

# The system for testing different versions of the PHP

Mariia Yu. Tiahunova, Halyna H. Kyrychek and Yevhenii D. Turianskyi

National University “Zaporizhzhia Polytechnic”, 64 Zhukovskiy Str., Zaporizhzhia, 69063, Ukraine

## Abstract

The paper analyzes various versions of the popular PHP programming language. It is used in web development and there are many popular website engines and frameworks written in PHP. Many new and useful features of PHP 8, such as JIT compiler, error correction, etc., are described, which are useful for both users and developers. A testing system has been developed for different versions of PHP, which can be extended by other modules if necessary. The result of the system is time data reflecting the speed of the selected PHP version.

## Keywords

PHP, programming languages, testing system, speed

## 1. Introduction

Nowadays, there are many programming languages such as JavaScript, goLang, Java, etc. They are different in purpose thus their use is also limited. When we talk about web development languages, we can't mention PHP.

PHP is a widely used language for web implementations, which occupies more than 78% of the market for server solutions [1]. Its development began back in 1994 by Rasmus Lerdorf as a “Personal Home Page” for his own use and has continued to grow ever since [2].

PHP is an open source language, meaning users can modify or modify it. It can be used and distributed to other users and organizations.

PHP's syntax is similar to C. Elements such as associative arrays and the foreach loop are borrowed from Perl. The execution of the program does not require the description of variables, modules, or etc, but this language supports OOP (fully from version 5). The program begins with the operator `<?php`, which indicates the start of script execution. There is also a shortened version to output the string `<?=`. Execution ends with the `?>` operator, but if the entire file consists only of PHP code, then you do not need to use the closing operator.

Variable names begin with the `$` character, without specifying the type of the variable itself, and are case sensitive.

---

*doors-2023: 3rd Edge Computing Workshop, April 7, 2023, Zhytomyr, Ukraine*

✉ mary.tyagunova@gmail.com (M. Yu. Tiahunova); kirgal08@gmail.com (H. H. Kyrychek);

zhenya7990@gmail.com (Y. D. Turianskyi)


🌐 <https://csn.zntu.edu.ua/tyagunova-maria-uriivna> (M. Yu. Tiahunova);

<http://csn.zntu.edu.ua/galina-grigorivna-kirichek> (H. H. Kyrychek); <https://www.facebook.com/evjoni/> (Y. D. Turianskyi)

🆔 0000-0002-9166-5897 (M. Yu. Tiahunova); 0000-0002-0405-7122 (H. H. Kyrychek); 0000-0001-8109-7868 (Y. D. Turianskyi)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

When comparing PHP to JavaScript and its newer server-side implementations, PHP still comes out on top. PHP is interpreted by the web server into HTML code, which is transmitted to the client side. Unlike JavaScript, the user does not have access to the PHP code, which is an advantage from a security point of view, but significantly impairs the interactivity of the pages. However, nothing prohibits the use of PHP to generate JavaScript code that will be executed on the client side.

Many features are built into PHP that make it possible not to write them multi-line like in C or Pascal. There are libraries for working with many databases, such as MySQL, PostgreSQL, mSQL, Oracle, dbm, Hyperware, Informix, InterBase, Sybase, etc.

There is also the PHP Data Object – PDO module, which provides a simple and consistent interface for accessing databases, which has many methods for using databases and protection against SQL injections [3].

The attribute syntax consists of several parts. First, an attribute declaration always begins with the #[and ends]. Inside an enumeration of one or more comma-separated attributes. Attributes can be specified using partial, full, and absolute names.

Attribute arguments are optional, but if present, they are enclosed in braces (). Attribute arguments can be either specific values or constant expressions. Both positional and named argument syntax can be used for arguments. When an attribute is requested using the Reflection API, its name is interpreted as a class name, and arguments are passed to its constructor. Thus, for each attribute there must be a corresponding class.

However, there is a big problem with the PHP community. Many websites are still using outdated and no longer supported versions of PHP. According to W3Techs [4], 38.8% of websites still run on PHP 5.6 and below leading to worse performance, no security breach closing support releases and not having newly added functionality that can provide necessary improvements out of the box. The point of this article is to show the speed differences between the PHP versions as an additional argument to developer to upgrade and support existing web products.

This is achieved in work by developing a system, with the help of the results of which determine the most productive version of PHP among the specified ones. To achieve this the following tasks must be completed:

- analyze the features of different versions PHP;
- design a performance analysis system;
- select mathematical and other operations for performance research;
- implement the system;
- analyze the performance of PHP of different versions on selected operations and visualize the obtained results;
- analyze the obtained results.

## **2. Language versions and their features**

The keyword class was reserved even in the third version of the language, in the fourth it was already possible to create classes and objects of these classes. Nevertheless, until the fifth version, the principles of OOP were not fully implemented, since all methods and properties were open privacy and were not a cheap wish on the part of the runtime.

The use of classes in PHP is similar to C, a class is declared by the keyword `class`, it can have properties and methods of a certain privacy, namely `public`, which are available from anywhere, `protected`, which the class itself and its successors have access to, and `private`, which are not no one has access except the class itself. PHP supports all OOP mechanisms such as encapsulation, polymorphism, and inheritance. Also the use of such keywords as `final`, `abstract`, `extend`, `implement` [5].

