FITPED

PHP fundamentals

Jozef Kapusta (UP)
Ľubomír Benko (UKF)
Zenón José Hernández-Figueroa (ULPGC)
José Daniel González-Domínguez (ULPGC)
Juan Carlos Rodríguez-del-Pino (ULPGC)
Ján Skalka (UKF)
Dominik Halvoník (UKF)
Arkadiusz Nowakowski (UŚ)
Tomáš Hála (MENDELU)

www.fitped.eu

2021

Co-funded by the Erasmus+ Programme of the European Union

Work-Based Learning in Future IT Professionals Education
(Grant. no. 2018-1-SK01-KA203-046382)

PHP Fundamentals

Published on

November 2021

Authors:

Jozef Kapusta | Pedagogical University of Cracow, Poland

Ľubomír Benko | Constantine the Philosopher University in Nitra, Slovakia

Zenón José Hernández-Figueroa | University of Las Palmas de Gran Canaria, Spain

José Daniel González-Domínguez | University of Las Palmas de Gran Canaria, Spain

Juan Carlos Rodríguez-del-Pino | University of Las Palmas de Gran Canaria, Spain

Ján Skalka | Constantine the Philosopher University in Nitra, Slovakia

Dominik Halvoník | Constantine the Philosopher University in Nitra, Slovakia

Arkadiusz Nowakowski | University of Silesia in Katowice, Poland

Tomáš Hála | Mendel University in Brno, Czech Republic

Reviewers:

Martin Drlík | Constantine the Philosopher University in Nitra, Slovakia

Cyril Klimeš | Mendel University in Brno, Czech Republic

Piet Kommers | Helix5, Netherland

Eugenia Smyrnova-Trybulska | University of Silesia in Katowice, Poland

Peter Švec | Teacher.sk, Slovakia

Graphics

Ľubomír Benko | Constantine the Philosopher University in Nitra, Slovakia

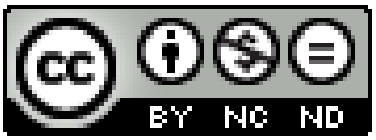
David Sabol | Constantine the Philosopher University in Nitra, Slovakia

Erasmus+ FITPED

Work-Based Learning in Future IT Professionals Education

Project 2018-1-SK01-KA203-046382

Co-funded by the
Erasmus+ Programme
of the European Union



ISBN 978-80-558-1789-7

Table of Contents

PHP Introduction

Chapter 1

1.1 Client-side vs. server-side programming

📖 1.1.1

A **script**, in general, is a **series of commands** (programs) or instructions executed in another program or application. Scripting is typical for web applications where HTML language is a stem. It is basically the only universal language that all web browsers understand and work with. Because of the relatively limited capabilities of HTML language, it is often supplemented by scripts (i.e. parts of the code directly written to the HTML page), but must be interpreted separately.

Two basic principles can be applied for programming web applications. They are the so-called **client-side** and **server-side programming**. The main difference between the two approaches is in the way scripts are interpreted. Server-side scripts are interpreted on the server. It means that a language interpreter must be installed on the server to generate HTML code from the script and send it to the client. By client, we mean mainly a web browser, i.e. Mozilla Firefox, Opera, etc. In client-side programming, the entire source code is sent if it is requested by the client and it is then interpreted using a web browser. Although it may seem at first glance that it is not important who actually interprets the script, this way of interpreting significantly limits or favours all operations that can be executed on a website. In practice, it is necessary to analyze what I expect from the developed website and then, according to these requirements, choose an appropriate approach to programming and interpretation.

It should also be noted that the method of interpretation is determined by the programming language used. Well, if the programmer chooses a programming language in which will create the functionality of the website, so he actually chose the way of its interpretation and thus decided to program on the client or server-side. Technologies that provide searching, interpretation, loading, and displaying a web page can tell, by the end of the file, how the script should be interpreted.

Given that each approach has many advantages as well as disadvantages, the current trend is a combination of both approaches.

📝 1.1.2

What approach are we talking about when the source script is interpreted by the web browser?

- Client-side programming
- Server-side programming

📖 1.1.3

Both approaches (client vs. server scripting) differ mainly in the following areas:

1. Source code visibility

In server-side programming, the script is interpreted and results in an HTML page. The user does not see the source code of the script, he can only see the result (web page), the script is visible only to the servers. In the case of client-side programming, the web client receives all source code from the server, which then interprets it. Therefore, the user can easily view the code. This is particularly appreciated by users learning how to script, as they can learn from real solutions.

2. Code processing

While in client programming web browser is sufficient and moreover, we can also browse pages with scripts (i.e. from USB, CD, etc.), for scripts written in languages for programming on the server side it is necessary to connect to the Internet and connection to the server where the page is with the interpreter (this is automatically provided by the web browser). Creating local versions of applications that need an interpreter on a web server is impossible for the average user.

3. Efficiency

In practice, both approaches are often combined. Server-side programming allows us to create more complicated pages, with the ability to personalize and connect to a database. However, many problems can be solved directly on the client without the need to load the server. The advantage of client interpretation is speed, i.e. operations such as form validation, visual effects, etc. can be solved quickly, as each client can interpret them alone without a server. Client-side scripting requires no interaction with the server.

4. Security

Because the code interpreted from the server is sent only as generated HTML when programming on the server-side, application security is only a question of developer skill. In the case of client scripts, where practically "the whole world" can see the source code, security is questionable.

📝 1.1.4

Choose which access is best for the following applications:

(a) I want a simple check to see if the quick form elements are filled.

_____

(b) I need a database connection page with login and password verification.

———

- Client-side programming
- Server-side programming

🕮 1.1.5

Client-side scripting is performed to interpret and display code that can run on a web browser without the need to process server-side code. Basically, these types of scripts are embedded in HTML documents. Client-side scripting can be used to check a user form for errors before submitting the form, for simple effects, and so on. Effective client-side scripting can significantly reduce server load.

If we disregard the basic languages that are also realized using a web browser, i.e. HTML and CSS, so the basic client-side scripting languages are JavaScript. Web creators and web standards developers must ensure that client-side languages work properly on every web browser.

1.1.6

Which languages are used for client-side programming?

- JavaScript
- VBScript
- ASP
- PHP
- Python

🕮 1.1.7

Server-side scripting is a programming technique in which the source code executes and interprets the software installed on the server. Server-side programming most often solves operations such as customizing a website, dynamically changing website content, accessing a database, etc.

Server-side scripting creates a communication link between the server and the client (user). When a browser makes a request to a web page that has an extension associated with a server language, the web server processes the script before sending the page to the browser. Script processing often involves extracting information from the database, simple calculations or selecting the appropriate

content to display to the client. The interpreter creates an HTML page from the script and sends it to the browser.

Note: In the past, server-side scripting has been implemented by so-called CGI scripts (Common Gateway Interface). CGI was designed to execute scripts from programming languages like C++ or Perl on web pages.

Because server programming does not restrict web browsers and is actually a matter of installing and setting up a particular web server, there is a wide variety of programming languages now.

The best known of these is probably PHP, which we will deal with in this tutorial. Other languages are ASP.net, Ruby on Rails, ColdFusion, Python, and so on.

1.1.8

In which approach to web programming can we choose from a variety of languages?

- Server-side programming
- Client-side programming

1.2 HTTP protocol

1.2.1

As a typical representative of server-side programming, PHP needs several technologies for its "normal" functioning. The first and probably the most basic is a web server. A web server can also be seen as a technical device from a hardware perspective, but from our - programming - perspective, we can understand a web server as an application that runs on a computer (server).

A web server is software that understands URLs (web addresses) and HTTP (the protocol your browser uses to display web pages, an acronym derived from the Hypertext Transfer Protocol). It is accessible for example through the domain names of the website (e.g. ukf.sk).

The function of its work is to respond to client requests - web browsers. If the browser needs a file that is located on a web server (most often a web page that is an HTML file), the browser requests that file via HTTP. The initial communication is the lowest level of communication, activation of the domain name system and finding the server with the necessary IP address. This is followed by communication between the web client (Firefox, Mozilla, Opera, etc.) and the

webserver. All communication is governed by a set of rules that are determined by the HTTP protocol. After accepting the request, the webserver (software) looks for the requested document (if it does not return the pending request with the error number and description) and sends the document to the requesting browser, even via HTTP.

1.2.2

What is the set of rules that govern the entire communication between a web browser and a web server?

- HTTP
- etiquette
- netiquette
- TCP/IP

1.2.3

Hypertext Transfer Protocol is a protocol for transferring documents between web servers and clients. It is the primary method of transporting information on the web. HTTP is a protocol that defines requests and responses between clients and servers. An HTTP client (referred to as a user agent), like a web browser, typically begins a request by establishing a TCP connection. The HTTP server waits for the client to send a request string such as "GET / HTTP / 1.1" (requesting the webserver start page) followed by a series of headers. Some headers are optional. After accepting the request, the server sends a response string like "200 OK" followed by headers along with the message itself, the body of which is the contents of the requested file, an error message, or other information.

1.2.4

Who sends a web document request within the HTTP protocol?

- Web client
- Web server

1.2.5

The HTTP protocol is quite simple. It contains basic methods (commands) for page requests, most commonly used:

- GET - on request a resource/file with its URL
- POST - similar to a GET request, except that a message body is usually added, containing key-value pairs from the HTML form.
- HEAD - on request a document with no requiring the message body, only headers. It is used to get metainformation about the document.

In addition to the above commands, the HTTP protocol also includes e.g. PUT, DELETE, TRACE, OPTIONS, etc.

🕮 1.2.6

Like the social protocol, it defines "behaviour" in a company, HTTP is a set of rules for "behaviour" and communication in the world of the web. For example, here is the format of a web page request that is normally sent by web browsers.

```
GET /docs/index.html HTTP/1.1
Host: www.ukf.sk
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

When the request message reaches the server, the server can do one of the following:

- The server finds the requested file in the server's document directory and returns the requested file to the client.
- The server detects that the client requests a script page, maps the request to a program saved on the server, executes the program and returns the program output to the client.
- The request cannot be accepted, the server returns an error message.

The example of an HTTP response message:

```
HTTP/1.1 200 OK
Date: Sun, 18 Jan 2019 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2018 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug
```

```
<html><body><h1>It works!</h1></body></html>
```

1.2.7

Which of the following actions can be performed by the request server:

- Finds the desired file and sends it to the client.
- Sends a script to interpret the related applications and sends the generated document to the client.
- Sends an error message if it does not find a file.
- If it does not find the file, it gets angry and turns off the entire server.
- If the client does not greet him, he will stop talking to them.

1.2.8

The first response line of a web server is called the **status line**, followed by optional response headers. The status line has the following syntax:

```
HTTP-version status-code reason-phrase
```

The status-code is a three-digit number generated by the server that reflects the result of the request, **the reason-phrase** provides a brief explanation of the status code. Common status-code and reason-phrase can be:

```
HTTP/1.1 200 OK
HTTP/1.0 404 Not Found
HTTP/1.1 403 Forbidden
```

1.2.9

The status code is a three-digit number with the first digit characterizing the code category. It can belong to one of the following categories:

- 1xx (Information): Request received, the server continues the process.
- 2xx (Success): The request has been successfully received and is processed.
- 3xx (Redirection): Further action is required to complete the request.
- 4xx (Client Error): The request contains bad syntax or cannot be processed.
- 5xx (Server Error): Server-side error.

1.3 Useful technologies

📖 1.3.1

When programming on the server-side, all the web pages or necessary scripts are located on the webserver. A web server can be understood as a technical device (computer) or as an application running on a technical device. When creating PHP, we can rely on the existing web hosting service, when the question of the server as a device and applications solve the provider for us. The second option is to obtain a web server (technical equipment) and web server application.

From the currently available web server applications, the most popular are:

- Apache HTTP Server
- Internet Information Services
- Sun Java System Web Server

It is needed to run these applications. Each of these applications has a folder that is visible, where it finds all the HTML pages and scripts provided by the client. For Apache HTTP Server, it is the HTDOCS folder (usually found in the web server file folder). Programmers need to place all their scripts in this folder. Files outside are not visible to the webserver and therefore not available.

📝 1.3.2

The application Apache HTTP Server belongs to a group of applications called:

- Web server
- FTP server
- Database server
- HTML server
- Applications server

📖 1.3.3

If we have created HTML pages or PHP pages, we save all such pages in the webserver folder (in the case of Apache HTTP Server in the **HTDOCS** folder). To check the correctness, we can call the domain address (or IP address) of our server in the web client (Firefox, Opera, Chrome…).

Programmers often do not upload their pages directly to the server but use the local version of the server directly on their computer to create them. They install a web server application on their computer, save the scripts in the **HTDOCS** folder,

and then call the scripts that are created. If the web client and web server are on the same computer, to call the web pages in the web client is used **locahost**. That means that an IP address of 127.0.0.1 or a call to "localhost" is entered in the browser instead of the domain name to check the created page.

1.3.4

If the webserver and client are located on the same computer/technical device, what can we type as a URL into the browser?

- localhost
- www.local
- http://computer
- home
- www.home

1.3.5

The web server application is running on the technical device, and that makes the technical device the real web server. However, it should be noted that many web server applications are often running on such a web device, e.g. database server, FTP server, etc. And even, it is not uncommon to run multiple web server applications on such a single technical device. Therefore, if a client (web, database, FTP client, etc.) connects to a technical device, it must specify which server application it wants to communicate with. For this reason, web applications have a number called a port. The port specifies the server application. Ports 80 or 8080 are reserved for the webserver application (of course, other ports can be assigned to the webserver).

If the user does not specify a port in the URL (most users do not), the web client will automatically send a request to the server application running under port 80, because it is reserved for the webserver. If the webserver has a specific port, it must be specified in the URL address, followed by the port number, such as ":". http://www.ukf.sk:8080 or http://www.ukf.sk:5555.

1.3.6

We have a web server running on our computer that is configured to port 1234. What URL will we type into the web client to get a web page from our local computer?

- localhost:1234

- web:1234
- local
- computer:1234
- 1234

🕮 1.3.7

After typing the URL (or calling localhost) our web browser sends an HTTP request to a specific HTML page (e.g. GET http://ww.ukf.sk/students.html). Users usually do not specify a specific page in their request (they only type http://www.ukf.sk). Each web server has set HTML page names that are searched for and then sent to the client if the client does not specify a specific page in its request. They are actually the home pages of each site. These usually (depending on your web server settings) must be called index.html or default.html. Of course, similar names are also set in the case of scripts, e.g. for PHP, it is index.php or default.php.

1.4 PHP programming

🕮 1.4.1

PHP is a server-side programming language. Mostly, PHP scripts are embedded in HTML. It is used to manage dynamic content, databases, session tracking, even to create entire complex pages, e.g. CMS, E-shop.

Why do I need to create HTML pages dynamically? Imagine a school wanting to make student lists available on its website. If the creator of the page knew only HTML, he would create pages prvaci.html, druhaci.html, tretiaci.html, and so on. It would certainly work perfectly for the first year. For the second year, however, freshmen.html would have to change to second.html, second.html to third.html, remove quarter.html, create a new list of first.html, and so on. Obviously, the laboriousness and frequency of errors in copying and rewriting from databases or official lists would be high. When e.g. e-shops or more complex information systems, this approach is practically unimaginable. On the current web, there are not many so-called static pages, because dynamic pages dominate virtually the entire site. Those are pages where, depending on the user's request, time, and other parameters, a page is generated for each user with current content.

1.4.2

What is PHP?

- server-side programming language
- markup language
- content management system
- CGI framework

📖 1.4.3

PHP is integrated with a number of popular databases including MariaDB, PostgreSQL, Oracle, Sybase, Informix and Microsoft SQL Server. The language syntax is based on the C language.

The basic tasks of PHP include:

- PHP performs system functions, e.g. provides work with files in the system, it can create, open, read, write and close files.
- PHP can process forms, i.e. collect user input, send data via e-mail, view user calculation data from user data, and more.
- PHP can work with a database, i.e. add, delete, edit database inputs.
- Can access and edit cookies variables.
- PHP can encrypt data and can authenticate website users.

📝 1.4.4

What can PHP not do?

- Play hockey
- Access cookies
- Access database
- Create and delete files
- Read data from web forms

📖 1.4.5

PHP is designed primarily for dynamic web generation, i.e. for work with HTML. In most cases, PHP commands are inserted directly into HTML code. The interpreter must distinguish where the PHP script starts and ends. Therefore, PHP always starts with the **<?php** tag and ends with the **?>** tag.

Example:

```
<?php
   //here you can insert PHP commands
```

```
?>
```

1.4.6

A complete HTML page with PHP scripts can look like this:

```
<html>
 <title>Hello World program in PHP</title>
 <body>
   <?php
     //here you can insert PHP commands
   ?>
 </body>
</html>
```

1.4.7

Add the correct characters to insert the PHP script

```
<html>
 <title>Hello World program in PHP</title>
 <body>
   <_____php

     //here you can insert PHP commands
    _____>
 </body>
</html>
```

1.4.8

The first PHP command will be the echo command for us. Use this command to type the text into a web page. Let's not forget that, like other programming languages, every PHP text is surrounded by quotes or apostrophes.

```
<html>
 <title>Hello World program in PHP</title>
 <body>
   <?php
     echo "Hello world";
   ?>
 </body>
```

```
</html>
```

1.4.9

Insert the correct command for output the "First steps" text into a webpage using PHP.

```
<html>
 <title>Hello World program in PHP</title>
 <body>
   <?php
     _____ "First steps";
   ?>
 </body>
</html>
```

1.4.10

In order to test the created PHP script, it is necessary to take a few more steps. First, it is necessary to save the created script with the correct file extension. For PHP files, this is the .php extension. If we use Webhosting, it is sufficient to copy the created PHP file to Webhosting and display it in the web browser using the correct address.

In case we use our own solution, i.e. we have a web server (e.g. Apache) on our computer, it is necessary not only to save the file with the correct extension but also to save it in the correct directory that is reserved for the webserver. For the Apache web server, this is the HTDOCS folder. After saving the script in this directory, we can check the functionality of the script in a web browser. In the URL field, enter **localhost** or **localhost/script_name.php**

Strings and Output

Chapter 2

2.1 Strings and echo

📖 2.1.1

The **echo** function is a built-in PHP function for listing embedded texts on a web page. In current PHP, echo is not a typical function, it is rather a language construct that does not need parenthesis to write parameters.

If we use the echo function to write a text string, we must insert the string in quotation marks or apostrophes.

```
<?php
  echo "PHP is super";
?>
```

📝 2.1.2

Add the PHP code to correctly write the words "wow" including the echo command.

