

Zaawansowany PHP



v 1.6

Plan

- Interfejsy, klasy abstrakcyjne i finalne
 - Funkcja autoload
 - Zaawansowane działania na stringach
 - Wyrażenia regularne
 - Pliki
 - Wyjątki
- Filtry
 - Mail
 - XML
 - JSON
 - Header



Header

Nagłówki HTTP

Odpowiedzi serwera składają się nie tylko z przesyłanych danych, które są wyświetlane następnie przez przeglądarkę.

W komunikacji przeglądarki z serwerem używane są również nagłówki pozwalające przesłać dodatkowe informacje związane z połączeniem lub dodatkowe dane.

Nagłówki przyjmują postać klucz-wartość, zapisywane w postaci:

Klucz: wartość

Np:

Content-Type: application/json

Connection: keep-alive

Nagłówki dzielą się na:

- Nagłówki żądania czyli przesyłane od klienta do serwera, te nagłówki są wysyłane jako pierwsze.
- Nagłówki odpowiedzi czyli przesyłane przez serwer do klienta.

Nagłówki HTTP

NAGŁÓWEK	OPIS
Content-Type	Określa jakiego typu dane są przesyłane
Content-Length	Zawiera informację ile danych jest przesyłanych
Cookie	Przechowuje informacje o ciasteczkach wraz z ich zawartością
Location	W odpowiedzi nakazuje przeglądarce przejście pod inny adres
Last-Modified	Informuje kiedy ostatnio nastąpiła zmiana źródła np. obrazka
Content-Disposition	Pozwala wymusić na przeglądarce pobranie danych zamiast ich wyświetlenia
Host	Nagłówek obowiązkowy, wskazuje na adres jaki chcemy przesłać żądanie
Accept	Pozwala przekazać informację jakiego typu dane są akceptowane
User-Agent	Nazwa przeglądarki wraz z dodatkowymi informacjami na jej temat
Referer	Adres URI jaki przekierował nas na daną stronę np. po wyszukaniu w Google
Allow	Określa metody akceptowane przez serwer GET, POST, HEAD
Connection	Pozwala ustalić czy połączenie po zakończeniu żądania ma zostać zakończone czy potrzymane tzw. keep-alive

Nagłówki HTTP

Listę najczęściej używanych nagłówków można znaleźć m.in, tutaj:

http://pl.wikipedia.org/wiki/Lista_nag%C5%82%C3%B3wk%C3%B3w_HTTP

Oprócz standardowych nagłówków można wprowadzać też swoje własne, aby przekazać dodatkowe informacje (np. pozwalające na łatwiejsze debugowanie).

```
header("Moja-informacja: 123456");
```

Funkcja header

- Funkcja pozwalająca na kontrolowanie nagłówków wysyłanych przez serwer w odpowiedzi na zapytanie do naszego skryptu.
- Funkcja musi być użyta przez wysłaniem jakiegokolwiek innej treści do przeglądarki. Również informacji o błędach.
- Użyta później generuje ostrzeżenie (headers already sent) i nie modyfikuje już nagłówków.

```
void header ( string $string [, bool $replace = true  
              [, int $http_response_code ] ] )
```

```
header('My-header: http://coderslab.pl');
```

```
$someText = 'zadzialam';
```

```
echo $someText;
```

Skrypt wykona się prawidłowo

```
$someText = 'niezadzialam';
```

```
echo $someText;
```

```
header('My-header: http://coderslab.pl');
```

Skrypt zwróci błąd ponieważ przed wysłaniem nagłówka,
do przeglądarki przesłano dane przez funkcję echo

Popularne użycia

➤ Przekierowanie

```
header("Location: http://www.example.com/");
```

➤ Zmiana typu zwracanej zawartości (image, json, xml, pdf)

```
header('Content-Type: application/pdf');
```

➤ Polecanie przeglądarce zapisać zwracaną zawartość jako plik

```
header('Content-Disposition: attachment;  
    filename="downloaded.pdf" ');  
header('Content-Transfer-Encoding: binary');  
header('Content-Length: 27641');
```

➤ Cache

```
header("Cache-Control: no-cache, must-revalidate");  
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");  
header('Pragma: no-cache');
```

➤ Kontrola dostępu

```
header(' WWW-  
Authenticate: Basic realm="My Realm" ');  
header('HTTP/1.0 401 Unauthorized');
```

➤ Data ostatniej modyfikacji

```
$time = time() - 60; // or filemtime($fn), etc  
header('Last-Modified: ' . gmdate('D, d M Y H:i:s',  
$time) . ' GMT');
```


Inne funkcje

bool headers_sent ([string &\$file [, **int** &\$line]])

- Sprawdza, czy nagłówki zostały już wysłane, tzn., czy możemy jeszcze dodać własne nagłówki czy nie.
- Jeżeli zostaną podane zmienne **\$file** i **\$line**, to zostanie w nich zapisane, w którym miejscu wysłano pierwszą część odpowiedzi.

int http_response_code ([**int** \$response_code])

- Ustawia kod odpowiedzi HTTP, domyślny to 200.
- Tę funkcję można wykorzystać do zwrócenia błędu HTTP, np. 404, 500 lub też niestandardowej odpowiedzi z rodziny 2XX.

