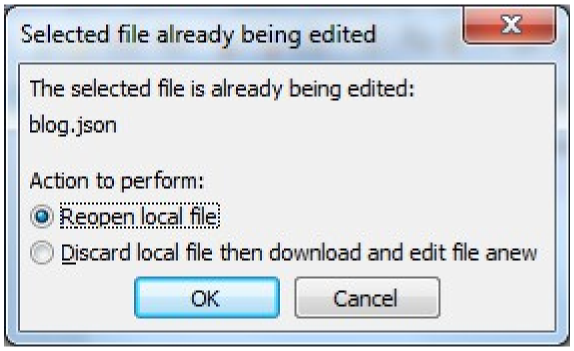
Assigned: 2013-09-20
Due: 2013-10-01 (Tuesday)

Turn In

You can test your PHP files on the *inclass* server space, but to turn in your work you just need to zip up all project files (.php5, .css, etc.) and submit them via Blackboard.

Tip: If FileZilla asks whether to reopen or discard a file...


Selected file already being edited
The selected file is already being edited:
blog.json
Action to perform:
(●) Reopen local file
( ) Discard local file then download and edit file anew

Click on *Discard local file then download and edit file anew.* If you try to Reopen, you might not see your changes appear!

Tip: Check the sample code from the class webpage if you get stuck on loading/saving files.

PHP Concepts

- Including other PHP files
- JSON File reading/writing
- Pulling data out of a form

Description

For this assignment we are going to write a simple blog, with a control panel and a main view. From the control panel, the author can enter a title, date, and the blog post. Blog posts are stored in a .json file which will be read in each time.

Files

For this assignment, we are going to have three main php files. You can add a CSS file if you would like, I have no requirements for styling.
Make sure your XHTML is valid.

File	Description
Control-panel.php5	Webpage, contains new blog post form
Blog.php5	Webpage, contains read-only view of all blog posts
Model.php5	Not a webpage: Just a script file. Contains functionality for reading and writing the .json blog post.

