

Podstawy PHP

Dzień I

v.1.3

Plan zajęć

- Wprowadzenie PHP
- Zmienne i stałe w PHP
- Typy danych
- Operatory w PHP
- Kontrola przepływu programu

Wprowadzenie do PHP

Wprowadzenie do PHP

- PHP to język programowania przeznaczony do tworzenia dynamicznych stron WWW (aplikacji WWW).
- PHP – PHP Hypertext Preprocesor (akronim rekursywny) lub Personal HomePage Tools (pierwsza wersja PHP stworzona przez Rasmusa Lerdorfa).
- Rdzeniem PHP jest silnik Zend wykonujący i nadzorujący wszystkie najistotniejsze operacje związane z przetwarzaniem kodu zawartego w skryptach.

Wprowadzenie do PHP

- PHP może współpracować z serwerem na dwa sposoby:
 - jako moduł serwera (szybkość wykonywania i większe bezpieczeństwo),
 - skrypt CGI (niezależność w stosunku do używanego serwera).
 - Obecnie zalecana jest konfiguracja PHP jako modułu serwera.
- Początkowo PHP był używany do obsługi formularzy HTML, ale z czasem jego funkcjonalność znacznie wzrosła, stał się kompletnym narzędziem do tworzenia aplikacji WWW.
 - Najważniejsze cechy:
 - prostota tworzenia i wdrażania aplikacji WWW,
 - obsługa wielu różnych systemów zarządzania bazą danych,
 - prosta składnia – nie trzeba dbać o jawne deklarowanie typu zmiennych i przydzielanie im pamięci,
 - pełna obsługa obiektowości.

Wersje PHP

Na kursie uczymy się PHP w wersji 5.6. Jest to najpopularniejsza obecnie wersja PHP.

Najnowszą wersją PHP jest wersja 7.

Główną zmianą jest przyspieszenia działania silnika Zend.

Jeżeli ktoś jest zainteresowany migracją z wersji 5.6 do wersji 7:

<http://php.net/manual/en/migration70.php>

Jeżeli ktoś jest zainteresowany funkcjonalnościami wprowadzonymi w wersji 7:

<http://php.net/manual/en/migration70.new-features.php>

Zmienne i stałe w PHP

Czym są zmienne?

- Zmienna to obszar w pamięci komputera, w którym programista może umieścić jakieś wartości, a następnie je odczytywać, inaczej – symbol, któremu możemy przypisać pewną wartość.
- W PHP (inaczej niż w innych językach programowania) zmienne nie muszą być deklarowane przed pierwszym użyciem. Zmienna zostaje utworzona i zdefiniowana w momencie jej pierwszego użycia.
- W PHP nie ma konieczności ustalania typu zmiennej. Jest on określany przez PHP w zależności od kontekstu, w jakim zmienna została użyta.
- Co więcej w PHP możemy łatwo zmieniać typ danych jaki trzyma w sobie zmienna.

Czym są zmienne?

Obok widzimy kilka przykładów przypisania wartości do zmiennych. Nie trzeba podawać typu zmiennej (czy jest to liczba, napis, wartość logiczna itp.) – PHP sam wykryje typ na podstawie przypisywanej wartości. Jest to tzw. typowanie słabe (dynamiczne).

Zmienne w zależności od ich zasięgu (obszaru obowiązywania) dzielimy na:

- lokalne,
- globalne,
- statyczne.

```
<?php
```

```
$noOfRepetitions= 1;
```

```
$someText= "Zmienna napisowa";
```

```
$valueOfPi= 3.14;
```

```
$agreement= true;
```

```
?>
```

Zmienne lokalne

Zmienne lokalne

- Zmienne lokalne mają ograniczony zasięg obowiązywania i można się do nich odwoływać tylko z tego obszaru.
- Wszystkie zmienne deklarowane wewnątrz funkcji są lokalne i dostępne tylko w ciele tej funkcji.
- Ich zawartość jest niszczone po wykonaniu funkcji.

<?php

```
function showInnerCost() {  
    $innerCost = 60;
```

```
    echo("Moje koszty własne = $innerCost");
```

```
}
```

```
showInnerCost();
```

```
echo("Ogólne koszty własne = $innerCost");
```

?>

Zmienna lokalna o zasięgu funkcji,
niewidoczna poza funkcją.

Spowoduje wyświetlenie Warninga (ostrzeżenia)
jako że zmienna nie ma wartości.

