responseText

javascript

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'process.php', true);

xhr.onload = function() {
    if (xhr.status == 200) {
        console.log(xhr.responseText);  // Raw JSON string
        var data = JSON.parse(xhr.responseText);  // Manually parse the JSON
        console.log(data);
    }
};

xhr.send();
```

# Homework 1

```


function getQueryString() {
   var formattedTermValue =
document.getElementById("term").value.trim().toLowerCase().replace(/[\s,]+/g,
'+');
   var formattedCityValue =
document.getElementById("city").value.trim().toLowerCase().replace(/[\s,]+/g,
'+');
   var levelLimitValue = document.getElementById("level").value;
   var queryString = "proxy.php?term=" + formattedTermValue + "&location=" +
formattedCityValue + "&limit=" + levelLimitValue;
   return queryString
}

function findRestaurants () {
    var xhr = new XMLHttpRequest();
    const qString = getQueryString();

    // Call yelp api
    xhr.open("GET", qString);
    xhr.setRequestHeader("Accept","application/json");
    xhr.onreadystatechange = function () {
        if (this.readyState == 4) {
            var json = JSON.parse(this.responseText);

            if (json.businesses && json.businesses.length > 0) {
                console.log(json.businesses);
                displayResults(json.businesses);
            }
            else{
                let outputDiv = document.getElementById('output');
                outputDiv.innerHTML = '<p>No businesses found for the given query.
</p>';
            }
        }
    };
    xhr.send(null);
}

async function displayResults(businesses) {
   // Clear previous results
   let outputDiv = document.getElementById('output');
   outputDiv.innerHTML = '';

    // Create ordered list
    const orderedList = document.createElement('ol');
    orderedList.style.paddingLeft = '20px';

    // Sort by rating (best business is 1)
    businesses.sort((a, b) => b.rating - a.rating);

    // Limit results to top 10 or less if less than 10 results
    const numResults = Math.min(businesses.length, 10);

    for (let i = 0; i < numResults; i++) {
        let business = businesses[i];

        // Create list item
        const listItem = document.createElement('li');
        listItem.style.marginBottom = '20px';

        // Create a card
        let card = document.createElement('div');
        card.style.border = "solid";
        card.style.padding = "15px";
        card.style.margin = "10px 50px 0px";
        card.style.boxShadow = "0 4px 8px 0 rgba(0, 0, 0, 0.1)";

        // Details container
        const details = document.createElement('div');
        details.style.flex = '1';

        // Add rank
        let rankingText = document.createElement('h2');
        rankingText.innerHTML = `Match: ${i+1}`;
        card.appendChild(rankingText);

        // Image
        let img = document.createElement('img');
        img.src = business.image_url;
        img.alt = `${business.name} image`;
        img.style.width = "100px";
        card.appendChild(img);

        // Name and Link
        let nameLink = document.createElement('a');
        nameLink.href = business.url;
        nameLink.target = "_blank";
        nameLink.innerHTML = business.name;
        nameLink.style.display = "block";
        card.appendChild(nameLink);

        // Categories
        let categories = business.categories.map(cat => cat.title).join(', ');
        let categoriesText = document.createElement('p');
        categoriesText.innerHTML = `Categories: ${categories}`;
        card.appendChild(categoriesText);

        // Price, Price sometimes isn't included for some reason
        if (business.price)
        {
            let priceText = document.createElement('p');
            priceText.innerHTML = `Price: ${business.price}`;
            card.appendChild(priceText);
        }

        // Rating
        let ratingText = document.createElement('p');
        ratingText.innerHTML = `Rating: ${business.rating} / 5`;
        card.appendChild(ratingText);

        // Address, formatted
        let addressText = document.createElement('p');
        addressText.innerHTML = `Address: ${business.location.address1},
${business.location.city}, ${business.location.zip_code}`;
        card.appendChild(addressText);

        // Phone number
        let phoneText = document.createElement('p');
        phoneText.innerHTML = `Phone: ${business.display_phone}`;
        card.appendChild(phoneText);

        // Append details to card
        card.appendChild(details);
        listItem.appendChild(card);
        orderedList.appendChild(listItem);
    }

    outputDiv.appendChild(orderedList);
}
```

# Homework 2

```
let map; // Google Map
let markers = []; // Google Map Markers
let infoWindow; // Popup Window
let currentSearchTerm = ''; // Search term in search bar at given moment

DEFAULT_LAT = 32.75
DEFAULT_LNG = -97.13
DEFAULT_ZOOM = 16

function initMap()
{
    map = new google.maps.Map
    (document.getElementById('map'),
        {
        center: { lat: DEFAULT_LAT, lng: DEFAULT_LNG },
        zoom: DEFAULT_ZOOM = 16
        }
    );