Any class can have many interfaces that it implements, but can inherit only one of the classes. To solve this problem, version 5.4.0 introduced a code reuse mechanism called `trait`. Many traits can be used in a class using the `use` word, but it is not possible to get a trait object. At runtime, the trait's code will be "mounted" to the class that uses it.

PHP 5 [6] was released back in 2004 and runs on the new Zend Engine II. The Zend engine is a set of components that make PHP. The most important component of Zend is the Zend Virtual Machine, which includes the Zend Compiler and Zend Executor components. We can also add the `OPCache` zend extension to the same category. These three components are the core of PHP, they are the critical and most complex parts of the source code. The Zend engine behind the interpreter has been completely redesigned in Zend Engine 2, paving the way for further improvements. PHP 5 included new features such as improved OOP support, PHP data objects, also known as PDO extensions (a lightweight and consistent database access interface), and much more.

Starting with PHP 8.0.0, constructor parameters can be used to set the appropriate properties of an object. It is a fairly common practice to give object properties the parameters passed to the constructor without doing any additional transformations. In this case, defining the properties of the class in the constructor allows you to significantly reduce the amount of template code.

Starting with PHP 8.0.0, properties and methods can also be accessed using the "nullsafe" operator: `?->`. The nullsafe operator works in the same way as a property or method access, except that if the variable that contained an object of class actually returns null, then null is returned instead of an exception being thrown. If the dereference is part of a string, the rest of the string is skipped. Similar to the conclusion of each call in `is_null()`, but more compact.

Performance improvements continued: the hash table (PHP's main data structure) was optimized, specialization of certain opcodes in the Zend VM and specialization of certain sequences in the compiler were implemented, the optimizer (`OpCache` component) was continuously improved, and many other changes were made.

According to PHP developer Dmytro Stogov: "JIT is extremely simple, but in any case it increases the level of complexity of PHP, the risk of new bugs appearing, and the cost of development and maintenance" [7]. The proposal to introduce JIT to PHP 8 received 50 votes out of 52 [7]. A short list of additional approved improvements that are included in PHP 8 is given in Zandstra [8], Prettyman [9].

To carry out complex testing, it is necessary to develop a system that would perform operations that require only one computer resource. These components are the CPU and hard disk. The results of the tests will be time indicators, by which it will be possible to compare different versions of the PHP language. Accordingly, shorter execution time is better [10].

### 3. Development of a testing system

#### 3.1. Choosing a web server

To implement the system as a web page, it is necessary to have a web server that would perform the necessary actions on a request by URL address.

Apache and Nginx were considered among the local servers. These are the two most common open source web servers in the world. Together, they serve more than 50% of the world's traffic. Both solutions are able to work with different work programs and interact with other applications to implement a complete web stack. Although Apache and Nginx have many similar qualities, they cannot be considered as completely interchangeable solutions, as each has its own capabilities.

Apache provides several multi-processing modules (MPM), which are responsible for how the client's request will be processed. This allows administrators to define connection handling policies [11].

Nginx came later than Apache, so its developer was more aware of the competitive issues that sites face when scaling. Thanks to this knowledge, Nginx was originally designed based on asynchronous non-blocking event-driven algorithms. Nginx creates worker processes, each of which can serve thousands of connections. Workers achieve this result thanks to a mechanism based on a fast cycle in which events are checked and processed. Separating the main work from connection processing allows each Worker to do its job and be distracted from connection processing only when a new event occurs.

Looking at real-life examples, the main differences between Apache and Nginx are how they handle requests to static and dynamic content. Apache can serve static content using standard file-based methods. Productivity of such operations depends on the selected MPM [12].

Nginx does not have the ability to handle dynamic content requests on its own. To handle requests to PHP or other dynamic content, Nginx must pass the request to an external processor for execution, wait for the response to be generated, and receive it. The result can then be sent to the client.

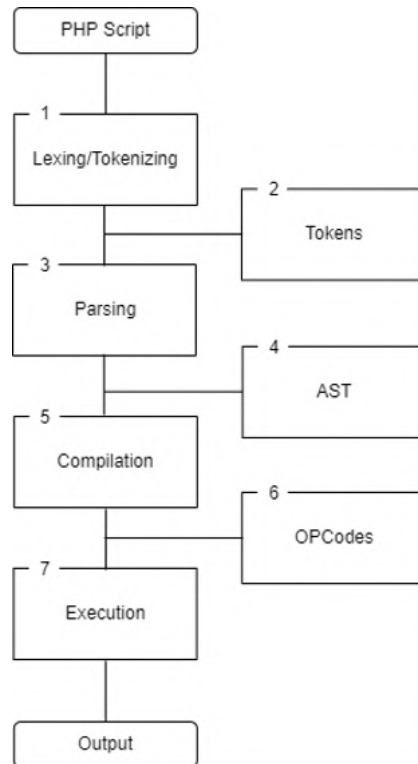
#### 3.2. System design

To test different versions of PHP, it is necessary to implement a system that includes classes that would present each type of test.