```
<?php
  _____ "_____";
?>
```

📖 2.1.3

Individual echo commands can be called in PHP whenever needed. The condition is to separate the commands with a semicolon. Inserting a semicolon at the end of commands is required in multiple languages, not just PHP.

```
<?php
  echo "PHP ";
  echo "is ";
  echo "super ";
?>
```

Note that the language interpreter checks the semicolon at the end of the command/function as the only termination. From the interpreter's perspective, the following code is also correct:

```
<?php
  echo "PHP "; echo "is "; echo "super ";
?>
```

However, writing each command/function on a new line is a good practice for programmers, especially for reasons of clarity and better readability of the source code.

2.1.4

What character is used in PHP to indicate the termination of an instruction?

- ;
- .
- -
- ,
- ?
- %

2.1.5

Putting multiple echo commands on separate lines does not mean that even the output, i.e. listing and website will be in separate rows.

The following code:

```
<?php
  echo "PHP ";
  echo "is ";
  echo "super ";
?>
```

will appear on a webpage with the only line:

```
PHP is super
```

2.1.6

The **echo** command inserts strings into a page that is displayed in a web browser. HTML code is generated for the user. Typically, HTML tags are inserted into **echo** command strings. E.g. tag **
** to insert a new line.

```
<?php
  echo "PHP <br>";
  echo "is <br>";
  echo "super <br>";
```

```
?>
```

2.1.7

Which of the following echo commands has the correct syntax?

- echo "<|h1> holiday<|/h1>";
- echo "(<|h1> holiday<|/h1>");
- echo (<|h1> holiday<|/h1>);
- echo <|h1> holiday<|/h1>;

2.1.8

Insert the correct tags for the new line into the echo commands (with the end-of-command character) in the source code so that the output is formatted as follows:

```
Hello,
this is echo
result.
```

Source code:

```
<?php
  echo "Hello, _____
  echo "this is _____
  echo "echo <br>";
  echo "result.";
?>
```

- <|hr>;
-
";
- <|space>;
- n/r/;
- \n\r;
- ";

2.1.9

Like the **
** tag, other HTML tags can be inserted into the echo string.

Format “PHP is super” so that the word “super” will appear in bold. Use the **** HTML tag for bold.

```
<?php
    echo "PHP is _____super_____";
?>
```

2.2 Comments

📖 2.2.1

Commenting is an important part of every programming language despite that it is not read as part of the program. Its only purpose is to be read by someone who is editing or updating the code.

Comments are used to let others know what was done in the code. If you are working with a group of people or plan to share your code with others, the comments tell the other programmers what you are doing at each step. This makes it much easier for them to work with and edit your code if needed.

Comments are used also for yourself to remind you what you did in your code. Although you may just be writing a shortcode for yourself and don't see the need for comments, go ahead and add them in any way. Most programmers have experienced coming back to edit their own work a year or two later and having to figure out what they did. Comments can remind you of your thoughts when you wrote the code.

There are several ways to add comments in PHP code which will be introduced in the following lesson.

📖 2.2.2

Commenting in PHP is similar to comments that are used in HTML and other programming languages. PHP supports *C*, *C++* and *Unix shell-style* commenting. The PHP comment syntax always begins with a special character sequence and the whole text that is placed between the characters.

Commenting in HTML is done in the following form:

```
<!-- HTML tags and words between these characters are taken as
comments -->
```

The HTML syntax has only one type of comment but the PHP offers a better variance. Let's focus on **one-line** (or also known as single-line) comments. These comments are the most used because they are placed right beside the commands. The one-line comment tells the interpreter to ignore everything that occurs on that line to the right of the comment. PHP has two types of one-line commenting. To do

a one-line comment you have to type "//" or "#" and all the text to the right will be ignored by the PHP interpreter.

```
<?php
echo "Hello, I'm John."; // This is a one-line comment
# This is also a comment
?>
```

2.2.3

Select the right comment command:

- echo "Wof, wof!";
- // echo "Wof, wof!";
- // the dog will bark
- $name = "John";
- // $number = 5;

2.2.4

There are several ways to add a comment in PHP code. The **one-line comment** can be also used in a separate line like shown in the examples:

```
<?php
echo "Hello, I'm John."
// introduce yourself - this is a comment
echo "Nice to meet you, John."
?>
```

The alternative use of the sharp "#" character is shown here:

```
<?php
echo "Hello, I'm John."
# introduce yourself - this is also a comment
echo "Nice to meet you, John."
?>
```

2.2.5

Complete the code that way to print out the following text:

```
Hello, I'm John!
```

```
<?php
echo "Hello, I'm John!"; _____ this is a comment using the two
slashes
_____ echo "Nice to meet you!";  this is a comment using the
sharp
?>
```

📖 2.2.6

One-line comments have also some issues that can sometimes occur. Because it is a one-line comment that means that it can end with the end of the line and that the code on the next line will be executed (if it is not another one-line comment). The other option to end the one-line comment is the PHP closing tag *?>*. If the PHP closing tag occurs in the same line as the one-line comment then the following command (after the closing tag) will be executed. Let's look at the following example:

```
<?php
// echo "Hello "; ?> echo "I'm John."
```

This will result in the following printout:

```
I'm John.
```

that is because the PHP closing tag closes also the one-line comment.

📝 2.2.7

One-line comments can be written using the characters "//" or "#".

- True
- False

📖 2.2.8

Similar to HTML comments also in PHP can be used a **character sequence to comment multiple lines** or large scope of code. The multiple line PHP comment begins with "/* " and ends with "*/".

```
<?php
/* This code will print out the greeting to the place
in which I reside on. In other words, the World. */
echo "Hello World!";
/* echo "My name is John!";
echo "No way! My name is Peter!";
*/
?>
```

2.2.9

Complete the commands to make the comment through all the lines.

```
<?php
echo "Hello, I'm John.";

_____
 this line contains a comment
 that is three lines long
 and should be closed using the corresponding characters
_____
?>
```

2.2.10

What will be the output of the following code? Pay attention to the closing of the comments and PHP. Write the resulting output in one line where each sentence is separated by a space.

```
<?php
 // echo "Hello, I'm John.";
 echo "Nice to meet you." # introduction
 /*
  echo "Welcome to our school.";
  echo "It's awesome that you have come here."
 */
 # echo "What subject is your favorite?"; ?> echo "It's Math."
```

- Nice to meet you. echo "It's Math."
- Nice to meet you.

Operators

Chapter 3

3.1 Operators

📖 3.1.1

Operators are used to performing operations on given values. Using the operators we can take some values to perform operations and receive results. For example, 2 + 2 = 4 in this expression '+' is an operator. The expression consists of two values 2 and 2 and performs addition on them to get the result 4.

PHP supports various types of operators:

- Arithmetic Operators
- Logical or Relational Operators
- Comparison Operators
- Conditional Operators
- Assignment Operators
- Spaceship Operators (Introduced in PHP 7)
- Array Operators
- Increment/Decrement Operators
- String Operators

Arithmetic and Increment/Decrement operators

Operator	Description	Example	Result
+	addition	$a + $b	sum of values
-	subtraction	$a - $b	difference of values
*	multiplication	$a * $b	product of values
/	division	$a / $b	quotient of values
%	modulo	$a % $b	remainder of division of values
++	increment	$a++	incrementation by one
--	decrement	$a--	decrementation by one
**	exponentiation	$a ** $b	$a raised by the power $b

📝 3.1.2

Select the correct result with the operator if the variables are defined as follows:

```
$a = 4;
$b = 3;
$c = 0;
$d = -6;
```

- $a + $b = 7
- $a - $c = 0
- $c + $a = 4
- $d * $c = 0
- $a +- $b = 4
- $d / $a = -1.5
- $b % $a = 3
- $a * $d = -24
- $c - $d = -6
- $a / $b = 1

📖 3.1.3

Logical or Relationship operators are used to operate with conditional statements and expressions. Conditional statements are based on conditions. Also, a condition can either be met or cannot be met so the result of a conditional statement can either be *true* or *false*.

Logical operators

Operator	Description	Example	Result
and	logical and	$a and $b	true if both variables are true
or	logical or	$a or $b	true if either of the variable is true
xor	logical xor	$a xor $b	true if either of the variable is true but not both
&&	logical and	$a && $b	true if both are true
||	logical or	$a || $b	true if either is true
!	logical not	!$a	true if $a is not true

📖 3.1.4

The comparison operators are used to compare two elements and outputs the result in boolean form.

Comparison operators

Operator	Description	Example	Result
==	equal to	$a == $b	true if $a is equal to $b
!=	not equal to	$a != $b	true if $a is not equal to $b
<>	not equal to	$a <> $b	true if $a is not equal to $b
===	identical	$a === $b	true if $a is equal to $b and they are of the same type
!==	not identical	$a !== $b	true if $a is not equal to $b or they are not the same type
<	less than	$a < $b	true if $a is lower than $b
>	grater than	$a > $b	true if $a is greater than $b
<=	less than or equal to	$a <= $b	true if $a is lower than or euqal to $b
>=	greater than or equal to	$a >= $b	true if $a is greater than or euqal to $b

📖 3.1.5

There is an operator called conditional operator. These operators are used to compare two values and take either of the results simultaneously, depending on whether the outcome is TRUE or FALSE. These are also used as a shorthand notation for if…else statement.

Conditional operators

Operator	Description	Example	Result
?:	conditional expression	($a>$b) ? $a: $b	if condition is true then return $a otherwise return $b

📖 3.1.6

The assignment operators are used to assign values to a different variable, with or without mid-operations. You have to be cautious of the number of "=". If you want to compare two values you have to use == and if you assign you to use only one =.

Assignment operators

Operator	Description	Example	Similar to
=	assign	$a = $b	$a = $b
+=	add then assign	$a += $b	$a = $a + $b
-=	subtract then assign	$a -= $b	$a = $a - $b
*=	multiply then assign	$a *= $b	$a = $a * $b
/=	divide then assign	$a /= $b	$a = $a / $b
%=	divide then assign (modulo)	$a %= $b	$a = $a % $b

📖 3.1.7

The array operators are used only if you work with arrays. They are similar to the operators we have already listed but using with arrays they have a little other functionality.

Array operators

Operator	Description	Example	Result
+	union	$a + $b	union of $a and $b
==	euqal to	$a == $b	true if $a and $b have the same key/value pair
!=	not equal to	$a != $b	true if $a is not equal to $b
===	identical	$a === $b	true if $a and $b have the same key/value pair in the same order and of the same types
!==	not identical	$a !== $b	true if $a is not identical to $b
<>	not equal to	$a <> $b	true if $a is not equal to $b

📖 3.1.8

The string operators are implemented upon strings.

String operators

Operator	Description	Example	Result
.	concatenation	$a . $b	concatenates values of $a and $b
.=	concentation	$a .= $b	first it concatenates and then it assigns to the value

📖 3.1.9

PHP 7 has introduced a new kind of operator called spaceship operator (). These operators are used to compare values but instead of returning the boolean result, it returns integer values. If both the operands are equal, it returns 0. If the right operand is greater, it returns -1. If the left operand is greater, it returns 1.

Spaceship operators

Operator	Description	Example	Result
<=>	<	$a <=> $b	identical to -1 if right is greater
<=>	>	$a <=> $b	identical to 1 if left is greater
<=>	<=	$a <=> $b	identical to -1 or 0 if right is greater or both equal
<=>	>=	$a <=> $b	identical to 1 or 0 if left is greater or both equal
<=>	==	$a <=> $b	identical to 0 if both equal
<=>	!=	$a <=> $b	not identical to 0

📝 3.1.10

Select the correct answer for the following input variables:

```
$a = 10
$b = -5
$c = 3
$d = 10
```

- $a > $b
- ($a > $b) && ($b > $c)
- $a === $d
- $a != $c
- $c >= $d
- $b <|= $c
- $c-- //result will be 3, and $c takes the value 2
- ($a <|=> $d) === 1

3.2 Operators (programs)

⌨ 3.2.1 Arithmetic operators

Print a comma-separated list of the results of the operations between *$x* a *$y*: addition, subtraction, multiplication, division and modulo.

```
Input : 4 2
Output: 6,2,8,2,0
```

⌨ 3.2.2 Set of instructions

Assume you have *$x* as the input, make the following instruction:

- add 2 to *x*,
- subtract 4,
- multiply by 5,
- increment value by one,
- divide by 3.

Print the final result.

⌨ 3.2.3 Find the maximum

Find and print the maximum between two variables *$x* and *$y* (your input).

Conditions

Chapter 4

4.1 If/else statement

📖 4.1.1

The conditional statements are also available in PHP like in most programming languages. Conditions are used to divide the actions that are performed based on the result of a logical or comparative test. This means you can create test conditions in the form of expressions that evaluates either *true* or *false*. Based on the results of these tests you can perform certain actions.

There are several statements in PHP that you can use to make decisions:

- the **if** statement
- the **if... else** statement
- the **if... elseif... else** statement
- the **switch... case** statement

📖 4.1.2

The if statement

The *if* statement is used to execute a block of code only if the specified condition evaluates to true.

```
if (condition) {
    code to be executed if condition is true;
}
```

📝 4.1.3

Complete the code that way to print out the following text if the current day is Friday:

```
Have a nice weekend!
```

```
<?php
$day = date("D");
if($day _____ _____) {
   echo "Have a nice weekend!";
}
?>
```

📖 4.1.4

The if... else statement

The *if... else* statement executes code when a condition is true and another code when that condition is false.

```
if (condition) {
   code to be executed if condition is true;
} else {
   code to be executed if condition is false;
}
```

📝 4.1.5

Complete the code that way to print out the text "Have a nice weekend!" if the current day is Friday otherwise it will output "Have a nice day!":

```
<?php
$day = date("D");
_____($day=="Fri") {
   echo "Have a nice weekend!";
} _____ {
   echo "Have a nice day!"
}
?>
```

📖 4.1.6

The if... elseif... else statement

If you want to execute some code when one of the several conditions are true use the elseif statement. Using this statement you can use more than two conditions.

```
if (condition) {
   code to be executed if this condition is true;
} elseif (condition) {
   code to be executed if this condition is true and the first
is not;
} else {
   code to be executed if all conditions are false;
}
```

4.1.7

Complete the code that way to print out the text "Good morning!" if the current hour is less than 10, and "Good day!" if the current time is less than 20. Otherwise, it will output "Good night!":

```
<?php
$time = date("H");

if ($time < "10") {
   echo _____;
} _____ ($time < "20") {
   echo "Good day!";
} _____ {
   echo "Good night!";
}
?>
```

4.1.8

The conditional/ternary operator

As mentioned in the previous chapter there is also a conditional (or ternary) operator that provides a shorthand way of writing the *if... else* statement. The ternary operator is represented by the question mark *?* symbol and it takes three operands: a condition to check; a result if the condition is met; and a result if the condition is not met.

```
statement (condition) ? code to return if true : code to
return if false;
```

4.1.9

Complete the code that way to print out the text "Have a nice weekend!" if the current day is Friday otherwise it will output "Have a nice day!". Use the ternary operator.

```
<?php
$day = date("D");
echo (_____) _____ "Have a nice weekend!" _____ "Have a nice
day!";
?>
```

📖 4.1.10

PHP 7 introduced a new null coalescing operator *??* that can be used as a shorthand where you need to use a ternary operator in conjunction with *isset()* function. To better understand this consider the following line of code:

```
<?php
$day = isset($_GET['day']) ? $_GET['day'] : "day not
selected";
?>
```

Using the null coalescing operator the same code could be rewritten following:

```
<?php
$day = $_GET['day'] ?? "day not selected"
?>
```

4.2 If/else statement (programs)

⌨ 4.2.1 Checking a year

Write code that realises the following algorithm:

```
if ($year is not divisible by 4) then (print "common year")
else if ($year is not divisible by 100) then (print "leap
year")
else if ($year is not divisible by 400) then (print "common
year")
else (print "leap year")
```

⌨ 4.2.2 Comparing two numbers

Write a program that finds and prints the larger of the two different integers, making sure that the program gives the information if the numbers are matched.

```
Input : 5 4
Output: higher is number 5
```

```
Input : 10 10
Output: both numbers are equal
```

4.2.3 Even and odd numbers

Write a program that finds and prints information whether the number is even or odd.

```
Input : 5
Output: odd
```

```
Input : 10
Output: even
```

4.2.4 Absolute value

Write a program that will write an absolute value of the given number.

```
Input : 6
Output: 6
```

```
Input : -3
Output: 3
```

4.2.5 Division

Write a program that writes a division of two numbers also with the division remainder (dr.). Do not forget to deal with the division by zero

```
Input : 5 4
Output: 1, dr. 1
```

```
Input : 10 0
Output: division by zero
```

4.2.6 Maximum of three numbers

Write a program that finds the maximum of three given numbers.

```
Input : 5 4 7
Output: 7
```

```
Input : 10 10 10
Output: 10
```

4.2.7 Time of day

Write a program that returns the time of day (night/day) based on the given hour (1-12) and time period (a.m./p.m.). Let's assume that the sun sets down and comes up at 6:00 a.m./p.m.

```
Input : 5 a.m.
Output: night
```

```
Input : 5 p.m.
Output: day
```

4.2.8 Diagnosis

Write a program that will decide whether the given temperature is a fever (>37.5°C), high fever (>38.5°C) or normal.

```
Input : 38.8
Output: high fever
```

```
Input : 36.7
Output: normal
```

4.2.9 Leap year

Write a program that finds out whether the given year is a leap year. A leap year is a multiple of 4, and if it is a multiple of 100, it must also be a multiple of 400.

```
Input : 2012
Output: leap year
```

```
Input : 2019
Output: normal year
```

4.2.10 Digits

Write a program that finds whether the given number has a higher digit sum or digit product.

```
Input : 111
Output: digit sum
```

```
Input : 1234
```

```
Output: digit product
```

4.2.11 Similar numbers

Write a program that finds out whether the three given numbers are equal.

```
Input : 5 4 5
Output: False
```

```
Input : 10 10 10
Output: True
```

4.3 Switch

4.3.1

The *switch* statement is used similarly to the *if* statement but the difference is that using the *switch* statement you can perform different actions based on different conditions. PHP *switch* statement compares a variable or an expression against many different values and executes a code block based on the value it equals to.

