AJAX, PHP, and SQL

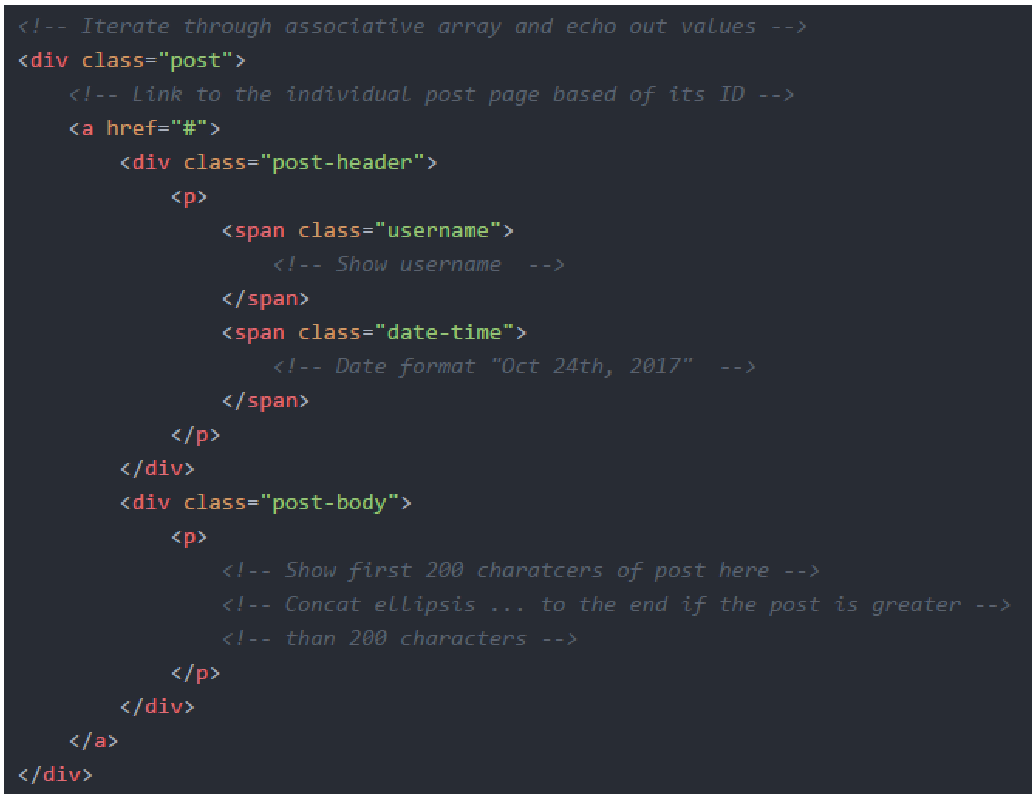
Using what you have learnt these last few weeks, your task is to complete the construction of the blog.

Exercise 1 – Retrieving posts and displaying with partials

Inside of **functions.inc** create a function called **get_posts**, and pass no parameters. This function is to return the last 10 posts submitted to the database. So your SQL statement shouldn't have a WHERE clause but will need to be ordered and have a limit of 10. If nothing is found, then return **false.**

The function should return the collected rows, if any which will then be passed to process the partial **indexPost.inc** in **includes/partials.** In case you've forgotten, partials are just parts of a HTML document that is logically separated from the main document and is included when used. It's better to create partial files and include them rather than parsing HTML as strings simply because they are easier to modify.