Building the Blog

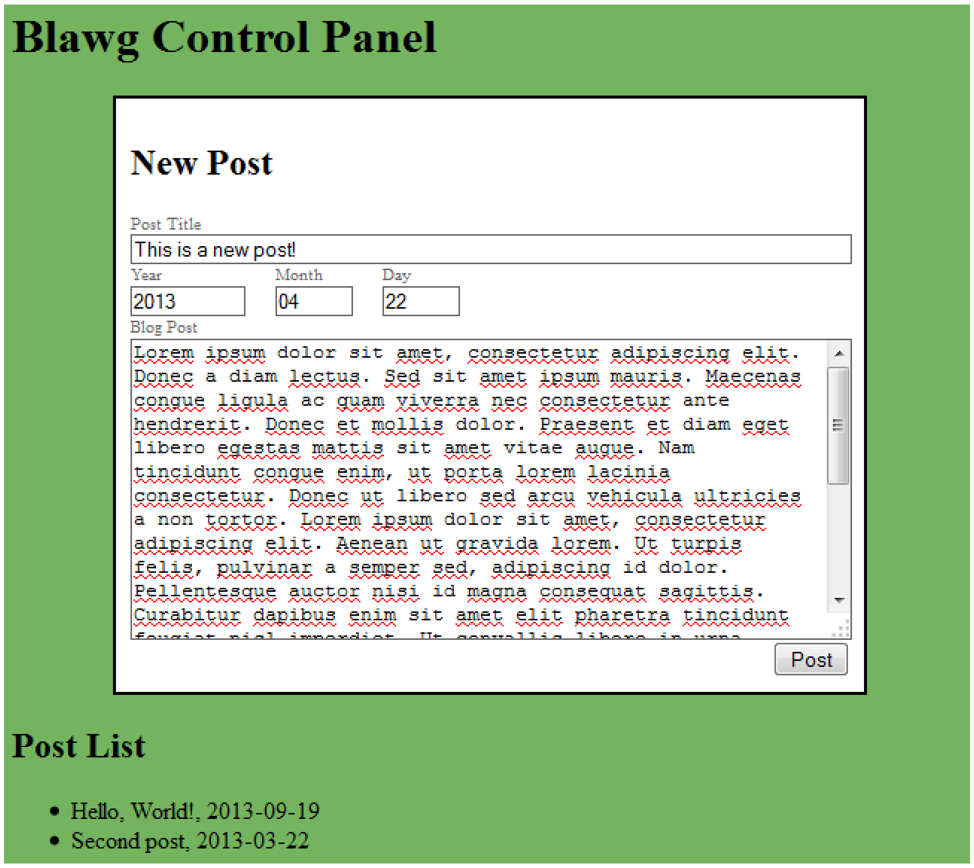
The Control Panel

Start by building the XHTML for the control panel. Minimum required XHTML is as follows:

```
<!DOCTYPE html>
<html>
     <head>
          <title>Blog Control Panel</title>
     </head>

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

Within the blog, we are going to have two main sections: A list of posts that have already been posted, and a form where the user will enter the new post information.


Blawg Control Panel

New Post

Post Title: This is a new post!

Year: 2013 Month: 04 Day: 22

Blog Post:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Donec a diam lectus. Sed sit amet ipsum mauris. Maecenas
congue ligula ac quam viverra nec consectetur ante
hendrerit. Donec et mollis dolor. Praesent et diam eget
libero egestas mattis sit amet vitae augue. Nam
tincidunt congue enim, ut porta lorem lacinia
consectetur. Donec ut libero sed arcu vehicula ultricies
a non tortor. Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Aenean ut gravida lorem. Ut turpis
felis, pulvinar a semper sed, adipiscing id dolor.
Pellentesque auctor nisi id magna consequat sagittis.
Curabitur dapibus enim sit amet elit pharetra tincidunt
```

Post List

- Hello, World!, 2013-09-19
- Second post, 2013-03-22

You can style each page however you would like.

Create only one form element, and make sure to set its **method** attribute to **post**.

Then, create the following form elements:

Element Type	**Element Name**	**Element ID**	**Description**
Input type text	post[title]	post-title	Will store the title of the post
Label for post-title			
Input type number	post[date]	post-year	Stores the year for the post's timestamp
Label for post-year			
Input type number	post[date]	post-month	Stores the month for the post's timestamp
Label for post-month			
Input type number	post[date]	post-day	Stores the day for the post's timestamp
Label for post-day			
Text area	post[content]	post-content	The actual blog post text

Label for post-content			
Input type submit	new-post		The submit button

For the **Post List** section of the page, create an unordered list, but don't add any list elements yet. We will come back to this later.

Once the form and post list are set up, make sure to add

```
<? include_once( "model.php5" ); ?>
```

to the top of your file (after the DOCTYPE tag)

Make sure to upload your file to the server!

The Model

The model.php5 file is a script file, containing back-end functionality for handling blog posts. It will be shared by both the control-panel and the blog pages. Keeping functionality separate from the markup is usually a good idea, just like keeping your .css styles and .js scripts outside of the XHTML markup.

Within the Model, first create two variables: A variable named **blogPath** which is set to "blog.json", and a variable named **blogPosts**, which is a new array.

Then, create two comments in your page so we can keep things sectioned off:

```
<?
$blogPath = "blog.json";
$blogPosts = array();

/* Load Posts */

/* Save Posts */
?>
```

On both pages, we will always load the existing posts, if there are any. On the control panel, it will just display a simple list of previous posts (we are not adding edit functionality). On the blog page, it will display all blog posts on one page. The blog posts will be loaded out of *blog.json* and the json_decode function will be used to turn it from a JSON string to a PHP array.