The following image (figure 1) shows a visual representation of the basic PHP execution process.

PHP execution is a 4-step process [13]:

- lexing/tokenization – the interpreter reads the code and creates a set of tokens;
- parsing – the interpreter checks whether the script matches the syntax and uses tokens to build an abstract syntax tree, a hierarchical representation of the PHP source code structure;
- compilation – the interpreter traverses the tree and translates the AST nodes into low-level Zend opcodes, which are numeric identifiers that define the types of instructions executed by the Zend virtual machine;



**Figure 1:** Representation of the main execution process.

- interpretation – opcodes are interpreted and run on the Zend VM.

OPcache improves PHP performance by storing precompiled script bytes in shared memory, eliminating the need to load and parse PHP scripts for each request (figure 2).

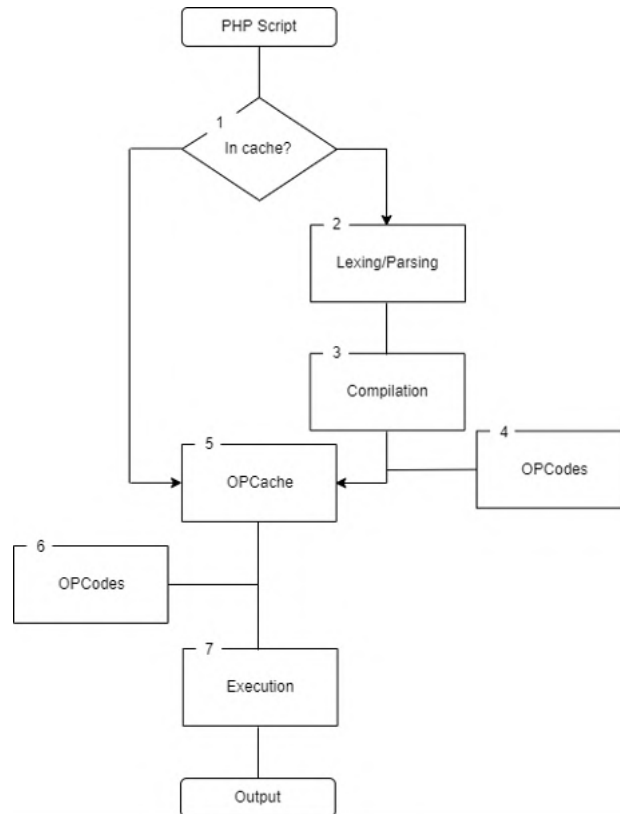
The project does not require any frameworks. Structurally, the project will consist of a configuration file, a class for each module to be tested, a class for displaying results on the screen, and an executable file – the starting point.

The script can be called both from the console and as a web page. Calling the initial file will in turn call an object of the Benchmark class, which starts the initialization of information about the enabled modules for testing. Next, all the modules will be performed one by one with the output of the results of their work [14].

To implement the project and meet the need for future scaling, we will broadly plan the project tree. It will consist of a configuration directory, modules, and the home classes for those modules.

First in line is the configuration file. It contains input data about the requested modules to be executed, as well as parameters to them. In this way, we can enable, disable or expand the functionality of the project without changing the existing code. Those modules considered to be CPU and Disk operations since they are considered as heavy one's and access to which was changed from version to version of PHP.

Among the functions there were selected 12 math operations and 12 string conversions.



**Figure 2:** Algorithm for improving PHP performance using OPcache.

Those are `abs`, `acos`, `asin`, `atan`, `bindec`, `floor`, `exp`, `sin`, `tan`, `is_finite`, `is_nan`, `sqrt` for math and `addslashes`, `chunk_split`, `metaphone`, `strip_tags`, `md5`, `sha1`, `strtoupper`, `strtolower`, `strrev`, `strlen`, `soundex` and `ord` for string conversion.

The next section is the directory of the actual project. The first owned folder contains the `Config` class, which is responsible for receiving from it the data obtained from the `YamlConfigLoader` – the configuration file parsing class.

The following directory contains helper classes for system operation, such as:

- `Directory` – creates a temporary directory during tests of writing files to disk and deletes them after the test;
- `Size` – contains information about file weight units, as well as byte and format conversion methods;
- `Table` – exists for the generating table borders for the presentation view;
- `Utility` – implements the function of dynamically adding an argument for variable SQL operations;
- `Visual` – a separate class designed to display information with a line separator parameter.

The following directory contains the modules that will be executed, namely:

- CPU – contains the logic of performing arithmetic operations, converting lines, calculating the execution time of cycles and if-else expressions.
- Disk – creates files of a given weight, writes them to a temporary directory and deletes them after time measurements.
- MySQL – a class for implementing and calculating the execution time of a random operation.
- PHP – providing information about the version of PHP.

Among the functions that will be executed by the CPU are such string conversion functions as: `addslashes`, `chunk_split`, `metaphone`, `strip_tags`, `md5`, `sha1`, `strtoupper`, `strtolower`, `strrev`, `strlen`, `soundex`, `ord`.