Kod odpowiedzi serwera

Za pomocą nagłówków możemy także kontrolować odpowiedź serwera przesyłaną do przeglądarki.

Najpopularniejsze kody odpowiedzi to:

- **200** – wszystko ok
- **403** – brak autoryzacji, strona nie dostępna bez podania danych autoryzacyjnych
- **404** – nie znaleziono strony pod podanym adresem
- **500** – wewnętrzny błąd serwera, należy przejrzeć logi serwera
- **301** – strona przekierowana na stałe pod inny adres
- **101** – zmiana protokołu np. przy używaniu WebSocket

- Zmiana kodu odpowiedzi serwera
`header('HTTP/1.1 404 Not Found');`
`header('HTTP/1.1 403 Forbidden');`
`header('HTTP/1.1 301 Moved Permanently');`
`header('HTTP/1.1 500 Internal Server Error');`



Pliki

Pliki

Otwieranie pliku

fopen – otwiera dany plik z możliwością wyboru trybu. Zwraca wskaźnik do pliku, który będziemy wykorzystywać przy operacjach na tym pliku.

Lista możliwych trybów:

- **r** – tylko odczyt,
- **w** – tylko zapis, zaczyna zapisywać od początku pliku i czyści jego zawartość (nadpisuje). Jeżeli plik nie istnieje, to go tworzy.
- **a** – tylko zapis, zaczyna zapisywać na końcu pliku (dodaje),
- **r+** , **w+** – to samo co **r+w**,
- **a+** – to samo co **r+a**.

```
$handle = fopen("/home/resource.txt", "r")
```

Odczyt

- **fread** – odczytuje zadaną liczbę bajtów z pliku,
- **fgets** – odczytuje jedną linię z pliku,
- **fgetc** – odczytuje jeden znak z pliku,
- **fpassthru** – odczytuje plik do końca i zawartość wysyła do bufora wyjściowego.

```
$contents = fread($handle, filesize($filename));  
$buffer = fgets($handle);
```

Pliki

Zapis

fputs , **fwrite** – zapisuje dany string do pliku, zwraca liczbę bajtów.

```
$length = fwrite($handle, $text);
```

```
$file = fopen("example.txt", "w");  
echo(fwrite($file, "Hello World. Testing!"));  
fclose($file);
```

Wykrywanie końca pliku

feof – funkcja zwraca **true**, jeżeli osiągnięty został koniec pliku (End Of File).

```
$handle = fopen('somefile.txt', 'r');  
if(feof($handle)) {  
    echo("koniec pliku");  
}
```

Pliki

Zamykanie pliku

Każdy wskaźnik do pliku powinien być zamknięty funkcją **fclose**, jeżeli skończyliśmy wykonywać operację na pliku.

```
$handle = fopen('somefile.txt', 'r');
```

```
fclose($handle);
```

Na skróty

Funkcja **file** odczytuje cały plik o podanej nazwie i zwraca jego zawartość w postaci tablicy. Jeden wiersz tablicy odpowiada jednej linii tekstu.

```
$array = file('somefile.txt');
```

Na skróty

file_get_contents – odczytuje cały plik o podanej nazwie i zwraca jego zawartość. Pozwala w prosty sposób wczytać zawartość pliku.

```
$file = file_get_contents('people.txt');
```

file_put_contents – analogicznie funkcja pozwala w prosty sposób zapisać do pliku dany tekst. Domyślnie funkcja nadpisuje plik!

```
file_put_contents($file, $txt);
```

```
file_put_contents($file, $txt, FILE_APPEND |  
LOCK_EX);
```


Nie tylko pliki

Niektóre funkcje obsługujące pliki można zastosować również do adresów URL. Są to m.in.:

- **fopen**,
- **file_get_contents**,
- **file**,
- **file_exists**,
- **filesize**.

Dzięki temu w prosty sposób możemy również pobrać dane przez protokół HTTP lub FTP.

Należy pamiętać że do działania z adresami URL konieczna może być zmiana konfiguracji PHP na serwerze.

```
$homepage = file_get_contents('http://www.ex.mo/');
```

```
$handle = fopen("http://www.ex.mo/", "r");
```

```
$handle = fopen("ftp://user:password@ex.mm/  
somefile.txt", "w");
```

Operacje na plikach

- **rename** – zmienia nazwę pliku
 - **unlink** – usuwa plik
 - **filesize** – zwraca wielkość pliku w bajtach
-
- **filetype** – zwraca typ zasobu (file, dir, link, char itp.)
 - **fstat** – zwraca tabelę z informacjami o pliku (m.in. wielkość, czas utworzenia, modyfikacji, dostępu)

Operacje na katalogach

- **mkdir** – tworzy katalog
- **rmdir** – usuwa katalog

is_*

- **is_dir**
 - sprawdza, czy argument jest katalogiem
- **is_file**
 - sprawdza, czy argument jest plikiem
- **is_link**
 - sprawdza, czy argument jest linkiem
- **is_readable**
 - sprawdza, czy można odczytać plik
- **is_writable**
 - sprawdza, czy można zapisywać do pliku.