Zmienne globalne

- W odróżnieniu od zmiennych lokalnych zmienne globalne są dostępne w całym skrypcie.
- Aby odwołać się do zmiennej globalnej w funkcji należy wewnątrz niej wskazać, że zmienna ta jest zmienną globalną i ma być dostępna w ciele tej funkcji.

```
<?php
    $globalCost= 100;

    function addGlobalCostToInnerCost() {
        global $globalCost;
        $innerCost= 60;

        return $globalCost + $innerCost;
    }

    echo("Suma kosztów=" .addGlobalCostToInnerCost() );
?>
```

Zmienne globalne

<?php

\$globalCost= 100;

Zmienna globalna o zasięgu całego skryptu,
ale niewidoczna wewnątrz funkcji.

function addGlobalCostToInnerCost() {

global \$globalCost;

\$innerCost= 60;

return \$globalCost + \$innerCost;

}

Deklaracja powodująca że zmienna globalna jest
widoczna w środku naszej funkcji.

echo("Suma kosztów=" . addGlobalCostToInnerCost());

?>

Zmienne statyczne

- Zmienne statyczne to zmienne, których wartość nie jest niszczone wraz z końcem funkcji (w przeciwieństwie do zmiennych lokalnych).
- Przy kolejnym wywołaniu funkcji zmienna statyczna będzie miała taką wartość, jaką miała po poprzednim wykonaniu tej funkcji.
- Zmienną statyczną deklarujemy, dodając przed jej nazwą słowo kluczowe **static**.
- Można je opisać jako zmienne z pamięcią.

```
<?php
```

```
function count() {  
    static $counter = 0;  
    echo($counter);  
    $counter = $counter + 1;  
    echo('<br>');  
}
```

```
count();
```

```
count();
```

```
count();
```

```
?>
```

Zmienna statyczna

\$counter jest tworzone – wartość jest równa 0

\$counter już istnieje – wartość jest równa 1

\$counter już istnieje – wartość jest równa 2

Stałe

- Stałe to wartości, które nie będą się nigdy zmieniać. Jeśli potrzebujemy zadeklarować jakąś wartość, która w trakcie wykonywania skryptu nie będzie zmieniana, należy użyć stałej.
- Do deklaracji stałej używany konstrukcji **define('NAZWA_STALEJ', wartosc_stalej);**
- Nazwy stałej nie poprzedzamy znakiem \$. Przyjęło się też, że nazwy stałych piszemy wielkimi literami. Stałe mogą przechowywać ciągi znaków, wartości liczbowe i logiczne. Mają zawsze zasięg globalny.

```
<?php
define('LICZBA_PI', 3.14);

$promien = 15.5;
$pole = LICZBA_PI * $promien * $promien;
echo('Pole koła wynosi ' . $pole);
?>
```

Hint:

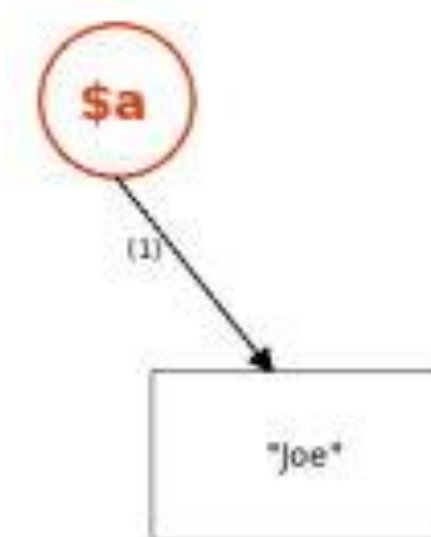
Zwyczajowo stałe definiujemy w oddzielnym pliku który potem jest załączany do naszego kodu

Zmienne – referencje

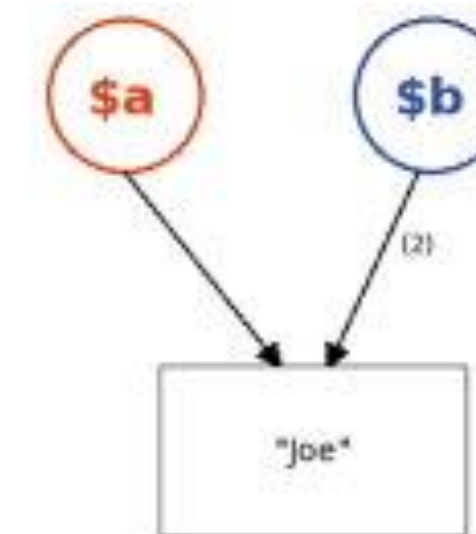
- Mechanizm referencji pozwala na stworzenie dwóch zmiennych odwołujących się do tej samej zawartości.
- Referencję oznaczamy za pomocą symbolu & stawianego przed nazwą zmiennej.
- O referencjach można myśleć jako o aliasach do zmiennej. Mamy jedną zmienną która po prostu mam dwie nazwy.

```
<?php
$a = "Joe";
$b = &$a;
echo($a); // Joe
echo($b); // Joe
$b = "Bob";
echo($b); // Bob
echo($a); // Bob
?>
```