Among the functions that will be performed by the CPU are such mathematical functions as: `abs`, `acos`, `asin`, `atan`, `bindec`, `floor`, `exp`, `sin`, `tan`, `is_finite`, `is_nan`, `sqrt`.

The features identified are chosen because of their prevalence of use among their categories.

The following file volumes were written to disk: 512, 1024, 2048, 4096, 8192, 16384, 32678, 65536.

Next is the Benchmark class, which includes all the modules and starts their execution.

In the root directory there is a file – the starting point for performing all tasks.

To use the project, you need to have an installed Composer – the application level package manager for PHP. With its help, we can use ready-made package solutions, so as not to implement them ourselves from scratch [15].

All defined packages, after they are requested by the composer, are placed in the vendor directory, which should not be added to Git commits, since the vendor folder can weigh many gigabytes. To work around this flaw, composer generates a `composer.json` file and a `composer.lock` file.

Json contains project dependencies for execution and is converted to a lock file, which is actually executed by the composer. Thus, only one file is needed to “transfer” the vendor, after cloning, or whatever, the project, we will only need to execute `composer install` and we will get all the dependencies to our local project.

You can get the results either by executing the `benchmark.php` file in the terminal as `php benchmark.php`, or by running a local server, on the web page of which we will see the same results – `php -S localhost:8000 benchmark.php`.

## 4. Implementation of the designed system

### 4.1. Project tree

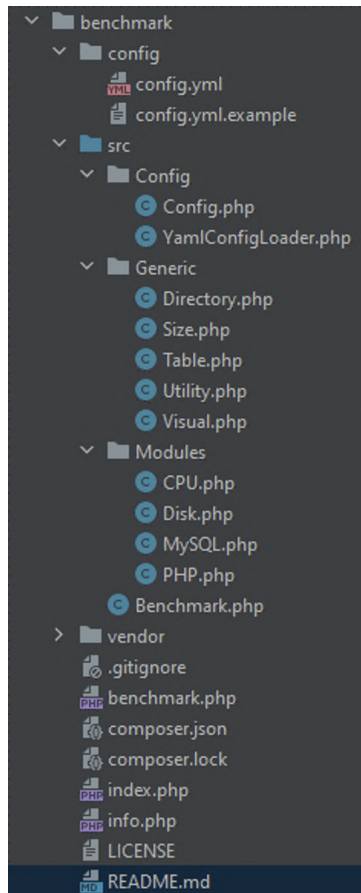
The project tree looks like this (figure 3):

Since an OOP approach was chosen for the implementation, all files are located according to their purpose and have namespaces to uniquely define each class.

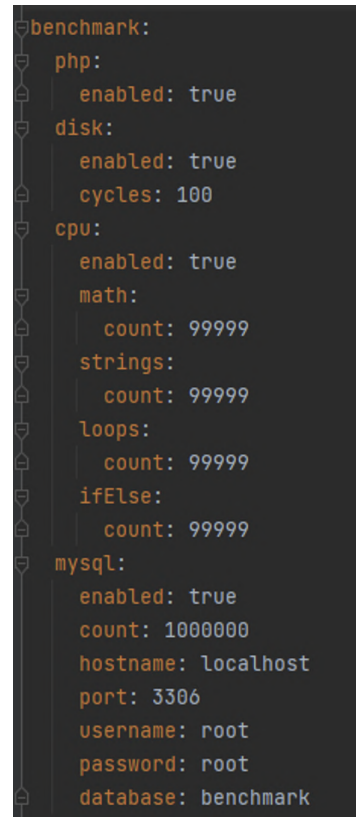
The configuration file has the following structure (figure 4). It allows us to define each module’s parameters such as:

- Enabled – defines whether the module will be used during the benchmark;

- Disk cycles – the amount of read/write operations;
- CPU math/strings/loops/ifElse count – the amount of recurring operations;
- MySQL – describes the access to the database, it's name and amount of allowed operations.



**Figure 3:** Structure of the project.



**Figure 4:** Configuration file.

As you can see, we can set the number of repetitions of each operation, enable or disable any module, and set database connection options.

## 4.2. Implementation of classes

The Config class has several methods, but the main one is constructor initialization.

Initialization of the constructor of the Config class:

```
public function __construct() {
    $locator = new FileLocator([
        __DIR__ . DS . '..' . DS . '..' . DS . $this->directory
    ]);
    try {
```



```

        $loader = new YamlConfigLoader($locator);
        $this->config = $loader->load(
            $locator->locate($this->file)
        );
    } catch (\Exception $e) {
        Visual::print($this->file . ' could not be loaded, please copy
        ' . $this->directory . '/config.yml.example to ' .
        $this->directory . '/' . $this->file);
    }
}

```

The result of this method is an object of the Config class with certain properties obtained using the YamlConfigLoader. The properties of this class will be used many times in the future.

### 4.3. Implementation of mathematical operations

Next, the modules and their supporting classes should be implemented. The first module is the CPU. In order not to implement all mathematical functions manually, they can be called in a loop. The implementation of mathematical operations is given below.