    // Initialize the popup information window
    infoWindow = new google.maps.InfoWindow();

    //  Update map, when it is moved or zoomed
    map.addListener
    ('idle', function()
        {
        findRestaurants();
        }
    );
}


/**
 * Ensures yelp only searches on the displayed map
 * @returns lat,long,radius
 */
function getMapBounds()
{
    const bounds = map.getBounds();
    const center = bounds.getCenter();
    const ne = bounds.getNorthEast();
    const radius = google.maps.geometry.spherical.computeDistanceBetween(center,
ne);
    const radiusInMeters = Math.min(Math.round(radius), 40000); // Round the
radius to the nearest integer

    var mapBounds =
    {
        latitude: center.lat(),
        longitude: center.lng(),
        radius: radiusInMeters
    };

    return mapBounds;
}


function getQueryString() {

    const searchTermInput =
document.getElementById("term").value.trim().toLowerCase();

    if (searchTermInput) {
        currentSearchTerm = searchTermInput;
    }

    if (!currentSearchTerm) {
        return '';
    }

    const formattedTermValue = currentSearchTerm.replace(/[\s,]+/g, '+');
    const bounds = getMapBounds();
    return `proxy.php?
term=${formattedTermValue}&latitude=${bounds.latitude}&longitude=${bounds.longitud
e}&radius=${bounds.radius}&limit=10`;
}


function findRestaurants(){
    const qString = getQueryString();
    if (!qString) {
        console.log('No search term, skipping search.');
        return;
    }

    clearMarkers();

    // Call Yelp API
    const xhr = new XMLHttpRequest();
    xhr.open("GET", qString);
    xhr.setRequestHeader("Accept", "application/json");
    xhr.onreadystatechange = function () {
        if (this.readyState == 4) {
            const json = JSON.parse(this.responseText);
            console.log(json);
            if (json.businesses && json.businesses.length > 0) {
                displayResults(json.businesses);
            }
            else {
                console.log('No businesses found');
            }
        }
    };
    xhr.send(null);
}
```

```
function clearMarkers() {
    for (let marker of markers) {
        marker.setMap(null);
    }
    markers = [];
}

function displayResults(businesses) {
    let outputDiv = document.getElementById('output');
    outputDiv.innerHTML = '';

    businesses.sort((a, b) => b.rating - a.rating);

    for (let i = 0; i < businesses.length; i++) {
        // Create new marker
        let business = businesses[i];
        let marker = new google.maps.Marker
        (
            {
                position:
                {
                    lat: business.coordinates.latitude,
                    lng: business.coordinates.longitude
                },
                label: `${i + 1}`,
                map: map
            }
        );

        // Event listener to the marker to display an info window when marker is
clicked
        google.maps.event.addListener
        (marker, 'click', function()
            {
                infoWindow.setContent(`
                    <div>
                    <h2>${business.name}</h2>
                    <img src="${business.image_url}" style="width:100px;"
alt="${business.name}">
                    <p>Rating: ${business.rating} / 5</p>
                    <p>${business.location.address1}, ${business.location.city}
</p>
                    </div>
                `);
                infoWindow.open(map, marker);
            }
        );
        markers.push(marker);
    }
}
```

# javascaript

**GEt, POST, UPDATYE, and DELETE**

```
// Function to initialize any necessary components (if needed)
function initialize() {
    console.log("Initialization complete.");
}

// GET Request: Fetch data from the server
function getData() {
    var ajax = new XMLHttpRequest();
    var searchTermValue = document.getElementById('search_phrase').value;
    var queryString = "process.php?phrase=" + encodeURIComponent(searchTermValue);

    ajax.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            console.log('GET request sent successfully!');
            console.log(`Response = ${this.response}`);
        }
    };

    // Open the GET request
    ajax.open("GET", queryString, true);

    // Send the GET request (no body needed for GET)
    ajax.send(null);
}


// POST Request: Insert data into the server (e.g., adding a new record)
function postData() {
    var ajax = new XMLHttpRequest();
    var username = document.getElementById('post_data').value;
    var url = "process.php";

    // Create the POST data (URL-encoded)
    var postData = "username=" + encodeURIComponent(username);

    ajax.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            console.log('POST request sent successfully!');
            console.log(`Response = ${this.response}`);
        }
    };

    // Open the POST request
    ajax.open("POST", url, true);

    // Set the content type to application/x-www-form-urlencoded
    ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