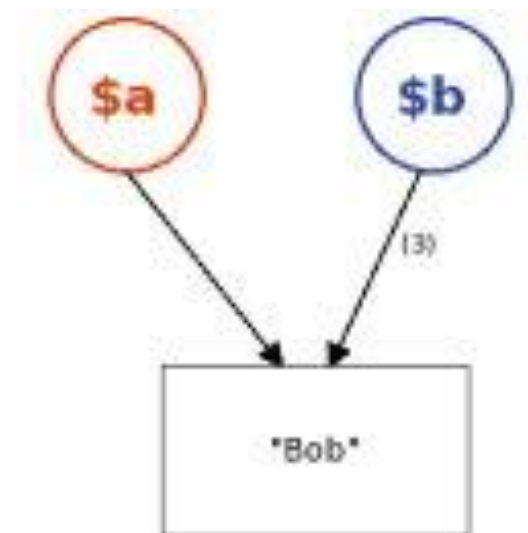
Zmienna **\$a** ma wartość 5.
Zmienna **\$b** odwołuje się do tej samej zawartości co zmienna **\$a**.



`$a = 'Joe';`



`$b = &$a;`



`$b = 'Bob';`

Typy danych

var_dump

Funkcja **var_dump** wyświetla podstawowe informacje o zmiennej przekazanej w parametrze.

Będzie bardzo przydatna podczas nauki i później podczas szukania błędów.

Funkcja ta pokazuje nam informację o tym jakiego typu jest zmienna i jaką dokładnie ma wartość.

Kod PHP

```
<?php
    var_dump("Jakiś napis");
    var_dump(34);
?>
```

Wynik na stronie

string 'Jakiś napis' (length=12)

int 34

Wartość zmiennej

Dodatkowe informacje
o zmiennej

Typ zmiennej

Typy danych

W PHP typy danych dzielimy na:

- skalarne: **boolean**, **integer**, **float**, **string**,
- złożone: **array**, **object**,
- specjalne: **resource**, **null**.

W trakcie działania skryptu zmienna będzie mieć swój typ w zależności od przypisywanej jej wartości.

Dokładny opis tych typów możecie znaleźć tutaj:
<http://php.net/manual/en/language.types.php>

```
<?php
$variable = 2;
var_dump($variable);

$variable = 2.3;
var_dump($variable);

$variable = 'Napis trzymany w zmiennej';
var_dump($variable);

$variable = true;
var_dump($variable);
?>
```

Typy danych – boolean

Typ **boolean** to typ logiczny przyjmujący tylko dwie wartości:

- **true**,
- **false**.

Wykorzystywany jest przy konstruowaniu wyrażeń logicznych oraz sprawdzaniu warunków.

```
<?php
    $isFormSubmitted = true;
    var_dump($isFormSubmitted);
?>
```

Typy danych – integer

- Typ **integer** służy do przechowywania liczb całkowitych (dodatnich i ujemnych).
- Można używać trzech formatów – dziesiętnego (domyślny), ósemkowego i szesnastkowego. Format ósemkowy uzyskujemy dzięki poprzedzeniu liczby znakiem 0, a szesnastkowy – znakami 0x.
- Maksymalny zakres typu całkowitego zależy od platformy sprzętowo-systemowej i zazwyczaj wynosi od -2^{31} do $2^{31}-1$.
- W przypadku przekroczenia zakresu wartość jest konwertowana na typ **float**.

Więcej o systemach liczbowych:

- http://pl.wikipedia.org/wiki/%C3%93semkowy_system_liczbowy (ósemkowy)
- http://pl.wikipedia.org/wiki/Szesnastkowy_system_liczbowy (szesnastkowy)

Typy danych – integer

`<?php`

`$sum = 17;`

10 w systemie dziesiętnym

`$level = -7;`

`$octagonal= 012;`

255 w systemie dziesiętnym

`$hexadecimal = 0xFF;`

`var_dump($sum);`

`var_dump($level);`

`var_dump($octagonal);`

`var_dump($hexadecimal);`

`?>`

Typy danych float

- Typ **float** służy do przechowywania liczb z wartością po przecinku (w informatyce zwanych liczbami zmiennoprzecinkowymi).
- Można używać dwóch formatów – dziesiętnego (z kropką dziesiętną) lub postaci wykładniczej liczby (jest on rzadziej używany).
- Maksymalny zakres typu całkowitego zależy od platformy sprzętowo-systemowej i zazwyczaj wynosi od $-1.8 * 10^{308}$ do $1.8 * 10^{308}$.