Method of execution of mathematical functions:

```

private function math(): void {
    foreach (self::$mathFunctions as $function) {
        $this->mathResults['x'][$function] = 0;
        $start = \microtime(true);
        for ($i = 0; $i < $this->mathCount; $i++) {
            \call_user_func_array($function, array($i));
        }
        $this->mathResults['x'][$function] +=
            (\microtime(true) - $start);
    }
}

```

As a result of its execution, we will receive an array with the names of operations and their time indicators. The same goes for string conversion functions.

Method for performing string conversion functions:

```

private function strings(): void {
    foreach (self::$stringFunctions as $function) {
        $this->stringsResults['x'][$function] = 0;
        $start = \microtime(true);
        for ($i = 0; $i < $this->stringsCount; $i++) {
            \call_user_func_array($function, array(self::$string));
        }
        $this->stringsResults['x'][$function] +=

```

```

        (\microtime(true) - $start);
    }
}

```

There are also separate implementations for checking loops and logical operators.  
Method for checking the speed of cycles:

```

private function loops(): void {
    $start = \microtime(true);

    for ($i = 0; $i < $this->loopsCount; ++$i);
    for ($i = 0; $i < $this->loopsCount; ++$i);

    $this->loopsResults = (\microtime(true) - $start);
}

```

The speed test method of the IfElse statement:

```

private function ifElse(): void {
    $start = \microtime(true);

    for ($i = 0; $i < $this->ifElseCount; $i++) {
        if ($i === -1) {
            ;
        } elseif ($i === -2) {
            ;
        } else if ($i === -3) {
            ;
        }
    }

    $this->ifElseResults = (\microtime(true) - $start);
}

```

As a result of their execution, we will receive their results as properties of the CPU class.  
The next module is Disk. It contains a list of weights for the files being created.  
A property of the Disk class that describes file volumes:

```

private $commonBlockSizesBytes = [
    512,
    1024,
    2048,
    4096,
    8192,
    16384,

```

```

        32678,
        65536,
    ];

```

The main method of the Disk class:

```

private function run(): void {
    $initial = \time();
    $tmpDirectoryPath = \realpath(self::$path) . self::$tmpDirectory;

    try {
        // Create subdirectory
        Directory::create($tmpDirectoryPath);
        foreach ($this->commonBlockSizesBytes as $bytes) {
            $this->counterFileCreation['Run'][$bytes] = 0;
        }
        for ($c = $this->cycles; $c >= 0; $c--) {
            // Generate files with different block sizes
            foreach ($this->commonBlockSizesBytes as $bytes) {
                $prefix = $initial . '_' . $bytes;
                $content = $this->getRandomBytes($bytes);
                // Start the timer (measure only disk interaction,
                //not string generation etc)
                $start = \microtime(true);
                $file = \tempnam($tmpDirectoryPath, $prefix);
                \file_put_contents($file, $content);
                // Stop timer & append time to timer array
                // with this block size
                $this->counterFileCreation['Run'][$bytes] +=
                    (\microtime(true) - $start);
            }
        }
        // Clean up
        Directory::removeRecursively($tmpDirectoryPath);
    } catch (\Exception $e) {
        Visual::print($e);
    }
}

```

Getting a given weight file:

```

private function getRandomBytes($bytes): string {
    return random_bytes($bytes);
}

```

When this code is executed, a temporary directory will be created in which the files with the tested weight size will be written. For each weight, a file is generated and written to disk, and the time used is added to the array along with the corresponding weight. The files are then deleted.

But since we have more than one file, the task of recursive deletion appears. To do this, we will create an auxiliary class Directory. It implements methods for creating a temporary directory and deleting files and the directory itself after they are written to disk. Their code is given below.

Recursive deletion of generated files and their directories:

```
public static function removeRecursively($path): void {
    if (\file_exists($path)) {
        $dir = \opendir($path);
        if (\is_resource($dir)) {
            while ($file = \readdir($dir)) {
                if (($file !== self::$rootPath) &&
                    ($file !== self::$parentPath)) {
                    $full = $path . DS . $file;
                    if (\is_dir($full)) {
                        self::removeRecursively($full);
                    } else {
                        \unlink($full);
                    }
                }
            }
            \closedir($dir);
        }
        \rmdir($path);
    }
}
```

Creating a temporary directory:

```
public static function create($path, int $permissions = 0755): bool {
    if (!\file_exists($path) && !mkdir($path, $permissions) &&
        !is_dir($path)) {
        throw new \RuntimeException('Could not create directory: ' . $path);
    }
}
```

In the following code, we open the directory and delete the path if it is a file, then exit the directory and delete the path itself.

Next is MySQL class. Its main methods are getting the current version of MySQL and executing an encoded random string.

Getting the current version of MySQL:

```
private function getVersion(): void {
    $this->version = \mysqli_get_server_version($this->connection);
}
```

Executing a random encoded string:

```
private function encodeRand(): void {
    $query = Utility::format($this->benchmarkQuery, [
        $this->config->get('benchmark.mysql.count'),
        $this->benchmarkText
    ]);
    $start = \microtime(true);
    \mysqli_query($this->connection, $query);
    $this->queryResults = (\microtime(true) - $start);
}
```

The next class is PHP. It implements the functions of obtaining the current parameters of the PHP environment. These methods include getting preloaded extensions, getting maximum file size before uploading, and memory limit.

