

# PHP Basic Syntax- Lab

This document defines a set of tasks to be done as a part of the PHP Basic Syntax lecture's exercises.

## Part I: Play with PHP

This first part will walk you through the PHP's basic syntax by making you solve some logical basic tasks. This is very important before going to build web applications. Please, be patient.

### Calculate Two Numbers

You will first start using PHP in CLI mode (console application). This means you will receive user input through the standard input and will return result into the standard output.

PHP, like most of the modern languages, supports argument passing on application start as well as waiting for user input.

Now we need to read the standard input. PHP relies heavily on [streams](#) (maybe as heavy as Java does). We can read a line from a resource handle by using the function [fgets\(\)](#). It might be usual for files, but other streams are also acceptable, thus we can pass the STDIN to it.

```
<?php
$operation = $argv[1];

$numberOne = fgets(STDIN);
```

There are **two possible problems** here. If we need to read a text, **we will receive the line terminating symbols** as part of the variable contents. We can **trim** them safely. As we read numbers, **we can just cast or convert to number**, which will remove the redundant extra symbols.

```
<?php
$operation = $argv[1];

$numberOne = intval(fgets(STDIN));
```

To read the second number, we need to read the second line as well

```
<?php
$operation = $argv[1];

$numberOne = intval(fgets(STDIN));
$numberTwo = intval(fgets(STDIN));
```

Now we need to build our logic. Keep in mind that **no matter if you output to a web page or to the CLI, you can use "echo" construction**.

```
<?php
$operation = $argv[1];

$numberOne = intval(fgets(STDIN));
$numberTwo = intval(fgets(STDIN));

if ($operation == "sum") {
    echo " == " . ($numberOne + $numberTwo);
} else if ($operation == "subtract") {
    echo " == " . ($numberOne - $numberTwo);
} else {
    echo " == Wrong operation supplied";
}
```

Let's try our program now:

```
RoYaL@DESKTOP-BFNQJQE MINGW64
$ php cli-test.php sum
22
33
== 55
RoYaL@DESKTOP-BFNQJQE MINGW64
$ php cli-test.php subtract
33
22
== 11
RoYaL@DESKTOP-BFNQJQE MINGW64
$ php cli-test.php multiply
2
3
== Wrong operation supplied
```

### Find Largest of Three Numbers

You are given three numbers from the standard input each on new line. Print on standard output the string "Max: ", followed by the largest number e.g. "Max: 4".

Let's find the largest number amongst the first and the second. Let's assume that the first number is the largest.

```
<?php
$numberOne = intval(fgets(STDIN));
$numberTwo = intval(fgets(STDIN));
$numberThree = intval(fgets(STDIN));

$largestFromOneTwo = $numberOne;
```

If the second is larger than the first, assign `$largestFromOneTwo` the value of `$numberTwo`.

```
<?php
$numberOne = intval(fgets(STDIN));
$numberTwo = intval(fgets(STDIN));
$numberThree = intval(fgets(STDIN));

$largestFromOneTwo = $numberOne;

if ($numberTwo > $numberOne) {
    $largestFromOneTwo = $numberTwo;
}
```

Now **the largest amongst all the numbers** is the largest **amongst the first two numbers and the third number**

```
if ($numberThree > $largestFromOneTwo) {
    echo "Max: " . $numberThree;
} else {
    echo "Max: " . $largestFromOneTwo;
}
```

Cool! It's ready!

Of course there's a built-in function that finds **max numbers between two numbers** and it has the **same performance as if/else**, so we will not kill the performance if we use it.

```
$numberOne = intval(fgets(STDIN));
$numberTwo = intval(fgets(STDIN));
$numberThree = intval(fgets(STDIN));

$largestFromOneTwo = max($numberOne, $numberTwo);
$largest = max($largestFromOneTwo, $numberThree);
echo "Max: " . $largest;
```

### Find Largest from Undefined Count of Numbers

You need to read **numbers (the numbers will be in range  $[1, 2^{31}]$ )** from the standard input **until receiving empty string** and find the maximum number amongst them. Print it to the standard output as in Task 2 (Max: n).