Upload pliku

- Protokół HTTP umożliwia przesyłanie plików tylko w zapytaniu typu POST.
- Jest to bardzo wygodna dla użytkowników forma przesyłania plików do naszej strony.
- Stosuje się ją z reguły dla małych i średnich plików ze względu na ograniczenia transferu.

Formularz

Aby na stronie umożliwić wybranie i przesłanie pliku stosujemy tag **<input>** typu **file**.

W formularzu wybieramy metodę **POST** i dodajemy rodzaj kodowania **enctype="multipart/form-data"**

```
<form action="upload.php" method="post"
      enctype="multipart/form-data">
  <input type="file" name="fileToUpload" id="fileToUpload">
  <input type="submit" value="Upload Img" name="submit">
</form>
```

Chrome 21	<input type="button" value="Choose File"/>	No file chosen
Safari 5	<input type="button" value="Choose File"/>	no file selected
FireFox 22	<input type="button" value="Browse..."/>	no file selected
Opera 15	<input type="button" value="Choose Files"/>	No file chosen
IE 8	<input type="text"/>	<input type="button" value="Browse..."/>

Zmienna `$_FILES`

Po stronie serwera informacje o przesłanych w żądaniu plikach znajdziemy w zmiennej superglobalnej `$_FILES`.

Klucz w tablicy jest identyczny jak atrybut `name` inputa typu `file` czyli w tym wypadku:

```
<input type="file" name="userfile"
id="fileToUpload">
```

- `$_FILES['userfile']['name']`
– oryginalna nazwa pliku.
- `$_FILES['userfile']['type']`
– typ mime pliku, np. `'image/gif'`.
- `$_FILES['userfile']['size']`
– wielkość pliku w bajtach.
- `$_FILES['userfile']['tmp_name']`
– tymczasowa nazwa pliku na serwerze.
- `$_FILES['userfile']['error']`
– kod błędu powiązany z plikiem.

Zapisanie pliku

- Plik po odebraniu przez serwer musi zostać zapisany w docelowe miejsce.
- Serwer WWW przechowuje plik w katalogu tymczasowym i jeżeli go nie przeniesiemy, zostanie skasowany po zakończeniu żądania.
- W tym celu korzystamy z `move_uploaded_file`.

```
$uploaddir = '/var/www/uploads/';
$uploadfile = $uploaddir . basename($_FILES
['userfile']['name']);

if(move_uploaded_file($_FILES['userfile']
['tmp_name'], $uploadfile)) {
    echo("File is valid, and was successfully
uploaded.\n");
} else {
    echo("Possible file upload attack!\n");
}
```


Upload wielu plików

W jednym żądaniu możemy przesłać wiele plików:

```
<input type="file" name="pictures[]" />
```

```
<input type="file" name="pictures[]" />
```

```
<input type="file" name="pictures[]" />
```

Zmienne konfiguracyjne

Serwer WWW ogranicza wielkość akceptowanych plików.

Możemy kontrolować ten limit za pomocą zmiennych konfiguracyjnych.

- **file_uploads**
– czy można uploadować pliki,
- **upload_max_filesize**
– maksymalna akceptowana wielkość pliku,
- **max_file_uploads**
– maksymalna liczba akceptowanych plików,
- **post_max_size**
– maksymalna wielkość danych dla żądania typu POST.

Popularne problemy

- Zbyt mała wartość **upload_max_filesize** lub **post_max_size**
- Zbyt mały limit pamięci **memory_limit**
- Zbyt krótki czas **max_execution_time**

Czas na zadania



Wykonajcie zadania z działu:

Pliki w PHP



Wyjątki

Definicja

Wyjątek (exception)

- Jest to mechanizm przepływu sterowania używany w mikroprocesorach oraz współczesnych językach programowania do obsługi zdarzeń wyjątkowych, w szczególności do obsługi błędów, których wystąpienie zmienia prawidłowy przebieg wykonywania programu.
- W momencie zajścia niespodziewanego zdarzenia generowany jest wyjątek, który **musi zostać obsłużony** poprzez zapamiętanie bieżącego stanu programu i przejście do procedury jego obsługi.

Innymi słowy

- **Wyjątek** jest to mechanizm pozwalający obsłużyć sytuację wyjątkową, w której dalsze wykonywanie programu nie jest możliwe lub wskazane.
- Po zgłoszeniu wyjątku praca programu jest przerywana i następuje próba obsłużenia wyjątku.
- Jeżeli wyjątek nie zostanie obsłużony, to następuje koniec pracy programu i zgłaszany jest błąd nieobsłużonego wyjątku.