Getting the maximum size of the uploaded file:

```
private function getMaxUploadBytes(): int {
    static $max_size = -1;
    if ($max_size < 0) {
        // Start with post_max_size.
        $post_max_size = Size::formatToBytes(\ini_get('post_max_size'));
        if ($post_max_size > 0) {
            $max_size = $post_max_size;
        }
        // If upload_max_size is less, then reduce. Except if upload_max_size
        // is zero, which indicates no limit.
        $upload_max = Size::formatToBytes(\ini_get('upload_max_filesize'));
        if ($upload_max > 0 && $upload_max < $max_size) {
            $max_size = $upload_max;
        }
    }
    return $max_size;
}
```

```
private function getMaxMemoryBytes(): int {
    return (int)Size::formatToBytes(\ini_get('memory_limit'));
}
```

To count the input values specified in the ini-file, a separate Size class was created, which has the following methods.

Converting bytes to format and vice versa:

```

public static function bytesToFormat(int $bytes): string {
    $power = $bytes > 0 ? \floor(\log($bytes, 1024)) : 0;
    return \number_format($bytes / (1024 ** $power), 2, '.',
        ',') . ' ' . self::$units[$power];
}

public static function formatToBytes(string $format): float {
    $unit = \preg_replace(self::$unitsRegexPattern, '', $format);
    $format = \preg_replace(self::$numberRegex, '', $format);
    return $unit ? \round($format * (1024 **
        \stripos(self::$unitsPattern, $unit[0]))) : \round($format);
}

```

Thus, we obtained the maximum permissible file weight, which is specified in the configuration file of the executable version of PHP.

In the root directory there is a file - the starting point for performing all tasks, which contains the following code:

The main executable:

```

<?php
use DI\ContainerBuilder;
use DI\DependencyException;
use DI\NotFoundException;
use Benchmark\Benchmark;
const DS = DIRECTORY_SEPARATOR;
require 'vendor' . DS . 'autoload.php';
echo '<pre>';
try {
    (new ContainerBuilder())->build()->get(Benchmark::class);
} catch (DependencyException | NotFoundException $e) {
    \print_r($e);
}
echo '</pre>' . "\n";

```

As a result of this section, a system has been developed, the result of which is the time intervals of the performed operations, which are presented in a visual form. An example of the obtained result is given below (figure 5).

The table contains time measured results for each operation in seconds, lower is better.

#### 4.4. Implementation of visualization of results

There is a separate method for displaying information that uses a method from another class, namely line representation, to improve reading comprehension. Its implementation is given below.

The method of outputting the resulting information:

```

== Generic PHP information

PHP version: 5.6.40-54+ubuntu20.04.1+deb.sury.org+1
Server: PHP 5.6.40-54+ubuntu20.04.1+deb.sury.org+1 Development Server
Maximum execution time: 30 seconds

Maximum memory size: 1.00 B (1 bytes)
Maximum upload size: 2.00 MB (2097152 bytes)

Modules loaded:

== Disk performance information
Results sorted by file size (in bytes) in milliseconds (less is better), for a total of 100 cycles:
+-----+-----+-----+-----+-----+-----+-----+-----+
| 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0.0077745914459229 | 0.0070915222167969 | 0.0076439380645752 | 0.0086843967437744 | 0.0097451210021973 | 0.011658906936646 | 0.013409852981567 | 0.017565727233887 |
+-----+-----+-----+-----+-----+-----+-----+-----+

== CPU performance information
Math operation results by function in milliseconds (less is better), for a total of 99999 cycles:
+-----+-----+-----+-----+-----+-----+-----+-----+
| abs | acos | asin | atan | bindec | floor | exp | sin | tan |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0.028146982192993 | 0.027643918991089 | 0.028227090835571 | 0.027456998825073 | 0.033526182174683 | 0.026874780654907 | 0.029170989990234 | 0.030976057052612 | 0.032726049423218 |
+-----+-----+-----+-----+-----+-----+-----+-----+

String operation results by function in milliseconds (less is better), for a total of 99999 cycles:
+-----+-----+-----+-----+-----+-----+-----+-----+
| addslashes | chunk_split | metaphone | strip_tags | md5 | sha1 | strtoupper | strtolower | strrev |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0.041321039199829 | 0.036525964736938 | 0.044986009597778 | 0.04217791557312 | 0.046206951141357 | 0.049161911010742 | 0.03412389755249 | 0.034275054931641 | 0.032958030700684 |
+-----+-----+-----+-----+-----+-----+-----+-----+

Loop operation results in milliseconds (less is better), for a total of 99999 cycles: 0.0038131340026855
If/Else operation results in milliseconds (less is better), for a total of 99999 cycles: 0.0036921501159668

== MySQL performance information
MySQL version:
Query (Encode + random) operation results in milliseconds (less is better), for a total of 100000 cycles: 4.5061111450195E-5