<?php

\$price = 9.99;

730 w systemie dziesiętnym

\$tempInCelcius = -2.5;

\$floatInExpForm = 7.3e2;

var_dump(\$price);

var_dump(\$floatInExpForm);

?>

Typy danych string

Typ **string** służy do przechowywania sekwencji (ciągu) znaków. Inaczej nazywany typem łańcuchowym.

Ograniczenie długości napisu to 2GB (czyli w praktyce go nie ma).

Ciągi znaków możemy tworzyć na trzy sposoby:

- za pomocą apostrofu,
- za pomocą cudzysłowu,
- za pomocą konstrukcji **heredoc**.
- za pomocą konstrukcji **newdoc**

```
<?php
```

```
$textQuote = 'To jest jakiś napis';
```

```
$textDoubleQuote = "To jest inny napis";
```

```
var_dump($textQuote);
```

```
var_dump($textDoubleQuote);
```

```
?>
```


Typy danych string

Apostrof

- PHP nie przetwarza w żaden sposób ciągu znaków zawartego między apostrofami. Jeżeli chcemy uzyskać apostrof w samym ciągu znaków, to należy go poprzedzić znakiem \.

<?php

```
$cena = 9.99;
```

Wynik: Cena to \$cena

```
$textQuote = 'Cena to $cena';
```

```
var_dump($textQuote);
```

?>

Cudzysłów

- PHP przetwarza ciąg znaków zawarty pomiędzy cudzysłowami, znajduje w nim nazwę zmiennej i zmienia ją na jej wartość.

<?php

```
$cena = 9.99;
```

Wynik: Cena to 9.99

```
$textDoubleQuote = "Cena to $cena";
```

```
var_dump($textDoubleQuote);
```

?>

Typy danych string

heredoc

- Składnia **heredoc** służy do wygodnego deklarowania zmiennych typu string składających się z wielu linii tekstu.
- Napis należy rozpocząć od sekwencji <<<, za którą musi nastąpić identyfikator. Następnie podajemy właściwy napis. W ostatniej linii wpisujemy identyfikator który stworzyliśmy na początku.
- W ostatniej linii nie mogą znaleźć się żadne inne znaki poza identyfikatorem i średnikiem.
- W przypadku składni **heredoc** nazwy zmiennych również są zmieniane na ich wartości.

```
<?php
```

```
$cena = 9.99;
```

```
$textHeredoc = <<<EOD
```

```
Cena naszego produktu to $cena  
heredoc służy głównie do napisów  
składających się z wielu linii
```

```
EOD;
```

```
var_dump($textHeredoc);
```

```
?>
```

Identyfikatorem może być dowolny ciąg liter. Przyjęte jest pisanie dużymi literami.

Heredoc wstawia wartość zmiennej do napisu

W ostatniej linii może się znajdować tylko identyfikator i średnik.

Typy danych array

- **array** (tablica) jest złożonym typem danych, pozwala na przechowywanie większej liczby elementów i odwoływanie się do nich przy użyciu jednej nazwy.
- W najprostszej postaci jest to indeksowana liczbami kolekcja wartości.
- Każdy indeks (klucz) odwołuje się do przypisanej mu wartości.
- Co indeksy (klucze) mają wartości liczone od 0.

```
<?php
```

```
$weekdays = array();
```

```
$months = [ ];
```

Tworzenie pustej tablicy

Od wersji PHP 5.4 można też tak definiować pustą tablicę

```
$weekdays[0] = 'poniedziałek';
```

```
$weekdays[1] = 'wtorek';
```

```
$weekdays[2] = 'środa';
```

```
$weekdays[3] = 'czwartek';
```

```
$weekdays[4] = 'piątek';
```

```
$weekdays[5] = 'sobota';
```

```
$weekdays[6] = 'niedziela';
```

```
var_dump($weekdays);
```

```
?>
```

Wprowadzanie wartości w odpowiednie komórki (pierwsza komórka ma indeks 0)