Wyjątki w PHP

```
function inverse($x) {  
    if (!$x) {  
        throw new Exception('Division by zero.');    }  
    else return 1/$x;  
}
```

```
try {
```

```
    $value = inverse(0);
```

```
} catch(Exception $e) {
```

```
    echo('Caught exception: ');
```

```
} finally {
```

```
    echo('First finally.');
```

```
}
```

Ten blok kodu zostanie "sprawdzony" czy nie rzuca wyjątku

Ten blok kodu zostanie wywołany jeśli wystąpi wyjątek

Ten blok kodu zostanie wywołany zawsze

Rzucanie wyjątku

- W PHP wyjątki są rzucane (**throw**).
- W momencie rzucenia wyjątku praca programu jest przerywana.
- Parametrem **throw** jest obiekt, który musi być klasy Exception (wbudowana klasa wyjątku) lub klasy po niej dziedziczącej.
- W klasie dziedziczącej po klasie Exception zawsze należy wywołać konstruktor klasy nadrzędnej.

```
class MyException extends Exception { }

class AdvException extends Exception {
    public function __construct($message = null,
                                $code = 0)
    {
        parent::__construct($message, $code);
    }
}
```

! Wywołanie konstruktora klasy nadrzędnej.

Klasa Exception

```
Exception {  
/* Właściwości */  
protected string $message;  
protected int $code;  
protected string $file;  
protected int $line;  
/* Metody */
```

```
public __construct ([ string $message =  
''' [, int $code = 0 [, Exception $previous =  
NULL ]]] )  
final public string getMessage(void)  
final public Exception getPrevious(void)  
final public mixed getCode(void)  
final public string getFile(void)  
final public int getLine(void)  
final public array getTrace(void)  
final public string getTraceAsString(void)  
public string __toString(void)  
final private void __clone(void)  
}
```

Rzucanie wyjątku

```
class MyException extends Exception { }
```

```
class AdvException extends Exception {  
    public function __construct($message = null, $code = 0)  
    {  
        parent::__construct($message, $code);  
    }  
}
```



Wywołanie konstruktora klasy nadrzędnej.

Obsługa wyjątku

- Aby przechwycić rzucony wyjątek używamy bloku **try....catch**.
- W bloku **try** znajduje się kod, dla którego spodziewamy się wyjątków.
- W bloku **catch** znajduje się kod obsługi wyjątku.
- W jednym bloku **try...catch** może występować wiele bloków **catch**, które definiują różne sposoby obsługi różnych wyjątków.
- Do obsługi wyjątku wybierany jest pierwszy napotkany blok **catch**, który oczekuje wyjątku danego typu (danej klasy).

```
try {  
    throw new TestException();  
}  
catch (TestException $e) {  
    echo('Caught TestException');  
}  
catch (Exception $e) {  
    echo('Caught Exception');  
}
```

Łapie tylko wyjątki
z klasy TestException
i jej pochodnych

Łapie wszystkie wyjątki

Zagnieżdżanie

- Bloki **try...catch** mogą być zagnieżdżone jeden w drugim.
- Dany blok **try...catch** nie musi łapać wszystkich wyjątków.
- Może to zrobić któryś z bloków nadrzędnych.
- W bloku **catch** złapany wyjątek może być rzucony ponownie (**rethrow**) lub rzucony zupełnie nowy wyjątek.
- W takim wypadku obsługą wyjątku zajmą się nadrzędne bloki **catch**.

```
try {  
    try {  
        throw new MyException('foo!');  
    } catch (MyException $e) {  
        /* rethrow it */  
        throw $e;  
    }  
} catch (Exception $e) {  
    var_dump($e->getMessage());  
}
```

finally

- Może się tak zdarzyć, że w bloku **try** przed rzuceniem wyjątku zostaną zaalokowane pewne zasoby wymagające zwolnienia nawet w przypadku wystąpienia wyjątku.
- Blok **finally** (od wersji PHP 5.5) jest wykonywany zawsze. Niezależnie od tego, czy został zgłoszony jakiś wyjątek czy nie.
- Umożliwia on zwolnienie takich zasobów (np. zamknięcie połączenia do bazy danych).

```
try {  
    inverse(0);  
} catch(Exception $e) {  
    echo('Caught exception');  
} finally {  
    echo('Finally.');
```


Czas na zadania



Wykonajcie zadania z działu:

Wyjątki



Filtry

Filtry

- Filtry umożliwiają walidację danych przetwarzanych przez skrypt.
- Dane otrzymywane w zapytaniu nie muszą odpowiadać naszym założeniom. Na przykład adres email może zawierać nieprawidłowe znaki lub mieć zły format.

filter_has_var – sprawdza, czy istnieje zmienna danego typu

type – INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, INPUT_ENV

filter_has_var(INPUT_GET, 'test')

Filtry

filter_var – filtruje zmienną i zwraca wartość po filtrowaniu (przetworzeniu) lub **false**, jeżeli nie udało się przefiltrować zmiennej.

```
filter_var('http://coderslab.pl',  
FILTER_VALIDATE_URL);
```

```
filter_var('email@something.com',  
FILTER_VALIDATE_EMAIL);
```

```
filter_var('79.123.123.3', FILTER_VALIDATE_IP)
```

