Back-end API Development

2024-2025 course material

Outline

1. Course structure
2. PHP as a back-end language
3. Cloning GIT repository
4. Installing PHP server locally
5. Running PHP server locally
6. Documentation
7. PHP tags
8. Variables
9. Value types
10. Showing results in browser

Outline

Outline

Outline

Course structure

- Git repository

  https://github.com/pascal-tm/back-end-api-development

  - **Slides**
    - Serve as the guide throughout the course

  - **Examples**
    - Mentioned throughout the course **in green**
    - In order to see these examples, you need to have a PHP server running
    - The examples can be found in the Git repository

Course structure

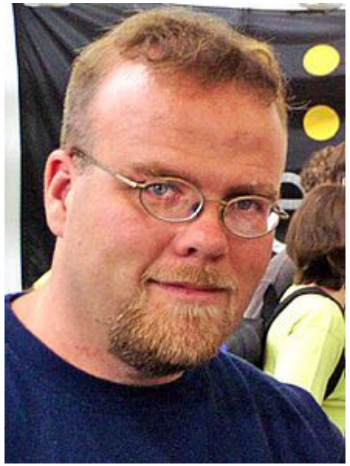
- Git repository:
    - **Exercises**
        - Mentioned throughout the course **in blue**
        - After every chapter/knowledge block
        - Needs to be made individually
        - Commit solution to your own personal git repository

Course structure

- Git repository:
  - **Revision exercises**
    - Mentioned throughout the course **in orange**
    - After every few chapters.
    - Combines all the previous
    - Needs to be made individually
    - Commit solution to your own personal git repository
      -

PHP as a back end language

- Scripting language (<-> programming language)
- 1994 IBM (Rasmus Lerdorf)
- Perl / C / Python
- **P**erson **H**ome **P**age (formerly)
- PHP: Hypertext Processor (recursive acronym)
- Server side
  - Alternatives:



PHP as a back end language

- Main goal:
  - Making HTML dynamic
    - HTML = static, requires a lot of manual labor to maintain every page
    - Leverage PHP to more easily maintain a large website
  - Processing information
    - creating a bridge between the user and the database

PHP as a back end language

- creating a API's
    - **A**pplication **P**rogramming **I**nterface
        - Create functions that allow interaction with your program
        - Lets applications communicate with each other in a controlled environment
            - <-> allowing direct access to the database
        - Usually returns result in JSON format (can also be XML)
        - ie. a function that will retrieve all the songs played by a specific user in the year 2024
            - https://developer.spotify.com/documentation/web-api
            - https://restful-api.dev/
            - ...

PHP as a back end language

- Multiplatform
    - Versatile
        - Runs on every server (linux/apache, windows/iss)
        - Runs on every local machine (linux, windows, mac, …)
    - Supports multiple database types
        - MySQL, SQLite, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, MongoDB, …
    - Free to use
    - *Easy* & quick to learn

Cloning git repository

- Download the course material from GitHub
  - https://github.com/pascal-tm/back-end-api-development
  - You can use any Git client
    - ie. https://desktop.github.com/download/

Installing PHP server locally (windows)

- Install Chocolatey and PHP
    - Chocolatey is a package manager
        - Allows easy installation of programs using the *command line*

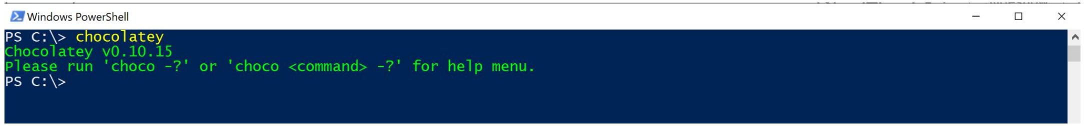
Installing PHP server locally (windows)

- Install Chocolatey & PHP using script
  - Search for **install-chocolatey-and-php.cmd** in the script folder of the repository and run it
    - Confirm every step (using A: allow all)
    - This should automatically install Chocolatey and PHP

Installing PHP server locally (windows)

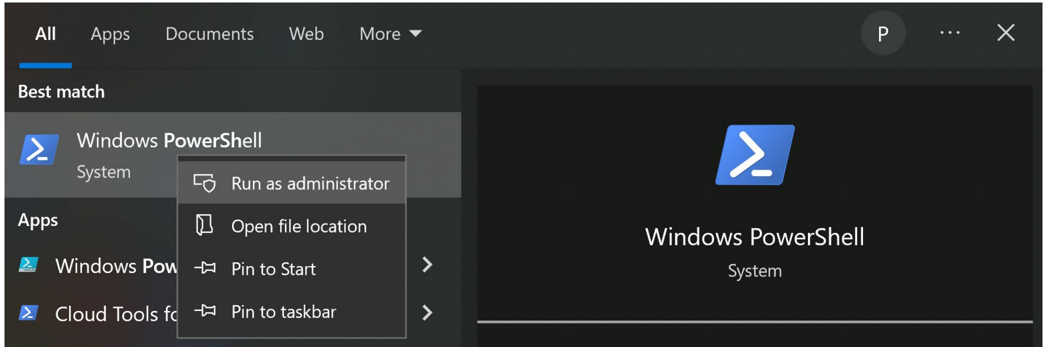
- Check if chocolatey has been installed correctly by executing the following command in Powershell

```
chocolatey
```



```
PS C:\> chocolatey
Chocolatey v0.10.15
Please run 'choco -?' or 'choco <command> -?' for help menu.
PS C:\>
```

  - Opening PowerShell: go to start, search for PowerShell, right click and *run as administrator*



Installing PHP server locally (windows)

- If this failed, you should execute the steps manually

- Open Windows Powershell
  - Start > Windows Powershell
  - Right click on the icon
  - ***Run as administrator***

- Follow the installation guide on https://chocolatey.org/install#install-step1

Installing PHP server locally (windows)

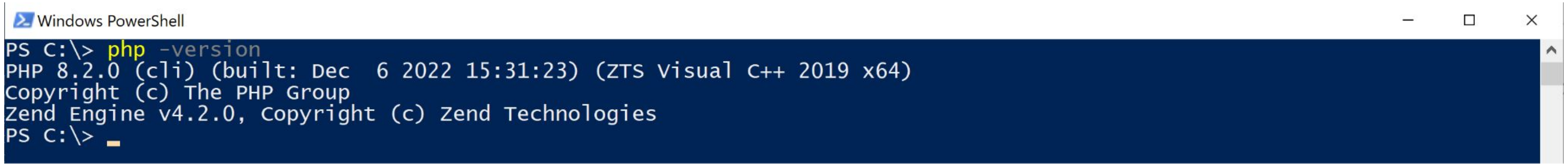
- Install PHP using chocolatey
  - https://community.chocolatey.org/packages/php
  - Open Powershell
    - Right click and make sure you **run it as administrator**
  - Execute the following command

    ```
    choco install php
    ```

    - If chocolatey asks for permission, make you you allow all (A)
  - Check if php is installed by executing the following command

    ```
    php -version
    ```

  - If you see the following result, you have installed PHP correctly



```
PS C:\> php -version
PHP 8.2.0 (cli) (built: Dec  6 2022 15:31:23) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.2.0, Copyright (c) Zend Technologies
PS C:\> _
```

Installing PHP server locally (mac)

- Mac
  - PHP should be installed by default
  - Open cmd (command line interface)
  - Make sure you have the latest version by executing the following commands

```
php -version
```

  - If it outputs anything above PHP 5, you are good to go

Running PHP server locally

- Running PHP server
    - You cannot open your PHP-files directly (ie. using a browser)
        - They need to be served using a server that knows how to read PHP files
        - Only when using a server, do you have correct access to the files using a browser
    - You only need to run the server once when booting to access your PHP files
        - But you need to restart your server every time you start a machine
    - Automatically using .bat/.bash
        - Windows: run **run-server.bat**
        - Mac: (**TODO**)