```
<?php
switch(variable) {
   case value1:
       code executed when value1 == variable;
       break;
   case value2:
       code executed when value2 == variable;
       break;
   default:
       code executed when variable does not meet any
condition;
       break;
}
?>
```

4.3.2

Let's examine the *switch* statement syntax in more detail.

First, you put a variable or expression that you want to test into the brackets after the *switch* statement. Then inside the curly brackets are multiple *case* constructs that contain values that are compared with the variable or expression. In case the

value of the variable or expression matches the value in a *case* construct, the code block in the corresponding *case* will be executed. If the value of the variable or expression does not match any value then the code block in the *default* constructor will be executed.

The *break* statement is used in each *case* or *default* construct to exit the entire *switch* statement. It is important because if the *break* is missing then the next construct will follow.

4.3.3

It is very important to understand that the *switch* statement is executed statement by statement therefore the order of the *case* construct is very important. If the value of the variable matches a value in the *case* construct, PHP will execute code black in that *case* construct and ends the *switch* statement if the *case* construct contains the *break* statement.

4.3.4

Complete the code that way to print out the text based on the random number we get for the variable.

```
<?php
$a = rand(0,3);
_____($a) {
   _____ 0: echo 'a = 0';
   break;
   _____: echo 'b = 1';
   break;
   case 2: echo 'c = 2';
   break;
   case 3: echo 'c = 3';
   _____;
}
?>
```

4.3.5

What will be the output of the following code:

```
<?php
$a = 1;
```

```
switch($a) {
   case 0: echo 'zero';
   break;
   default: echo 'default';
   break;
}
?>
```

4.3.6

Complete the code to get the correct switch statement:

```
<?php
$a = 3;
$b = -3;
$c = $a - $b;
_____($c) {
   case -3: echo 'negative';
   _____
   case 0: echo 'zero';
   break;
   case 3: echo 'positive';
   break;
   _____: echo 'out of scope';
   break;
}
?>
```

4.3.7

What will be the output of the following code:

```
<?php
$a = 3;
$b = -3;
$c = $a - $b;
switch($c) {
   case -6: echo 'negative';
   break;
   case 0: echo 'zero';
   break;
   case 6: echo 'positive';
   break;
```

```
   default: echo 'out of scope';
   break;
}
?>
```

4.3.8

Will the following switch statement output the following text?

```
one
```

```
<?php
$a = 1;
switch($a) {
   case 0: echo 'zero';
   case 1: echo 'one';
   case 2: echo 'two';
   default: echo 'none';
}
?>
```

- false
- true

4.3.9

It is also possible to check an expression or condition in the *switch* statement. This requires a little bit of an elegant solution. Let's have an example where we want to find out whether the randomly generated number is from the various intervals. So we have to assume that the condition is met and that's why we have to say that the *switch* statement has to search for a condition that returns *true*.

Let's see the following example:

```
<?php
$r = rand(-5,5);
switch(true) {
   case $r < 0: echo 'negative number';
   break;
   case $r == 0: echo 'zero';
   break;
   case $r > 0: echo 'positive number';
   break;
```

```
}
?>
```

4.3.10

Fill up the following code with the correct expressions:

```
<?php
$r = rand(0,10);
switch(_____) {
   case _____: echo 'zero';
   break;
   case _____: echo 'even number';
   break;
   case $r % 2 != 0: echo 'odd number';
   break;
}
?>
```

4.4 Switch (programs)

4.4.1 Count of even digits

Write a program that finds and prints the number of even digits for a given number.

```
Input : 15478
Output: 0:0 2:0 4:1 6:0 8:1
```

```
Input : 12124225
Output: 0:0 2:4 4:1 6:0 8:0
```

4.4.2 Days per month

Write a program that will write the number of days for a given month (**do not** take into account the leap year). Wrong input will result in Output: "wrong month"

```
Input : 5
Output: 31
```

```
Input : 13
Output: wrong month
```

4.4.3 Calculator

Write a program that, based on a mathematical operation (+, -, *, /), finds the sum, the difference, the product, and the proportion between the two entered numbers. The input is entered in the order of the mathematical operation and the next line of the number.

```
Input : + 5 4
Output: 9
```

```
Input : / 3 2
Output: 1
```

4.4.4 Day of month II.

Write a code which for the given number between 1-12 (corresponding to the number of the month) and the number from the interval 1900-2200 (corresponding to the number of the year) computes the number of days in this month (1, 3, 5, 7, 8, 10, 12 - 31 days; 4, 6, 9, 11 - 30 days, 2 - 28/29 days). Input is two integer numbers (month and year). Print number of days in this month. Be aware of leap years. If the data are out of bound then print -1.

```
Input : 2 1900
Output: 28
```

```
Input : 2 2000
Output: 29
```

4.4.5 Seasons

Write a program that returns for the given month number its corresponding season:

- 3-5 write "SPRING"
- 6-8 write "SUMMER"
- 9-11 write "AUTUMN"
- 12-2 write "WINTER"

If the number is out of range then write "wrong month".

```
Input : 5
Output: SPRING
```

```
Input : 13
Output: wrong month
```

4.4.6 Remaining days

Write a program that will return the remaining days till the end of the month. The input consists of the actual day and the month.

```
Input : 2 2
Output: 26
```

```
Input : 1 1
Output: 30
```

4.4.7 Age categories

Write a program that reads a person's age and writes whether it is a child (0-11 years old), a teenager (12-18 years old), a young man (19-35 years old), a middle-aged man (36-60 years old) or an old man (61 and over).

```
Input : 9
Output: child
```

```
Input : 89
Output: old man
```

4.4.8 Equation

Write a code that will compute solution to a system of linear equations of two variables. Input contains 6 values of type double (a1, b1, c1, a2, b2, c2) as values of equations- coefficients:

a1 x + b1 y = c1

a2 x + b2 y = c2

Print suitable values of double type:

- if the system has no solution, print 0

- if the system has one solution, print three numbers: 1 and values of x and y

- if the system has an infinite number of solutions, print Infinity

```
Input : 1 1 1 1 1 2
Output: 0
```

```
Input : 2 3 6 4 9 15
```

```
Output: 1 1.5 1.0
```

```
Input : 1 1 1 1 1 1
Output: Infinity
```

Loops

Chapter 5

5.1 For loop

📖 5.1.1

During writing code many times happens that you have to write the same code. Sometimes you have to write similar code and you lose time and have a long code. Because of that, we do use loops.

Loops are used when you want the same block of code to run over and over again in a row. Instead of adding several almost equal code lines in a script, we can use loops to perform a task like this.

The PHP syntax does use various loops:

- *for* - loops through a block of code a specified number of times
- *foreach* - loops through a block of code for each element in an array
- *while* - loops through a block of code as long as the specified condition is true
- *do... while* - loops through a block of code once, and then repeats the loop as long as the specified condition is true

📝 5.1.2

Is the following statement true or false?

Loops in PHP are used to execute the same block of code a specified number of times.

- true
- false

📖 5.1.3

The *for* loop statement is used when you know how many times you want to repeat a statement or block of statements. If you don't know how many times the code should run then it is hard to use this loop type.

```
<?php
for (initialization; condition; increment) {
   code to be executed;
}
?>
```

The parameters in the *for* statement are following:

- *initialization* - this part is used to set the start value for the counter of the number of loop iterations;
- *condition* - it is evaluated for each loop iteration if the condition is met, the loop continues, otherwise, it ends;
- *increment* - this increases the loop counter.

5.1.4

Is the following statement true or false?

The *for* loop is used when you know in advance how many times the script should run.

- true
- false

5.1.5

Complete the following code to get the correct syntax.

```
<?php
_____($i=0; $i<10; _____) {
   echo "$i <br>";
}
?>
```

5.1.6

The *for* loop statement can be also written without any expressions as long as you keep the semicolon.

```
for(;;) {
   code to be executed;
}
```

If we omit all the expressions in the *for* loop statement then it is important to terminate the loop otherwise we get an infinite loop. We can use a simple *if* condition. When the condition is met (*true*) then we terminate the loop using the *break* statement.

5.1.7

Fill up the following code so that you terminate the loop when the number is higher than 5.

```
<?php
for(;;) {
   $i = rand(0,10);
   echo $i . '<br />';
   if(_____)
      _____;
}
?>
```

5.1.8

The other loop type is a *foreach* loop. The *foreach* loop works only with arrays or public properties of an object and is used to loop through each key/value pair in the array.

```
foreach ($array as $value) {
   code to be executed;
}
```

For each loop iteration, the value of the current array element is assigned to value and the array pointer is moved by one until it reaches the last array element. If you use the *foreach* loop statement with other data types, you will get an error.

5.1.9

Fill up the following statement with the correct code.

```
<?php
$fruit = array("apple", "peach", "pear", "orange");

_____ ($fruit _____ $value) {
   echo "$value <br>";
}
?>
```

5.1.10

Create a loop that runs from 0 to 100.

```
<?php
_____($i = 0; _____; $i++) {
   echo $i;
}
?>
```

5.2 For (programs)

5.2.1 Uppercase and Lowercase

Write a program that prints the count of uppercase and lowercase letters and numbers in a given string.

```
Input : Hello
Output: Upper: 1, Lower: 4, Numbers: 0
```

```
Input : H3LL0
Output: Upper: 3, Lower: 0, Numbers: 2
```

5.2.2 Vowels

Write a program that prints the vowels of a given string.

```
Input : Hello John
Output: eoo
```

```
Input : How are you
Output: oaeou
```

5.2.3 Repeating the output

Write the code that prints 10 times after each other the word "hello"

```
Input :
Output: hello
hello
hello
hello
```

```
hello
hello
hello
hello
hello
hello
```

5.2.4 Numbered output

Write the code that prints the word "hello" 10 times with the number of the output.

```
Output: hello1
hello2
hello3
hello4
hello5
hello6
hello7
hello8
hello9
hello10
```

5.2.5 Sequence of numbers

Write the code that shows the sequence of numbers from 0 to a given number.

```
Input : 5
Output: 0 1 2 3 4 5
```

```
Input : 10
Output: 0 1 2 3 4 5 6 7 8 9 10
```

5.2.6 Sum of numbers

Write the code that returns the sum of the numbers in the interval two given numbers.

```
Input : 3 5
Output: 12
```

```
Input : 10 15
Output: 75
```

5.2.7 Print of sum numbers

Write the code to determine the sum of the first n natural numbers you enter on the input. Print continuous results on the console.

```
Input : 5
Output: 1
3
6
10
15
```

```
Input : 4
Output: 1
3
6
10
```

5.2.8 Factorial

Write the code that calculates the factorial *(n! = n * n-1)* ... *3*2*1)* of a given number. Print the interim result.

```
Input : 5
Output: 1
2
6
24
120
```

```
Input : 3
Output: 1
2
6
```

5.2.9 Fibonacci

Write the code that will compute the nth Fibonacci number according to the iterative algorithm. Input the integer number n, greater-or-equal 0 and less-or-equal 90. Print the nth Fibonacci number.

```
Input : 4
Output: 3
```

```
Input : 1
Output: 1
```

5.2.10 Product of numbers

Write the code that finds out the product of two given numbers without using the multiplication operation.

```
Input : 5 4
Output: 20
```

```
Input : 10 10
Output: 100
```

5.2.11 Product of interval

Write the code that finds out the product of integers located between two specified values. Ensure that it displays the variables in each cycle step during the run.

```
Input : 5 7
Output: 1 - 5
2 - 30
3 - 210
210
```

```
Input : 2 5
Output: 1 - 2
2 - 6
3 - 24
4 - 120
120
```

5.2.12 Multiplication table

Write the code that prints a small multiplication table for the given integer.

```
Input : 5
Output: 1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
```

```
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50
```

```
Input : -5
Output: 1 * -5 = -5
2 * -5 = -10
3 * -5 = -15
4 * -5 = -20
5 * -5 = -25
6 * -5 = -30
7 * -5 = -35
8 * -5 = -40
9 * -5 = -45
10 * -5 = -50
```

5.3 While and do while loops

🕮 5.3.1

As was mentioned in the previous lesson during writing code many times happens that you have to write the same code. Because of that, we do use loops. In the previous lesson, we worked with *for* and *foreach* loops, now we will work with *while* loops. Let's remind us what loops are.

Loops are used when you want the same block of code to run over and over again in a row. Instead of adding several almost equal code lines in a script, we can use loops to perform a task like this.

The PHP syntax does use various loops:

- *for* - loops through a block of code a specified number of times
- *foreach* - loops through a block of code for each element in an array
- *while* - loops through a block of code as long as the specified condition is true
- *do... while* - loops through a block of code once, and then repeats the loop as long as the specified condition is true

🕮 5.3.2

The *while* loop executes a block of code as long as the specified condition is met (*true*).

```
while (condition is true) {
   code to be executed;
}
```

If you compare it with the *for* loop you can see the main difference is in the way of thinking. When you want to use the *for* loop you usually have to know how many times the loop will be repeated. On the other hand, the *while* loop can be run as many times as the condition is met.

This specific *while* loop checks the condition at the beginning of each iteration. If the condition is not met (*false*) the loop terminates.

5.3.3

Complete the following code with the correct syntax.

```
<?php
$seconds = 10;

_____($seconds _____ 0) {
   if($seconds==0) echo "Happy New Year!";
   else echo "The New Year is in ". $seconds . " seconds.";
   $seconds--;
}
?>
```

5.3.4

Complete the code so that you will calculate the sum of all numbers from 0 to 10.

```
<?php
$number = 0;
$sum = 0;
_____($number <= 10) {
   $sum += $number;
   _____;
}
echo $sum;
?>
```

5.3.5

Is it true or false?

The following loop will be terminated when the condition is not met. Based on the code the condition will be certainly *false* at a specific run.

```
<?php
$num = 4;
while($num >= 2) {
   echo "The number is: $num <br>";
   $num++;
}
?>
```

- false
- true

5.3.6

The *do... while* loop will always execute the block of code once and then it will check the condition at the end. If the condition is met it will repeat the loop while the condition is true.

```
do {
   code to be executed;
} while (condition is true);
```

The difference between the *while* and *do... while* statement is where the condition is tested. The *while* statement has the condition at the beginning and the *do... while* has the condition at the end. This means that the *do... while* loop will certainly be run at least once even if the condition is not met at the first run.

5.3.7

Complete the following code with the correct syntax.

```
<?php
$num = 10;
_____ {
   echo $num.'<br>';
   $num--;
} _____ ($num > 0);
```

```
?>
```

5.3.8

Complete the following code so that you will print the numbers from 1 to 7.

```
<?php
$num = _____;
do {
   echo $num.'<br>';
   _____
} while (_____);
?>
```

5.3.9

How many times will be the following loop run? (Number of cycles)

```
<?php
$num = 0;
do {
   echo "*";
} while ($num > 0);
?>
```

5.3.10

Is it true or false?

The *while* or *do... while* loop can not be used if you don't know the number of iterations that should be run in the loop.

- false
- true

5.4 While (programs)

5.4.1 Decreasing numbers

Write the code that prints the numbers in decreasing order from a given number to 0. Use the while loop.

```
Input : 5
Output: 5
4
3
2
1
0
```

```
Input : 10
Output: 10
9
8
7
6
5
4
3
2
1
0
```

5.4.2 Sum of numbers

Write the code that returns the sum of numbers from 0. The given number will be the upper limit, if the sum of numbers is higher than the given number, print out the result that is lower than the given number. Use the while loop.

```
Input : 5
Output: 3
```

```
Input : 10
Output: 6
```

5.4.3 Remainder after division

Write the code that writes the remainder after division for the given numbers without the help of integer division functions and the remainder after division (do not use / or %). In the case of division by zero, write "Division by zero".

```
Input : 5 4
Output: 1
```

```
Input : 4 0
Output: Division by zero
```

5.4.4 Euclid's algorithm

Write the code that will compute the greatest common divisor of two integers using the subtraction-based version of Euclid's algorithm (which was Euclid's original version). In addition, the code should compute the smallest common multiple of these two integers (using the divisor computed within the first step). The input contains two integer numbers. Print the greatest common divisor and the smallest common multiple.

```
Input : 25 40
Output: 5 200
```

```
Input : 33 196
Output: 1 6468
```

5.4.5 Product of numbers

Write the code that finds out the product of two given numbers. Do not use the operation of multiplication!

```
Input : 5 4
Output: 20
```

```
Input : 10 10
Output: 100
```

5.4.6 Even numbers

Write the code that returns the count of even numbers between two given numbers. Use the while loop.

```
Input : 5 8
Output: 2
```

```
Input : 7 15
Output: 4
```

5.4.7 Digit sum

Write a program that finds out the sum of digits of a given number. Use the while loop.

```
Input : 1234
```

```
Output: 10
```

```
Input : 100
Output: 1
```

5.4.8 Number length

Write the code that finds out the length of a given number. Make note that each given number will end with 0.

```
Input : 520
Output: 3
```

```
Input : 10
Output: 2
```

5.4.9 Factorial

Write the code that calculates the factorial (n!=n*(n-1)* ... *3*2*1) of a given number. Use the *do... while* loop.

```
Input : 5
Output: 120
```

```
Input : 2
Output: 2
```

5.4.10 Fuel

Write the code that prints out whether the car has enough fuel. You have a given fuel and if the car has no fuel inform the user. Use a while loop wherein each iteration you spend 1 fuel.

```
Input : 5
Output: You have enough fuel.
You have enough fuel.
You have enough fuel.
You have enough fuel.
You have enough fuel.
You are out of fuel!
```

Logical Expression

Chapter 6

6.1 Booleans

📖 6.1.1

A boolean variable (also known as a bool) can have a value of either **true** or **false**. To declare a boolean use the constants **TRUE** and **FALSE**, which are case-insensitive.

```
$a = TRUE;
$b = true;
$c = trUE;
$d = TRue;
// All above variables are true.
$e = FALSE;
$f = false;
$g = faLSE;
$h = False;
// All above variables are false.
```

📖 6.1.2

Booleans are typically used to control structures like the **if** statement, the **for** loop and so on. The comparison operators (like the equality **==**operator) and logical operators (like the and operator **&&**) return a boolean value as a result. We usually combine control structures and operators together:

```
// It should display "1.01" if the variable $action equals to
"display_version"
if ($action == "display_version") {
    echo "1.01";
}
// The loop iterates till the condition `$i < 10` is fulfilled
(means it is evaluated as false).
for ($i = 0; $i < 10; $i++) {
    //...
}
```

The result of displaying a boolean value with the **echo** function is either **1** (true) or an empty string (false).