Filtry

filter_input – filtruje zmienną zewnętrzną i zwraca wartość po filtrowaniu (przetworzeniu).
Zwraca **false** – jeżeli nie udało się przefiltrować zmiennej, **null** – jeżeli zmienna nie istnieje.

```
filter_input(INPUT_POST, 'email',  
FILTER_VALIDATE_EMAIL);
```

```
filter_input(INPUT_GET, 'search',  
FILTER_SANITIZE_ENCODED);
```

Wybrane rodzaje filtrów

Walidacja

- FILTER_VALIDATE_BOOLEAN
- FILTER_VALIDATE_EMAIL
- FILTER_VALIDATE_INT
- FILTER_VALIDATE_IP
- FILTER_VALIDATE_URL

Czyszczenie

- FILTER_SANITIZE_EMAIL
- FILTER_SANITIZE_ENCODED
- FILTER_SANITIZE_STRING
- FILTER_SANITIZE_URL

Czas na zadania



Wykonajcie zadania z działu:

Filtry





Przestrzenie nazw w PHP

PHP Namespaces

Czym są przestrzenie nazw?

- Przestrzenie nazw pozwalają na tworzenie bardziej przejrzystego i głównie zhermetyzowanego kodu.
- Pomagają przy pracy z dużymi projektami gdzie mogą wystąpić dwie lub więcej klas o tej samej nazwie.

Definiowanie przestrzeni nazw

- Aby zdefiniować przestrzeń nazw należy umieścić jako pierwsza instrukcja pliku nazwę przestrzeni poprzedzoną słowem kluczowym **namespace**.
- Przestrzenie nazw powinny mieć swoje nazwy zgodne ze strukturą katalogów w jakich znajdują się pliki z klasami, nie jest to wymagane ale przynosi dużą korzyść o czym dowiedzie się na kolejnych slajdach.

```
namespace Foo;  
namespace Foo\Bar;  
namespace Foo\Bar\Baz;
```

PHP Namespaces

plik class.userAccount.php

```
namespace User\Account;  
class Register {  
    public function getFormData ()  
    {  
        return true;  
    }  
}
```

Fatal error: Class
'Register' not found!

```
$uReg = new Register();  
$uRegAccount = new User\Account\Register();
```

Teraz nasza klasa to
User\Account\Register a nie Register

plik class.userForum.php

```
namespace User\Forum;  
class Register {  
    public function getFormData ()  
    {  
        return true;  
    }  
}
```

```
$uRegForum = new User\Forum\Register();
```

PHP Namespaces

Spróbujmy utworzyć obiekty naszych klas

```
include('class.userAccount.php');
```

```
include('class.userForum.php');
```

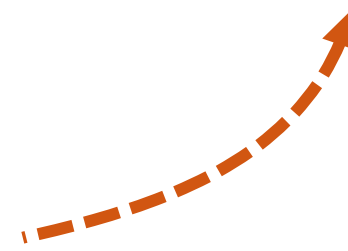
```
$uRegAccount = new User\Account\Register();
```

```
$uRegForum = new User\Forum\Register();
```

Utworzyliśmy obiekty klas o takiej samej nazwie dzięki przestrzeniom nazw.

Gdybyśmy ich nie użyli otrzymalibyśmy błąd

Fatal error: Cannot redeclare class Register



PHP Namespaces

Może się zdarzyć iż nasza przestrzeń nazw będzie bardzo długa co może spowodować nie czytelność kodu.

plik class.userAccountDhl.php

```
namespace User\Account\Shop\Shipping\Dhl;
```

```
class SendParcel {
```

```
    public function prepareParcel()
```

```
    {
```

```
        return true;
```

```
    }
```

```
}
```

Nasz kod staje się nie czytelny a co w wypadku jeśli będziemy mieli kilkanaście klas

```
$dhl = new
```

```
User\Account\Shop\Shipping\Dhl\SendParcel();
```

Dyrektywa **use** pozwala zdefiniować jakieś przestrzeni nazw aktualnie używamy

```
use User\Account\Shop\Shipping\Dhl;
```

```
include('class.userAccountDhl.php');
```

```
$dhl = new SendParcel();
```

Używając dyrektywy **use** PHP wie, że chodzi nam o klasę SendParcel która posiada przestrzeń nazw User\Account\Shop\Shipping\Dhl

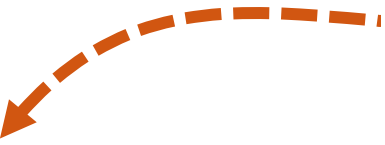
PHP Namespaces

W sytuacji jeśli mamy 2 klasy o tej samej nazwie i chcemy użyć dyrektywy **use** jest to możliwe ale musimy skorzystać z aliasów nazw.

```
use User\Account as UA;
```

```
use User\Forum as UF;
```

Nasza przestrzeń
ma alias UA




```
include('class.userAccount.php');
```

```
include('class.userForum.php');
```

```
$uRegAccount = new UA\Register();
```

```
$uRegForum = new UF\Register();
```

Korzystamy zarówno z przestrzeni nazw i aliasów aby wykluczyć błąd tej samej nazwy klasy.