Running PHP server locally

- Running PHP server
    - Manually
        - Open Powershell (Windows) or Cmd (Mac)
        - Navigate to the folder where the course is located

```
cd c:\location\of\course
```

        - Execute the following command

```
php -S 127.0.0.101:80
```

        - Open your browser and go to the following url http://127.0.0.101:80
            - You should now have access to all the examples and exercises

Running PHP server locally

- Setting up a vhost to use the http://backend.local url (execute only once)
  - Windows:
    - Using script:
      - In the scripts folder run **create-localhost-vhost.bat**
      - Follow steps
    - Manually
      - TODO

Running PHP server locally

- Setting up a vhost to use the http://backend.local url (execute only once)

  - Mac:
    - TODO

Running PHP server online

- Running PHP server online
  - Development usually happens on your local machine
    - Which is why we need to run a server that can translate PHP files to html
  - When uploading your files to a server, PHP should be already set up
  - When you upload your files to a webhosting you can access them directly using a browser
    - This is how the internet works.
      - Web hosting (online storage) -> runs server -> runs PHP -> serves PHP files -> returns result of executing PHP files -> user sees results in the browser (ie. a web page)

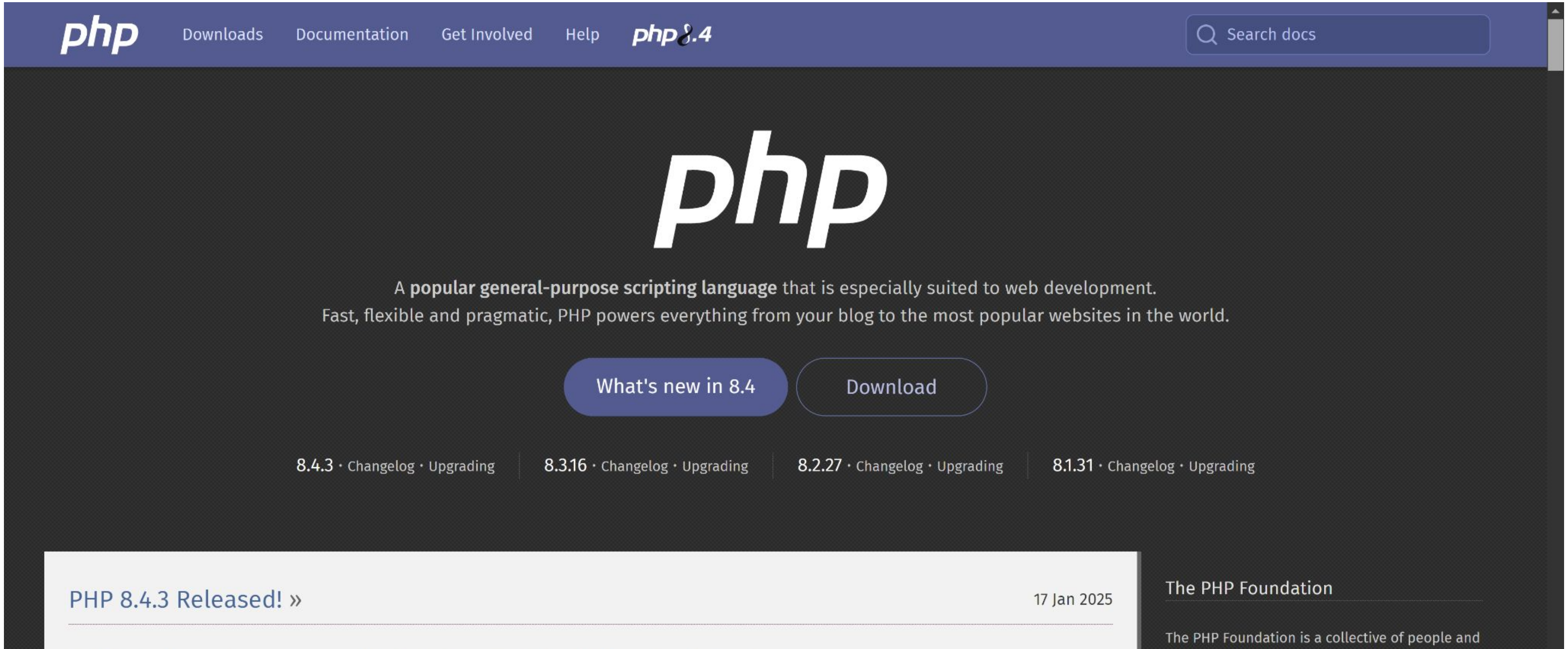
Documentation

- All the documentation on how to use PHP can be found at

https://www.php.net

- Let's analyse how to use it

Documentation


Downloads
Documentation
Get Involved
Help
php 8.4
Search docs
What's new in 8.4
Download
8.4.3 · Changelog · Upgrading
8.3.16 · Changelog · Upgrading
8.2.27 · Changelog · Upgrading
8.1.31 · Changelog · Upgrading

A **popular general-purpose scripting language** that is especially suited to web development.
Fast, flexible and pragmatic, PHP powers everything from your blog to the most popular websites in the world.

PHP 8.4.3 Released! »