First of all, if we don't know the exact count of the lines we will receive, **we cannot use a deterministic loop like "for" loop** (or at least not in the conventional scenario of a for loop). We can use a **"while" loop**, and once we receive an empty string to break out from the loop:

```
<?php
while (true) {
    $line = trim(fgets(STDIN));
    if (empty($line)) {
        break;
    }

    $number = intval($line);
}
```

Or we can simplify it **relying on three rules**:

1. While loop **exits once the condition is falsy** (empty string, 0, false, null...)
2. We will never receive 0 because of the range [1, 2<sup>31</sup>].
3. The **assignment process still produces output** and it's not void (as in Java/C#)

```
while ($number = intval(fgets(STDIN))) {
}
```

We can put all the numbers in an **array** and use the built-in utility function that finds max number in an array.

```
<?php
$numbers = [];
while ($number = intval(fgets(STDIN))) {
    $numbers[] = $number;
}
echo "Max: " . max($numbers);
```

And it will most probably work. **The solution is elegant, yet not so fast.**

The problem here is that **we are iterating through a set of input** and then the built-in function **iterates over the array to find the largest number**. If we receive **5000 numbers**, we will make **2x5000 = 10 000 iterations**. It might be crucial to the performance. So we need to optimize it.

Why don't we **search for the largest number at the same time we are reading the input**?

Comparing each number to the last largest number found. Once we are done reading the input we will have the largest number found.

```
<?php
$largest = 0;
while ($number = intval(fgets(STDIN))) {
    if ($number > $largest) {
        $largest = $number;
    }
}
echo "Max: $largest";
```

Or simply:

```
<?php
$largest = 0;
while ($number = intval(fgets(STDIN))) {
    $largest = max($largest, $number);
}
echo "Max: $largest";
```

### Largest Number Again

Having in mind the considerations in the Task 3, find the largest number, but this time **the input numbers might contain negatives**.

#### Examples

Input	Output
3 7 -4 5	Max: 7

Input	Output
-3 -2 -17 -33 -1	-1

### Count Letters

You will receive a single line from the standard input containing a word (or at least a set or characters). You need to print on the standard input **how many times each letter is found in order of the letter appearance**, in format {letter} -> {times}

Input	Output
apple	a -> 1 p -> 2 l -> 1 e -> 1

Input	Output
appearance	a -> 3 p -> 2 e -> 2 r -> 1 n -> 1 c -> 1

An [associative array](#) is good for this purpose. It can keep each **letter as a key** and the **times it has appeared as value**. When you **find a letter for first time**, **set its value to 0**. This will **ensure you will never try to increment non-existing keys**. The main logic then will be “**increment the value by that key in the array**”. If it was just found, it will be 0 and incrementing it will become 1 (finding letter for first time means exactly that – it has occurred 1 time). If it was found before it will increment the old value e.g. from 1 to 2

```
if (!array_key_exists($letter, $letters)) {
    $letters[$letter] = 0;
}
$letters[$letter]++;
```

Read about [strlen\(\)](#) function in order to iterate over a string.

### Count Letters – Sorted

As in Task 5, but the output should be **sorted by the times a letter has occurred in descending order**, then in order of appearance.

Input	Output
apple	p -> 2 a -> 1 l -> 1 e -> 1

Input	Output
appearance	a -> 3 p -> 2 e -> 2 r -> 1 n -> 1 c -> 1

[Read about sorting an array](#). Choose the appropriate function (or you will have headache with the keys).

### Part II: Try the Web

So far you might have understood how PHP's I/O works. Once a WEB Server receives a request and needs to dispatch it to the PHP parser it carefully prepares the environment eligible for WEB development. One of the core problems is how to pass a user input to a PHP file, when it's part of the WEB Environment. The HTTP Standard defines some Request ways. One of them is the **query string** and the another is the **FORM data**.

We will discuss the form data later, and stick to the query string right now.

The query string is all that part you might have already seen **after the question mark in the address bar**. For instance, [http://site.com/script.php?name=john&last\\_name=smith](http://site.com/script.php?name=john&last_name=smith) has a query string, namely: “**name=john&last\_name=smith**”.

A part of the environment preparation is converting the query string to key/value pairs. You can find in the [\\$ GET superglobal variable](#) an associative array containing “**name**” as **key** and “**john**” as **its value**. Same applies for “**last\_name**” and “smith”.

The query string can be manually filled as in the above example or generated from FORM submission with method GET.

A **Form** is an **HTML element that contains a fieldset with name/value pairs** e.g. text fields, dropdowns, checkboxes and so forth. After the form is submitted the values of the fields are sent (along with their name attribute as keys) either as form data or query string depending on the “method” attribute of the form tag. The default method is “GET” which produces a query string.

### 1. Calculate Two Numbers

CLI is gone. No console anymore. We need a web interface. So let’s define one:

Operation:	Sum	▼
Number 1:	Sum	
Number 2:	Subtract	
<input type="button" value="Calculate!"/>		

From the above web page, a user can choose an operation between Sum and Subtract from a dropdown, then enter 2 numbers in text fields and could click “Calculate!” button.