Dzięki połączeniu przestrzeni nazw, dyrektywy use oraz aliasów możemy tworzyć czytelny kod bez obaw o jego kolizję z innymi klasami o tych samych nazwach.

PHP Namespaces

Dlaczego tak ważne jest aby nazwy przestrzeni nazw były zgodne ze strukturą katalogów.
Założmy następującą strukturę katalogów:

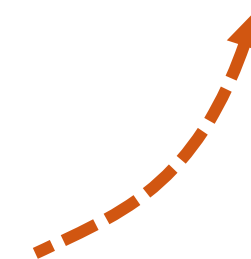
```
classes
|__ User
|___ Account
|_____ Register.php
|___ Forum
|_____ Register.php
```

Nasza przestrzeń to przykładowo **User\Account** i plik z klasą z tej przestrzeni znajduje się w katalogu **classes/User/Account/Register.php**

Możemy skorzystać teraz z funkcji **autoload**

```
function __autoload($class)
{
    $cl = str_replace('\\', '/', $class);
    require_once(__DIR__ . '/classes/' . $cl . '.php');
}
```

```
$uRegAccount = new User\Account\Register();
```



Dzięki zachowaniu nazw przestrzeni zgodnych ze strukturą katalogów funkcja __autoload wie, że ma załadować plik Register.php z katalogów User/Account



Instalowanie bibliotek za pomocą Composer

Composer – co to jest?

- System wspomagający zarządzanie zależnościami aplikacji w PHP.
 - Załóżmy, że wykonanie skryptu **index.php** jest zależne od istnienia pliku **funkcje.php**.
 - Plik **funkcje.php** jest swego rodzaju biblioteką programistyczną.
 - Biblioteka to plik (lub zestaw plików) dostarczający funkcje, klasy, metody lub dane, które mogą zostać wykorzystane w kodzie źródłowym programu.
- Załóżmy, że chcemy w naszym skrypcie użyć jakiejś biblioteki udostępnionej w internecie.
 - Nie chcemy fizycznie pobierać tej biblioteki, mamy tylko zdefiniować, jakie biblioteki są wymagane przez nasz skrypt.
 - Pobranie tych bibliotek zajmie się Composer.

Composer – instalacja i konfiguracja

➤ System Linux

- w katalogu swojego projektu wykonaj polecenie:
curl -s https://getcomposer.org/installer | php

➤ System Windows

- pobierz i uruchom instalator
<https://getcomposer.org/Composer-Setup.exe>

- W katalogu swojego projektu utwórz plik `composer.json` – będzie on zawierał wpisy, które poinformują Composera, jakich bibliotek wymaga nasz skrypt.

Można również wywołać z konsoli komendę
composer init

- Załóżmy, że chcemy pobrać bibliotekę służącą do zapisywania (logowania) informacji o działaniu skryptu do jakiegoś loga (pliku `.log`).

- Z listy dostępnej na tej stronie:
<https://packagist.org/explore>
– wybieramy bibliotekę Monolog:
<https://packagist.org/packages/monolog/monolog>



Strona packagist.org jest wyszukiwarką pakietów dostępnych do instalacji

Composer – konfiguracja

- **require** – jest słowem kluczowym oznaczającym, jakie biblioteki są wymagane w naszym projekcie.
- **monolog/monolog** – (autor/nazwa) to skrócona nazwa biblioteki.
- **1.0.*** – wersja danej biblioteki. Gwiazdka oznacza, że Composer ma zainstalować zawsze najnowszą wersję biblioteki.

composer.json

```
{  
  "require": {  
    "monolog/monolog": "1.0.*"  
  }  
}
```

Composer – instalacja bibliotek

- Po zdefiniowaniu zależności naszego skryptu możemy pozwolić Composerowi działać – pobierze on biblioteki, od których jest zależny nasz skrypt.
- **System Linux**
W katalogu swojego projektu wykonaj polecenie:
php composer.phar install
- **System Windows**
W katalogu swojego projektu wykonaj polecenie:
composer install
- Pobrane biblioteki Composer umieści w katalogu **vendor**, który zostanie utworzony w katalogu projektu.
- Zostanie także utworzony plik **vendor/autoload.php**, który musimy dołączyć do naszego skryptu, umieszczając na jego początku polecenie:
require 'vendor/autoload.php';
- Pamiętaj aby katalogu vendor nie dodawać do repozytorium ponieważ wówczas traci sens idea używania Composera.