For the save functionality, first it will check to see if the **new-post** submit button has been hit. From there, it will push the new blog post on to the array of blog posts, and then encode the array to json and save to the *blog.json* file.

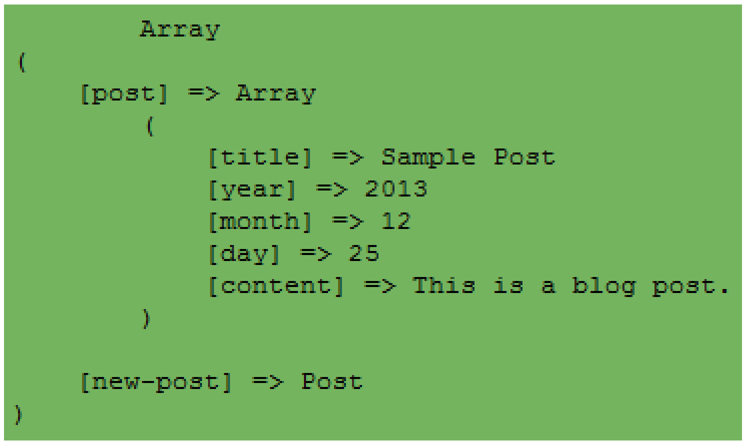
Since there are no posts to load in right now, let's start with saving the post. Note that our **form** element on the control panel is set to **method=”post”**. In PHP, we will use the **$_POST** array to look at the values entered into the form.

Let's start by looking at how this array is build. Add this code after our Save Posts comment:

```
/* Save post on form submit */
?>
<pre>
     <? print_r( $_POST ); ?>
</pre>
<?
```

Notice that we can end a PHP script at any time (with the closing ?>), add in XHTML, and then re-open the PHP script to call a function.

Now, upload the model.php5 to the server. Refresh the control-panel.php5 page, enter information into the various input fields, and then click Submit. You should see an array pop up after the refresh, which looks somewhat like this:



```
        Array
(
    [post] => Array
        (
            [title] => Sample Post
            [year] => 2013
            [month] => 12
            [day] => 25
            [content] => This is a blog post.
        )

    [new-post] => Post
)
```

Our $_POST array contains an element named “post”. Post itself is also an array, which contains “title”, “year”, “month”, “day”, and “content”.

There is also a second element within $_POST named “new-post”. This is the submit button we clicked.

Now, remember that our input boxes had names like this:

`post[day], post[title], post[content]`

By adding the square brackets and text inside, we're telling PHP that all these fields belong to the *post* array, and the value within the square brackets are the specific element within *post* that we're storing.

Now we know what is being posted to the server, but we need to actually do something with this information...

First, we need to check for the **new-post** flag. We can use PHP's **isset** function to do so. The **isset** function returns true or false, based on whether the variable passed in exists or not.
We can check

```
if ( isset( $_POST["new-post"] ) )
{
}
```

And, if the new-post button hadn't been pressed, this would return false. Otherwise, we know the new-post submit button was clicked and we should then save our blog post.

Saving our Blog Posts

The first time around, we won't have any *blog.json* file and no blog posts to pre-load; it's a blank slate. We can use the array in $_POST and encode it to JSON to store it in the file. However, we will have to come back to this later in order to account for pre-existing posts that we don't want to remove each time.

Within the `if ( isset( $_POST["new-post"] ) )` if block, we will do the following:

1. Convert the array to json and store it in a variable. (json_encode function)
2. Open a file to write to (fopen function).
   Store this as $fileWrite. This is the same sort of thing as $fileHandler in the sample code.
   We're using “fileWrite” here because we will also have a “fileRead”.
3. Write the json variable to the file (fwrite function)
4. Close the file (fclose function)

* Remember that we created $blogPath and $blogPosts variables at the top of this file! We're not using $blogPosts yet, but you will store the current post at the $blogPath.

Upload all of the changed files to the server via your FTP client. Then, go back to control-panel.php5 in your web browser, fill out the post information, and click Submit.