What we need to achieve is to take that input and perform the logical calculation.

Let’s see how the above web interface could be as HTML code:

```
<form method="get">
  <div>
    Operation:
    <select name="operation">
      <option value="sum">Sum</option>
      <option value="subtract">Subtract</option>
    </select>
  </div>
  <div>
    Number 1:
    <input type="text" name="number_one"/>
  </div>
  <div>
    Number 2:
    <input type="text" name="number_two"/>
  </div>
  <div>
    <input type="submit" name="calculate" value="Calculate!"/>
  </div>
</form>
```

It defines a FORM with method GET (the method can be omitted, as GET is the default). The fieldset in the form is as follows:

- Dropdown named “operation” with values either “sum” or “subtract”
- Text field “**number\_one**” with user value
- Text field “**number\_two**” with user value
- Submit button

This means that after the form submission the key “operation” will be in the \$\_GET superglobal with value either “sum” or “subtract” (to be honest, people can modify it, but we will talk about it later), as well as for “**number\_one**” and “**number\_two**”.

Let debug the \$\_GET superglobal and open our PHP file in the web browser

```
Number 1:

</div>
<div>
    Number 2:
    
</div>
<div>
    
</div>
</form>
<?php
var_dump($_GET);
```

Operation:

Number 1:

Number 2:

```
C:\wamp64\www\Scripting\calculator.php:22:
array (size=0)
    empty
```

Let's populate the form and click Calculate



Operation:	<input type="text" value="Sum"/>
Number 1:	<input type="text"/>
Number 2:	<input type="text"/>
<input type="button" value="Calculate!"/>	

C:\wamp64\www\Scripting\calculator.php:22:

```
array (size=4)
  'operation' => string 'subtract' (length=8)
  'number_one' => string '55' (length=2)
  'number_two' => string '33' (length=2)
  'calculate' => string 'Calculate!' (length=10)
```

Everything seems OK, so we can now take the input from the `$_GET` superglobal and create our logic

First of all, we need to make the calculation only when the submit button is clicked. We can determine that by the presence of its key in the `$_GET` superglobal.

```
    Number 2:
    <input type="text" name="number_two"/>
  </div>
  <div>
    <input type="submit" name="calculate" value="Calculate!"/>
  </div>
</form>
<?php
if (isset($_GET['calculate'])) {
}
```

In the conditional statement body, we can now perform the core logic

```

</div>
<div>
    <input type="submit" name="calculate" value="Calculate!"/>
</div>
</form>
<?php
if (isset($_GET['calculate'])) {
    $operation = $_GET['operation'];
    $numberOne = $_GET['number_one'];
    $numberTwo = $_GET['number_two'];
    if ($operation == "sum") {
        echo " == " . ($numberOne + $numberTwo);
    } else if ($operation == "subtract") {
        echo " == " . ($numberOne - $numberTwo);
    } else {
        echo " == Invalid operation supplied";
    }
}
}

```

And the output of the previous input is:

localhost/Scripting/calculator.php?operation=subtract&number\_one=55&num

Operation:

Number 1:

Number 2:

== 22

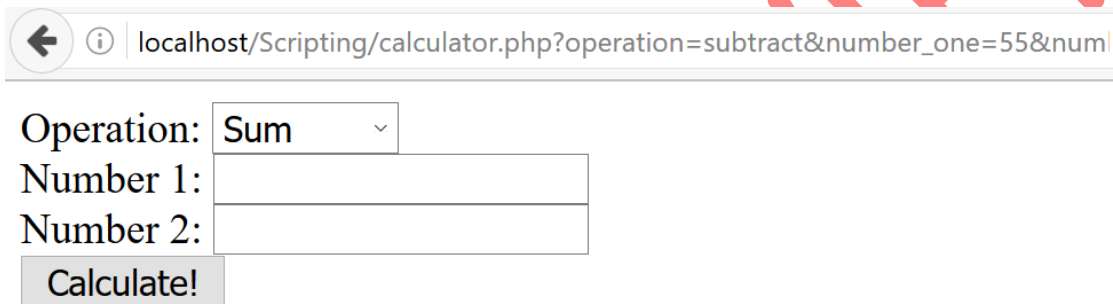
Whooila!