    // Send the POST data in the body
    ajax.send(postData);
}


// PUT/UPDATE Request: Update existing data on the server
function updateUser() {
    var ajax = new XMLHttpRequest();
    var username = document.getElementById('username').value;
    var newFirstName = document.getElementById('new_firstname').value;
    var url = "process.php";

    // Prepare the data to be sent (URL-encoded)
    var updateData = "username=" + encodeURIComponent(username) + "&firstname=" +
encodeURIComponent(newFirstName);

    ajax.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            console.log('PUT/UPDATE request sent successfully!');
            console.log(`Response = ${this.response}`);
        }
    };

    // Open the PUT/UPDATE request (note: we use POST in PHP as it doesn't
natively support PUT)
    ajax.open("POST", url, true);

    // Set the content type to application/x-www-form-urlencoded
    ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

    // Send the update data in the body
    ajax.send(updateData);
}


// DELETE Request: Remove data from the server
function deleteUser() {
    var ajax = new XMLHttpRequest();
    var username = document.getElementById('username').value;

    // Prepare the DELETE request with the data in the query string
    var url = "process.php?username=" + encodeURIComponent(username);

    ajax.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            console.log('DELETE request sent successfully!');
            console.log(`Response = ${this.response}`);
        }
    };

    // Open the DELETE request
    ajax.open("DELETE", url, true);
    // Send the DELETE request (usually no body is needed for DELETE)
    ajax.send(null);
}
```

# MySQL

#### Establishing db connection

```
session_start();

// Database credentials
$dbusername = "root";
$dbpassword = "";
$dbname = "mydb";
$conn = new mysqli("localhost", $dbusername, $dbpassword, $dbname);

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

#### Mysql connection ending

```
if ($sql->affected_rows > 0)
{
    echo "New user has been inserted successfully.";
}
else
{
    echo "Failed to insert the user.";
}
$conn->close();
```

#### Check if params are set

```
// Get requests
if ($_SERVER['REQUEST_METHOD'] === 'GET' && isset($_GET['<url_paramater>']) && ...
) {...}

// Post Requests
```

```
elseif ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['<post_data_name>'])
&& ...) { ... }

// Delete Requests
elseif ($_SERVER['REQUEST_METHOD'] === 'DELETE') {}
```

**Update Data**

```
$value1 = htmlspecialchars($_POST['<value>']);
$value2 = htmlspecialchars($_POST['<value>']);
$sql = $conn->prepare("UPDATE table SET value1 = ? WHERE value2 = ?");
// "ss" for two strings
$sql->bind_param("ss", $value1, $value2);
$sql->execute();
```

**Select Data**

```
// Handling GET request to retrieve user info
$username = htmlspecialchars($_GET['username']);

// Prepare SQL to retrieve user info
$sql = $conn->prepare("SELECT id, username, firstname, lastname FROM users WHERE
username = ?");
$sql->bind_param("s", $username); // "s" indicates the parameter is a string
$sql->execute();
$result = $sql->get_result();

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "User ID: " . $row['id'] . "<br>";
        echo "Username: " . $row['username'] . "<br>";
    }
} else {
    echo "No user found with the username: " . $username;
}
```

**INSERT data**

```
$username = "john_doe"; // Example data
$firstname = "John";
$lastname = "Doe";
```

```
// Prepare the SQL statement
$sql = $conn->prepare("INSERT INTO users (username, firstname, lastname) VALUES
(?, ?, ?)");
$sql->bind_param("sss", $username, $firstname, $lastname); // "sss" for three
strings
$sql->execute();

if ($sql->affected_rows > 0) {
    echo "New user inserted successfully.";
} else {
    echo "Failed to insert user.";
}

$sql->close();
```

**Delete**

```
$username = "john_doe"; // Example data

// Prepare the SQL statement
$sql = $conn->prepare("DELETE FROM users WHERE username = ?");
$sql->bind_param("s", $username); // "s" for a single string
$sql->execute();

if ($sql->affected_rows > 0) {
    echo "User deleted successfully.";
} else {
    echo "No user found with that username.";
}

$sql->close();
```

**Select Sort**

```
<?php
// Prepare the SQL statement to sort by last name
$sql = $conn->prepare("SELECT id, username, firstname, lastname FROM users ORDER
BY lastname ASC");
$sql->execute();
$result = $sql->get_result();

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"] . " - Name: " . $row["firstname"] . " " .
$row["lastname"] . "<br>";
    }
} else {
    echo "No users found.";
}

$sql->close();
?>
```