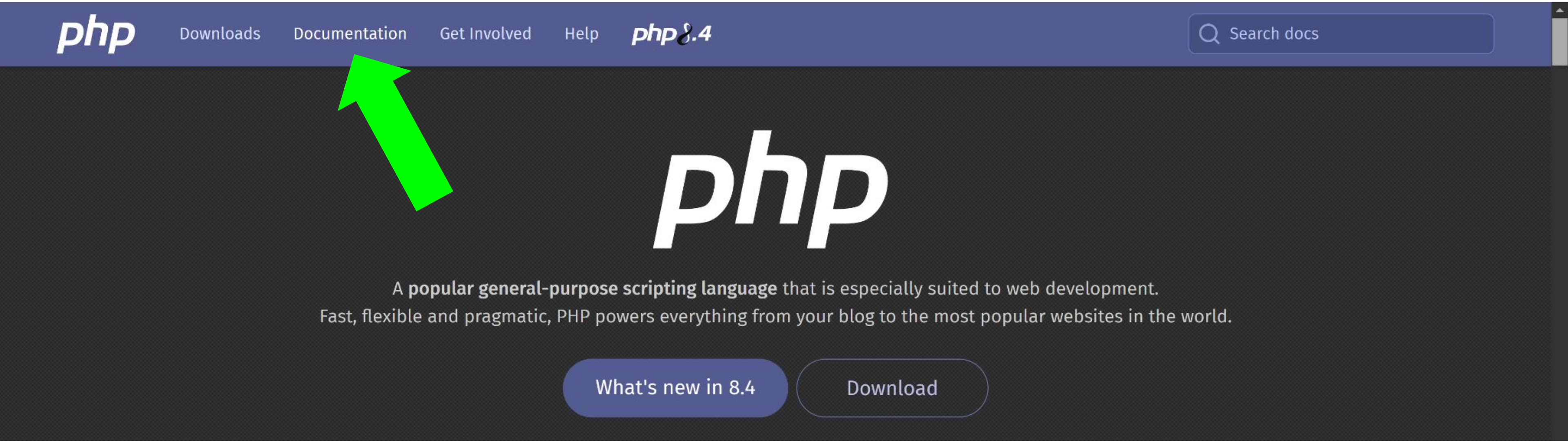
17 Jan 2025

The PHP Foundation

The PHP Foundation is a collective of people and

Documentation

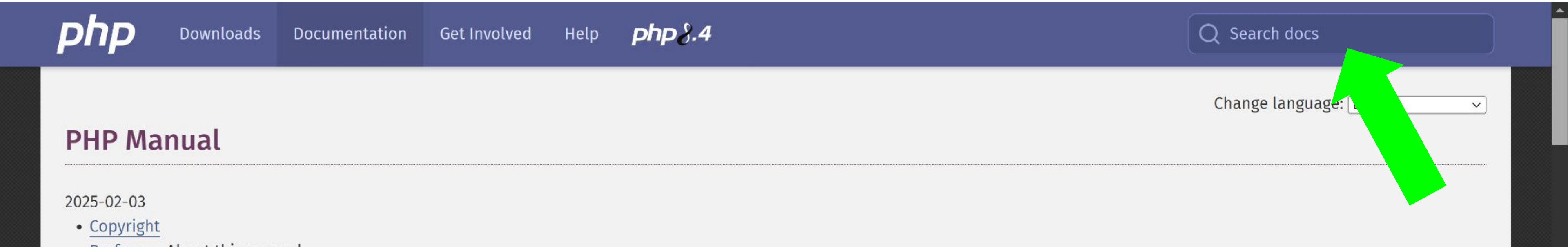
- Documentation: lists all the functionality PHP
  - Select English



A **popular general-purpose scripting language** that is especially suited to web development.
Fast, flexible and pragmatic, PHP powers everything from your blog to the most popular websites in the world.

Documentation

- *Search docs*
    - Ie. *echo*
- Or using a search engine of preference:
    - `echo php site:php.net` (if you know the functionality)
    - `ow to print something in PHP` (if you don't know the functionality, just describe what you want to achieve)



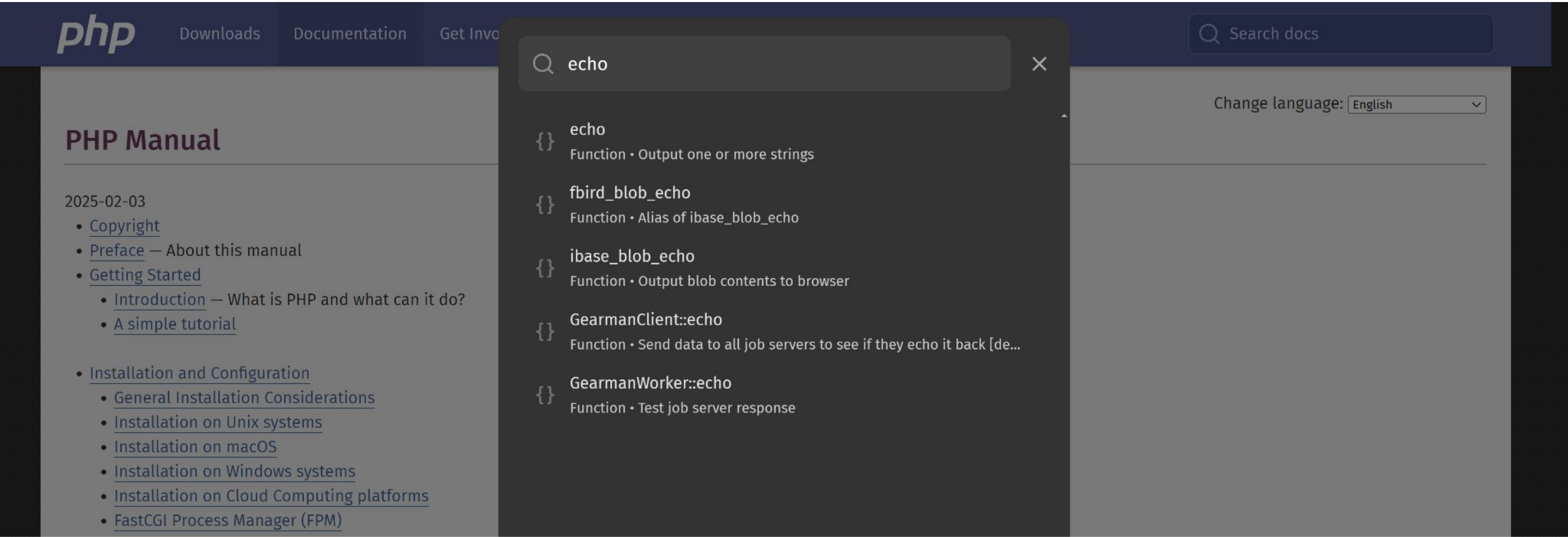
PHP Manual

2025-02-03

- Copyright

Documentation

- *Search docs*



echo

echo
Function • Output one or more strings

fbird_blob_echo
Function • Alias of ibase_blob_echo

ibase_blob_echo
Function • Output blob contents to browser

GearmanClient::echo
Function • Send data to all job servers to see if they echo it back [de...

GearmanWorker::echo
Function • Test job server response

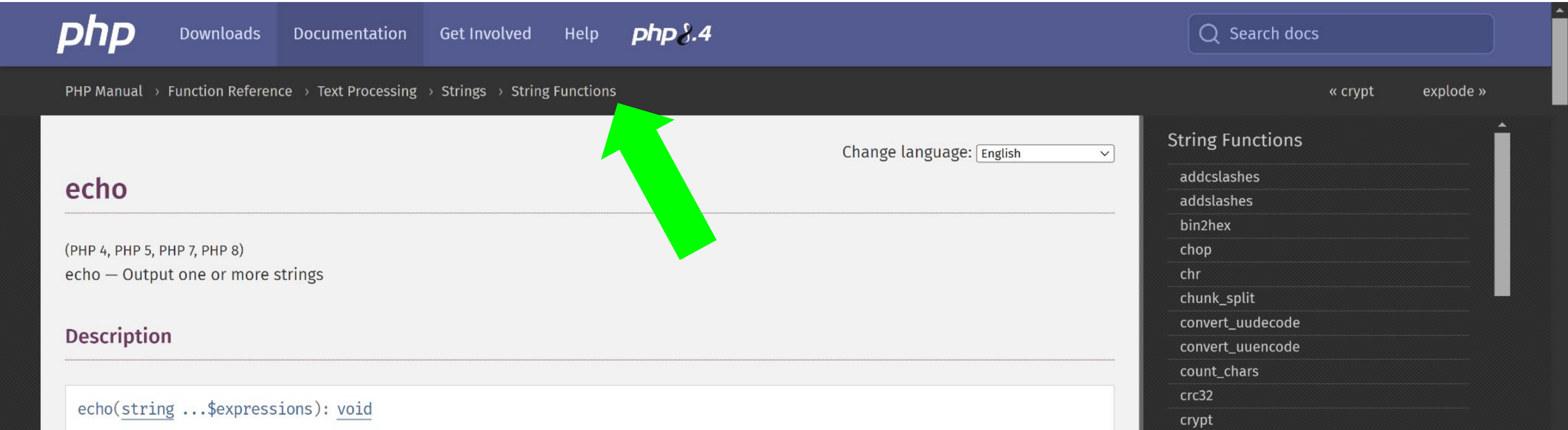
PHP Manual

2025-02-03

- Copyright
- Preface — About this manual
- Getting Started
  - Introduction — What is PHP and what can it do?
  - A simple tutorial
- Installation and Configuration
  - General Installation Considerations
  - Installation on Unix systems
  - Installation on macOS
  - Installation on Windows systems
  - Installation on Cloud Computing platforms
  - FastCGI Process Manager (FPM)

Documentation

- Search result found!
    - Breadcrumb shows family of functionality in which the result is located
    - Right pane shows all the functionality that falls under the same category



echo

(PHP 4, PHP 5, PHP 7, PHP 8)

echo — Output one or more strings

Description

```
echo(string ...$expressions): void
```

String Functions

- addcslashes
- addslashes
- bin2hex
- chop
- chr
- chunk_split
- convert_uudecode
- convert_uuencode
- count_chars
- crc32
- crypt