### 1. Code Quality

What if we want to output the result not prefixed by “==” but by a single bullet and in red color? Well, it’s not that hard. We just need to output the relevant HTML and the Web Browser will render it

```
</form>
<?php
if (isset($_GET['calculate'])) {
    $operation = $_GET['operation'];
    $numberOne = $_GET['number_one'];
    $numberTwo = $_GET['number_two'];
    echo "<ul>";
    if ($operation == "sum") {
        echo " <li style='color: red'> " . ($numberOne + $numberTwo) . "</li>";
    } else if ($operation == "subtract") {
        echo " <li style='color: red'> " . ($numberOne - $numberTwo) . "</li>";
    } else {
        echo " <li style='color: red'> Invalid operation supplied </li>";
    }
    echo "</ul>";
}
```

And its respective output:



← ⓘ localhost/Scripting/calculator.php?operation=subtract&number\_one=55&number\_two=33

Operation: Sum ▾

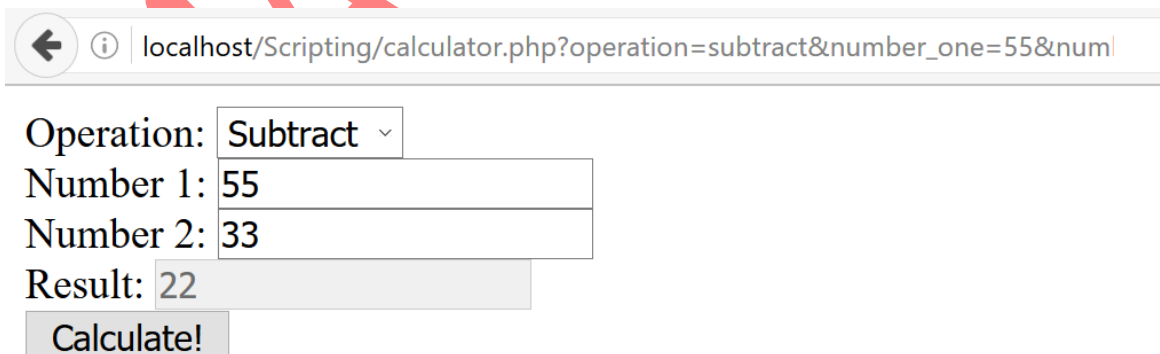
Number 1: 55

Number 2: 33

Calculate!

• 22

Good, eh? But **some of our HTML is wrapped in quotes, mixed in the PHP logic, making it harder to maintain.** It's not a single `<li>`, but what if it's the whole of your Facebook wall, with all the images, styles and videos? What if we want to output it to another text field (readonly) after the Number 2 field?



← ⓘ localhost/Scripting/calculator.php?operation=subtract&number\_one=55&number\_two=33

Operation: Subtract ▾

Number 1: 55

Number 2: 33

Result: 22

Calculate!

Well, we can write it that way:

```
</div>
<div>
    Number 2:
    <input type="text" name="number_two"/>
</div>
<?php
if (isset($_GET['calculate'])) {
    $operation = $_GET['operation'];
    $numberOne = $_GET['number_one'];
    $numberTwo = $_GET['number_two'];
    echo "<div>";
    echo "Result: ";
    if ($operation == "sum") {
        echo "<input type='text' disabled='disabled' readonly='readonly' value='". ($numberOne + $numberTwo)
    } else if ($operation == "subtract") {
        echo "<input type='text' disabled='disabled' readonly='readonly' value='". ($numberOne - $numberTwo)
    } else {
        echo "<input type='text' disabled='disabled' readonly='readonly' value='Invalid operation supplied'>
    }
    echo "</div>";
}
?>
<div>
    <input type="submit" name="calculate" value="Calculate!"/>
</div>
</form>
```

But **now it's totally unmaintainable**, with string concatenations and embedding big PHP logic between the HTML presentation.

The solution is called [Separation of Concerns](#).

**Never mix HTML and PHP!** Separate the logic and the HTML and only add PHP to the HTML when you need to output value, not whole dynamic HTML.

Let's create a PHP file that handles the `$_GET` request and renders the HTML. Using the `"include"` construction, one can embed one file's content in another file.

So our files might be called:

- `calculator.php`
- `calculator_frontend.php`

`calculator.php`:

```
<?php
if (isset($_GET['calculate'])) {
    $operation = $_GET['operation'];
    $numberOne = $_GET['number_one'];
    $numberTwo = $_GET['number_two'];
    $output = "";
    if ($operation == "sum") {
        $output = $numberOne + $numberTwo;
    } else if ($operation == "subtract") {
        $output = $numberOne - $numberTwo;
    } else {
        $output = 'Invalid operation supplied';
    }
}

include 'calculator_frontend.php';
```



Assigns output to the \$output variable and includes the content of the **calculator\_frontend** file, which should contain the HTML.