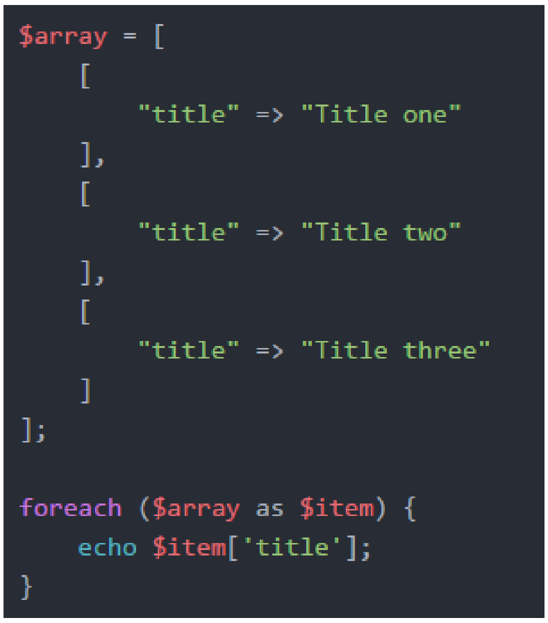
Have a look at the comments in **indexPost.inc** and include the PHP code accordingly.



```
<!-- Iterate through associative array and echo out values -->
<div class="post">
    <!-- Link to the individual post page based of its ID -->
    <a href="#">
        <div class="post-header">
            <p>
                <span class="username">
                    <!-- Show username  -->
                </span>
                <span class="date-time">
                    <!-- Date format "Oct 24th, 2017"  -->
                </span>
            </p>
        </div>
        <div class="post-body">
            <p>
                <!-- Show first 200 charatcers of post here -->
                <!-- Concat ellipsis ... to the end if the post is greater -->
                <!-- than 200 characters -->
            </p>
        </div>
    </a>
</div>
```

The date format isn't necessary for the practical, but it's a good learning experience with the native functions. Typically you wouldn't change the format on the backend because it's determined by the time-zone settings in the server, so it is always best to use a library like moment.js to handle date/time formatting on the front-end because they can handle converting times to local times.

Since the returned data is a 2D array, where the parent array is 0-based indexed, and the child arrays are associative, you will have to use the **foreach loop** like so:

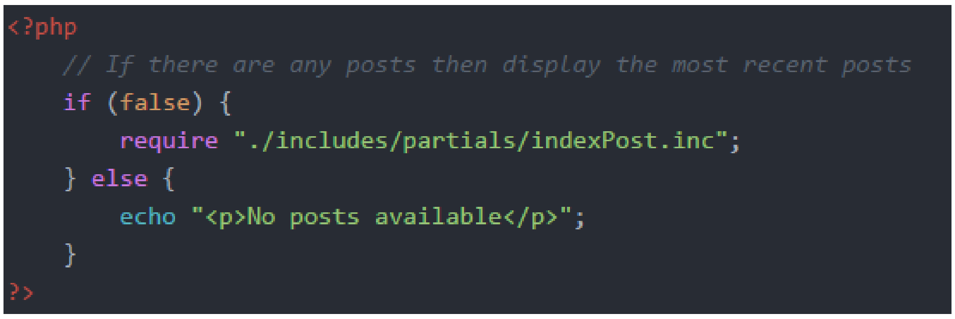


```php
$array = [
    [
        "title" => "Title one"
    ],
    [
        "title" => "Title two"
    ],
    [
        "title" => "Title three"
    ]
];

foreach ($array as $item) {
    echo $item['title'];
}
```

The above code is nothing more than an example of how it works. The **foreach loop** loops through the array **$array** for each of its elements **$item**, which are the associative arrays. Then you access the values using the key.

In **index.php** you will see this block of code:



```php
<?php
    // If there are any posts then display the most recent posts
    if (false) {
        require "./includes/partials/indexPost.inc";
    } else {
        echo "<p>No posts available</p>";
    }
?>
```

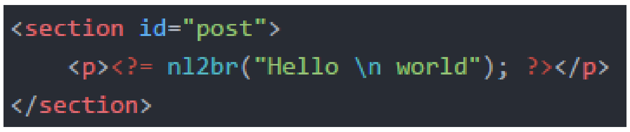
Call the function **get_posts** and store the results into a variable. Replace the Boolean value **false** with your variable, then pass your variable into the **foreach** loop.

Exercise 2 – Viewing a blog post

Going back to the previous exercise you were required to create a link to the blog post. The url should look like this: **blog_post.php?id={$item['id']}** where $item['id'] is the id of the post. The file **blog_post.php** is the page where we can show the post by itself. In order to do that you need to create a function to get the details of the post by its ID. Create a function called **get_post($id)** where **$id** is the URL parameter id. Remember to use **$_GET** in order to get the post id from the URL.

You must display the post's **title**, **username** of poster, **date** of submission, and the **post** itself.