**select wild card**

```
<?php
$username = "john"; // Example: search for users starting with "john"

// Prepare the SQL statement with a wildcard
$sql = $conn->prepare("SELECT id, username, firstname, lastname FROM users WHERE
username LIKE ?");
$searchTerm = $username . "%"; // Adds the wildcard to the search term
$sql->bind_param("s", $searchTerm); // "s" for string
$sql->execute();
$result = $sql->get_result();

if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"] . " - Name: " . $row["firstname"] . " " .
$row["lastname"] . "<br>";
    }
} else {
    echo "No user found.";
}

$sql->close();
?>
```

**Select + JSON areturn**

```
<?php

// Database credentials
$dbusername = "root";
$dbpassword = "";
$dbname = "mydb";

// Create connection
$conn = new mysqli("localhost", $dbusername, $dbpassword, $dbname);

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

// Prepare the SELECT statement (for example, selecting all users)
$sql = $conn->prepare("SELECT id, username, firstname, lastname FROM users");
$sql->execute();
$result = $sql->get_result();

// Create an array to hold the results
$users = [];

if ($result->num_rows > 0) {
    // Fetch each row and add it to the $users array
    while ($row = $result->fetch_assoc()) {
        $users[] = $row;
    }
}

// Convert the result array to JSON and return it
header('Content-Type: application/json');
echo json_encode($users);

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

## HTML

**New page, get**

```
<!-- Form with GET method -->
    <form action="process.php" method="GET">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" placeholder="Enter your
username" required><br>

        <label for="age">Age:</label>
        <input type="number" id="age" name="age" placeholder="Enter your age"
required><br>

        <!-- Submit button -->
        <input type="submit" value="Submit">
    </form>
    <!--process.php?username=john&age=30
    $username = $_GET['username'];
    $age = $_GET['age'];
    -->
```

**New page, post**

```
 <!-- Form with POST method -->
    <form action="process.php" method="POST">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" placeholder="Enter your
username" required><br>

        <label for="age">Age:</label>
        <input type="number" id="age" name="age" placeholder="Enter your age"
required><br>

        <!-- Submit button -->
        <input type="submit" value="Submit">
    </form>

    <!-- $username = $_POST['username'];
$age = $_POST['age']; -->
```

**Proccessing forms in php for new pages**

```
<?php
// Check the request method
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    // Handle GET request
    $username = $_GET['username'];
    $age = $_GET['age'];
    echo "GET request received.<br>";
    echo "Username: $username<br>";
    echo "Age: $age<br>";
} elseif ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Handle POST request
    $username = $_POST['username'];
    $age = $_POST['age'];
    echo "POST request received.<br>";
    echo "Username: $username<br>";
    echo "Age: $age<br>";
} else {
    echo "Invalid request method.";
}
?>
```

Sure! Here's a **basic CSS cheat sheet** that covers fundamental CSS concepts, properties, and examples. This will help you quickly style your web pages during an exam or project.

---

## CSS Cheat Sheet

### 1. **Basic CSS Syntax**

```
selector {
    property: value;
}
```

- **Selector**: Targets the HTML element(s) to style.
- **Property**: The CSS property you want to change (e.g., `color`, `font-size`).
- **Value**: The value of the property (e.g., `red`, `16px`).

2. **CSS Selectors**

- **Element Selector**: Targets all elements of a type.

  ```
  p {
      color: blue;
  }
  ```

- **Class Selector**: Targets elements with a specific class.

  ```
  .classname {
      color: green;
  }
  ```

  In HTML:

  ```
  <p class="classname">This is a paragraph.</p>
  ```

- **ID Selector**: Targets a specific element with an ID.

  ```
  #unique-id {
      font-size: 18px;
  }
  ```

  In HTML:

  ```
  <div id="unique-id">This is a div.</div>
  ```

3. **Text Styling**

```css
p {
    color: #333;           /* Text color */
    font-size: 16px;        /* Font size */
    font-family: Arial, sans-serif;  /* Font family */
    text-align: center;     /* Center-align text */
    font-weight: bold;      /* Make text bold */
    text-decoration: underline;  /* Underline text */
}
```

4. **Background Styling**

```css
body {
    background-color: #f0f0f0;  /* Background color */
    background-image: url('image.jpg');  /* Background image */
    background-repeat: no-repeat;  /* Prevent repeating */
    background-size: cover;  /* Cover entire background */
}
```

5. **Box Model**

- The box model controls how padding, borders, and margins are applied to elements.

```css
div {
    width: 300px;           /* Element width */
    height: 200px;          /* Element height */
    padding: 20px;          /* Space inside the element */
    margin: 10px;           /* Space outside the element */
    border: 1px solid #ccc; /* Border around the element */
}
```

6. **Positioning Elements**

- **Static (default)**: Elements are positioned according to the normal document flow.
- **Relative**: Positioned relative to its normal position.
- **Absolute**: Positioned relative to the nearest positioned ancestor.
- **Fixed**: Positioned relative to the viewport (i.e., the window).