Documentation

- Search result found!
  - Description
    - What it does
  - Parameters
    - What needs to be passed to the function, if applicable
  - Return values
    - What it returns
  - Examples
    - Coding examples to show it in action
  - Notes
    - Additional notes, ie. something is *deprecated* (=should not be used anymore, is going to be phased out in later versions)
  - See also
    - Related functionalities
  - User contributed notes
    - Useful additional notes/bugs/behaviors are usually mentioned here

PHP Tags

- PHP files
  - Extension: .php
    - (or .phtml which combines html & php, rarely used, does the same as .php)
  - Content
    - PHP code starting with <?php and ending with ?>
    - HTML/CSS/JS/JSON … can still be mixed

  - Output can be shown as
    - HTML, JS, CSS, JSON, …
    - JPEG, PNG, GIF, …
    - pretty much every extension imaginable

PHP tags

- Script block
  - Starts with **<?php** and ends with **?>**
  - The end of every line of code should be followed up by a **;** **(!!!)**
    - Exception: the last line before the closing **?>** tag
  - **Example**: **php tag code block**

- Integrate PHP & HTML
  - Inline PHP
    - HTML document with script tags
    - Try to keep PHP and HTML separated
    - **Example**: **php tag inline syntax**

  - Full PHP (**not recommended**)
    - PHP script that contains all the logic and HTML combined
    - **Example**: **php tag non recommended syntax**

Variables

- A name for a value
  - In stead of using the value, we use the name
  - A placeholder for a value
  - Allows *portability* of the value, without having to change it in every place

$$X = 5$$

Variables

$$\underbrace{X}_{\text{variable}} = \underbrace{5}_{\text{value}}$$

Variables

$$\underbrace{X}_{\text{variable}} = \underbrace{5 + y}_{\text{expression}}$$

Variables

- Variable notation
  - Always starts with a **$** sign then followed by the name of the variable
  - The name **has to start with a letter** OR and **underscore** (_)
    - Variables starting with an _ are usually reserved for variables that belong to a class (see **Classes**)
  - Variable name can only consist of:
    - letters
    - numbers
    - underscores (_)

Variables

- Variable notation
  - Can not contain any spaces
  - The name is case sensitive (ie. `$test` and `$Test` are two different variables)
  - Type declaration not necessary (loosely typed)
  - The variable is created when the value is assigned to it
  - You can reassign any value to the variable
  - Variables starting with a capital (ie. `$Cars`) are usually **reserved for classes**

Variables

- Variable notation
    - Separating words in a variable name
        - mumble case: `$checkinsystem`
        - camel case: `$checkInSystem`(**preferred method**)
        - snake case: `$check_in_system`
    - Make the name of the variable as descriptive as possible
        - Do not use things like:
            - `$text1`
            - `$number1`
            - `$t`
        - It should be clear what the variable represents by reading the name
        - Describe what the value represents:
            - `$finalGrade`
            - `$welcomeText`

Value types

- Boolean
  - `$hasBeenSent = false;`
    - `false` can also be represented as a `0`
  - `$hasBeenSent = true;`
    - `true` can also be represented as `1`

  - Note:
    - `1 == true` will evaluate to **`true`**
    - `1 === true` will evaluate to **`false`**
      - This because the === operator ALSO checks whether the types are the same
      - In this case the 1 is an integer and the true is a boolean type

Value types

- Integers
  - `$yourGrade = 20;`
- Floats (integers containing a number after the decimal)
  - Use a period (.)
  - `$myGrade = 19.5;`
- Strings
  - Enclosed using single quotes (') or double quotes (")
  - `$car = "Pole start";`
  - `$car = 'Pole star';`

- **Example: value types bool int float and strings**

Showing results in browser

- Echo
    - Only for strings (not arrays, objects, ...)
        - Inside PHP tags
            - printed on the location of the echo in the document
            - Not recommended
            - **Example: php tag code block**

        - Inside HTML
            - `<?php echo 'test'; ?>`
            - `<?= 'test'; ?>`
                - `<?=` is shorthand for `<?php echo`
            - **Example: php tag inline syntax**
            - (also possible to use inline conditionals and looping statements, see later chapters)

- **Exercise: debug errors**

Showing results in browser

- `var_dump`
  - ie. `var_dump $myGrade;`
  - *Dumps* the value of the variable
  - Shows additional information about the variable
    - Type
    - If array, it shows the content of that array
    - stack trace
    - ...
  - Used for *debugging*
  - Do not use this in production!
    - (*production* means an environment that is used by clients)

String functions

- **Escaping PHP-reserved characters in strings**
  - Sometimes a string can contain a ' or a " which could unwillingly end the value.
    - Solution:
      - Use ' in stead of " or visa versa
        - ie. `$quote = '"I had a dream"';`
      - **Escape** the special characters using a backslash (\)
        - Ie. `$quote = "\"I had a dream\"";`
  - **Example: string escaping**

String functions

- **Concatenation**
  - Combining strings or variables containing strings
  - using a period (.)
    - `$fullText = 'This is ' . 'a string';`
    - `$fullText = $welcome . $name;`
  - **Example: string functions concatenation**
  - **Exercise: string functions concatenation**

String functions

- strlen()
  - Returns the length of a string

  - **Example: string function strlen**

- strpos()
  - Checks if a string occurs in another string
  - If the string has been found
    - Returns the position of the string (a number starting from the first character being 0)
  - If the string hasn't been found
    - Returns false

  - **Example: string function strpos**

- **Exercise: string functions**

Operators

Arithmetic Operators	Description	Example	Result
+	add	$x = 3; $x = $x + 4;	 7
-	subtract	$x = 5; $x = 8 - $x;	 3
*	multiply	$x = 3; $x = $x * 2;	 6
/	divide	$x = 6; $x = $x / 2;	 3
%	modulo	$x = 10; $x = $x % 4; $x = $x % 5; $x = $x % 3;	 2 (rest) 0 1 (rest)
++	increment	$x = 3; $x++;	 4
--	decrement	$x = 3; $x--;	 2

- Example: operators arithmetic

Operators

Assignment operators	Example	Result
=	$x = 3;	3
+=	$x = 3; $x += 5;	(3 + 5) = 8
-=	$x = 3; $x -= 1;	(3 -1) = 2
*=	$x = 3; $x *= 2;	(3 * 2) = 6
/=	$x = 6; $x /= 2;	(6 / 2) = 3
.=	$x = 3; $x .= 2;	(3 . 2) = 32
%=	$x = 8; $x %= 4;	(8 % 2) = 0

- **Remark:** if possible, choose assignment operator (=more concise) instead of arithmetic operator

Operators

Comparison operators	description	example	result
==	Equals to	"test" == "test" "4" == 4	true true
===	Equals to and also checks data type	"4" === 4	false
!=	Does not equal to	4 != 3	true
>	Is bigger than	5 > 9	false
<	Is smallar than	3 < 9	true
>=	Is bigger than or equal to	8 >= 9	false
<=	Is smaller that or equal to	8 <= 8	true

- **Example: operators comparison**

Operators

Logic operators	description	example	result
&&	and	1 == 4 && 1 == 1 1 == 1 && 1 < 2	false true
||	or	1 == 4 || 1 == 1 1 == 1 || 1 < 2	true false
!	negate	!true !(1==4)	false true

- Example: operators logic
- Exercise: string functions

Conditional statements

- **Four different conditional statements**
  - If
  - If … else …
  - If … elseif … else …
  - switch

Conditional statements

- If statement
    - Syntax

        -
          ```
          if (condition)
          {
              Code to execute
          }
          ```