If you don't get any error messages, go back to the FTP client and refresh (F5). See if *blog.json* was created.

```
[{"title":"Hello,
World!","year":"2013","month":"12","day":"25","content":"This is a
sample blog post."}]
```

Sample blog.json file

Loading existing blog posts

Now that we have one post saved, let's write the logic to load existing posts. Once this is written, it will work whether there is 1 post or 100, since it will store them in an array.

Back in model.php5, under the Load Posts comment, we will first need to check to see if *blog.json* exists. If it does not, we will get errors when trying to load it for a read. (If the file doesn't exist, we don't need to load *existing* blog posts, anyway!)

You can check to see if a file at some path exists with the **file_exists** function. This will return true or false.

```
if ( file_exists( $blogPath ) )
{
}
```

Now do the following:

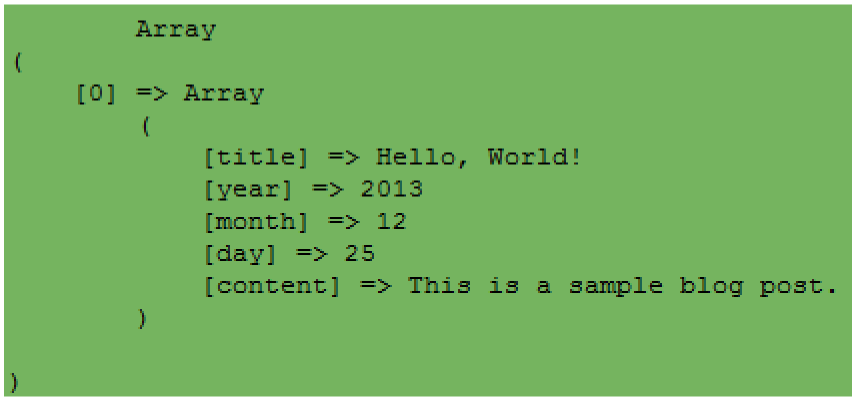
1. Load the file into a $fileRead variable using fopen.
2. Remember that we already declared a $blogPosts array at the top of the file. Use json_decode to store the contents of the $fileRead to this array.
   Make sure to convert the JSON to an **associative array** by setting the **second argument** in the json_decode function to **true**.
3. Close the $fileRead file

Now, any existing blog posts should be stored in $blogPosts.

Add this to your model.php5 file after the if block:

```
?>
<pre>
     <? print_r( $blogPosts ); ?>
</pre>
<?
```

Upload all changes to the server, then open the control-panel.php5. You should see an array similar to last time, except this array was loaded from a file, not posted from the form.



```
        Array
(
    [0] => Array
        (
            [title] => Hello, World!
            [year] => 2013
            [month] => 12
            [day] => 25
            [content] => This is a sample blog post.
        )

)
```

Notice how the first element is [0]. Any additional blog posts will be [1], [2], etc. We can use a for-each loop to iterate through all posts on the blog page now!

The Blog Page

Before we continue adding functionality to the control panel, let's go ahead and load our blog post into the blog page.

Remember to start of with the XHTML base, plus the include_once for our model:

```
<!DOCTYPE html>
<? include_once( "model.php5" ); ?>
<html>
     <head>
          <title>Blog</title>
     </head>

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

By including model.php5, the blog post loaded above (shown in the control panel) will automatically also be loaded on this blog page.

You can style this how you would like later, but for now, create a foreach loop within the <body> tags:

```
<body>

     <h1>My Blog</h1>

     <?
     foreach( $blogPosts as $post )
     {
     }
     ?>

</body>
```

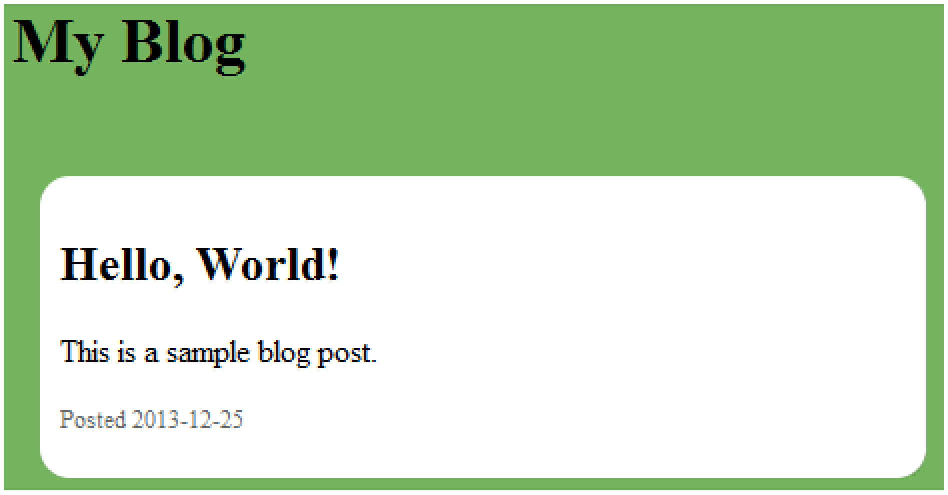
Remember that PHP's foreach loop is in the opposite order from Python's. Python has

for single-item in array

and PHP has

foreach array as single-item

Output each element of the **$post** element: title, content, year, month, and day.


My Blog

Hello, World!

This is a sample blog post.

Posted 2013-12-25

Now, let's finish up by making sure we *append* new blog posts to the *blog.json* file, rather than overwriting it every time.

Saving, revisited.

Back in model.php5, we are going to modify the save section.

Now that we have the file loading working, all existing blog posts will *already* be loaded in every time a web page is loaded. We're half way there.

Now, we just need to make sure that we append any new blog posts to the end before saving the file.

Add this step to the beginning:

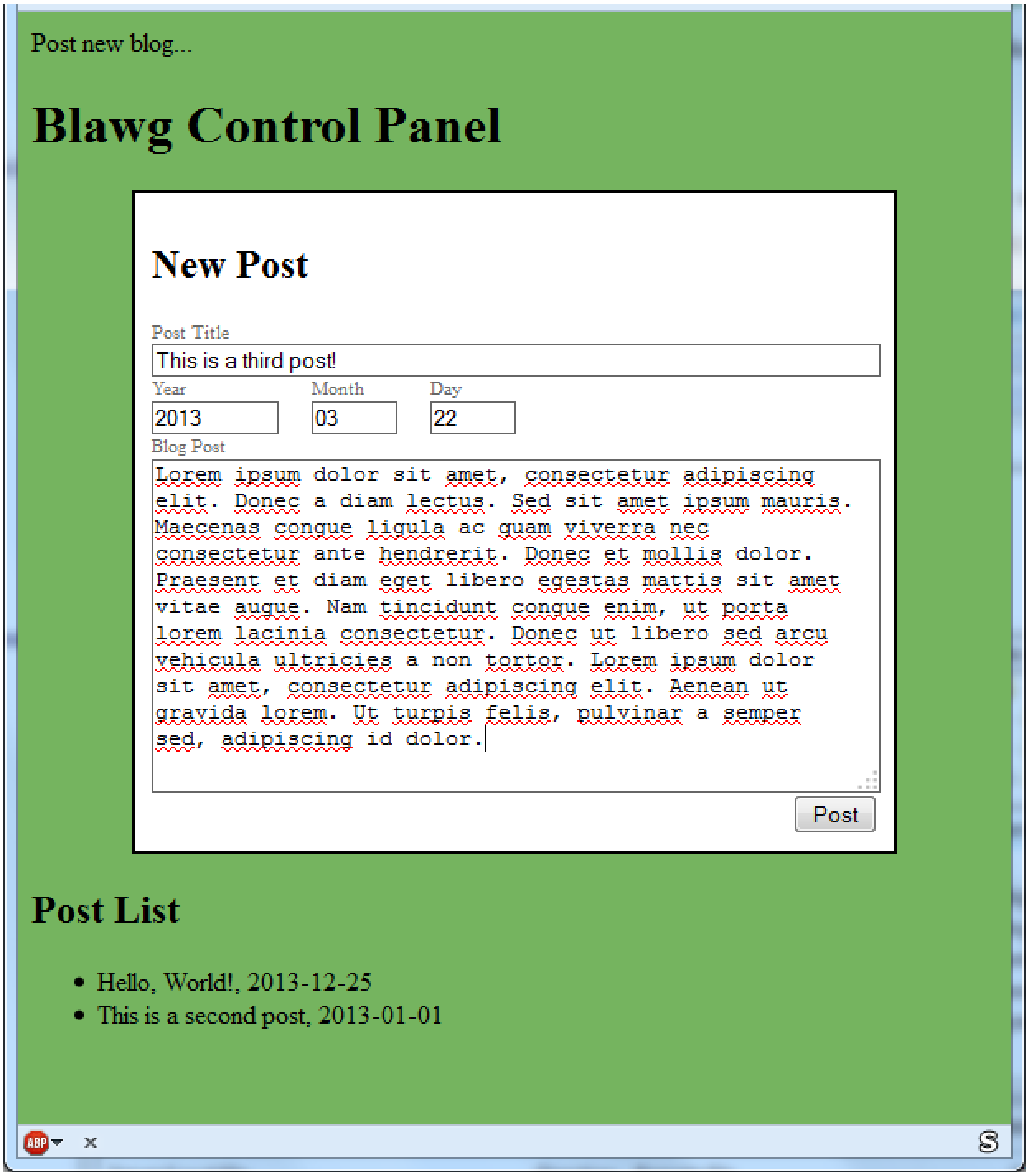