Composer – użycie

```
<?php
require 'vendor/autoload.php';
use Monolog\Logger;
use Monolog\Handler\StreamHandler;
echo('Prosty przykład użycia loggera Monolog<br>');

// utworzenie kanału logowania
$log = new Logger('infoLogger');
$log->pushHandler(new StreamHandler(__DIR__ . '/log/info.log', Logger::DEBUG));
$log->addInfo('skrypt rozpoczął działanie');

for($i = 0; $i < 10; $i++) {
    $log->addInfo($i . '. obrót pętli');
}
$log->addInfo('skrypt zakończył działanie');
?>
```



Mail

Wysłanie maila

```
telnet: > telnet mx1.example.com smtp
telnet: Connected to mx1.example.com.
server: 220 mx1.example.com ESMTP server ready Tue, 20 Jan 2004 22:33:36 +0200
client: HELO client.example.com
server: 250 mx1.example.com
client: MAIL from: <sender@example.com>
server: 250 Sender <sender@example.com> Ok
client: RCPT to: <recipient@example.com>
server: 250 Recipient <recipient@example.com> Ok
client: DATA
server: 354 Ok Send data ending with <CRLF>.<CRLF>
client: From: sender@example.com
client: To: recipient@example.com
client: Subject: Test message
client:
client: This is a test message.
client: .
server: 250 Message received: 20040120203404.CCCC18555.mx1.example.com@client.example.com
client: QUIT
server: 221 mx1.example.com ESMTP server closing connection
```


Funkcja mail

bool **mail** (string **\$to** , string **\$subject** , string **\$message** [, string **\$additional_headers**])

Wysyła wiadomość poczty elektronicznej adresowaną do **\$to** o temacie **\$subject** i treści: **\$message**
\$additional_headers – dodatkowe nagłówki, takie jak **From**, **Bcc**, **Cc**, **Reply-To**, **X-Mailer**.

Przykład

```
$to = 'nobody@example.com';  
$subject = 'the subject';  
$message = 'hello';  
$headers = 'From: webmaster@example.com'  
          . "\r\n" . 'Reply-To:  
webmaster@example.com' . "\r\n" . 'X- Mailer:  
PHP/' . phpversion();  
  
mail($to, $subject, $message, $headers);
```

Przykład

```
$message = '<html><body>Hello world</body></html>';
$to = 'mary@example.com' . ',' . 'kelly@example.com';
// To send HTML mail, the Content-type header must be set
$headers = "MIME-Version: 1.0" . "\r\n";
$headers .= 'Content-type: text/html; charset=iso-8859-1' . "\r\n";
// Additional headers
$headers .= 'To: Mary <mary@example.com>, Kelly <kelly@example.com>' . "\r\n";
$headers .= 'From: Birthday <birthday@example.com>' . "\r\n";
$headers .= 'Cc: john@example.com' . "\r\n";
$headers .= 'Bcc: bill@example.com' . "\r\n";

// Mail it
mail($to, $subject, $message, $headers);
```

Nagłówek

- **From** – pole określające nadawcę np.: From: Jan Kowalski jan.kowalski@gmail.com
- **Cc** – adresat kopi wiadomości, np.: Cc: Jan Kowalski jan.kowalski@gmail.com
- **Bcc** – adresat ukrytej kopi wiadomości
- **Reply-To** – adres do odpowiedzi
- **Content-type** – typ zawartości, np.: Content-type: text/html; charset=UTF-8
- **X-Mailer** – identyfikuje program wysyłający wiadomość

PHPMailer

PHPMailer jest biblioteką napisaną w PHP, która umożliwia wysyłanie maila w prosty sposób i oferuje szereg zaawansowanych opcji.

- Wysyłanie maili tekstowych i HTML
- Wysyłka za pośrednictwem SMTP
- Wspiera kodowanie UTF-8
- Wspiera "podpisywanie" DKIM i S/MIME

Instalacja PHPMailera za pomocą Composer'a:

```
composer require phpmailer/phpmailer
```

<https://github.com/PHPMailer/PHPMailer>

```
$mail = new PHPMailer;  
$mail->setFrom('from@example.com', 'Mailer');  
$mail->addAddress('joe@example.net', 'Joe User');  
$mail->addReplyTo('info@example.com', 'Information');  
  
$mail->addAttachment('/var/tmp/file.tar.gz');  
$mail->isHTML(true);  
  
$mail->Subject = 'Here is the subject';  
$mail->Body = 'This is the HTML <b>message body</b>';  
$mail->AltBody = 'This is the body in plain text for non-  
HTML mail clients';  
if(!$mail->send()){  
    echo 'Message could not be sent.';  
    echo 'Mailer Error: ' . $mail->ErrorInfo;  
} else {  
    echo 'Message has been sent';  
}
```


Czas na zadania



Wykonajcie zadania z działu:

Maile





XML

XML

Dekodowanie

Oto dwie podstawowe metody czytania danych XML.

W postaci drzewa (SimpleXML)

- Proste, łatwe do zaimplementowania.
- Naturalne.
- Wymaga dużo pamięci i zasobów.

Poprzez eventy

- Szybkie i wydajne.
- Skomplikowane i trudne w utrzymaniu.

```
<?xml version="1.0" encoding="UTF-8"?>
<ksiazka-telefoniczna kategoria="bohaterowie
książek">
  <!-- komentarz -->
  <osoba charakter="dobry">
    <imie>Ambroży</imie>
    <nazwisko>Kleks</nazwisko>
    <telefon>123-456-789</telefon>
  </osoba>
  <osoba charakter="zły">
    <imie>Alojzy</imie>
    <nazwisko>Bąbel</nazwisko>
    <telefon/>
  </osoba>
</ksiazka-telefoniczna>
```


SimpleXML

Wczytywanie:

- **simplexml_load_file** – wczytuje strukturę z pliku,
- **simplexml_load_string** – wczytuje strukturę ze stringu np. zmiennej zawierającej XML.