Conditional statements

- **Shorthand if statement**
  - **Syntax**
    - `(condition) ? code to execute when true : code to execute when false`
  - Primarily used
    - In HTML to print small pieces of text, like classnames
    - To quickly assign numbers or strings to a variable
  - **Example: conditional statements if**
  - **Exercise: conditional statements if**

Conditional statements

- if else
  - Syntax

    -
      ```
      if (condition)
      {
           Code to execute when condition is true
      }
      else
      {
           Code to execute when condition is false
      }
      ```
  - Example: conditional statements if else
  - Exercise: conditional statements if else

Conditional statements

- if else if
  - Syntax

    -
      ```
      if ( condition )
      {
            Code to execute when this condition is true
      }
      else if( condition2 )
      {
            Code to execute when this condition is true
      }
      else
      {
            Code to execute when none of the conditions above are true
      }
      ```
  - Example: conditional statements if else if
  - Exercise: conditional statements if else if

Conditional statements

- Switch
  - Syntax

    -
      ```
      switch ($variable)
      {
          case "red":
              Code to execute when this case is true
              break;

          case "green":
              Code to execute when this case is true
              break;

          default:
              Code to execute when this case is true
      }
      ```

Conditional statements

- **switch**
  - Is used only to compare the value of one variable, not for elaborate conditions
    - operators are not allowed in the condition
    - `break;` is required if you don't want the code below to execute
      - Sometimes you might want this, when the variable needs to execute the same code for different cases
    - `default` is used when none of the cases match
      - Not required

  - **Example: conditional statements switch**

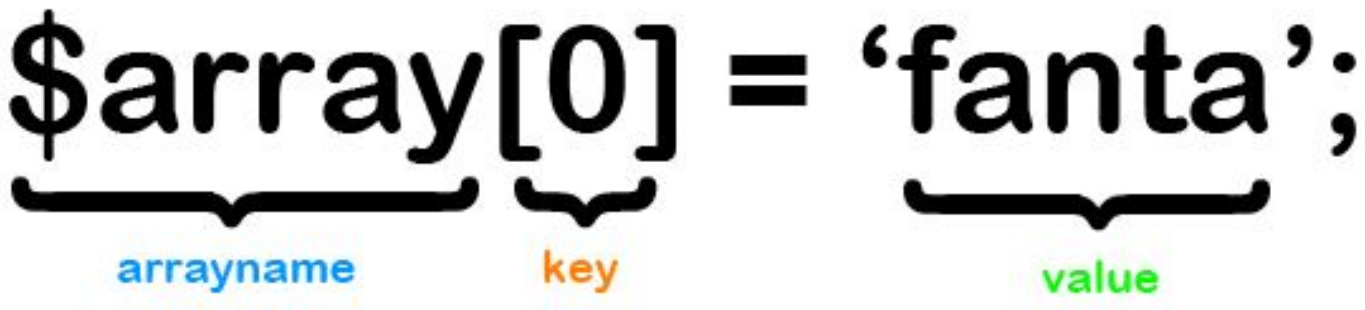
  - **Exercise: conditional statements switch**

Arrays

- An array is a variable that can contain multiple values
- There are two types of arrays:
    - **Numeric arrays** (always starts counting at 0)
        - `$soda[0] = 'Dr Pepper';`
    - **Associative array**
        - `$soda['type'] = 'diet';`

Arrays

- Syntax



```
$array[0] = ‘fanta’;
```

Arrays

- The purpose of an array is to collect values that belong together
    - Ie.
      ```
      $soda1 = 'Dr Pepper';
      $soda2 = 'Fanta';
      $soda3 = 'Evian';
      ```

      Better:

      ```
      $sodas[0] = 'Dr Pepper';
      $sodas[1] = 'Fanta';
      $sodas[2] = 'Evian';
      ```

- This allows you to loop over the array to print all the values, without having to know how many values are inside (see loops)

Arrays

- Creating numeric arrays:
    - `$sodas = array('Dr Pepper', 'Fanta', 'Evian');`

    OR:

    ```
    $sodas[] = 'Dr Pepper';
    $sodas[] = 'Fanta';
    $sodas[] = 'Evian';
    ```

- Using the $array[] notation allows you to add new items at the end of the array without having to know how many values it already contains

Arrays

- Calling a value at a specific location of an array:
    - `$user = ('name' => 'Lynch', 'surname' => 'David');`

      OR:

      ```
      $user['name'] = 'Lynch';
      $user['surname'] = 'David';
      ```

- Usually used to describe different characteristics of the same object (can also be done using classes, see later)

Arrays

- Looking up the value of a specific array entry
  - `$sodas['DrPepper']`
  - `$sodas['DrPepper'];`
- Debugging an array to see it's full content (don't use this in production!)
  - `var_dump($sodas);`
  - `echo '<pre>'; print_r($sodas); echo '</pre>';` (for pretty printing)
- Using the `$array[]` notation allows you to add new items at the end of the array without having to know how many values are already inside
- **Example: arrays assignment**

Arrays

- Multidimensional arrays
  - Array in an array
  - Creating multidimensional arrays:

```
$sodas = array(
    'Cola'   =>  array( 'Regular', 'Zero', 'Light' ),
    'Fanta' =>   array( 'Regular', 'Lemon')
);
```

  - **Example: arrays multidimensional**

  - **Exercise: arrays basic**

Array functions

- count()
  - Counts the amount of values in the arrays
  - Returns the amount of values (returns 0 if the array is empty)
  - **Example: array function count**
- in_array()
  - Checks if the value occurs in an array
  - Returns `true` if found, `false` if not
  - **Example: array function in array**
- isset()
  - Checks if the key is set **AND** the value is not null
  - Returns `true` if found, `false` if not
  - **Example: array function isset**
- **Exercise: arrays functions**

Looping statements

- Looping statements executes a code block during a specified amount of times or until a specific condition is met

- Four types of looping statements
  - while
  - do … while
  - for
  - foreach

Looping statements

- **while**
  - executes a code block as long as the while condition is met
  - Syntax

    -
      ```
      while (condition)
      {
              Code to execute
      }
      ```
  - Usually used to execute code blocks based on conditions that require you to wait for something to be completed. Ie. establishing a connection to a database
    - This action is **blocking** or **synchronous** meaning that it will not proceed until the code block has been fully executed.
  - The code that is executed usually also changes something in the condition
    - Beware of infinite loops!
  - **Example looping statements while**
  - **Exercise looping statements while**

Looping statements

- **do while**
  - Difference with while loop is that the code block is executed first and only then is the condition being checked. As long as the condition evaluates to true, the code block will keep executing.