```
$a = "test";
$b = $a == "test";
$c = $a == "tost";
echo $b; // Output: 1
```

```
echo $c; // Output:
```

The operators and structures are presented in other sections of this course.

6.1.3

Assign the correct boolean value to the variables to display: "Fitped" at the Output:

```
$a = _____;
$b = _____;
$c = _____;
if (($a || $b) && (! $b || $c) && ($b || ! $c) && ! $c) {
    echo "Fitped";
}
```

6.1.4

Assign the correct boolean value to the variables to display: "Fitped" at the Output:

```
$a = _____;
$b = _____;
$c = _____;
if ($a || $b && ! $c || ! $b) {
    echo "An wrong answer";
} else {
    echo "Fitped";
}
```

6.1.5

In PHP, if a function, operator or control structure requires a boolean, then a passed value will be automatically converted to it. Moreover, we can covert a value explicitly with the **(bool)** or **(boolean)** cast.

```
var_dump((bool) ""); // Output: bool(false)
var_dump((boolean) ""); // Output: bool(false)
```

The rules of the automatic conversion can be tricky, hence it is worth knowing them. Let's start with non-boolean values that are considered **false**.

```
var_dump((bool) "");       // bool(false), an empty string
var_dump((bool) "0");      // bool(false), the string "0"
```

```
var_dump((bool) 0);       // bool(false), the integers 0 and -
0 (zero)
var_dump((bool) 0.0);     // bool(false), the floats 0.0 and -
0.0 (zero)
var_dump((bool) array()); // bool(false), an array with zero
elements
var_dump((bool) null);    // bool(false), the special type
NULL
```

Every other value is considered **true**.

```
var_dump((bool) "Fitped"); // bool(true)
var_dump((bool) 1);         // bool(true)
var_dump((bool) -5);        // bool(true)
var_dump((bool) 0.01);      // bool(true)
var_dump((bool) array(1)); // bool(true)
```

Considering the above rules the following code will not display: "Fitped".

```
if ("a" && 0.0) {
    echo "Fitped";
}
```

There are more rules, which require introducing more complex concepts, so we do not mention them here. The interested reader can find more information in the PHP manual.

6.1.6

Find the output of the following code:

```
$a = -1;
$b = "false";
$c = "0";
if ($a && $b && $c) {
    echo "Fitped A";
} else {
    echo "Fitped B";
}
```

6.1.7

Find the output of the following code:

```
$a = "0.0";
$b = "-1";
$c = 0.0;
if ($a && $b && ! $c) {
    echo "Fitped A";
} else {
    echo "Fitped B";
}
```

📖 6.1.8

The standard library consists of two functions that are dedicated to booleans. The first is **boolval(mixed $var)**, which returns the boolean value of the passed variable (according to the previously mentioned converting rules). The second function is **is_bool(mixed $var)**, which tests if the passed argument is a boolean.

```
var_dump(boolval(0));   // bool(false)
var_dump(boolval(1));   // bool(true)
var_dump(boolval("0")); // bool(false)
var_dump(boolval("a")); // bool(true)
$a = true;
$b = false;
$c = 0;
var_dump(is_bool($a)); // bool(true)
var_dump(is_bool($b)); // bool(true)
var_dump(is_bool($c)); // bool(false)
```

📝 6.1.9

Find the output of the following code:

```
$a = FaLsE;
$b = "test";
$c = (bool) "0";
if (! $a && boolval($b) && is_bool($c)) {
    echo "Fitped A";
} else {
    echo "Fitped B";
}
```

6.1.10

Which of the following sentences is correct?

- The declaration: "$a = tRUE;" is syntactically invalid.
- The string: "0" is considered as false.
- The string: "0.0" is considered as false.
- The expression: "is_bool(0)" returns false.
- The expression: "boolval(array(0))" returns false.
- The expression: "echo boolval(-0)" returns an empty string.

6.2 Booleans (programs)

6.2.1 Are all true? (boolean casting)

Assume you have 3 variables: *$a, $b, $c*, display the message: "Success" if all variables are considered as true and display: "Failure" otherwise.

```
Input : a 1 -4
Output: Success
```

```
Input : 0 1 b
Output: Failure
```

6.2.2 Are all true? (boolean casting with a special rule)

Assume you have 3 variables: *$a, $b, $c*, display the message: "Success" if all variables are considered as true and display: "Failure" otherwise. **BUT** add a special rule, i.e. "0.0" is now considered as **false** not true (similar to "0").

```
Input : a 1 -4
Output: Success
```

```
Input : 0.0 1 b
Output: Failure
```

6.2.3 Boolean expression

Assume you have 3 input variables: *$a*, *$b*, *$c*, and 3 local variables: *$x*, *$y* and *$z*. Input variables have two possible values: "AND" and "OR" (strings that should be converted into && and || operators), whereas local variables are booleans. Your task

is to concatenate all variables into a single expression, if the expression is considered as **true**, then display "Success" and display: "Failure" otherwise.

The expression has a form: *$x $a $y $b ($x $c $z) $a ($y $b $x).*

```
Example 1: if $a="AND", $b="OR" and $c="AND", the concatenated
expression is: true && false || (true && false) && (false ||
false) and the message "Failure" will be displayed.
```

```
Example 2: if $a="OR", $b="OR" and $c="OR", the concatenated
expression is: true || false || (true || false) || (false ||
false) and the message: "Success" will be displayed.
```

assigment_answer.php

```
<|?php
// your variables, do not remove this line
list($a, $b, $c) = explode(" ", trim(fgets(STDIN)));
// local variables, do not remove those lines
$x = true; $y = false; $z = false;
// write your code here
```

6.2.4 Success if greater

Assume you have 3 variables: *$a, $b, $c*, display the message: "Success" if *$a > $b > $c* and all variables are considered as true, otherwise display "Failure".

```
Input : 4 3 2
Output: Success
```

```
Input : 2 1 0
Output: Failure
```

6.2.5 Vowel validation

Assume you have one variable: *$a*, display the message: "Success" if the variable value is a vowel (a, e, i, o, or u, case-insensitively), otherwise display "Failure".

```
Input : a
Output: Success
```

```
Input : c
Output: Failure
```

Arrays

Chapter 7

7.1 Arrays

🕮 7.1.1

A PHP's array is actually an ordered map, which is a data structure associating values to keys. If you know the concept of an array from other programming languages, like C++, you can think about a PHP's array as an ordinary array, list, hash table, dictionary, collection, stack, queue and so on. As array values can be other arrays, you can create a tree structure and multidimensional arrays as well. To sum up, PHP's arrays are multipurpose and provide a convenient method to store data in applications.

As the scope of this course is very limited, we encourage the interested reader to find out more information about the above-mentioned data structures.

🕮 7.1.2

An array can be created using the **array()** constructor or a square brackets **[ ]** (which is more popular nowadays). Both methods take any number of comma-separated values or *keys=>values* pairs as arguments. Arrays are mutable, hence they can be modified dynamically after their creation.

```
$arr1 = [1,2,3,4,5];
$arr2 = ["a","b","c"];
$arr3 = ["a",1,"c",4];
$arr4 = [1=>"a",2=>"b",3=>"c"];
$arr5 = ["a"=>1,"b"=>2,"c"=>3];

// All above examples can be written with the `array()`
construct.
$arr1 = array(1,2,3,4,5);

// We can add an optional comma at the end of arguments.
$arr1 = [1,2,3,4,5,];
$arr2 = ["a","b","c",];
```

A value can be of any type, whereas a key need to be either an integer or a string. Using keys in brackets allow to set and get a value of an array's element.

```
$arr = [];       // an empty array
$arr[0] = "a";   // sets a new value
echo $arr[0];    // gets a value, Output: a
var_dump($arr); // Output: array(1) { [0]=> string(1) "a" }
```

It is also possible to convert any of the other types, like integer, float, string, boolean and resource, into arrays.

```
$arr = (array) "abc";
var_dump($arr); // Output: array(1) { [0]=> string(3) "abc" }
$arr = (array) 1;
var_dump($arr); // Output: array(1) { [0]=> int(1) }
```

📖 7.1.3

As we mentioned in the previous part, keys can be either an integer or a string, yet there is an automatic conversion mechanism that converts invalid data types into valid ones.

- Strings containing valid decimal integers will be cast to the integer type (except the numbers preceded by a + sign).
- Floats are converted to integers, which means that the fractional part will be truncated.
- Booleans are converted to integers: **true** will be converted to 1 and **false** to 0.
- The special value **null** is converted to an empty string.
- Arrays and objects cannot be converted to valid keys.

```
var_dump(["5"=>1]);   // Output: array(1) { [5]=> int(1) }
var_dump(["+5"=>1]);  // Output: array(1) { ["+5"]=> int(1) }
var_dump(["-5"=>1]);  // Output: array(1) { [-5]=> int(1) }
var_dump(["a"=>1]);   // Output: array(1) { ["a"]=> int(1) }
var_dump([0.1=>1]);   // Output: array(1) { [0]=> int(1) }
var_dump([3.14=>1]);  // Output: array(1) { [3]=> int(1) }
var_dump([false=>1]); // Output: array(1) { [0]=> int(1) }
var_dump([true=>1]);  // Output: array(1) { [1]=> int(1) }
var_dump([null=>1]);  // Output: array(1) { [""]=> int(1) }
```

The declaration of keys is optional, if they are not specified, PHP will use the increment of the largest previously used integer key starting from zero.

```
var_dump([1,2,3]);       // Output: array(3) { [0]=> int(1),
[1]=> int(2), [2]=> int(3) }
var_dump([5=>1,2,3]);    // Output: array(3) { [5]=> int(1),
[6]=> int(2), [7]=> int(3) }
var_dump([1,7=>2,3]);    // Output: array(3) { [0]=> int(1),
[7]=> int(2), [8]=> int(3) }
var_dump([5=>1,2=>2,3]); // Output: array(3) { [5]=> int(1),
[7]=> int(2), [6]=> int(3) }
```

Setting a value with an empty bracket (a key is not provided) will add a new element with the next integer key according to the mentioned above rule.

```
$arr = ["a" => 1];
$arr[] = 2;
var_dump($arr); // Output: array(2) { ["a"]=> int(1), [0]=>
int(2) }
$arr = [3 => 1];
$arr[] = 2;
var_dump($arr); // Output: array(2) { [3]=> int(1), [4]=>
int(2) }
```

It is possible to use the same key a few times while creating an array, the value will be overwritten by the last occurrence.

```
$arr = [0 => 1, 0 => 2, 0 => 3];
var_dump($arr); // Output: array(1) { [0]=> int(3) }
```

📖 7.1.4

An array's element can be accessed using either the **array[key]** (more popular) or **array{key}** syntax.

```
$arr = ["a"];
echo $arr[0]; // Output: a
echo $arr{0}; // Output: a
```

We can use more brackets to get access to nested arrays.

```
$arr = [
    "multi" => [
        "dimensional" => [
            "array" => "a"
        ]
    ]
];
echo $arr["multi"]["dimensional"]["array"]; // Output: a
```

It is possible to array dereference the result of a function or method call directly.

```
function getArray() {
    return [1, 2, 3];
}
echo getArray()[0]; // Output: 1
```

If the key does not exist in an array, the special value **NULL** will be returned.

```
$arr = [];
var_dump($arr[0]); // Output: NULL
```

7.1.5

Find the output of the following code:

```
$arr = [
    5 => "F", "i", 2 => "t", "p", "e", "d"
];
echo $arr[5];
```

7.1.6

Find the output of the following code:

```
$arr = [
    1 => "F", 1.5 => "i", 2 => "t", 2.5 => "p", 3 => "e", 3.5
=> "d"
];
echo $arr[2];
```

7.1.7

Find the output of the following code:

```
$arr = [
    "1" => "F", "-2" => "i", "+3" => "t", true => "p", -5 =>
"e", "0.5" => "d"
];
echo $arr[0];
echo $arr[1];
```

Remember that the expression: "*echo null;*" displays an empty string.

7.1.8

The **count(mixed $array)** function returns the number of elements in an array (or other countable objects).

```
$arr = [1,2,3];
echo count($arr); // Output: 3
```

The **unset(mixed $var)** function removes the key from an array.

```
$arr = [1,2,3];
unset($arr[1]);
var_dump($arr); // Output: array(2) { [0]=> int(1), [2]=>
int(3) }
```

The **array_merge(array $array1 ...)** function merges one or more arrays. If the arrays have the same string keys, then the later value for that key will overwrite the previous one. However, if the arrays contain numeric keys, the later value will not overwrite the original value but will be appended.

```
$arr1 = ["a" => 1];
$arr2 = ["a" => 2];
var_dump(array_merge($arr1, $arr2)); // Output: array(1) {
["a"]=> int(2) }
$arr1 = [0 => 1];
$arr2 = [0 => 2];
var_dump(array_merge($arr1, $arr2)); // Output: array(2) { 0
=> int(1), 1 => int(2) }
```

If you want to append elements from the second array to the first array without re-indexing and with elements overwriting, use the + array union operator.

```
$arr1 = [0 => 1];
$arr2 = [0 => 2];
var_dump($arr1 + $arr2); // Output: array(1) { 0 => int(2) }
```

7.1.9

Fill the gaps to display: "Fitped" at the Output:

```
$arr = [
    _____=> true,
    _____.5 => false,
    false,
];
_____($arr[_____]);
if (_____($arr) === 2 && $arr["a"] && $arr[3] === false) {
    echo "Fitped";
}
```

7.1.10

Which of the following sentences is correct?

- An array's key can be either a string or integer only.
- The expression: "$arr=[1]; echo $arr{0};" returns the special value NULL.
- The expression: "$arr=["6.5"]; echo $arr[6];" returns an empty string.
- The expression: "[1]+[1]" returns the new array "[1,1]".
- The expression: ' (array) "a b c" ' returns the new array ' ["a","b","c"] '.
- The expression: ' $arr=[null=>1]; echo $arr[""]; ' returns: "1".

7.2 Arrays (programs)

7.2.1 Printing an array's value

Assume you have the array *$arr*, print the value of the third element from the array.

```
Input : [1,2,3,4,5]
Output: 3
```

7.2.2 Printing an array's keys

Assume you have the array *$arr* with numerical keys, print the keys (comma-separated) that values containing the value: 3.

```
Input : [1,2,3,3,5]
Output: 2,3
```

7.3 Foreach

7.3.1

The foreach loop allows iterating over elements of an array (or an object, if it is traversable). In each iteration, we get access to an array element through a key and value pair.

```
$arr = array("a", "b", "c");
foreach ($arr as $value) {
    echo $value;
}
```

The above code should give you the following Output:

```
abc
```

In the above example, keys are skipped but we will see how to use them later. The foreach loop operates over internal structures of arrays and therefore types of keys are not important.

📖 7.3.2

The foreach loop has two syntaxes:

```
foreach (array_expression as $value) {
    statement
}
foreach (array_expression as $key => $value) {
    statement
}
```

The use of a key (an index) is what distinguished the second syntax from the former. In both, we get access to an element's value through the $value variable. You can use any name for the *$key* and *$value* variables but it is a good practice to give them meaningful names.

```
$letter = ["p", "h", "p"];
foreach ($letter as $letter) {
    echo $letter;
}
// Output: `php`

foreach ($letter as $key => $letter) {
    echo $key;
}
// Output: `012`
```

An array expression can be a variable that refers to an array (or a traversable object) or an array using the array syntax.

```
foreach (["p", "h", "p"] as $letter) {
    echo $letter;
}
// Output: `php`
```

You can also use the less popular syntax of the foreach loop.

```
foreach(["p", "h", "p"] as $letter):
```

```
    echo $letter;
endforeach;
// Output: `php`
```

You can use a **break** control structure to exit from the foreach loop.

```
foreach (["p", "h", "p"] as $letter) {
    echo $letter;
    break;
}
// Output: `p`
```

7.3.3

Fill the gaps with the correct expressions.

```
$arr = ["a", 1, "c"];
foreach (_____ _____ $value) {
   echo _____;
}
```

7.3.4

Is it possible to iterate over an array with non-numerical keys using the foreach loop?

- True
- False

7.3.5

In each iteration, the foreach loop 'creates' variables, which store a key and value of the current element. The variables names are provided in the loop syntax. We can use them to get an element's value, but we cannot modify the original array.

```
$arr = [0, 1, 2];
foreach ($arr as $value) {
    $value = 3;
}
var_dump($arr); // Output: array(3) { [0]=> int(0), [1]=>
int(1), [2]=> int(2) }
```

In order to directly modify array elements within the loop precede *$value* with the reference symbol **&**.

```
$arr = [0, 1, 2];
foreach ($arr as &$value) {
    $value = 3;
}
var_dump($arr); // Output: array(3) { [0]=> int(3), [1]=>
int(3), [2]=> int(3) }
```

It is worth noting that the reference of a *$value* and the last array element remain even after the foreach loop.

```
$arr = [0, 1, 2];
foreach ($arr as $value) {
}
echo $value; // Output: 2
foreach ($arr as &$value) {
}
$value = 3;
var_dump($arr); // Output: array(3) { [0]=> int(0), [1]=>
int(0), [2]=> int(3) }
```

📖 7.3.6

The foreach loop with the additional construct **list(...)** can be used to 'unpack' nested arrays into variables.