```

**Figure 5:** Representative table of the results of operations.

```

public static function print(string $input, string $delimiter = "\n\n"):
void {
    \print_r($input . $delimiter);
}

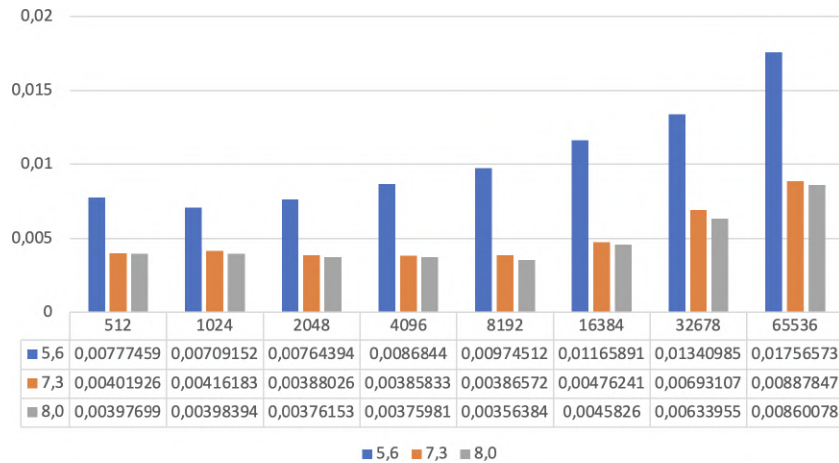
```

## 5. Analysis of the obtained results

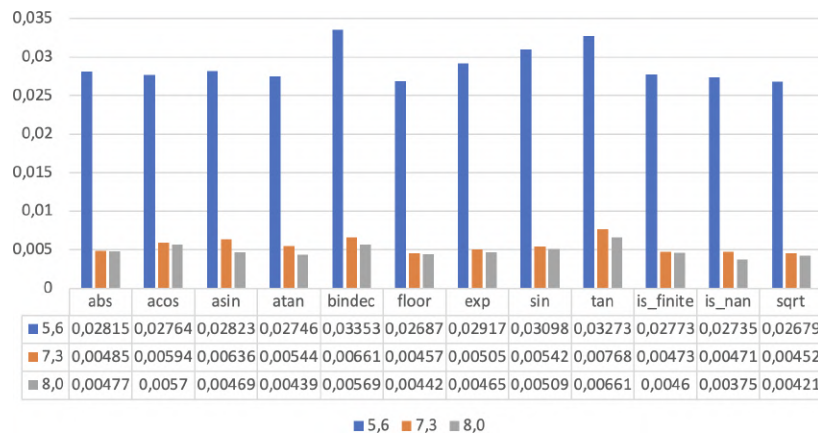
Benchmarks are used to achieve a competitive result with any other similar item. In our case, the difference is the version of PHP for the fastest execution. All results were entered into an Excel table, where comparative graphs were constructed.

Tests were conducted on the performance of disk (figure 6) and processor operations (figure 7) and string conversion depending on the language version (figure 8). The results of the analysis are shown below and represent the operations for each version of PHP, to identify the fastest. For all tests, a lower value is better. Tests were performed on a local machine with: Ryzen 5600H, 2x8 GB 3200 MHz CL22 PC4-25600 RAM, Samsung M.2 512 GB SSD, Ubuntu 21.04 [16]. PHP versions used: 5.6.40, 7.3.31, 8.0.1

Checking the speed of cycle operations (figure 9) and speed test of If/Else statements (figure 10).



**Figure 6:** Disk write tests (seconds, lower is better).

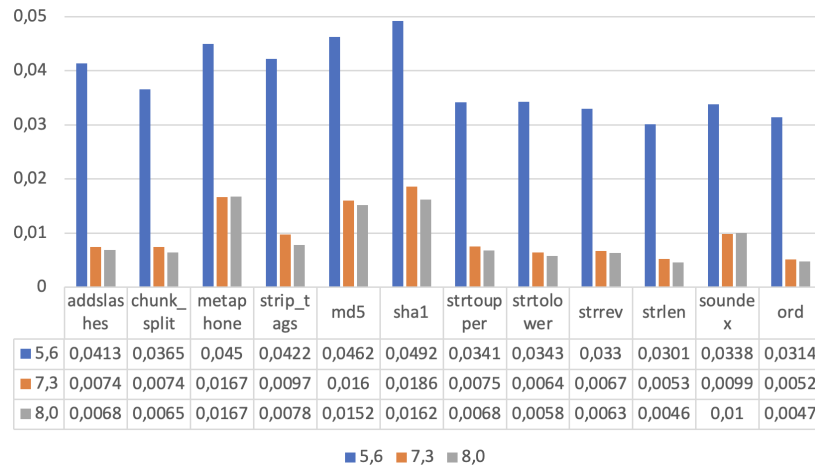


**Figure 7:** CPU math operations tests (seconds, lower is better).

## 6. Conclusion

PHP will continue to be a popular language for the foreseeable future. It is used in web development, and there are many popular website engines and frameworks written in PHP. PHP 8 has become faster and more reliable. Compared to previous versions, PHP 8 has many new and useful features, such as JIT compiler, bug fixes, etc., which will definitely benefit both users and developers. A benchmark system has been developed for different versions of PHP, which can be extended by other modules if necessary. The result of the system's operation is time data reflecting the speed of the selected PHP version. These results might not convince developers to keep PHP version up-to-date, but may give a focus on improvements they might seek to achieve using older versions.