  - Syntax

    -
      ```
      do
      {
              Code to execute
      }
      while (condition)
      ```

  - **Example looping statements do while**

Looping statements

- **for**
  - executes a code block x amount of times using a specific increment
  - Syntax

    -
      ```
      for (init; condition; increment)
      {
          Code to execute
      }
      ```
  - **Example looping statements for**
  - **Exercise looping statements for**

Looping statements

- **foreach**
  - specifically used to loop through an array
  - Syntax
    - Looping through it retrieving just the value

    -
      ```
      foreach ($array as $arrayValue)
      {
          Code to execute using the $arrayValue
      }
      ```
    - **Remark**: The name of `$arrayValue` can be anything that describes the value

Looping statements

- **foreach**
  - specificall y used to loop through an array
  - Syntax
    - Looping through it retrieving the key and the value
    - Mostly used for associative arrays

    -
      ```
      foreach ($array as $arrayKey => arrayValue)
      {
           Code to execute using the $arrayKey and $arrayValue
      }
      ```
    - **Remark**: The name of `$arrayKey` and `$arrayValue` can be anything that describes the key/value
  - **Example looping statements foreach**
  - **Exercise looping statements foreach**

Looping statements

- **Alternative syntax for looping statements to be used in HTML**
  - if/for/foreach/while/switch have alternative syntaxes to be used inside HTML

  - Syntax

    -
      ```
      <?php if($a == 5): ?>
           A is equal to 5
      <?php endif; ?>
      ```

    - After every opening statement a : is needed

    - To close the statement endif/endofr/endforeach/endwhile/endswitch

  - **Example looping statements alternative syntax**

Functions

- Used to easily reproduce certain functionality
  - When you see you are rewriting the same code over and over, it's probably best to put this functionality in a function
- Syntax

```
function functionName( $parameter )
{
    Code to execute
}
```

- **Example functions declaration**
- **Exercise functions basic**

Functions

- A function always starts with the `function` word followed by the function name which you can choose yourself

- A function name always starts with a letter or an underscore any other or characters

- A function name should describe what the function does.
    - Do not use abbreviations

Functions

- The variables that are coming from the outside can be caught in so called parameters:
  - A function can have between no parameters and an infinite amount
    - `function exampleOfNoParameters(  )`
    - `function exampleOfOneParameters( $`**`parameter1`**` )`
    - `function exampleOfMultipleParameters( $`**`parameter1`**`, $`**`parameter2`**` )`
  - When the parameter is assigned a value, ie. when the function is called it is called an **argument**
  - A parameter can have a default parameter value
  - Try

Functions

- When the parameter is assigned a value, ie. when the function is called it is called an **argument**
  - `callFunction( "teacher" );`

- A parameter can have a default parameter value
  - function defaultParameterValue ($role = "teacher") { … }
  - The assigned value parameters always come at the end when it is combined with parameters that don't have a default value:

```
function defaultParameterValue ( $name, $role = "teacher") { … }
```

Calling the function:

```
    defaultParameterValue( "Pascal");
```

Functions

- Scope of variables
  - All the code that is written in de code block of the function is scoped to that function
  - This means that the variables can not be accessed outside of the function
  - These are so called **local variables**
  - There are also **global variables**, which are accessible from everywhere

    - To access global variables use the following:

      ```
      $name = "Pascal"
      function editName() {
           global $name;
           echo $name;
      }
      ```

      Only use this for configuration variables, it is usually better to use parameters to make variables accessible from inside the function code block.

  - **Example functions scope**

Functions

- Scope of variables
  - Static variables are variables that are declared inside of the function and will retain their value when the function is called again
    - When calling the function everything is reset and the values of the variables will be set back to their original value
    - You can overwrite this reset by using the `static` keyword

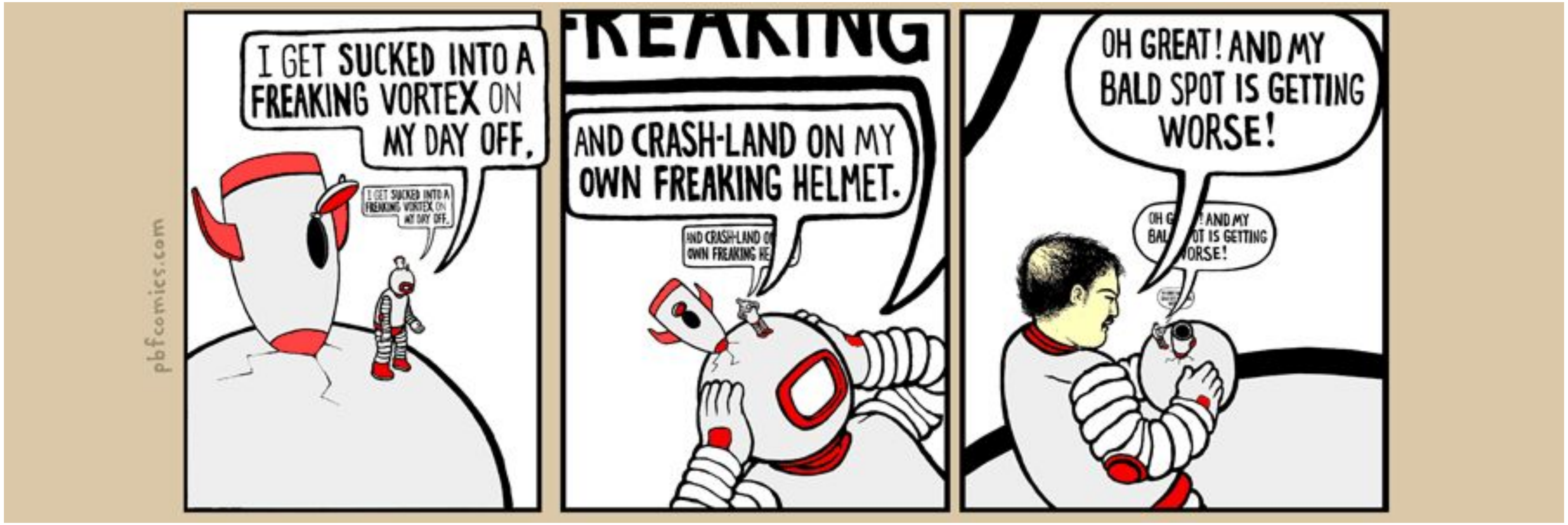
  - These are usually used in classes and should be avoided using in functions. The preferred way is a combination of parameters and return statements

- **Example functions scope static**

- **Exercise functions advanced**

Functions

- Recursive functions
  - Recursive functions are functions that call themselves


pbfcomics.com
I GET SUCKED INTO A FREAKING VORTEX ON MY DAY OFF.
I GET SUCKED INTO A FREAKING VORTEX ON MY DAY OFF.
FREAKING
AND CRASH-LAND ON MY OWN FREAKING HELMET.
AND CRASH-LAND ON OWN FREAKING HE
OH GREAT! AND MY BALD SPOT IS GETTING WORSE!
OH G ! AND MY BAL OT IS GETTING ORSE!

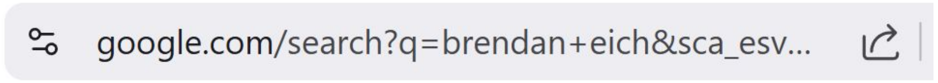
Functions

- Recursive functions
    - Call themselves
    - Beware! This can cause infinite loops if not applied correctly
        - At some point the function needs to stop calling itself

- **Example functions recursive simple**
- **Example functions recursive return**
- **Example functions recursive advanced**
- **Exercise functions recursion**

Superglobals $_GET

- $_GET variable
  - A way to send data to the server
  - Visible to everyone
  - Visible in the URL
    - Advantage: can be easily shared.



google.com/search?q=brendan+eich&sca_esv...

  - Limited amount of characters: max. 2000 chars

Superglobals $_GET

- $_GET variable
  - In the URL
    - Use a **?** to indicated that GET-variables are being added to the url
    - Everything after the first **?** will be interpreted as a GET-variable
    - The value of the GET-variables are in the format **key=string**
    - Separate multiple variables by using the **&** sign

  - Syntax
    - One variable:
      https://www.url.com/**?ref=logo**
      https://www.spotify.com/index.php**?ref=logo**

    - Multiple variables
      https://www.instagram.com/**?profile=me&ref=google**

Superglobals $_GET

- $_GET variable
  - Usage
    - You can create the url manually by concatenating the variables in the correct format in the URL-string (there are also functions that will convert arrays to get-variable notation…)
    - You can create an HTML form that will allow user input:

```
<form action="validate.php" method="GET">
     <input type="text" name="email">
</form>
```

- **action** is the page the data needs to be sent to
- **method** is the way the data needs to be sent
- **name** of the input field will be used as the key in the $_GET variable

- BEWARE! Not safe to pass user sensitive data like passwords
  - The data is passed through the URL and will be visible in the history.