```
$arr = [
    ["p", 1], ["h", 2], ["p", 3]
];
foreach ($arr as list($a, $b)) {
    echo $b . $a;
}
// Output: 1p2h3p
```

📝 7.3.7

Find the output of the following code:

```
$arr = ["a","b","c"];
foreach ($arr as $key=>$value) {
    $arr[$key] = "d";
}
```

```
foreach ($arr as $value) {
    echo $value;
}
```

7.3.8

Find the output of the following code:

```
$arr = ["a","b","c"];
foreach ($arr as $key=>&$value) {
    $value=$key;
}
foreach ($arr as $v) {
    echo $v;
}
```

7.3.9

Fill the gaps to display: "Fitped" at the Output:

```
$arr = [[1,2],[3,4],[5,6]];
$test = "";
_____ ($arr as _____(_____,_____)) {
    $test .= $a+$b; // strings concatenation
}
if ($test === "3711") {
    echo "Fitped";
}
```

7.3.10

Which of the following sentences is correct?

- An foreach loop can iterate over arrays only.
- The expression: "$arr=[1,2,3]; foreach($arr as $key) echo $key;" returns: "012".
- In order to modify an array directly, you need to use the reference symbol &.
- In order to modify an array directly, you need to use the additional construct list.
- The expression: "$arr=[1,2,3]; foreach($arr as $value); echo $value;" returns: "3".

7.4 Foreach (programs)

7.4.1 Sum of numbers

Assume you have the array *$arr*, use the *foreach* loop and print the sum of the values from the array.

7.4.2 Minimum

Assume you have the array *$arr*, use the *foreach* loop and print the minimum value from the array-s values.

7.4.3 Maximum

Assume you have the array *$arr*, use the *foreach* loop and print the maximum value from the array-s values.

7.4.4 Mean

Assume you have the array *$arr*, use the *foreach* loop and print the mean of the values from the array.

7.4.5 Lower than the mean

Assume you have the array *$arr*, use the *foreach* loop and print how many values are lower than the mean of the values.

7.4.6 Greater than the mean

Assume you have the array *$arr*, use the *foreach* loop and print how many values are greater than the mean of the values.

7.4.7 Two dimensional minimum

Assume you have the two-dimensional array *$arr*, use the *foreach* loop and print the minimum value.

7.4.8 Numbers counter

Assume you have the array *$arr*, use the *foreach* loop and print how many times certain numbers are repeated in the array values (comma-separated + hyphen).

7.4.9 Checking keys

Assume you have the associative array *$arr* (i.e. an array with key/value pairs), use the *foreach* loop and print the sum of the array-s values that keys are started with the letter *a*.

7.4.10 Find dollars

Assume you have the array *$arr*, use the *foreach* loop and replace all values that are equal $ with the number 5. Print the new array's elements (comma-separated).

7.4.11 Find the largest key

Assume you have the associative array *$arr* (i.e. an array with key/value pairs), use the *foreach* loop and print the largest key.

assigment_answer.php

```
<|?php
  $input = explode(" ",trim(fgets(STDIN)));
  $keys = eval("return" . $input[0] . ";");
  $values = eval("return" . $input[1] . ";");
  $arr = array_combine($keys, $values);
  // your array, do not remove those lines
  // write your code here
```

7.4.12 Merge arrays

Assume you have two associative arrays *$arr1* and *$arr2*, use the *foreach* loop to merge (by index) them, i.e. all elements from the second array replace the first array-s elements, if only they have the same keys. Print the merged array's elements (comma-separated).

7.4.13 Reverse an array

Assume you have the associative array *$arr* (i.e. an array with key/value pairs), use the *foreach* loop and print the values in the reverse order (comma-separated).

assigment_answer.php

```
<|?php
  $input = explode(" ",trim(fgets(STDIN)));
$keys = eval("return " . $input[0] . ";");
$values = eval("return " . $input[1] . ";");
$arr = array_combine($keys, $values); // your array, do not
remove those lines
// write your code here
```

Parsing and Errors

Chapter 8

8.1 Parsing and errors

📖 8.1.1

Each error in PHP has a so-called level, which determines the origin of an error and its meaningfulness (i.e. if an error halts an application or not). Levels are just integers but we identify them by predefined constants.

Let's begin with the following example:

```
if $x==1 {
    echo "Fitped";
}
```

The execution of this code will be halted with the message:

```
Parse error: syntax error, unexpected '$x'... expecting '('
```

Of course, I have forgotten to add brackets, which has ended with a parse error. The parse error (the integer value 4, the constant **E_PARSE**) is a compile-time error, which means it occurs during code reading and analysis by the PHP interpreter. Compile-time errors are due to inaccuracies in code. Parse errors occur when the syntax of the code is invalid, e.g. missing brackets or a semicolon.

Other examples of compile-time errors are **E_COMPILE_ERROR** (the integer value 64) and **E_COMPILE_WARNING**, which are generated by the Zend Scripting Engine (i.e. the scripting engine that interprets PHP code).

The difference between **warning** and **errors** is that the former does not halt executing, whereas the latter does it. There are also **notices**, which are like warnings but indicate only a possibility of an error.

📖 8.1.2

During the PHP's initial startup two types of errors can be thrown: **E_CORE_ERROR** (the integer value 16) and **E_CORE_WARNING** (the integer value 32). The occurrence of these errors means that the configuration is invalid and the core of PHP cannot initialize the engine or the engine itself is broken due to other reasons.

📖 8.1.3

E_ERROR (the integer value 1), **E_WARNING** (the integer value 2) and **E_NOTICE** (the integer value 8) are run-time errors, which means they occur during execution. All of

them suggest issues in the source code, e.g. an undeclared variable, memory allocation problems and so on.

```
var_dump([] + 1);
// FATAL ERROR Unsupported operand types on line number ...
echo array_merge([], $arr);
// NOTICE Undefined variable: arr on line number ...
// WARNING array_merge(): Argument #2 is not an array on line
number ...
```

📖 8.1.4

All error integers values are used to build up a bitmask that specifies which errors to report. The bitwise operators are used to combine or mask the values, e.g. *E_ERROR (1) | E_PARSE (4) = 5*.

To sets which PHP errors should be reported use the **error_reporting([int $level])** function. If the optional **$level** is not provided, the function returns the current reporting error level.

There is the special constant **E_ALL** (the integer value 32767), which means all errors and warnings.

```
// Turn off all error reporting.
error_reporting(0);

// Report run-time errors.
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);

// Report all errors except E_NOTICE.
error_reporting(E_ALL & ~E_NOTICE);

// Report all PHP errors.
error_reporting(E_ALL);
```

You can also set the reporting error level with the **ini_set(...)** function.

```
ini_set('error_reporting', E_ALL);
```

Moreover, with the **ini_set** function, you can also control errors displaying.

```
// Determines wheter errors should be printed to the screen.
ini_set('display_errors', '1');
```

📖 8.1.5

You can receive **E_DEPRECATED** (the integer value 8192) warnings, which are run-time notices about code that will not work in future versions.

```
$fn = create_function('$a,$b', 'return "ln($a) + ln($b) = " .
log($a * $b);');
// PHP Deprecated:  Function create_function() is deprecated
in ...
```

📖 8.1.6

An error can be triggered by a user through the **trigger_error(string $error_msg, [, int $error_type])** function.

```
if ($divisor == 0) {
    trigger_error("Cannot divide by zero", E_USER_ERROR);
}
```

The **$error type** can have the following values: **E_USER_ERROR** (the integer value 256), **E_USER_WARNING** (the integer value 512),**E_USER_NOTICE** (the integer value 1024) and **E_USER_DEPRECATED** (the integer value 16348).

📖 8.1.7

Apart from errors displaying, all error messages can be saved in log files (which are ordinary text files). To turn on this feature, the option **log_errors** must be set to **1** (by using the **ini_set** function).

```
ini_set('log_errors', '1');
```

The path to logs is stored as the configuration option **error_log**. The following code print out its current value:

```
echo ini_get("error_log");
```

There is also the possibility to send a custom message to text logs through the **error_log(string $message ...)** function.

```
error_log("Database not available!");
```

📖 8.1.8

PHP allows us to set (by using the **set_error_handler** function) a custom error handler, which is an ordinary function with few arguments (see the example below). Depending on the boolean value that is returned from the function, the default PHP error handler will be executed.

```
function custom_error_handler($errno, $errstr, $errfile,
$errline)
{
    if ($errno == E_USER_ERROR) {
        // Don't execute PHP internal error handler.
        return true;
    }
    // Do execute PHP internal error handler.
    return false;
}
set_error_handler("custom_error_handler"); // Sets a custom
error handler.
```

📖 8.1.9

In the previous sections, the function **var_dump** was used. The function allows printing the value and type of a variable.

```
var_dump(1); // Output: int(1)
```

There is also the possibility of printing all functions call, included/required files and evaluated code with the function **debug_print_backtrace**.

```
function a() {
    b();
}
function b() {
    c();
}
function c(){
    debug_print_backtrace();
}
a();
// Output:
// #0 c() called at ...
// #1 b() called at ...
// #2 a() called at ...
// #3 eval() called at ...
```

The above functions are great tools for identifying simple errors in the source code. Regardless, for more complex applications you should use a debugger.

8.1.10

Which of the following sentences is correct?

- The expression: "error_reporting(E_ALL | ~E_WARNING)" turns on all errors and warnings except E_WARNING.
- It is possible bypassing the standard PHP error handler.
- The function "error_log" allows saving custom messages in log files.
- The E_DEPRECATED warnings suggest the requirement of changing the source code in the future.
- Warnings halt execution.
- The E_PARSE errors are run-time errors.

8.2 Errors (programs)

8.2.1 Run-time errors

Set the error reporting level to show run-time errors and warnings only.

8.2.2 User's errors

Set the error reporting level to show all a user-s errors.

8.2.3 No errors

Set the error reporting level to not display any errors and warnings.

8.2.4 No errors logging

Turn off errors logging with the function *ini_set*.

8.2.5 Set a custom error handler

Set the *custom_error_handler* function as the error handler.

assigment_answer.php

```
<|?php // do not remove code below
  function custom_error_handler($errno, $errstr, $errfile,
$errline) {
  echo "Success";
}
// write your code here

// do not remove code below
trigger_error("", E_USER_ERROR);
```

Forms

Chapter 9

9.1 Forms and forms elements

🕮 9.1.1

In this short section, HTML forms will be presented, but as distinct from an HTML course, forms will be generated through PHP's echo function. Of course, they still will be ordinary HTML tags. This section requires basic knowledge of HTML, hence it is recommended to take part in an HTML course before.

HTML forms are used for collecting data from a user (e.g. registration data), afterwards the collected data are posted to a backend application written in PHP (or some other technologies). The backend application performs required processing on the passed data, such as validation and saving to a database.

🕮 9.1.2

A form needs to be started with the ***<form>*** tag.

```
echo '<form>';
echo '</form>';
// Output: <form></form>
```

The form attributes specification:

- the **action** attribute specifies where to send the form data (the attribute is optional, if not provided, the data will be processed on the same page).
- the **method** attribute specifies the HTTP method to use when sending the form (can be either *GET*, which is the default value or *POST*).
- the **target** attribute specifies the target window or frame, where the response of the backend application will be displayed.
- the **enctype** attribute determines how the browser will encode the data before sending them to the server. There are two possible values: *application/x-www-form-urlencoded* (the default value, good for simple scenarios) and *multipart/form-data* (used when uploading binary data, e.g. images, word files).

```
echo '<form action="index.php" method="get">';
echo '</form>';
```

In the above example, the data will be sent to the script: **index.php**, so if the domain name equals **http://www.test.pl**, the data will be processed by the script: **http://www.test.pl/index.php**.

The difference between the *GET* and *POST* methods will be explained in the next section.

9.1.3

Complete the following form code:

```
echo '<_____ _____="index.php" _____="post">';
echo '</form>';:
```

9.1.4

The ***<input>*** tag specifies a basic form control for entering single-line user data, which needs to be used within a ***<form>*** element. An input field can vary in many ways, depending on **the type attribute**.

```
echo '<form>';
echo 'Username: <input type="text" name="username"><br>';
echo 'Password: <input type="password" name="password"><br>';
echo '<input type="submit" value="Sign in">';
echo '</form>';
```

The code above will produce the following form:

The input tag's attributes specification:

- the **type** attribute specifies the type of input control, the default type is *text*.
- the **name** attribute specifies a name of the control, which is sent to the server to be recognized and get the value.
- the **value** attribute provides an initial value.

The available types are as follows:

- button,
- checkbox,
- color,
- date,
- datetime-local,
- email,
- file,
- hidden,
- image,
- month,

- number,
- password,
- radio,
- range,
- reset,
- search,
- submit,
- tel,
- text,
- time,
- url,
- week.

9.1.5

Complete the following input code:

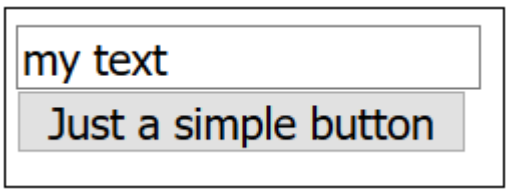
```
echo '<input _____="text" _____="An initial value">';
```

9.1.6

In the previous slide, the general form of the ***<input>*** tag was presented. In this part, we focus on buttons. There are four buttons types: *button*, *reset*,*image__ and _submit*. The buttons' value attribute contains text that is used as the button's label (if you do not specify a value, the button will have a default label, chosen by the user agent, e.g. a browser).

The *button* type renders as a simple push button, which can be programmed to control web page functionality.

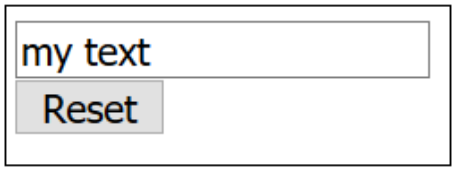
```
<form>
 <input type="text"> <br>
 <input type="button" value="Just a simple button">
</form>
```



The *reset* type renders as a button, which the default click event resets all of the inputs in the form to their initial values.

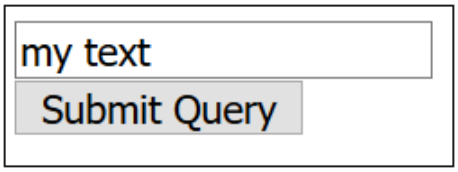
```
<form>
```

```
 <input type="text"> <br>
 <input type="reset">
</form>
```



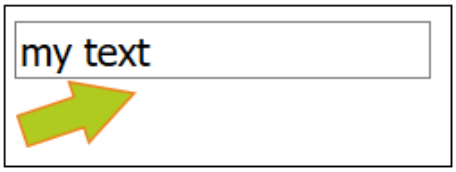
The *submit* type renders as a button, which the default click event submits the form to the server.

```
<form>
 <input type="text"> <br>
 <input type="submit">
</form>
```



The *image* type renders as a submit button, which has a graphical form.

```
<form>
 <input type="text"> <br>
 <input type="image"
src="https://priscilla.fitped.eu/images/my_arrow.svg">
</form>
```



9.1.7

If you want to submit a form, you should use an input with the type:

____

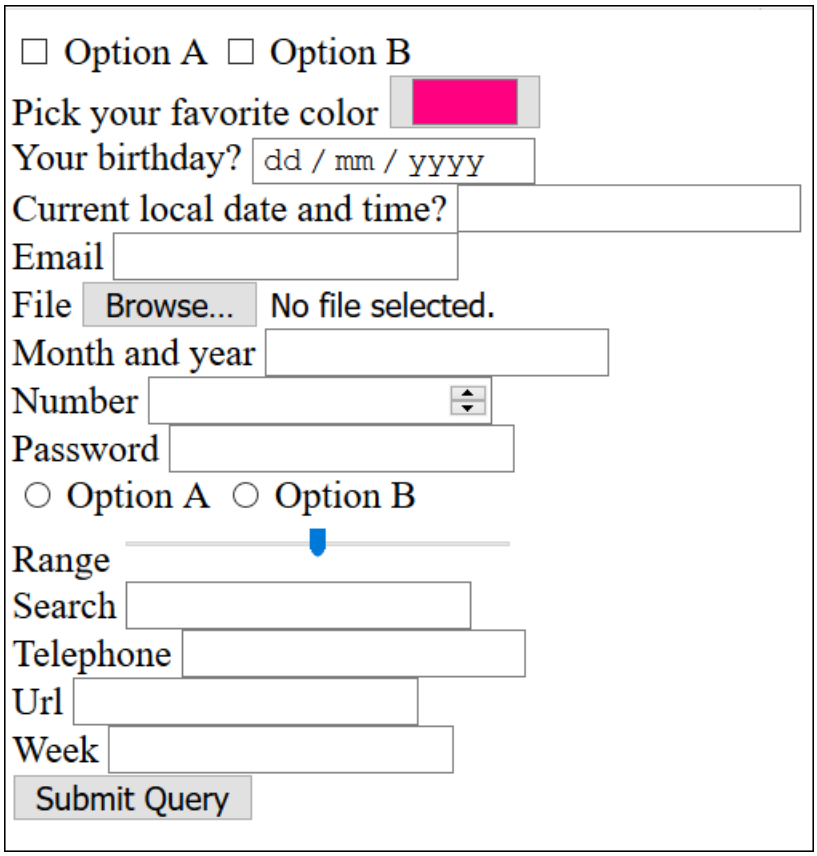
- send
- reset
- submit

📖 9.1.8

Read the example:

```
echo '<form>
echo '<input type="checkbox"> Option A <input type="checkbox">
Option B<br>';
echo 'Pick your favorite color <input type="color"><br>';
echo 'Your birthday? <input type="date"><br>';
echo 'Current local date and time? <input type="datetime-
local"><br>';
echo 'Email <input type="email"><br>';
echo 'File <input type="file"><br>';
echo 'Month and year <input type="month"><br>';
echo 'Number <input type="number"><br>';
echo 'Password <input type="password"><br>';
echo '<input type="radio" name="radio"> Option A <input
type="radio" name="radio"> Option B<br>';
echo 'Range <input type="range"><br>';
echo 'Search <input type="search"><br>';
echo 'Telephone <input type="tel><br>';
echo 'Time <input type="time"><br>';
echo 'Url <input type="url"><br>';
echo 'Week <input type="week"><br>';
echo '<input type="submit">';
echo '</form>';
```

The code above will produce the following form:



The available types for inputs and their descriptions are as follows:

- **checkbox** A check box allowing single values to be selected/deselected.
- **color** A control for specifying a color.
- **date** A control for entering a date (year, month, and day, with no time).
- **datetime-local** A control for entering a date and time, with no time zone.
- **email** A field for editing an e-mail address.
- **file** A control that lets the user select a file.
- **hidden** A control that is not displayed but whose value is submitted to the server.
- **month** A control for entering a month and year, with no time zone.
- **number** A control for entering a number.
- **password** A single-line text field whose value is obscured.
- **radio** A radio button, allowing a single value to be selected out of multiple choices.
- **range** A control for entering a number whose exact value is not important.
- **reset** A button that resets the contents of the form to default values.
- **search** A single-line text field for entering search strings.
- **tel** A control for entering a telephone number.
- **text** A single-line text field.
- **time** A control for entering a time value with no time zone.
- **url** A field for entering a URL.
- **week** A control for entering a date consisting of a week-year number and a week number with no time zone.

9.1.9

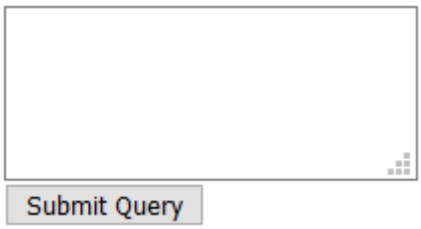
The ***<textarea>*** tag can hold an unlimited number of characters, and the text renders in a fixed-width font. The size of the input can be specified by the cols and rows attributes.