1. Remember that we have existing posts in the $blogPosts variable. We want to store the $_POST[“post”] data at the *end of this array*. Use the array_push function to push the new post onto the array of existing posts.

Our posts are in the form of an array. Now we can continue as before by converting the array to json (json_encode), opening a file to write to, and *overwriting the existing blog.json*.

Note that by overwriting blog.json, we're still maintaining the *old posts* because the old posts were part of the $blogPosts array that was loaded earlier in the model file.

Screenshots

Control Panel with two existing posts:



Post new blog...

Blawg Control Panel

New Post

Post Title
This is a third post!

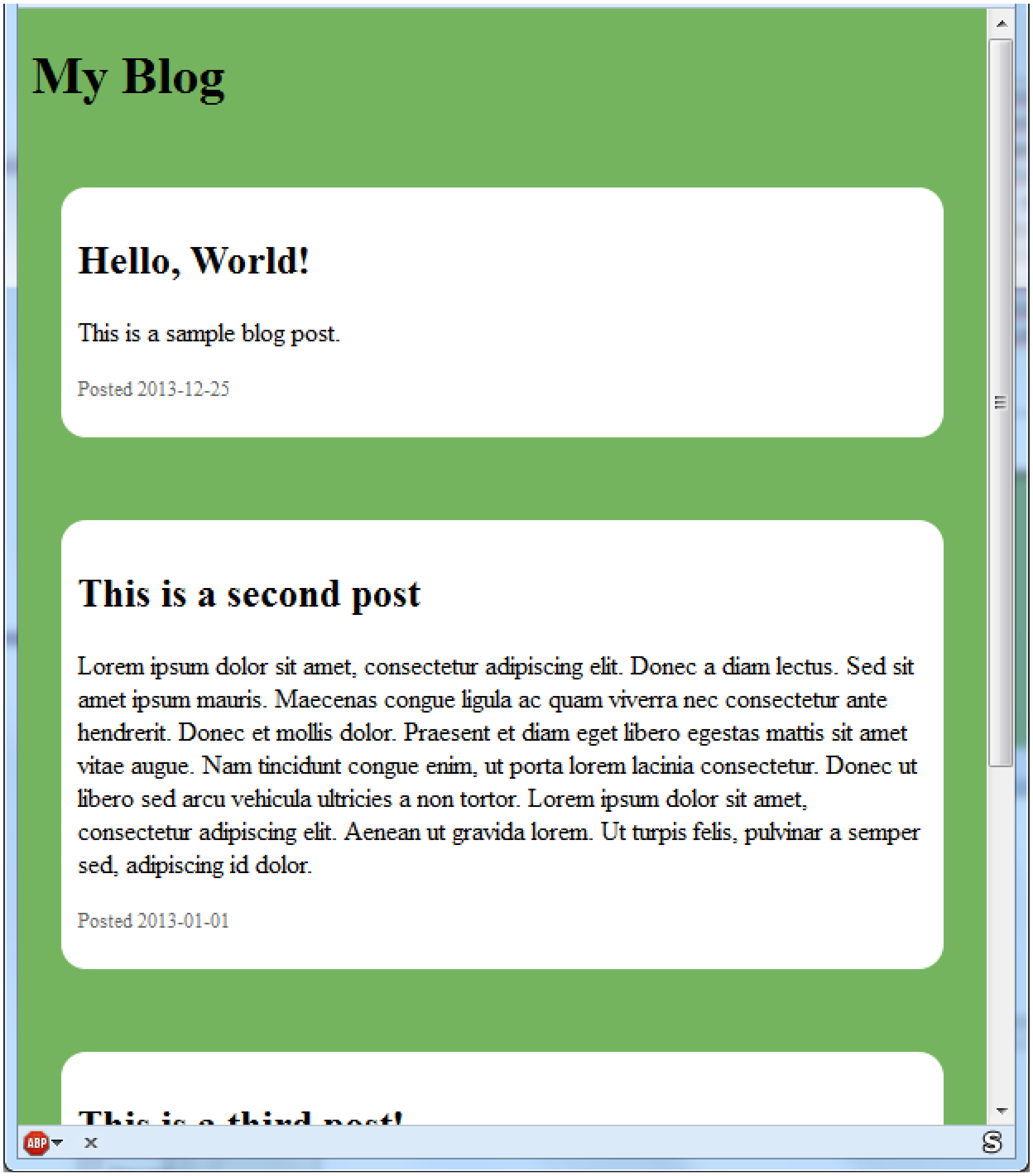
Year 2013 Month 03 Day 22

Blog Post
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec a diam lectus. Sed sit amet ipsum mauris. Maecenas congue ligula ac quam viverra nec consectetur ante hendrerit. Donec et mollis dolor. Praesent et diam eget libero egestas mattis sit amet vitae augue. Nam tincidunt congue enim, ut porta lorem lacinia consectetur. Donec ut libero sed arcu vehicula ultricies a non tortor. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean ut gravida lorem. Ut turpis felis, pulvinar a semper sed, adipiscing id dolor.

Post List

- Hello, World!, 2013-12-25
- This is a second post, 2013-01-01

Blog page with multiple posts:



My Blog

Hello, World!

This is a sample blog post.

Posted 2013-12-25

This is a second post

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec a diam lectus. Sed sit amet ipsum mauris. Maecenas congue ligula ac quam viverra nec consectetur ante hendrerit. Donec et mollis dolor. Praesent et diam eget libero egestas mattis sit amet vitae augue. Nam tincidunt congue enim, ut porta lorem lacinia consectetur. Donec ut libero sed arcu vehicula ultricies a non tortor. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean ut gravida lorem. Ut turpis felis, pulvinar a semper sed, adipiscing id dolor.

Posted 2013-01-01

This is a third post!