Superglobals $_GET

- $_GET variable
    - Usage
        - Retreiving data from the $_GET variable in php
            - `$_GET['email']`
            - This means $_GET is an array
            - use `isset($_GET['email'])` to validate whether the $_GET variable has been set

        - **Example superglobals get**
        - **Exercise superglobals get**

Superglobals $_POST

- $_POST variable
    - Is sent as a payload using the browser
    - Not visible in the url (still visible using the dev tools, but relatively safe)
    - Unlimited information size
    - Used to send sensitive data to the server (ie. user data)

Superglobals $_POST

- $_POST variable
    - Sending data to the server
        - You can create an HTML form that will allow user input:

```
<form action="validate.php" method="POST">
    <input type="text" name="email">
</form>
```

- **action** is the page the data needs to be sent to
- **method** is the way the data needs to be sent
- **name** of the input field will be used as the key in the $_GET variable

Superglobal $_POST

- $_POST variable
  - Usage
    - You can create an HTML form that will allow user input:

```
<form action="validate.php" method="get">
    <input type="text" name="email">
</form>
```

- **action** is the page the data needs to be sent to
- **method** is the way the data needs to be sent
- **name** of the input field will be used as the key in the $_GET variable

- BEWARE! Not safe to pass user sensitive data like passwords
  - The data is passed through the URL and will be visible in the history.

Superglobals validation

- When $_GET or $_POST are being called without being assigned
  - It will return an error message:

Notice: Undefined index: … in … on line …

- Checking whether a specific superglobal key can be used:

```
if ( isset( $_POST[ 'key' ] ) ) { // Code to execute }
```

- **Example superglobals validation**
- **Exercise superglobals validation**

Debugging

- Xdebug to add breakpoints, show pretty prints of var_dump, debug remote instances of code

- Installation
    - Download the correct XDebug version
    - To see which version of php you have, open a command/terminal window and execute the following command:

```
php --version
```

Debugging

- Xdebug to add breakpoints, show pretty prints of var_dump, debug remote instances of code

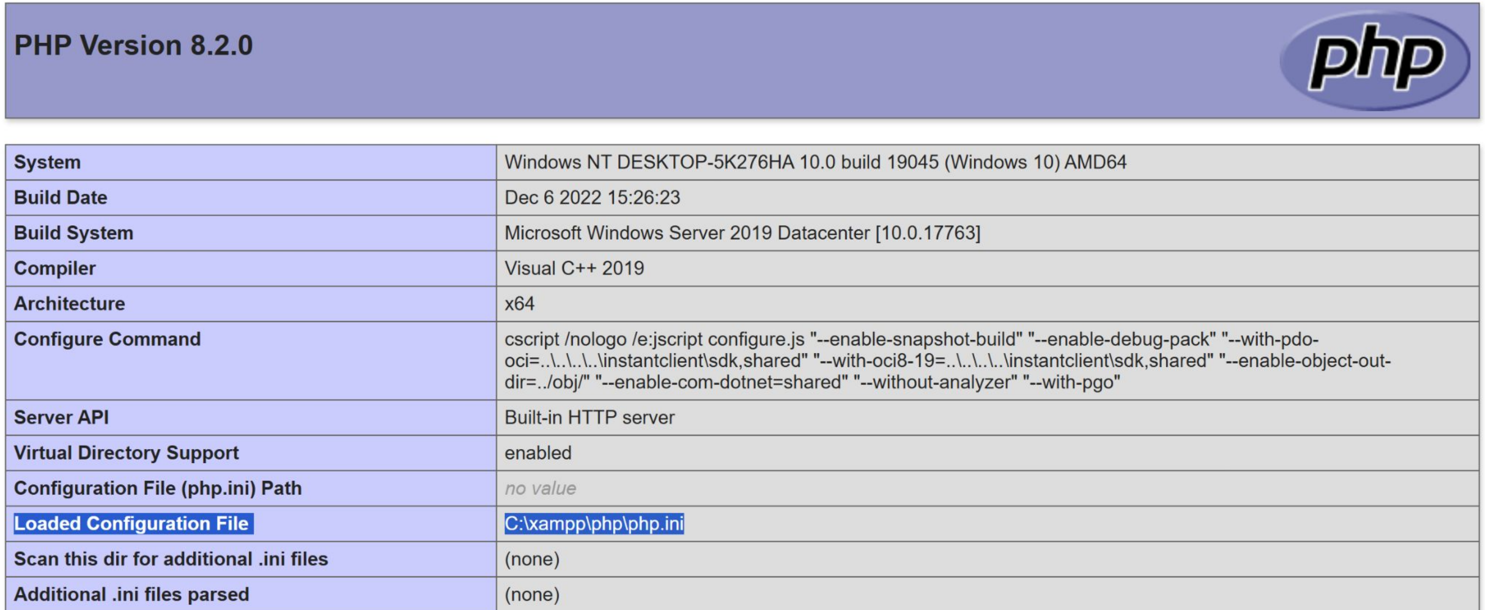
- Installation
  - First you need to know which PHP version you are using
    - open a command/terminal window and execute the following command:

      ```
      php --version
      ```

  - Download the correct XDebug from the following URL: https://xdebug.org/download#releases

Debugging

- Installation
    - Look for the line that mentions where your **php.ini** is located at



PHP Version 8.2.0

System	Windows NT DESKTOP-5K276HA 10.0 build 19045 (Windows 10) AMD64
Build Date	Dec 6 2022 15:26:23
Build System	Microsoft Windows Server 2019 Datacenter [10.0.17763]
Compiler	Visual C++ 2019
Architecture	x64
Configure Command	cscript /nologo /e:jscript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=..\..\..\..\instantclient\sdk,shared" "--with-oci8-19=..\..\..\..\instantclient\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Built-in HTTP server
Virtual Directory Support	enabled
Configuration File (php.ini) Path	*no value*
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)

Debugging

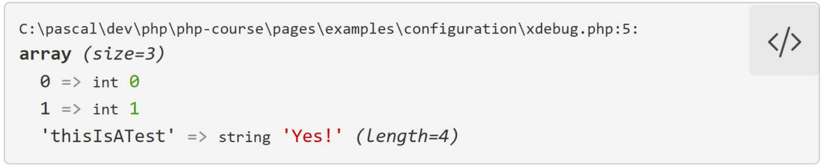
- Installation
  - Navigate to the folder that contains this php.ini file
  - Go to the `ext` folder
    - Put the XDebug .dll file in this folder
    - Rename the file to php_xdebug.dll
  - Open the php.ini file
    - Search for output_buffering and change the value to the following

      ```
      output_buffering=off
      ```

    - At the end of the file add the following configuration:

      ```
      [XDebug]
      zend_extension=xdebug
      xdebug.mode=develop
      xdebug.start_with_request=trigger
      ```

    - Restart your php server.
    - **Example: configuration xdebug** should now look like this:



```
C:\pascal\dev\php\php-course\pages\examples\configuration\xdebug.php:5:
array (size=3)
  0 => int 0
  1 => int 1
  'thisIsATest' => string 'Yes!' (length=4)
```

Revision Exercise 1

Time

- time() represents the amount of seconds that have passed since the Unix Epoch, January 1, 1970 in a so called **timestamp**
    - Unix epoch is the fixed reference point for computers. Everything before that will have a negative timestamp everything after a positive one
- microtime() represents the amount of milliseconds that have passed since the Unix epoch
- mktime($hours, $minutes, $seconds, $month, $day, $year) returns a timestamp based on a numerical date
    - **NOTE**: American notation, so month before day)
- date() converts a timestamp to a human readable format which can be manipulated using formatting parameters
- The timestamp is always calculated on the clock of the server the application is running on. You local machine might be in a different timezone that the server, which can sometimes cause issues when uploading your application. Take this into account!
- **Example time functions**
- **Exercise time functions**

Cookies

- $_COOKIE is a superglobal array that is stored **temporarily** on the client to retain specific data without having to put in a database.

```
    setcookie('testCookie', 'value', time() + 3600);
```

    => cookie expires in 1h (=3600 seconds

- It's an array, so any key value pairs can be stored there

- When no expiration date is given, the cookie will expire when the browser is closed.