```
echo '<form>';
echo '<textarea rows="4" cols="50">';
echo '</textarea><br>';
echo '<input type="submit">';
echo '</form>';
```

The result of the code is a set of HTML commands:

```
<form>
 <textarea rows="4" cols="50"></textarea><br>
 <input type="submit">
</form>
```

...and a web page:



9.1.10

Which of the following sentences is correct?

- The attribute that specifies where to send the form data is called action.
- The input's value attribute provides an initial value.
- The input that has the type: image is used for uploading images.
- The input's name attribute is the input's label.
- The type: radio allows selecting out multiple values.
- If you want to hide the input field, you should use the type: hidden.

9.2 Forms (programs)

9.2.1 Make a simple form

Make a simple form and put two text inputs and a submit button into it. Ask for the user's **name** and **address** (the names of the fields).

⌨ 9.2.2 Make a more complex form

Ask for a **name**, **address**, **city**, **state, zip**. Make a simple form and put a submit button into it. Change the submit button label for the text "Place your order".

⌨ 9.2.3 More form controls

Ask for a **name**, **address**, **city**, **state**, **zip** and the **subscription** period (you have two options: *1 year* and *2 years*, make those form controls as radio inputs). Make a simple form and put a submit button into it.

⌨ 9.2.4 Password

Ask for an **email** and **password**. Make a simple form and put a submit button into it. Use the appropriate input types.

⌨ 9.2.5 Textarea

Ask for a **name**, **address**, **city**, **state**, **zip, comment** (as a textarea). Make a simple form and put a submit button into it. Change the submit button label for the text "Place your order".

9.3 Submitting a form

🕮 9.3.1

After the click of submit button, the form data will be sent to a web server. The data can be passed with one of two methods: GET and POST. The method is chosen based on the form tag attribute.

🕮 9.3.2

Both GET and POST methods create an array, i.e. a set of key/value pairs, where keys are the names of the form controls and values are the input data from the user.

In PHP, the data are available through superglobals arrays **$_GET** and **$_POST**, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$_GET is an array of variables passed to the current script via the URL query string, e.g. "*?a=1&b=2*".

$_POST is an array of variables passed to the current script via the HTTP POST method, where the body of the request contains key/value pairs, e.g. "*a=1&b=2*".

9.3.3

If you want to submit data in the URL query string you should use the _____ method.

- POST
- GET
- SEND

9.3.4

If you want to submit data in the request body of the HTTP, you should use the _____ method.

- POST
- SEND
- GET

9.3.5

You need to validate form data to protect your script from malicious code. This section does not contain any form validation (will be presented in the next section), it just shows how you can send and retrieve form data.

9.3.6

Data sent with the GET method are visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters.

GET should NEVER be used for sending passwords or other sensitive information! GET may be used for sending non-sensitive data. The GET method is useful for sending information like pagination, searching criterion, sorting, filtering, etc.

🕮 9.3.7

Data sent with the POST method are invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.

Moreover, POST supports advanced functionality such as support for multi-part binary input while uploading files to the server. It is recommended to use the POST method for sending data (almost always).

🕮 9.3.8

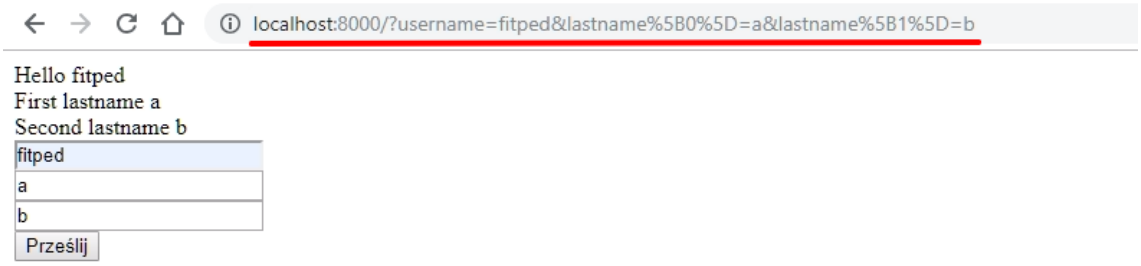
Data sent with the GET method are accessible through the **$_GET** variable, which is an ordinary array. See the example below:

```
if (isset($_GET['username'])) {
    echo 'Hello ' . $_GET['username'] . '<br>';
    echo 'First lastname ' . $_GET['lastname'][0] . '<br>';
    echo 'Second lastname ' . $_GET['lastname'][1] . '<br>';
}
echo '<form method="get">';
echo '<input type="text" name="username"><br>';
echo '<input type="text" name="lastname[0]"><br>';
echo '<input type="text" name="lastname[1]"><br>';
echo '<input type="submit">';
echo '</form>';
```

If you print the *$_GET* variable (after submitting the form), you should get the following Output:

```
var_dump($_GET);
// Output: array(2) { ["username"]=> string(6) "fitped"
["lastname"]=> array(2) { [0]=> string(1) "a" [1]=> string(1)
"b" } }
```

As you can see, it is possible to create multidimensional arrays with form controls naming (i.e. *lastname[0]* and *lastname[1]*).



📖 9.3.9

Data sent with the POST method are accessible through the **$_POST** variable, which is an ordinary array as well. See the example below:

```
if (isset($_POST['username'])) {
    echo 'Hello ' . $_POST['username'] . '<br>';
    echo 'First lastname ' . $_POST['lastname'][0] . '<br>';
    echo 'Second lastname ' . $_POST['lastname'][1] . '<br>';
}
echo '<form method="post">';
echo '<input type="text" name="username"><br>';
echo '<input type="text" name="lastname[0]"><br>';
echo '<input type="text" name="lastname[1]"><br>';
echo '<input type="submit">';
echo '</form>';
```

If you print the *$_POST* variable (after submitting the form), you should get the following Output:

```
var_dump($_POST);
// Output: array(2) { ["username"]=> string(6) "fitped"
["lastname"]=> array(2) { [0]=> string(1) "a" [1]=> string(1)
"b" } }
```

It is possible sending files with the POST method. You need to add the enctype: *multipart/form-data* to the form.

The upload files are accessible through the superglobal variable **$_FILES**. See the example below:

```
var_dump($_FILES);
echo '<form method="post" enctype="multipart/form-data">';
echo '<input type="file" name="testfile"><br>';
echo '<input type="submit">';
echo '</form>';
```

The Output:

```
array(1) {
    ["testfile"]=> array(5) {
        ["name"]=> string(8) "test.txt"
        ["type"]=> string(10) "text/plain"
        ["tmp_name"]=> string(60) "[...]\Temp\phpF2C1.tmp"
        ["error"]=> int(0)
        ["size"]=> int(0)
    }
}
```

9.3.10

Which of the following sentences is correct?

- Data sent with the GET method are invisible to others.
- Data sent with the POST method are visible to others.
- The files should be sent with the POST method.
- The files are accessible through the variable $_FILES.
- GET has limits on the amount of information to send.
- It is possible to create multidimensional arrays $_GET and $_POST.

9.4 Submitting a form (programs)

9.4.1 GET values

Print as a comma-separated list all values that are passed by the GET method.

assigment_answer.php

```
<|?php $input = explode(" ",trim(fgets(STDIN)));
$keys = json_decode($input[0],true);
$values = json_decode($input[1],true);
$_GET = array_combine($keys, $values);
// your $_GET input, do not remove those lines
// write your code here
```

9.4.2 GET keys

Print as a comma-separated list all keys (form controls names) that are passed by the GET method.

assigment_answer.php

```
<|?php
$input = explode(" ",trim(fgets(STDIN)));
$keys = json_decode($input[0],true);
$values = json_decode($input[1],true);
$_GET = array_combine($keys, $values); // your $_GET input, do
not remove those lines
// write your code here
```

9.4.3 POST

Assume you have the following form: Print the message "Success" if the username is equal "Fitped" and display the message "Failure" otherwise.

assigment_answer.php

```
<|?php $input = explode(" ",trim(fgets(STDIN)));
$keys = json_decode($input[0],true);
$values = json_decode($input[1],true);
$_POST = array_combine($keys, $values);
// your $_POST input, do not remove those lines
// write your code here
```

9.4.4 GET values - counter

Print the number of the passed values by the GET method.

assigment_answer.php

```
<|?php
$input = explode(" ",trim(fgets(STDIN)));
$keys = json_decode($input[0],true);
$values = json_decode($input[1],true);
$_GET = array_combine($keys, $values); // your $_GET input, do
not remove those lines
// write your code here
```

9.4.5 Telephones

Assume you have the following form: Print all telephones numbers as a comma-separated list.

Exercises

Chapter 10

10.1 String and echo I. (programs)

⌨ 10.1.1 First steps - echo I.

Write a PHP script which output will be: "First steps"

```
Output: First steps
```

⌨ 10.1.2 First steps - echo II.

Write a PHP script that returns the following: "wow a PHP script"

```
Input :
Output: wow a PHP script
```

⌨ 10.1.3 First steps III.

Write a PHP script that gives the following output in separated lines.

```
Input :PHP
Output:PHP is
a
good
choice
```

⌨ 10.1.4 Welcome string

Write a PHP script that returns the following output.

```
Input : Welcome
Output: Welcome in PHP programming!
```

⌨ 10.1.5 Solar system

Write a PHP script that outputs the first 4 planets in our solar system in new lines in the following way.

```
Input :
Output:Mercury
Venus
Earth
Mars
```

10.2 String and echo II. (programs)

10.2.1 Uppercase string

Write a PHP script to transform the input value to all uppercase letters.

```
Input : abc
Output: ABC
```

10.2.2 Uppercase string

Write a PHP script to make the first character of the input string an uppercase.

```
Input : abc
Output: Abc
```

```
Input : 123
Output: 123
```

```
Input : THIS IS PHP
Output: THIS IS PHP
```

10.2.3 Lowercase string

Write a PHP script to transform the input value to all lowercase letters.

```
Input : ABC
Output: abc
```

10.2.4 Split string

Write a PHP script to split the input string in the following way:

```
Input : abc
Output: abc
```

```
Input : 1111
Output: 111:1
```

```
Input : abcabcabc
Output: abc:abc:abc
```

10.2.5 Length of the string

Write a PHP script to return the length of the given string.

```
Input : abc
Output: 3
```

```
Input : 1111
Output: 4
```

```
Input : abcabcabc
Output: 9
```

10.2.6 Number of words in the input string

Write a PHP script to return the number of words of the given string.

```
Input : abc
Output: 1
```

```
Input : 1111
Output: 0
```

```
Input : abc abcabc
Output: 2
```

10.2.7 Return a reverse string

Write a PHP script to return the reverse of the given string.

```
Input : abc
Output: cba
```

```
Input : 1111
Output: 1111
```

```
Input : abc abcabc
Output: cbacba cba
```

10.2.8 Replace string

Replace the "fruits" word to "pizza" in each given input in the following way:

```
Input : fruitsaregood
Output: pizzaaregood
```

```
Input : fruits123
Output: pizza123
```

10.2.9 Repeat a string several times

Write a PHP script to repeat the given string several times:

```
Input : abc 2
Output: abcabc
```

```
Input : bbb 4
Output: bbbbbbbbbbbb
```

```
Input : abcabcabc 0
Output:
```

10.2.10 Substring

Write a PHP script that returns the first 3 characters from the given string in the following way:

```
Input : abc
Output: abc
```

```
Input : 12345
Output: 123
```

```
Input : PHP is the best option
Output: PHP
```

10.3 String manipulation (programs)

10.3.1 String concatenation

Concatenate the input strings in the following way.

```
Input : 5 4
Output: 54
```

```
Input : SDA ADS
Output: SDAADS
```

10.3.2 Remove a given part of the string

Write a PHP script to remove a concrete part of the string from the beginning:

```
Input : abc 2
Output: c
```

```
Input : String 4
Output: ng
```

```
Input : abcabcabc 0
Output: abcabcabc
```

10.3.3 Comment

Write a PHP script that inserts into comment the given input string and returns an empty string.

```
Input : abc
Output:
```

```
Input : 123
Output:
```

```
Input : this is a PHP script
Output:
```

Functions I.

Chapter 11

11.1 Introduction to functions

📖 11.1.1

Imagine a situation where a programmer e.g. creates the code that calculates a complex math problem. He would like to use the result of this code (even with other input values) several times. This is the ideal situation for creating functions. Functions are used when we need to run the same code multiple times. In addition to saving space, their main advantage is to make the source code more transparent.

A function is a block of commands that can be reused in a program. This function does not execute immediately when the page is loaded. It will be executed only by calling the **function**.

The function declaration begins with the **function** keyword.

```
function FunctionName(){
 // function code
}
```

📖 11.1.2

The function name cannot begin with a special symbol or number. Only letters or underscores are allowed. It should also be noted that the function name in PHP is **not case-sensitive**, there is no difference between uppercase and lowercase letters.

The following script will result in an error and is incorrect. It is actually a double declaration of the same function.

```
<?php
   function myfunction(){
       echo 'call: myfunction';
   }

   function MyFunction(){
       echo 'call: MyFunction';
   }
?>
```

📝 11.1.3

Which of the following is the correct name for the function?

- _my_function()
- 47function()
- 47_my_function()
- $my_function()

📖 11.1.4

It should be noted that in real programming, you hardly need to create your own PHP function since there are already more than 1000 built-in functions created for different areas and you can call them according to your requirements. Custom functions, however, are an excellent tool for making the code more transparent and also a good start for object-oriented programming in PHP.

The work with custom functions consists of two parts:

- Creating your own PHP function.
- Calling a custom PHP function.

It's very easy to create your own PHP function. When creating a function, the entire function code should be enclosed in {a} brackets.

Suppose you want to create a PHP function that writes a simple message to your browser when you call it. The following example creates a function called **showGreeting()** and then calls it as soon as it is created.

```
 <?php
        /* Defining a PHP Function */
        function showGreeting() {
           echo "Good morning!";
        }

        /* Calling a PHP Function */
        showGreeting();
?>
```

📝 11.1.5

What does the following script list in the document?

```
<?php
   function greeting_for_friend() {
      echo "Hello!";
   }
```

```
   function greeting_formal() {
      echo "Good morning!";
   }

   greeting_for_friend();
?>
```

- Hello!
- Good morning!

11.1.6

In PHP, we also have the option to insert parameters inside a function. You can use as many parameters as you want. These parameters act as variables in your function.

The following example has two integer parameters, sums them and then writes that sum together with the comment to the document.

```
 <?php
     function sumFunction($num1, $num2) {
           $sum = $num1 + $num2;
           echo "Sum of the two numbers is: $sum";
     }

     sumFunction(10, 20);
?>
```

11.1.7

Enter the correct function name and parameter order to make the following script work correctly. The **age()** function will count the number of years for the specified people and write the result to the document using echo.

```
<?php
   function _____(_____, _____) {
      $year = 2019;
      echo $fname.' is '.($year-$birth).' years old <br>';
   }

   age('Miloslav', 1975);
   age('Jarmila', 1992);
```

```
   age('Martina', 1979);
?>
```

11.1.8

Typical functions do not usually write their results directly into a document. Their task is to perform the calculation and send the result to the programmer, who practically decide how to use it in the script. The **return** command is used to send the result of the function.

In the example, we created a currency conversion function, i.e. to calculate how much EUR we get for USD (US dollar).

```
<?php
    function usd_to_eur($usd){
        $eur = $usd * 0.9;
        return $eur;
    }
    echo "1 USD = " . usd_to_eur(1) . " EUR <br>";
    echo "50 USD = " . usd_to_eur(50) . " EUR <br>";
    echo "1000 USD = " . usd_to_eur(1000) . " EUR <br>";
?>
```

11.1.9

Fill in the correct line to send the **multiplication()** result.

```
<?php
   function multiplication($number1,$number2){
         $result = $number1 * $number2;

         _____
   }

   echo "6 * 12 = " . multiplication(6, 12) . "<br>";
   echo "7 * 4 = " . multiplication(7, 4) . "<br>";
   echo "15 * 3 = " . multiplication(15, 3);
?>
```

11.2 String functions (1)

📖 11.2.1

Strings are referred to as the basic data types in PHP. Also, multiple scripts create string output to the page. There are several useful functions to work with. We have previously found out that strings are written in quotation marks (") or in apostrophes (').

Although it appears at first sight that the quotation marks or apostrophes are identical, they are not. If we use quotation marks, we can use in strings so-called escape sequence as well as direct calling of variable content. This is not possible with apostrophes. Apostrophe strings are the simplest, and PHP doesn't add anything to them. In theory, they should also be slightly faster in processing than other ways of writing strings.

E.g.:

```
<?php
   $nick = "Bubba";

   echo "One rainy evening $nick talked with Forrest. <br>";
   //Output: One rainy evening Bubba talked with Forrest.

   echo 'One rainy evening $nick talked with Forrest. <br>';
   //Output: One rainy evening $nick talked with Forrest.
?>
```

📝 11.2.2

Use the correct character for the strings to make the following code correct.

```
<?php
   $partner1 = 'Bubba';
   $partner2 = 'Gump';

   echo _____The $partner1  $partner2 Shrimp Company_____;

   //Output: The Bubba Gump Shrimp Company
?>
```

11.2.3

In PHP, the situation is solved if we need to write an apostrophe/quote character in the string. The solution may be or a combination of quotation marks and apostrophes or the use of a backslash.

```
<?php
   echo "The character apostrophe ( ' ) is important in php
<br>";
   //Output: The character apostrophe ( ' ) is important in
php

   echo 'The character apostrophe ( \' ) is important in php
<br>';
   //Output: The character apostrophe ( ' ) is important in
php
 ?>
```

Of course, if the apostrophe character can also be printed, there is a special way of listing the slash. For slash listing it is necessary to double the slash, i.e. type twice slash \\

```
<?php
   echo 'The character slash: \\ <br>';
   echo 'Two slashes: \\ \\<br>';
?>
```

These pairs of characters that begin with the backslash \ are called **escape sequences**. This is virtually the only limitation of the apostrophic chain. For this reason, several programmers prefer apostrophes to quotes when working with strings.

11.2.4

What does the following script show on the screen?

```
<?php
   echo '\\ \\ \\';
?>
```

- 3 backslashes
- 6 backslashes
- Nothing
- 2 backslashes

- only 1 slash

📖 11.2.5

PHP creators tried to make sure that all characters could be written using quoted strings. For this reason, is used, the so-called **escape sequence**. These are the character sequences that begin with the backslash character \ and allow all characters to be written. To repeat, we write two backslashes to write a backslash \\, to write quotations we use **\"**. To write the dollar, we use **\$**.

Sometimes writing a character in hex code is useful. It is written in pairs of **\x** characters and continues with a hexadecimal number. For example, you can type a space as **\x20** in hexadecimal.

E.g.:

```
<?php
   echo "h\x65llo\040goodbye";
   //Output: hello goodbye
?>
```

📖 11.2.6

There are several useful functions for working with strings. Some of the selected ones will be presented below. The first functions are **strtolower()** and **strtoupper()**. They are used to convert all characters of a string to uppercase or lowercase.