Obiekty **SimpleXML** tworzą drzewo, którego struktura odpowiada strukturze kodu XML.

Każdemu elementowi XML odpowiada jeden obiekt **SimpleXML**, zaś atrybuty są zwracane w postaci tablicy asocjacyjnej.

SimpleXMLElement wymaga, aby drzewo dokumentu XML zmieściło się w pamięci.

Jeśli kod XML zawiera więcej takich samych elementów wtedy elementy te zostaną umieszczone w tablicy (w naszym przykładzie **osoby**), co zobaczysz na następnym slajdzie.

Założmy, że nasz plik z poprzedniego slajdu nazywa się `ksiazka.xml`

Przykład

```
$ksiazka = simplexml_load_file('ksiazka.xml');
```

```
$ksiazka->getName();
```



Książka telefoniczna

```
$ksiazka->osoba[0]['charakter'];
```



dobry

```
$ksiazka->osoba[1]->nazwisko;
```



Bąbel

```
SimpleXMLElement Object (
    [@attributes] => Array (
        [kategoria] => bohaterowie książek
    )
    [comment] => SimpleXMLElement Object ( )
    [osoba] => Array (
        [0] => SimpleXMLElement Object (
            [@attributes] => Array (
                [charakter] => dobry
            )
            [imie] => Ambroży
            [nazwisko] => Kleks
            [telefon] => 123-456-789 )
        [1] => SimpleXMLElement Object (
            [@attributes] => Array (
                [charakter] => zły
            )
            [imie] => Alojzy
            [nazwisko] => Bąbel
            [telefon] => SimpleXMLElement Object ( )
        )
    )
)
```

SimpleXML z xpath

Funkcja **xpath** pozwala na przeszukiwania drzewa XML przy wykorzystaniu zapytań **xpath**.

WYRAŻENIE	ZNACZENIE
osoba	Znajduje wszystkie węzły osoba .
osoba/imie	Znajduje wszystkie węzły imie , które są dziećmi węzła typu osoba .
osoba/imie[1]	Wybiera pierwszy węzeł imie będący dzieckiem węzła osoba .
osoba/*	Wybiera wszystkie dzieci węzła osoba .
//osoba[@charakter]	Wybiera wszystkie węzły osoba z atrybutem charakter .
osoba/imie[text()]	Wybiera tekst węzła imie .

```
$ksiazka = simplexml_load_file('ksiazka.xml');  
$osoby = $ksiazka->xpath('osoba');  
foreach($osoby as $osoba){  
    echo $osoba->imie . ' ' . $osoba->nazwisko;  
};
```

XMLReader

- Czyta dokument XML węzeł po węźle.
- Nie ma możliwości wglądu w cały dokument, a tylko w bieżący węzeł.
- Ponieważ nie wczytuje całego pliku do pamięci, pozwala na odczytanie nawet bardzo dużych plików.

```
$xml = file_get_contents('ksiazka.xml');  
$ksiazka = new XMLReader();  
$ksiazka->xml($xml);  
  
while( $ksiazka->read() ) {  
    echo($ksiazka->name);  
  
    if($ksiazka->hasValue ) {  
        echo($ksiazka->value);  
    }  
  
    if($ksiazka->name == 'osoba') {  
        echo($ksiazka->getAttribute ('charakter'));  
    }  
}
```

XMLReader

WŁASNOŚĆ	ZNACZENIE
attributeCount	Liczba atrybutów w węźle
baseURI	Adres URI węzła
depth	Głębokość węzła w dokumencie XML
hasAttributes	Czy element ma atrybuty
hasValue	Czy element ma wartość tekstową
isDefault	Czy atrybut jest domyślny
isEmptyElement	Czy tag HTML jest pusty
localName	Nazwa lokalna węzła
name	Pełna kwalifikowana nazwa węzła
namespaceURI	Adres URI przestrzeni nazw węzła ze zbioru
nodeType	Kod reprezentujący typ węzła.
prefix	Prefix przestrzeni nazw węzła.
value	Wartość tekstowa węzła.
xmlLang	Zakres xml:lang w którym znajduje się wezeł

XMLReader

METODA	ZNACZENIE
XMLReader::open()	Pobierz URI zawierający dokument XML, który ma być analizowany.
XMLReader::close()	Zamknij dokument
XMLReader::read()	Przejdź do następnego węzła.
XMLReader::next()	Przejdź do następnego węzła, pomiń poddrzewa.
XMLReader::getAttribute())	Pobierz wartość atrybutu.
XMLReader::expand()	Zwróć kopię bieżącego węzła jako dokument DOM.
XMLReader::readString()	Pobiera wartość aktualnego węzła jako string

Czas na zadania



Wykonajcie zadania z działu:

XML