The function **nl2br** stands for newline-to-break and converts all newline characters in string to be converted to the **<br>** element. We use this as a quick and easy way to format paragraphs from the backend.



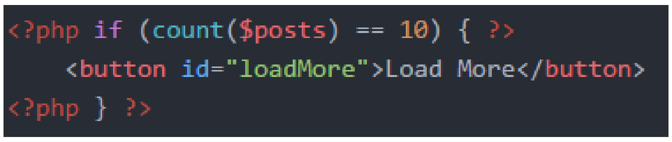
```
<section id="post">
    <p><?= nl2br("Hello \n world"); ?></p>
</section>
```

Optional: If no post was found then display an error message stating nothing was found.

Exercise 3 – Showing more posts with AJAX

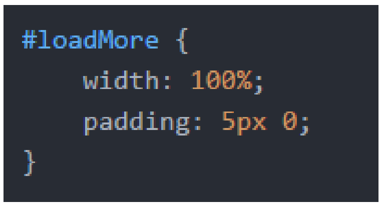
From exercise 1 we can show the 10 most recently made posts. What we want to do is display the next 10 posts with AJAX. We have to use AJAX in this instance because using the standard form submission would just reload the page. You could extend the list this way, but it's inefficient and clumsy.

First go back to the indexPost.inc partial and add the following code at the bottom of the file outside of the loop. You can change the **$posts** variable to the name the variable that stores your array.



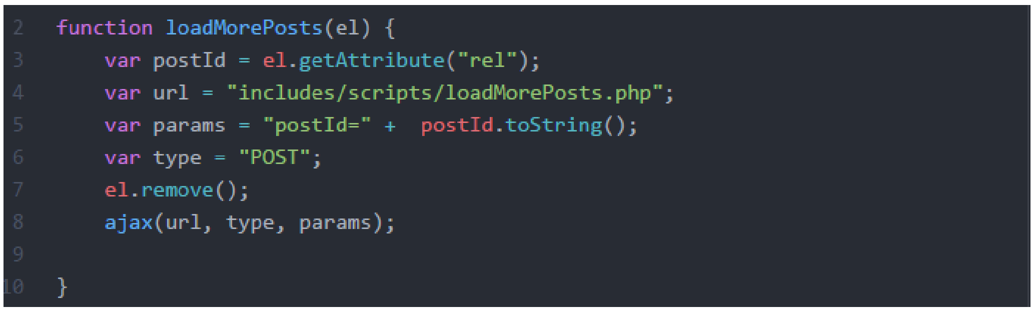
```
<?php if (count($posts) == 10) { ?>
    <button id="loadMore">Load More</button>
<?php } ?>
```

For the sake of basic aesthetics, add the following CSS code to index.css.



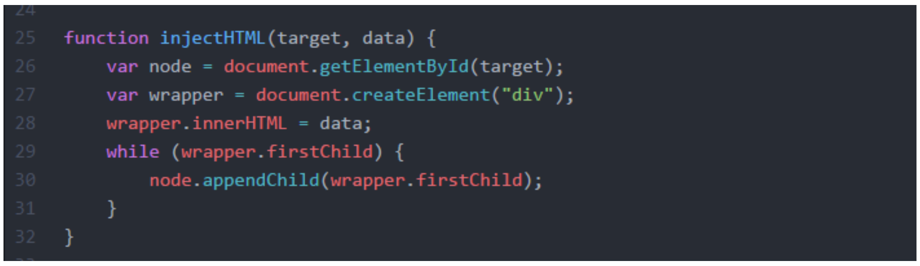
```
#loadMore {
    width: 100%;
    padding: 5px 0;
}
```

Now in **index.js** add the following code for to perform the AJAX request.



```
function loadMorePosts(el) {
    var postId = el.getAttribute("rel");
    var url = "includes/scripts/loadMorePosts.php";
    var params = "postId=" +  postId.toString();
    var type = "POST";
    el.remove();
    ajax(url, type, params);

}
```