**Figure 8:** CPU string conversion tests (seconds, lower is better).

Loop operation results in milliseconds (less is better), for a total of 99999 cycles:	
5,6	0.0030131340026855
7,3	0.0012149810791016
8,0	0.00072503089904785

If/Else operation results in milliseconds (less is better), for a total of 99999 cycles:	
5,6	0.0036921501159668
7,3	0.0016400814056396
8,0	0.0012068748474121

**Figure 9:** Checking the speed of cycle operations. **Figure 10:** Speed test of If/Else statements.

## References

- [1] M. Laaziri, K. Benmoussa, S. Khouliji, M. L. Kerkeb, A Comparative study of PHP frameworks performance, *Procedia Manufacturing* 32 (2019) 864–871. doi:10.1016/j.promfg.2019.02.295, 12th International Conference Interdisciplinarity in Engineering, INTER-ENG 2018, 4–5 October 2018, Tirgu Mures, Romania.
- [2] S. I. Adam, S. Andolo, A New PHP Web Application Development Framework Based on MVC Architectural Pattern and Ajax Technology, in: *2019 1st International Conference on Cybernetics and Intelligent System (ICORIS)*, volume 1, 2019, pp. 45–50. doi:10.1109/ICORIS.2019.8874912.
- [3] I. Fedorchenko, A. Oliinyk, A. Stepanenko, T. Zaiko, S. Shylo, A. Svyrydenko, Development of the modified methods to train a neural network to solve the task on recognition of road users, *Eastern-European Journal of Enterprise Technologies* 2 (2019) 46–55. doi:10.15587/1729-4061.2019.164789.
- [4] K. I. Bagwan, S. Ghule, A Modern Review On Laravel - PHP Framework, *Iconic Research And Engineering Journals* 2 (2019) 1–3. URL: <https://www.irejournals.com/paper-details/1701266>.
- [5] M. O’Leary, PHP, in: *Cyber Operations: Building, Defending, and Attacking Modern Computer Networks*, Apress, Berkeley, CA, 2019, pp. 983–1037. doi:10.1007/

978-1-4842-4294-0\_20.

- [6] A. P. Adi, Panduan Cepat Belajar HTML, PHP, dan MySQL, Elex Media Komputindo, 2020. URL: <https://openlibrary.telkomuniversity.ac.id/home/catalog/id/161999/slug/panduan-cepat-belajar-html-php-dan-mysql.html>.
- [7] C. Daniele, What's New in PHP 8 (Features, Improvements, and the JIT Compiler), 2022. URL: <https://kinsta.com/blog/php-8/>.
- [8] M. Zandstra, PHP 8 Objects, Patterns, and Practice: Mastering OO Enhancements, Design Patterns, and Essential Development Tools, Springer, 2021.
- [9] S. Prettyman, Learn PHP 8: Using MySQL, JavaScript, CSS3, and HTML5, Apress, Berkeley, CA, 2020.
- [10] M. Y. Tiahunova, H. H. Kyrychek, T. O. Bohatyrova, D. D. Moshynets, System and method of automatic collection of objects in the room, CEUR Workshop Proceedings 3077 (2021) 174–186. URL: <https://ceur-ws.org/Vol-3077/paper10.pdf>.
- [11] R. Yenduri, M. Al-khassaweneh, PHP: Vulnerabilities and Solutions, in: 2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), 2022, pp. 391–396. doi:10.1109/MIUCC55081.2022.9781790.
- [12] M. Tiahunova, O. Tronkina, G. Kirichek, S. Skrupsky, The Neural Network for Emotions Recognition under Special Conditions, in: S. Subbotin (Ed.), Proceedings of The Fourth International Workshop on Computer Modeling and Intelligent Systems (CMIS-2021), Zaporizhzhia, Ukraine, April 27, 2021, volume 2864 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 121–134. URL: <https://ceur-ws.org/Vol-2864/paper11.pdf>.
- [13] G. Engebret, PHP 8 Revealed: Use Attributes, the JIT Compiler, Union Types, and More for Web Development, Apress, Berkeley, CA, 2021. doi:10.1007/978-1-4842-6818-6.
- [14] H. Su, L. Xu, H. Chao, F. Li, Z. Yuan, J. Zhou, W. Huo, A Sanitizer-centric Analysis to Detect Cross-Site Scripting in PHP Programs, in: 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), 2022, pp. 355–365. doi:10.1109/ISSRE55969.2022.00042.
- [15] S. Semerikov, A. Striuk, L. Striuk, M. Striuk, H. Shalatska, Sustainability in Software Engineering Education: a case of general professional competencies 166 (2020) 10036. doi:10.1051/e3sconf/202016610036.
- [16] R. Dharsan, M. Krishanthini, C. Traveena, L. Anubama, D. I. D. Silva, D. Thisuru, Analyzing PHP Project - Medicare, International Journal of Engineering and Management Research 12 (2022) 432–440. doi:10.31033/ijemr.12.5.55.