```
<?php
   echo strtolower('FOREST');
   //Output: forest

   echo '<br>';

   echo strtoupper('Bubba');
   //Output: BUBBA
?>
```

Similar functions are **ucfirst()** and **lcfirst()**, which are used to convert the first character in a string to uppercase or to convert the first character in a string to a lowercase letter.

```
<?php
   echo ucfirst('bubba');
   //Output: Bubba
```

```
   echo '<br>';

   echo lcfirst('BUBBA');
   //Output: bUBBA
?>
```

11.2.7

Add the correct functions to the script where the output is:

„RUN, Forrest, RUN!“

```
<?php
   $first = _____( 'run');
   $second = _____( 'run');
   echo $first.', Forest, '.$second.'!';
?>
```

11.2.8

Like other programming languages, strings can be accessed as character arrays. I.e. string can be understood as an array, where each element of the array is a character with a specified order - index.

In the example, there is the solution of the output of the first and second characters in the string. Note that the first character has an index of 0, the second has an index of 1, and so on.

```
<?php
   $nick = 'Forest';
   echo 'First char: '.$nick[0];
   echo '<br>';
   echo 'Second char: '.$nick[1];
?>
```

11.2.9

There is a saved string **'RUN'** in the variable **$go**. Print each letter of the variable into a new line.

```
<?php
```

```
   $go = 'RUN';
   echo $go[_____].'<br>';
   echo $go[_____].'<br>';
   echo _____.'<br>';
?>
```

11.2.10

When working with indexes it is often a problem to "keep an eye on" the maximum index, i.e. we cannot enter an index higher than the number of characters in the string, otherwise, the script will end up with an error. Not only for this problem is the **strlen()** function useful. The function is used to calculate the number of characters in a string. The character spacing is also counted.

```
<?php
   $go = 'RUN, Forrest, RUN!';
   echo 'String $go have '.strlen($go).' characters.';
?>
```

11.2.11

Use the **strlen()** function to determine the number of characters in **$nick** and write the first and last characters of the variable.

Remember that the index starts at 0, so the first character has an index of 0.

```
<?php
   $nick = 'Forest';
   $count = _____($nick);
   echo 'First character: '.$nick[_____];
   echo '<br>';
   echo 'Last character: '.$nick[_____];
?>
```

11.2.12

Complete the script to work correctly and the string in the variable *$nick* will print each letter (character) of the string to a new line.

```
<?php
   $nick = 'Forest';
   $count = _____($nick);
```

```
   for($i=0;$i<=$count-1;$i++){
      echo $nick[_____];
      echo '<br>';
   }
?>
```

11.3 String functions (2)

11.3.1

If we know how to "break" strings into an array of characters, it may be useful to "break" strings into array elements, e.g. the most common case is break sentences into words. The **explode()** function is used to convert strings into an array variable. The function has two mandatory parameters, i.e. a separator, actually a sub-string to divide the string. The second parameter is the string itself. The last optional parameter specifies the number of array elements to return.

The example:

```
<?php
   $sentence = 'My Mama always said you've got to put the past
behind you before you can move on.';
   $words = explode(' ', $sentence);
   print_r($words);
?>
```

Note: In the example, we used the **print_r()** function. The function displays information about the variable in a way that is human readable. It is often used to display the contents of an array or variables where it is not known their data type or have no information about.

11.3.2

Use the correct function to split the numbers of the variable *$numbers* into array elements. Remember to specify the correct separator.

```
<?php
   $numbers = '0,1,2,3,4,5';
   $arr = _____(',', $numbers);
   print_r($arr);
?>
```

11.3.3

What does the following script do?

```
<?php
   $sentence = 'My Mama always said you've got to put the past
behind you before you can move on.';
   $words = explode(' ', $sentence);
   $min = 1000;
   foreach ($words as $word) {
        $len = strlen($word);
        if ($len<$min){
            $min = $len;
         }
   }
   echo $min;
?>
```

- Calculates and prints the length of the shortest word in the sentence.
- It counts the length of all characters of the sentence and prints them.
- It counts the number of words and prints the count.

11.3.4

You can use the **substr()** function to select a part of a string. The function has three basic parameters. The first is the string itself, the second is the starting point position, and the third is the number of characters to return.

If the third parameter is not specified, the function returns all characters from the position specified in the second parameter to the end of the string. It may also be helpful to set the second parameter as a negative number. In this case, the function takes the number of characters from the end of the string as the starting point position.

E.g.:

```
<?php
   echo substr("Forrest Gump",8)."<br>";
   //Output: Gump

   echo substr("Forrest Gump",11)."<br>";
   //Output: p

   echo substr("Forrest Gump",-4)."<br>";
```

```
   //Output: Gump

   echo substr("Forrest Gump",-1)."<br>";
   //Output: p
?>
```

11.3.5

Set the second and third parameters correctly to print only 'rest' with **substr()** function.

```
<?php
   echo substr("Forrest Gump",_____,_____)."<br>";
   //Output: rest
?>
```

11.3.6

When working with text editors, we have discovered the replacement function, i.e. exchanging part of the string for another. This functionality is provided by PHP function **str_replace()**. The function is used to find and replace a substring. The function accepts three arguments. The first argument is the text to be replaced (i.e. the text in the original string), the second argument is the replacement text, and the third argument is the analyzed text.

In the example in 'go, Forrest, go!' we replace the word 'go' with the new word 'run'. Note that the function executes all replacements, i.e. in our example, it will replace both occurrences of the word 'go'.

```
<?php
   $output = str_replace('go','run','go, Forrest, go!');
   echo $output;
   //Output: run, Forrest, run!
?>
```

11.3.7

Add the correct function and select the correct order of the arguments so that the function replaces the word 'surprise' with the word 'miracles' in the string saved in the *$sentence* variable.

```
<?php
```

```
   $sentence = 'My mama always told me that surprise happen
every day.';
   $output = _____('_____','_____',$sentence);
   echo $output;
   //Output: My mama always told me that miracles happen every
day.
?>
```

11.3.8

The last function we introduce is **str_pos()**. The function is used to find the position of the character(s) in the string. This function accepts two arguments. The first is the searching string, the second is the searched string.

```
<?php
   $sentence = 'My mama always told me that miracles happen
every day.';
   echo 'The word mama starts at the position:
'.strpos($sentence,'mama');
?>
```

11.3.9

What will be the output of the following script?

```
<?php
   $sent= 'My mama always told me that miracles happen every
day.';
   echo strpos($sent, 'miracles');
?>
```

- 28
- 5
- 3
- 10

Functions II.

Chapter 12

12.1 Array functions

📖 12.1.1

Arrays as a data structure, mostly consisting of the same elements, are often used and very useful in PHP. For this reason, we will continue to pay attention to it. We have already learned about the basic arrays information in the previous chapters.

First, we want to point out how arrays can be searched as a whole. The first way is to determine the array size (using the ***count()*** function) and use the for a ***loop***.

```
<?php
   $friends = array("Bubba", "Dan", "Jenny");
   $all = count($friends);

   for($i = 0; $i < $all; $i++) {
      echo $friends[$i];
      echo "<br>";
   }
?>
```

📝 12.1.2

Complete the correct script to print the contents (individual elements) of the numeric array in rows.

```
<?php
$numbers = array(1,2,3,4,6);
$count_num = count($numbers);

for($i = 0; $i < $count_num; $i++) {
   echo $numbers[_____];
   echo "<br>";
}
?>
```

📖 12.1.3

If we are working with an associative array, we do not know the array indexes, or we work with an "unknown" array, it is not possible to use the for loop with an index. In such cases, we can search the array using a foreach loop. Many programmers prefer this method for the reason of simplicity and avoiding index problems.

```
<?php
    $friends = array("Bubba", "Dan", "Jenny");
    foreach($friends as $f) {
         echo $f."<br>";
    }
 ?>
```

12.1.4

Complete the correct script to list the contents (individual elements) of the numbered array in rows. Set the correct variables in the **foreach** cycle.

```
<?php
   $numbers = array(1,2,3,4,6);

   foreach(_____ as _____) {
      echo $num."<br>";
   }
?>
```

12.1.5

In the case of associative arrays, we often need to determine not only the value of an array element but also its index. In the following example, we show an example of reading both the index and the value of an array element.

```
<?php
   $birth = array("Miloslav"=>"1975", "Jarmila"=>"1992",
"Martina"=>"1979");
   foreach($birth as $x => $x_value) {
       echo $x . " was born in " . $x_value;
       echo "<br>";
   }
?>
```

12.1.6

A function for sorting array elements - **sort()** - is a useful function when working with arrays. For the control statements in the example, we use the **print_r()** function, which can format data in variables in a readable way.

```
<?php
   $friends = array("Jenny", "Bubba", "Dan");
```

```
   //output before sorting
   print_r($friends);
   echo "<br>";

   //sorting
   sort($friends);

   //array output after sorting
   print_r($friends);
?>
```

12.1.7

Use the correct function to sort the array elements saved in the **$numbers** variable.

```
<?php
   $numbers = array(4,2,5,1,3);

   _____(_____);

   print_r($numbers);
?>
```

12.1.8

At the end of this section, we note that the **sort()** function is used to sort the array elements ascending, i.e. from A to Z for text values and from the smallest to the largest for numeric values.

The function of a descending order is **rsort()**. Its use is the same as for **sort()**. Similarly, for associative arrays, some functions sort arrays by associative indexes (ascending and descending) - **ksort()**, **krsort()**, or a special function for associative arrays for sorting by values (ascending and descending) - **asort()**, **arsort()**.

12.2 Date and time functions

12.2.1

Any larger information system can handle time data and work with them. Date and time are so much a part of everyday life that it is easy to work with without thinking.

PHP provides tools for working with these data structures that make it easy to manipulate dates or times.

It should be noted that most technologies in computers work with so-called *unix time*. This number represents the number of seconds that have passed since midnight on January 1, 1970. This moment is known as the UNIX epoch and the number of seconds that have passed is referred to as the timestamp.

The basic function in PHP is the **time()** function, which displays an integer - the time stamp of the current date and time.

```
<?php
   echo time();
?>
```

The result of the **time()** function is quite complicated but PHP offers tools to convert the timestamp into a form more readable for the general user.

12.2.2

Add the correct function to load the current time in unix time format and use it correctly in the script

```
<?php
   $stamp = _____();
   echo 'Since 1st January 1970 has passed '.$stamp.'
seconds.';
?>
```

12.2.3

The first option to add a readable format for the current time function is the **date()** function. The function returns a formatted string representing the date. The function has two arguments, the first one defines the output format and the second one is the time stamp.

```
date(format,timestamp)
```

The first format parameter of the **date()** function determines how to format the date (or time). Here are some of the characters commonly used for dates:

- **d** - represents the day of the month (01 to 31)
- **m** - represents the month (01 to 12)
- **Y** - represents the year (in four digits)

- **l** (small 'L') - represents the day of the week

Other characters, such as "/", "." or "-" can be inserted between characters to put additional formatting.

```
<?php
   echo "Unix time starts from ".date("Y/m/d",1)."<br>";
   echo "Unix time starts from ".date("d. m. Y",1)."<br>";
   echo "Unix time starts from ".date("d-m-Y",1)."<br>";
   echo "Unix time starts from ".date("l",1);
?>
```

Note: The second parameter is the time stamp. For number 1, it is the time since the seconds begin to count in unix time.

12.2.4

Add the correct formatting characters "d", "m" or "Y" to generate the correct output in the script:

```
<?php
   echo "Unix time starts in month ".date("_____",1)."<br>";
   echo "Unix time starts in year ".date("_____",1)."<br>";
?>
```

12.2.5

If the second parameter is omitted in the **date()** function, the function uses the current date and time.

The two commands in the example give the same result. In the first one, we used the **time()** function as the parameter, in the second the parameter is omitted.

```
<?php
   echo "Today's date ".date("d. m. Y",time())."<br>";
   echo "Today's date ".date("d. m. Y")."<br>";
?>
```

12.2.6

If you know that the day has 85400 seconds, which command to display tomorrow's date is correct?

- echo "Tomorrow will be ".date("d. m. Y",time()+85400);
- echo "Tomorrow will be ".date("d. m. Y",85400);
- echo "Tomorrow will be ".date("d. m. Y",date()+85400);
- echo "Tomorrow will be ".date("d. m. Y",+85400);
- echo "Tomorrow will be ".date("d. m. Y",day()+85400);

12.2.7

As the date, time can be displayed using the **date()** function with the time formatting characters. The most commonly used characters are:

- **H** - hours- 24-hour format (00 to 23)
- **h** - hours- 12-hour format with leading zeros (01 to 12)
- **i** - minutes with leading zeros (00 to 59)
- **s** - seconds with leading zeros (00 to 59)
- **a** - the letters am or pm

```
<?php
   echo "The time is " . date("h:i:s a");
?>
```

12.2.8

Which of the below commands will output a simple time in the format *hours:minutes* (e.g. 14:58), with the time in the 24-hour format (00 to 23).

- echo "The time is " . date("H:i");
- echo "The time is " . date("H:i:s a");
- echo "The time is " . date("h:i:s a");
- echo "The time is " . date("H:i:s");

12.2.9

The **date()** function is usually sufficient to display the current time. Complications may happen if we need to work with a time or date different than the current one. Recalculating Unix time is very difficult. PHP uses two functions to convert time to timestamp - unix time. The first is the **mktime()** function, which converts the specified date and time (specified in each function parameter) into a timestamp. The following sequence of parameters must be ordered in the function.

```
mktime(hour,minute,second,month,day,year)
```

E.g.:

```
<?php
   $d=mktime(18, 00,00, 12, 24, 2019);
   echo "Christmas dinner will start at: " . date("H:i:s
(d.m.Y)", $d);
?>
```

12.2.10

Add the correct function to print the departure of the bus at 12:14:54pm (11.07.2020).

```
<?php
   $d=_____(12, 14, 54, 11, 7, 2020);
   echo "The bus departures: " . date("h:i:sa (d.m.Y)", $d);
?>
```

12.2.11

There is an interesting function **strtotime()**, which is used to convert the English text date and time into a timestamp. The function accepts a parameter as a string in English that represents a date and time description, e.g. “now” indicates the current date.

The example below is an interesting parameter creation for **strtotime()**. Note, however, that the function is not perfect, so be sure to always check the strings you enter.

```
<?php
   $d=strtotime("14:35 July 10, 2019");
   echo date("Y-m-d H:i:sa", $d). "<br>";

   $d=strtotime("tomorrow");
   echo date("Y-m-d h:i:sa", $d) . "<br>";

   $d=strtotime("next Saturday");
   echo date("Y-m-d h:i:sa", $d) . "<br>";

   $d=strtotime("+3 Months");
```

```
   echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

📖 12.2.12

There is also a **getdate()** function in PHP. An optional parameter of the function is a timestamp. The function provides an associative string containing date and time information as a return value. If you omit the time stamp, it works with the current timestamp returned by the *time()* function.

The associative string that is the return value of a function has the following indexes:

- **seconds** - seconds (0-59)
- **minutes** - minutes (0 - 59)
- **hours** - hours (0 - 23)
- **mday** - day (1 - 31)
- **mon** - month (1 - 12)
- **year** - in 4-digit format, e.g. 2019
- **weekday** - day of the week (e.g. Thursday)
- **month** - month of the year (e.g. January)

```
<?php
   $date = getdate();
   echo "Today's date: ".$date['mday'] .
"/".$date['mon']."/".$date['year'];
?>
```

12.3 Password functions (password_hash, password_verify)

📖 12.3.1

PHP, as a server-side scripting language, is responsible for the entire so-called back-end functionality of the website. Usually, authentication is the most important part of the web system. This is also the most common case where developers make mistakes, then it becomes unconsciously vulnerable.

The first problem is the storage and use of user passwords, which can lead to an unauthorized person gaining access to the database and the entire system is at risk. When working with passwords it is often used so-called **password hashing**. It is a method that encrypts a user password (a string consisting of characters of

different lengths) into a string with a fixed (same for all passwords) length, adds random characters, and so on.

📖 12.3.2

PHP has several functions for password hashing. The first function is **md5()**, which calculates the MD5 hash. The function has two parameters, the first is the string to be encrypted, the second is optional. If it is set to TRUE, the output string will be a 16-character binary format, if FALSE then a 32-character hexadecimal number. This is also the default setting if we do not specify the second parameter.

```
<?php
   $pass = "supersecure";
   echo "32 character hex number: ".md5($pass)."<br>";
?>
```

Sha1() and **hash()** have similar functionality but with a different type of hash.

📖 12.3.3

Another level of security that programmers often use is so-called **salt**. Salt or salting passwords is a way to make passwords more secure even when system users underestimate passwords and use only simple strings as passwords. Salt is nothing but a long string of different characters entered by the programmer (most often generated). This tends to be uniform for the whole system. Whenever you need to work with passwords, the salt programmer adds to a string in which the password (at the beginning or at the end) is before the use of the selected hash function.

```
<?php
   // variable $salt has to be set
   $pass = "supersecure";
   $hash = md5($pass . $salt);
?>
```

It is important to note that any enhancement of the system “protection” is directly focused on a certain area of security. Using salt is useful if the user password database is leaked (stolen). If programmers used salt, unauthorized use of these passwords is virtually impossible (unless the salt string is stolen). Salt mainly protects against this type of attack. E.g. it is practically unusable when attacking the system by generating user (administrator) passwords.

📖 12.3.4

Password hash technologies such as MD5 and SHA1 are currently considered older and weaker, although a large number of developers still use them. The currently recognized best practice for hash passwords is the so-called **bcrypt**. However, working with this password hash can be complicated. There is a **password_hash()** function in PHP that simplifies working with hash passwords.

The first parameter is the password string that must be hashed, and the second parameter specifies the algorithm that should be used to generate the hash. The default algorithm (*PASSWORD_DEFAULT* parameter) is currently **bcrypt**, but a stronger algorithm may be added as the default algorithm later in the future to generate a larger string. You can also directly enter *PASSWORD_BCRYPT* as the second parameter.

The advantage of this function is that it is the whole API, which e.g. automatically takes care of adding salt. If you want to work with your own salt string, you can do so in the third function parameter.

```
<?php

   $hash = password_hash($password, PASSWORD_DEFAULT);

?>
```

📖 12.3.5

At the end of this section, we provide a useful password verification function. The **password_verify()** function has two parameters. The first is the password to be verified (string), the second is the hash. The function returns true if the hash matches the specified password.