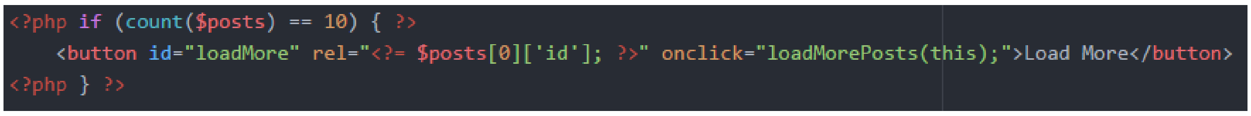
```
function injectHTML(target, data) {
    var node = document.getElementById(target);
    var wrapper = document.createElement("div");
    wrapper.innerHTML = data;
    while (wrapper.firstChild) {
        node.appendChild(wrapper.firstChild);
    }
}
```

You will notice that the function **ajax** doesn't exist yet, which is something you will have to complete. Create a function called **ajax** and inside of it use the **fetch API** to complete an asynchronous request to the target url. Alternatively you can just include the **fetch API** where the **ajax** function is called. Look at the following URL for more details. We are looking to perform a **POST request**.

https://developers.google.com/web/updates/2015/03/introduction-to-fetch

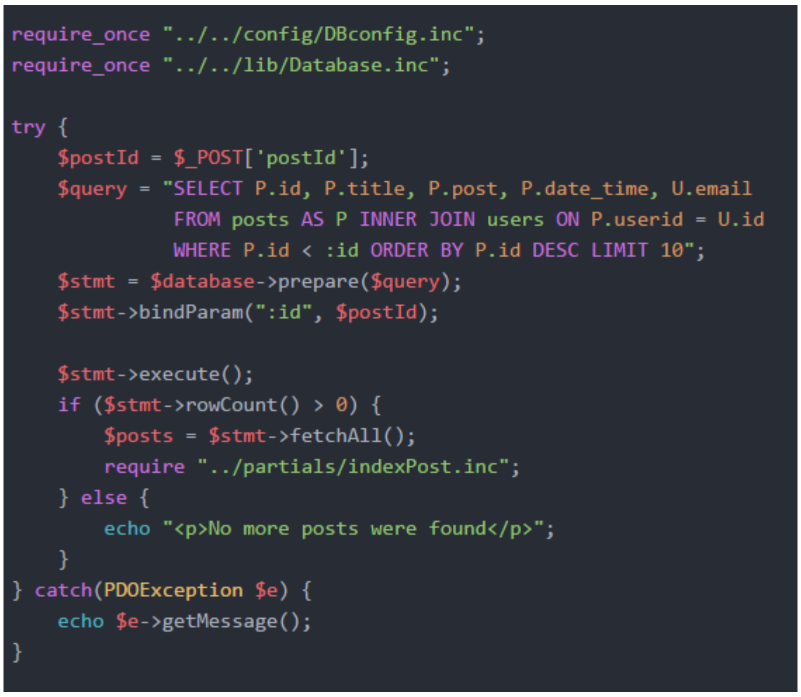
What this will do is send a post request to the server to execute the script **loadMore.php.** This script will receive one parameter, which is the ID of the last post received from the SELECT query.

Add a **rel** attribute and an **onclick** event to the load more button to contain the last ID like so:



```
<?php if (count($posts) == 10) { ?>
    <button id="loadMore" rel="<?= $posts[0]['id']; ?>" onclick="loadMorePosts(this);">Load More</button>
<?php } ?>
```

We use this value to determine what records we want to return as a constraint. Create a new file in includes/scripts called **loadMorePosts.php** and include the following:



```
require_once "../../config/DBconfig.inc";
require_once "../../lib/Database.inc";

try {
    $postId = $_POST['postId'];
    $query = "SELECT P.id, P.title, P.post, P.date_time, U.email
              FROM posts AS P INNER JOIN users ON P.userid = U.id
              WHERE P.id < :id ORDER BY P.id DESC LIMIT 10";
    $stmt = $database->prepare($query);
    $stmt->bindParam(":id", $postId);

    $stmt->execute();
    if ($stmt->rowCount() > 0) {
        $posts = $stmt->fetchAll();
        require "../partials/indexPost.inc";
    } else {
        echo "<p>No more posts were found</p>";
    }
} catch(PDOException $e) {
    echo $e->getMessage();
}
```

Does this code work? If not, try to debug and fix the code. Otherwise great!