- To remove a cookie, the expiration date needs to be in the past

```
    setcookie('testCookie', '', time() + 3600);
```

Cookies

- Calling the cookie values:

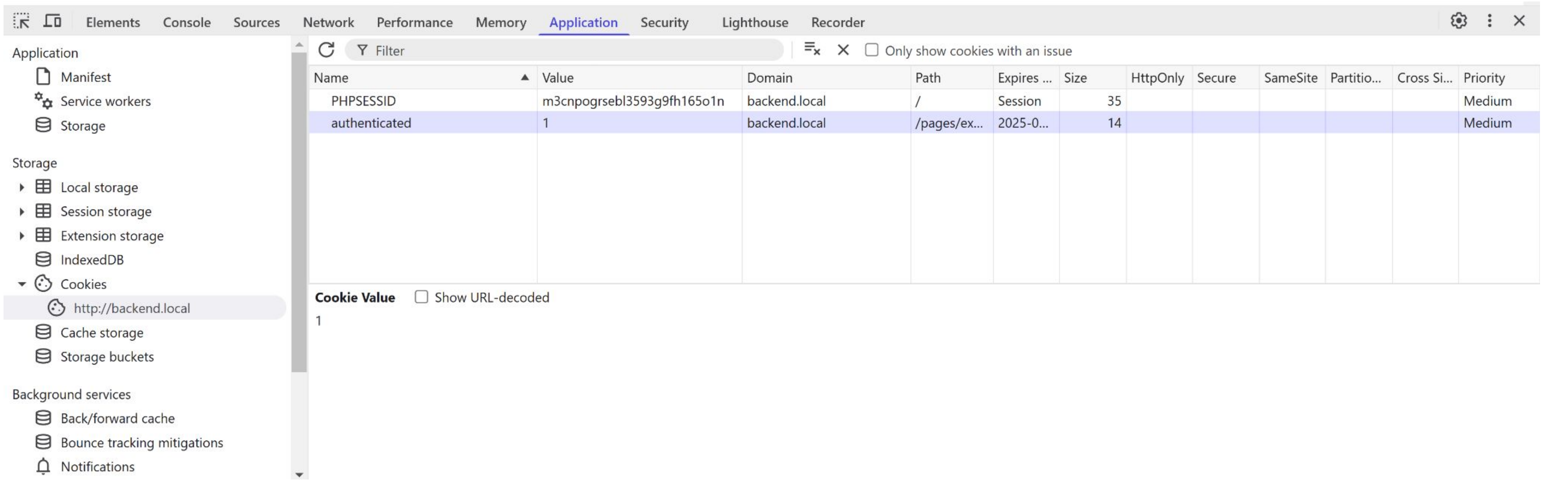
```
$_COOKIE['value'];
```

- A cookie needs to be set/unset at the top of the code block, before any other code.
  - When removing a cookie using PHP, you need to force a refresh for the changes to take place
    - `header('location: url');`

- **Example: cookie initialization**
- **Exercise: cookie basic**

Cookies

- Inspecting the values of cookies using developer tools:



Name	Value	Domain	Path	Expires ...	Size	HttpOnly	Secure	SameSite	Partitio...	Cross Si...	Priority
PHPSESSID	m3cnpogrsebl3593g9fh165o1n	backend.local	/	Session	35						Medium
authenticated	1	backend.local	/pages/ex...	2025-0...	14						Medium

Cookie Value

1

- **Example: cookie initialization**
- **Exercise: cookie basic**

Sessions

- $_SESSION is a superglobal array that is stored **temporarily** on the server to remember data without having to put it in a database.

- It uses cookies with a unique identifier as a value to access the session data saved on the server

- The session is bound to the domain name and cannot be accessed by any other instance
  - the actual session data is stored on the server

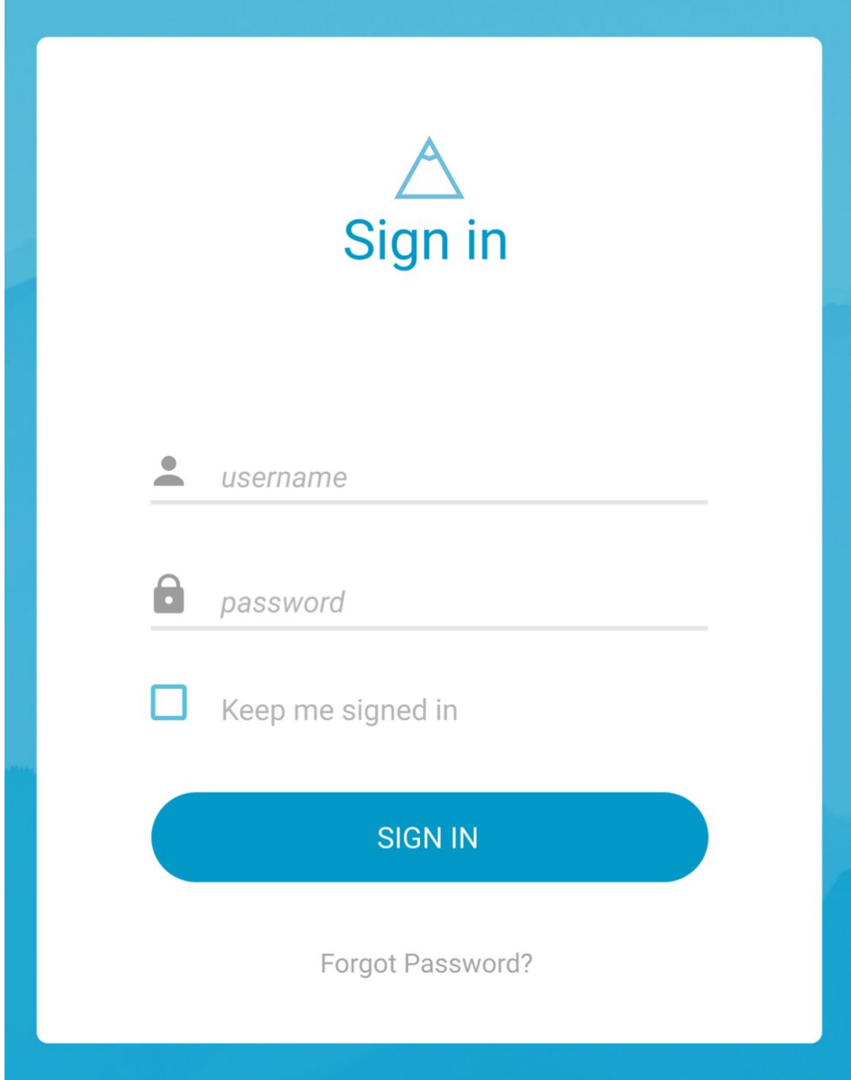
- A session expires after the user has closed it's browser window, unless this has been programmed to be retained longer

Sessions

- Unchecking "Keep me signed in" will create a sessions that will expire after the user closes all browser windows that are on that domain

- Checking the "Keep me signed in" checkbox will keep the sessions/cookies alive for as long as has been programmed, usually 31 days.

- Example: session initialization
- Example: session advanced
- Exercise: session basic



Database: SQLite

- Download SQLite studio
  https://github.com/pawelsalawa/sqlitestudio/releases
-