**calculator\_frontend.php:**

```
<form method="get">
    <div>
        Operation:
        <select name="operation">
            <option value="sum">Sum</option>
            <option value="subtract">Subtract</option>
        </select>
    </div>
    <div>
        Number 1:
        <input type="text" name="number_one"/>
    </div>
    <div>
        Number 2:
        <input type="text" name="number_two"/>
    </div>
    <?php if (isset($output)): ?>
        <div>
            Result:
            <input type="text" disabled="disabled" readonly="readonly" value="<?= $output; ?>"/>
        </div>
    <?php endif; ?>
    <div>
        <input type="submit" name="calculate" value="Calculate!"/>
    </div>
</form>
```

As you can see, we have opened PHP only for conditional statement and for echo statement. Once to check if \$output is set and once to print the value of \$output variable inside the HTML's value attribute.

The output is just the same:



 localhost/Scripting/calculator.php?operation=subtract&number\_one=55&num

Operation:

Number 1:

Number 2:

Result:

### Render Students Info

You are given an HTML form that contains three input fields. The first one is a dropdown containing delimiters (“,” “|”, “&”). The second one containing student names separated by the selected delimiter. The third one – student ages separated by the selected delimiter.

Delimiter:

Names:

Ages:

With the following HTML

```

<form method="get">
  <div>
    Delimiter:
    <select name="delimiter">
      <option value=","></option>
      <option value="|"></option>
      <option value="&"></option>
    </select>
  </div>
  <div>
    Names:
    <input type="text" name="names"/>
  </div>
  <div>
    Ages:
    <input type="text" name="ages"/>
  </div>
  <div>
    <input type="submit" name="filter" value="Filter!"/>
  </div>
</form>
  
```

You need to output an HTML table with headers “Name” and “Age” and each student as a row in this table

Delimiter:

Names:

Ages:

Name	Age
John	22
Peter	24
Wilma	16
Sarah	33

1. Separate the HTML and the PHP
2. Split the input from Names and Ages by the selected delimiter
3. Render a table only if names and ages arrays are initialized
4. Use foreach/endforeach or for/endfor to introduce new table rows

```

<?php if (isset($names, $ages)): ?>
    <table border="1" cellpadding="0">
        <thead>
            <tr>
                <th>Name</th>
                <th>Age</th>
            </tr>
        </thead>
        <tbody>
            <?php for ($i = 0; $i < count($names); $i++): ?>
                <tr>
                    <td><?= $names[$i]; ?></td>
                    <td><?= $ages[$i]; ?></td>
                </tr>
            <?php endfor; ?>
        </tbody>
    </table>
<?php endif; ?>

```

### Filter Legal Students

From the above task, render the HTML table only with these students are at least 18 years old.

### \* Paginate Students

Render max 5 students per page and two hyperlinks under the table – “Previous” and “Next”. The “Previous” hyperlink should not exist when there are no previous pages (e.g. we are on the first page) and the “Next” hyperlink should not exist when there are no next pages (e.g. when we are on the last page).

After clicking on “Next” hyperlink, the next chunk of 5 students is rendered. After clicking on “Previous” the previous chunk of 5 students is rendered.

HINT: Consider persisting the students' collection between requests, you may send it back with the GET request not-preferable, but is the only one taught so far) or persist it in a session/cookie (or even a file).

## \*\* Paginate Students (2)

Decorate more the last task. Between the hyperlinks "Previous" and "Next" render hyperlinks with pages. Display exactly 3 hyperlinks with pages. But be careful! In some ways there might not be 3 pages until the end of the collection, so you need to render the previous pages. The only exception of the rule is when there aren't 3 pages at all.

For instance, if we have 16 students, we will need 4 pages. This means on the first request we will render:

[1] [2] [3] [Next]

The red color means the page is the current page and cannot be clicked

The green color means the page is available for clicking

When clicking on "Next" or "2" the pager will be like:

[Previous] [2] [3] [4] [Next]

But when clicking on "3" there is no fifth page, so in order the slider to show exactly 3 pages, we need to make it remain the same way, just make "2" clickable in favor of "3":

[Previous] [2] [3] [4] [Next]

The next step is to go to the last page. Clicking on "4" will make the slider look like:

[Previous] [2] [3] [4]

The exception of the rule is when we have for example 9 students. The only thing we can do here is to render 2 pages (which is normal, no results for 3 pages ☺):

[1] [2] [Next]

[Previous] [1] [2]



MÃ SINH VIÊN