Note that in a real system, a hash is saved in a database and must be retrieved from the database. In our example, it is typed directly in a string variable.

```
<?php
   $password = "my_pass";
   $hash =
'$2y$10$fIQf0N1XeSzWqiOCdBPc7e5wp.fbyp9x6FUNbbRULG9eZDqNI.w5q'
;
   if (password_verify($password, $hash)){
        // Success!
        echo "OK";
    }else {
        // Invalid credentials
        echo "Problem";
```

```
    }
 ?>
```

12.3.6

Enter the correct function and order of parameters to verify that the given password matches the *$pass* variable with the hash saved in the *$hash* variable.

```
<?php
   $pass = "secrete";
   $hash =
'$2y$10$5Vdma7NrZBqLfbMljFhiWuTiTLIEahwn9Srj8K0zCZNTJboDM18gi'
;
   if (_____(_____, _____)) {
         // Success!
        echo "OK";
   }else{
        // Invalid credentials
        echo "Problem";
   }
 ?>
```

Cookies and Sessions

Chapter 13

13.1 Cookies

📖 13.1.1

The problem with PHP (as well as other server-side programming languages) is working with **global variables**. A larger web application consists of multiple PHP files, each of which runs separately. And not to be so easy, each file is run and generated separately for each visitor to the website.

The first option for saving global variables (especially for each user) is so-called **cookies**. A cookie is a small text file that a web server saves on a client computer. Files are saved in a dedicated folder on the user's computer, containing the name of the web page from which they originated, validity, and set variables (along with variable values). After the cookie is set, other scripts from the domain can read the values of the set variables. Cookies can only be read from the domain from which they were written. For example, a cookie set using http://up.krakow.pl cannot be read from http://www.up.warsow.pl or http://www.ukf.sk. Therefore, a cookie can only be visible to the server that created it. Other users cannot see its value.

📖 13.1.2

Cookies make it easy to use the site, for example, the site remembers that you are logged in with your nickname and you do not need to re-enter your login information the next time you load.

Today's world is characterized by connected services on the web. Most websites on the Internet can also display elements from other domains, most often an ad or social networking link or traffic measurement tool. These domains can also create their own cookies. These are referred to as third-party cookies.

Most web browsers have options to disable cookies, third-party cookies, or both. In this case, PHP scripts must rely on sending global variables by URL or in headers.

📝 13.1.3

What are third-party cookies?

- These are cookies which are places on a page different than the site on which they are currently located.
- These are cookies that are located in a special folder on the server.
- These are cookies on other servers that have been specially encrypted by a third party.

13.1.4

We create a cookie using the **setcookie()** function. Most often this function is called with three parameters, e.g.:

```
<?php
   setcookie("my_variable", "value", time() + 15);
 ?>
```

Using this function call, we set the cookie variable *my_value* with the value. The last parameter is the validity of the cookie. To keep order in cookies on client computers, each variable is set to be valid, i.e. until it is saved on the client computer. The client computer system has a mechanism that periodically clears an expired cookie. Most often, the cookie is set using the *time()* function, which returns the current date and time. We then add the time in seconds to this value. The cookie in the example will be deleted after 15 seconds. After this time, we can no longer read its value.

13.1.5

At what time will the following cookie be deleted?

```
<?php
   setcookie("time", "40", time() + 60);
 ?>
```

- 1 minute
- 40 minutes
- 40 seconds
- 60 hours
- 60 days

13.1.6

Cookies are part of the HTTP header. Therefore, in the PHP script, if cookies are not set before other outputs, the cookies will not be sent.

This problem can be solved by the use of the so-called **output buffering** function, or very simply by following the rule that **setcookie()** must be used before the <html> tag.

To modify an existing cookie, **setcookie** is called again for the same variable.

13.1.7

Set the cookie variable “nick” with the value “Bubba” for 2 minutes.

```
<?php
   _____("_____", "_____", time() + 120);
 ?>
```

13.1.8

The setcookie function has several parameters (although we set only the variable name, value, and expiration time to the standard). All the parameters of the function are as follows:

```
setcookie(name, value, expire, path, domain, secure, httponly)
```

Their meaning is given in the table.

Parameter	Description	Data type
name	The name of the cookie variable.	String
value	The value saved in the client computer.	String
expire	Validation cookie that is given in unix time format, i.e. number of seconds from 1.1.1970 (Unix Epoch).	Integer
path	Path on th server where will be the cookie available.	String
domain	The domain for which the cookie is available.	String
secure	If the value is TRUE, the cookie is available only for a secured connection.	Boolean
httponly	If the value is TRUE, the cookie is available only through HTTP protocol. Scripting language such as JavaScript will not have access to the cookie.	Boolean

13.1.9

You can read the cookie value using the **$_COOKIE** associative array. Obviously, in this array, we always find the values of the variables set for a particular client, i.e. web browser. The values in the array can be read by any script (PHP file) that accesses it from the domain that created it. To display all variables and their values, we can use the **print_r()** function, which displays information about the variable in a readable format. It is often used to list the array where the programmer

does not know exactly the indexes or associative index. We can use it to read the value of a cookie as follows:

```
<?php
     print_r($_COOKIE);
?>
```

13.1.10

In most cases, you do not just need to view the set cookies, but you need to read and continue to use them. To read a cookie, just enter the cookie variable name in the **$_COOKIE** associative array, e.g.

```
<?php
     echo $_COOKIE[“nick”];
?>
```

If the script tries to read a cookie that is not created, the script will result in an error. It is therefore a good practice to check that a cookie is set before using a cookie. The **isset()** function returns a boolean return value as a result of checking the existence of a variable. It is also often used with cookie variables.

```
<?php
 if(isset($_COOKIE[“nick“])) {
    echo "You are logged as '" . $_COOKIE["nick"] . "'";
 } else {
    echo "Cookie is not created";
 }
 ?>
```

13.1.11

Verify that the “last name” cookie is created and write the value using the echo command.

```
<?php
   if(isset($_COOKIE["_____"])) {
       _____ "Hi '" . $_COOKIE["last_name"];
   }
?>
```

13.1.12

What function do we use to clear a cookie?

- setcookie
- delete_cookie
- cookie_out
- erasecookie

13.1.13

What does the following source code print in the document?

```
<?php
   setcookie("nick", "bubba", time() + 3600);
   setcookie("age", "32", time() - 3600);
?>
 <html>
 <body>

 <?php
    if(!isset($_COOKIE["age"])) {
      echo "The age is unknown !";
    }else{
      echo "Nick '" . $_COOKIE["nick"] . "' is " .
$_COOKIE["age"];
    }
 ?>

 </body>
 </html>
```

- The age is unknown
- Nick bubba is 32
- Nothing

13.2 Sessions

13.2.1

A technology similar to cookies (which even uses cookies to function) is sessions. A session is a way to store information (in variables) to be used on multiple pages.

These are global variables stored on the server. Each session is assigned a unique ID, which is used to retrieve stored values on the server. Whenever a session is created, a cookie containing a unique session ID is stored on the user's computer and returned with each hit. Unlike a cookie, this is a way of storing global variables on a server; only the ID for accessing these variables is stored on the client.

If the client browser does not support cookies, the unique session ID will work with the URL. In this case, they are still saving to the server and to access them are used sessions. Compared to a cookie, a session has the capacity to store larger data.

The session can be used in the following situations:

- To upload values from one page to another.
- In browsers that do not support cookies, that is, as an alternative to cookies.
- For storing global variables in an efficient and secure way compared to selling them via URL
- For the development of more complex applications, e.g. editing items in the shopping cart
- in e-shops etc.

13.2.2

What technology is involved if only IDs are saved on client computers and are used to store other data on the server?

- Sessions
- Cookies
- database

13.2.3

To start working with sessions, use the **session_start()** function. Like cookies, this session function must be preceded by HTML tags.

Next, the work with variables in the session is done using the associative array **$_SESSION**.

```
<?php
  // Start the session
  session_start();
?>
<!DOCTYPE html>
  <html>
    <body>
```

```
      <?php
        $_SESSION["person"] = "Forest";
        $_SESSION["friend"] = "Bubba";
      ?>

</body>
</html>
```

13.2.4

In the script, create the *wife* variable and set its value to "Jenny"

```
<?php
  // Start the session
  session_start();
?>
<!DOCTYPE html>
<html>
  <body>

    <?php
      $_SESSION["_____"] = "_____";
    ?>

  </body>
</html>
```

13.2.5

Read the "person" and "wife" variables in the next HTML script.

```
<?php
  // Start the session
  session_start();
?>
<!DOCTYPE html>
<html>
  <body>

    <?php
      echo $_SESSION["_____"] . "and his wife" .
$_SESSION["_____"];
```

```
      echo " Just married!";
    ?>
  </body>
</html>
```

📖 13.2.6

The **session_unset()** and **session_destroy()** functions are used to delete all global variables.

```
<?php
   session_start();
?>
<!DOCTYPE html>
<html>
  <body>

   <?php
     // delete all variables session
     session_unset();

     // destroy session
     session_destroy();
   ?>

  </body>
</html>
```

📖 13.2.7

Sometimes it is enough to delete only one variable set by session. In this case, we can use the **unset()** functions.

```
<?php

  unset($_SESSION['product']); //destroy product session item

?>
```

Files

Chapter 14

14.1 Files

📖 14.1.1

Files are the basic method of storing data on a server. Instead of the often complicated work with a database, we can put simple data in files. In the following, we will focus on working with text files, usually with the type *.txt*. We will gradually learn the functions of reading and writing text files.

If we want to work with an existing file, it is a common part of scripts to check if the file is even at that location, if the file exists. The **file_exists()** function can do this check, where we specify the file name as a parameter.

```
<?php
  $exist = file_exists("my file");
?>
```

📝 14.1.2

Fill in the correct code: if there is a "letter.txt" file, it will output a message with an **echo** function.

```
<?php
if (_____('letter.txt'))
 {
     _____ 'The file exists!';
 }
?>
```

📖 14.1.3

There are several functions for working with files, even some of them have similar functionality. The basic and practically universal function is **readfile()**, which reads the text content of the file.

```
 <?php
    echo readfile('letter.txt');
 ?>
```

📝 14.1.4

Which function can we use to load a text file?

- readfile
- view_file
- see_file
- file_readln

14.1.5

Make sure there is a 'letter.txt' file. If it exists, read its contents using **readfile()**.

```
<?php
if (_____('letter.txt')) {
      _____;
 }
?>
```

14.1.6

A similar function, with the same result, is the **file_get_contents()** function. This function is also used to read the entire contents of a file but unlike **readfile()** it does not display it. Text files use the line separator as "\n" but in HTML it is
. For this reason, it is mostly used with HMTL <pre> tags, which write contents to HTML without further formatting.

```
<?php
      echo "<pre>";
      echo file_get_contents("letter.txt");
      echo "</pre>";
?>
```

14.1.7

The functions **readfile()** or **file_get_contents()** read the contents of the file as a whole. There are other functions to read the contents of the file.

We use the **fopen()** function to start working with the file.

The arguments of the function include the name of the file to be opened in the so-called mode in which we want to open the file. The result of the **fopen()** function is a pointer to an open file.

```
<?php
   $file = fopen('letter.txt',`r`);
?>
```

14.1.8

The **fopen()** function is used to open a file. In the second parameter, we specify in which mode we want to open the file.

We have the following modes to choose from:

- **(r)** - Opens the file as read-only, reads the file from start. Returns *false* if it does not exist.
- **(r+)** - Opens a file to read and write, reads the file from start. Returns *false* if it does not exist.
- **(w)** - Open the file for writing, start writing from the beginning of the file, if the file does not exist, attempt to create it, if it exists, delete its original content.
- **(w+)** - Opens the file for both read and write, starts writing from the beginning of the file if it does not exist, tries to create it, if it exists, deletes its original content.
- **(a)** - Opens the file for writing, writes to the end of the file, i.e. it is used for writing, adding information to the file, the original content will be preserved if the file does not exist, try to create it.
- **(a+)** - Opens the file for both write and read, writes to the end of the file, i.e. it is used for writing, adding information to the file, the original content will be preserved if the file does not exist, try to create it.

14.1.9

What mode is used to open the file, if the file already exists with the contents, we just want to write more content into the file (we will not read its contents)?

- a
- a+
- w
- w+

14.1.10

When working with a file, it is necessary to close the file correctly. Closing correctly ensures that changes to the file are saved. In PHP, closing a file is done using the **fclose()** function. The function parameter is a pointer to an open file.

```
<?php
   $my_file = fopen('letter.txt',`r`);
   //work with a file
```

```
   fclose($my_file)
?>
```

When you include the file existence check function, then the practical work with the file may look like this.

```
<?php
   $my_file = 'letter.txt';
   if (file_exists($file_name)){
      $my_file = fopen('letter.txt',`r`);
      //work with a file
      fclose($my_file)
    }else{
      echo 'File '.$file_name.' not found!';
    }
?>
```

The second alternative could be the following script:

```
<?php
  $file_name = 'letter.txt';
  $my_file = fopen($file_name, "r") or die("Cannot open
file!");
  //work with a file
  fclose($my_file);
 ?>
```

The **die()** function is called if an error occurs. It prints the set error message, and the message terminates the execution of the script.

14.1.11

Add the correct functions to open and close the file.

```
<?php
  $file_name = 'letter.txt';
  $file1 = _____($file_name, "r") or die("Cannot open file!");
  //work with a file
  _____(_____);
 ?>
```

14.1.12

The previously mentioned text file reading functions were able to read the entire file as a single text array. However, we often need to find out where a new line is in the file (so we can assign it a special format, not just
). For this purpose, **fgets()** is used to read the file line by line. Its parameter is a *pointer* to the file. Each function call reads one line of the file (the function starts at the beginning of the file, i.e. the first call is the first line, then the file line pointer moves to the second line, etc.).

```
<?php
   $sub = fopen('letter.txt', 'r') or die('Cannot open the
file!');
   $line = fgets($sub);
   echo 'The first line: '.$line.'<br>';
   fclose($sub);
?>
```

14.1.13

Fill in the script to print the first and second line to the document from the file.

```
<?php
   $sub = fopen('letter.txt', 'r') or die('Cannot open the
file!');
   $line1 = fgets($sub);
   $line2 = _____(_____);
   echo 'The first line: '.$line1.'<br>';
   echo 'The second line: '.$line2;
   fclose($sub);
?>
```

14.1.14

An unknown number of file lines may be a problem for **fgets()**. If you try to read a non-existent line, the script will not work correctly. It is therefore recommended that we always check that a line exists. The **feof()** function is known, which checks whether the end of the file has been reached - End-of-file (EOF). This function is useful for looping through files whose size (number of lines) is unknown.

```
<?php
   $file = fopen('letter.txt', 'r') or die("Cannot open the
file!");
   // checking if the end is reached
   while(!feof($file)) {
```

```
    $line = fgets($file);
    echo $line . "<br>";
  }
  fclose($file);
?>
```

14.1.15

What function is used to determine if we reached the end of a file?

- feof()
- ef()
- end_file()
- end_of_file()

14.1.16

Fill in the correct order of commands to load the contents of the entire text file.

```
<?php

  _____

  _____

     $line = fgets($file);
     echo $line . "<br>";
  _____

 _____
?>
```

- }
- $file = fopen('letter.txt', 'r') or die('Cannot open the file!');
- fclose($file);
- while(!feof($file)) {

14.2 Files – other functions

14.2.1

PHP specifications include using the same function to open a file as well as create it. This function is **fopen()**. If you use the **fopen()** function in mode (**a**) or (**w**) on a file that does not exist, it creates it.

Sometimes an error may occur when creating a file. Then you need to check and allow access to PHP files to write information to your hard drive.

After the file is created, you need to fill it with content. The **fwrite()** function is used to write data to a file. The first parameter of the **fwrite()** function contains the name of the file to write data to, and the second parameter is the string to write to.

```
<?php
   $new_file = fopen("letter.txt", "w") or die("Cannot open
the file!");
   $introduction = "Dear Jenny \n";
   fwrite($new_file, $introduction);
   fclose($new_file);
 ?>
```

The sign "*\n*" is a line-end character for text file types.

The **fopen** function also has a third parameter, the so-called "*$length*". This parameter specifies the maximum write length. The parameter is optional, if the length argument is specified, writing to the file stops when the specified byte size is reached or the end of the string being written, whichever comes first.

14.2.2

Fill in the correct **fwrite()** parameters

```
<?php
   $file = fopen("newcomers.txt", "w") or die("Cannot open the
file!");
   $line = "Bubba\n";
   fwrite(_____, _____);
   fclose($file);
 ?>
```

14.2.3

By calling **fwrite()** again, we can write more lines. Add “Forrest” on the second line and “Captain Dan” on the third line

```
<?php
   $file = fopen("newcomers.txt", "w") or die("Cannot create
the file!");

   $line = "Bubba\n";
   fwrite($file, $line);
   $line = " Forrest \n";
   fwrite(_____, _____);
   $line= " Captain Dan \n";
   fwrite(_____, _____);

   fclose($file);
 ?>
```

14.2.4

It should be remembered that the way data is written to the file is determined by the file open mode. When creating a new file, we specify the “w” or “w+” mode. This mode is also used if the file already exists, but we want to overwrite it, i.e. delete all of its content and replace it with a new one.

For an existing file to which we write data, we use the mode to open “a” or “a+”.

It is a good practice to separate lines with the "\n" character. However, its use is not necessary. We add it if we assume that the created file will be opened by users in other text editors, e.g. in text editors and so on. If the file will be used (written and read) only PHP scripts with a print to the web browser and we will use a function different than **fgets()** for the printing, so we can replace the line end character “\n” with e.g. tag "
".

14.2.5

For completeness, we will introduce file operations functions. The first is the **copy()** function, which is used to copy files. It has two required parameters, the first specifies the file path and file name to copy. The second parameter specifies the path and file name of the newly created file.

```
<?php
   copy('letter.txt', 'letter_backup.txt') or die('Cannot copy
the file');
   echo 'The file was successfully copied to letter_backup ';
?>
```

📖 14.2.6

The last function is **unlink()**, which is used to delete a file. Its use can be as follows:

```
<?php
   if (!unlink("letter_backup.txt")){
      echo "Cannot delete the file";
   }else{
      echo "The file 'letter_backup.txt' was successfully
deleted";
   }
?>
```

PRISCILLA

priscilla.fitped.eu