```css
div {
    position: relative;   /* Position relative to its normal position */
    top: 10px;            /* Move down by 10px */
    left: 20px;           /* Move right by 20px */
}
```

7. **Flexbox Layout**

Flexbox makes it easier to design flexible and responsive layouts.

```
.container {
    display: flex;              /* Enable flexbox */
    justify-content: center;    /* Center horizontally */
    align-items: center;        /* Center vertically */
    flex-direction: row;        /* Arrange children in a row */
    gap: 10px;                  /* Space between flex items */
}
```

In HTML:

```
<div class="container">
    <div>Item 1</div>
    <div>Item 2</div>
    <div>Item 3</div>
</div>
```

8. **CSS Grid Layout**

Grid is a powerful layout system for creating complex, responsive designs.

```
.container {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;  /* Three equal columns */
    grid-gap: 10px;  /* Space between grid items */
}
```

In HTML:

```
<div class="container">
    <div>Item 1</div>
    <div>Item 2</div>
    <div>Item 3</div>
</div>
```

9. **Hover and Active States**

```
a:hover {
    color: red;              /* Change color when hovered */
}

button:active {
    background-color: green; /* Change background when clicked */
}
```

10. **Media Queries** (Responsive Design)

Use media queries to apply different styles for different screen sizes (e.g., mobile, tablet, desktop).

```
/* For screens smaller than 600px */
@media only screen and (max-width: 600px) {
    body {
        font-size: 14px;
    }
    .container {
        flex-direction: column;  /* Stack elements vertically */
    }
}
```

11. **Centering Elements**

- **Horizontally center block-level elements**:

  ```
  div {
      margin: 0 auto;
      width: 50%;
  }
  ```

- **Horizontally & vertically center with Flexbox**:

  ```
  .center {
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;  /* Full viewport height */
  }
  ```

12. **CSS Colors**

- **Hexadecimal**: `#ff0000` (red)
- **RGB**: `rgb(255, 0, 0)` (red)
- **RGBA** (with transparency): `rgba(255, 0, 0, 0.5)`
- **Named colors**: `red`, `blue`, `green`, etc.

13. **Borders, Padding, and Margins**

```
div {
    border: 2px solid black;    /* Solid black border */
    padding: 20px;              /* Space inside the element */
    margin: 10px;               /* Space outside the element */
}
```

### 14. **Z-Index (Stacking Elements)**

Used to control the vertical stacking order of elements (higher `z-index` appears on top).

```
div {
    position: absolute;
    z-index: 10;
}
```

### 15. **Fonts and Typography**

```
body {
    font-family: Arial, sans-serif;  /* Font family */
    font-size: 16px;                 /* Font size */
    font-weight: bold;               /* Font weight */
    line-height: 1.5;                /* Line height */
    letter-spacing: 1px;             /* Letter spacing */
}
```

### 16. **Buttons Styling**

```
button {
    background-color: #4CAF50; /* Green background */
    color: white;              /* White text */
    padding: 10px 20px;        /* Padding inside button */
    border: none;              /* Remove border */
    cursor: pointer;           /* Pointer cursor on hover */
}

button:hover {
    background-color: #45a049; /* Darker green on hover */
}
```

### 17. **Links Styling**

```
a {
    color: blue;            /* Link color */
    text-decoration: none;  /* Remove underline */
}

a:hover {
    color: red;             /* Change color on hover */
}
```

---

## Quick References:

- **Units**:
  - `px`: Pixels
  - `%`: Percentage
  - `em`, `rem`: Relative to font size (1em = current font size, 1rem = root font size)
  - `vh`, `vw`: Viewport height/width

- **Box Shadow**:

  ```
  div {
      box-shadow: 5px 5px 10px #888888;  /* X-offset, Y-offset, blur radius,
  color */
  }
  ```

- **Transitions** (for animations):

  ```
  div {
      transition: all 0.3s ease-in-out;  /* Applies transition to all
  properties */
  }

  div:hover {
      background-color: red;  /* Change on hover */
  }
  ```

- **Opacity**:

  ```
  img {
      opacity: 0.7;  /* 70% opacity */
  }
  ```

---

## Useful Resources:

- CSS Tricks

- MDN Web Docs: CSS
- W3Schools: CSS

---

This cheat sheet provides a quick reference to fundamental CSS properties and techniques, including selectors, layout systems, and responsive design techniques (media queries). This should cover most of the basic styles you'll need for an exam! Let me know if you'd like additional topics included.