Typy danych null

- **null** jest specjalnym typem danych. Wskazuje, że dana zmienna nie przechowuje żadnej wartości – jest pusta.
- Zmienne bez zadeklarowanej wartości są zmiennymi nullowymi.
- Typ **null** nie ma wartości i typu.
- W sensie logiki **null** jest fałszem (**false**)

Funkcja **unset** powoduje wyrzucenie zawartości zmiennej i wstawienie tam wartości **NULL**

```
<?php
$surname;
var_dump($surname);

$surname = 'Nowak';
var_dump($surname);

$surname = NULL;
var_dump($surname);

$surname = 'Kowalski';
var_dump($surname);

unset($surname);
var_dump($surname);
?>
```

Zmienna bez wartości ma w sobie **NULL**

Typy danych object

- **object** (obiekt) jest złożonym typem danych i podstawowym pojęciem paradygmatu programowania obiektowego.
- Więcej o obiektach będziecie mieli na zajęciach z OOP (Object Oriented Programming)

```
<?php
class Animal {
    private $weight;
    private $color;
    private $hungry;

    public function feed() {
        $this->hungry = false;
    }
}

$animal = new Animal();
$animal->feed();
?>
```

Typy danych resource

- **resource** (zasób) jest specjalnym typem danych.
- Wskazuje, że zmienna przechowuje odwołanie do zasobu zewnętrznego, np. pliku lub bazy danych.
- Odwołania te są tworzone za pomocą przeznaczonych w tym celu funkcji.

```
<?php
    $plikTekstowy = fopen('tekst.txt');
    $plikXml = new_xmlrpc("1.0");
    $bazaDanych = mysql_connect();

    var_dump($plikTekstowy);
    var_dump($plikXml);

?>
```

Operatory w PHP

Operatory

Operator jest symbolem określającym konkretną czynność wykonywaną na jakiejś zmiennej.

W PHP operatory dzielimy na zastosowanie:

- arytmetyczne,
- przypisania,
- porównania,
- inkrementacji/dekrementacji,
- logiczne,
- napisowe,
- tablicowe,

- bitowe,
- kontroli błędów,
- wykonania poleceń systemowych.

Innym kryterium podziału jest liczba argumentów (operandów), które przyjmuje operator:

- jednoargumentowe,
- dwuargumentowe,
- jeden operator trójargumentowy.

Priorytety operatorów

- Priorytety operatorów określają, jaka jest kolejność wykonywania reprezentowanych przez nie operacji.
- Priorytety operatorów arytmetycznych w PHP są zgodne z kolejnością wykonywania działań znaną z matematyki.
- Do zwiększenia priorytetu danego operatora można użyć operatora grupowania wyrażeń czyli nawiasów okrągłych.

```
<?php
    $result = 3 + 4 * 5;
    echo($result);

    $result2 = (3 + 4) * 5;
    echo($result2);
?>
```


Operatory arytmetyczne

Służą do wykonywania arytmetycznych operacji matematycznych.

Dostępne są następujące operatory: +, -, *, /, %.

Jedynym nowym operatorem może być % (modulo). Jest to operator który wylicza resztę z dzielenia.

```
<?php
```

```
$sum = 5 + 4;
```

```
$subtraction = 5 - 4;
```

```
$multiplication = 5 * 4;
```

```
$division = 5 / 4;
```

```
$modulo = 5 % 4;
```

```
$modulo = 4 % 5;
```

```
$modulo = 16 % 4;
```

```
?>
```

Operatory przypisania

Służą do przypisania zmiennej jakiejś wartości.
Dostępne są m.in. operatory: =, +=, -=, *=, /=, %=.

```
<?php
```

```
$variable = 5;
```

```
$variable += 2; // $variable= $variable + 2
```

```
$variable -= 3; // $variable = $variable - 3
```

```
$variable *= 4; // $variable = $variable * 4
```

```
$variable /= 2; // $variable = $variable / 2
```

```
$variable %= 5; // $variable = $variable % 5;
```

```
?>
```

Operatory równości

- Służą do sprawdzenia, czy zmienne są sobie równe.
- Operatory porównujące wartości zmiennych: `==`, `!=`, `<>`.
- Operatory porównujące wartości oraz typy zmiennych: `===`, `!==`.
- Wynikiem działania operatorów równości jest wartości logiczna (**boolean**) **true** lub **false**.

```
<?php
```

```
$A = 5;
```

```
$B = 9;
```

```
$C = 5;
```

```
$D = 5.0;
```

```
$A == $B;
```

Zwróci **false** (5 jest różne od 9)

```
$A != $B;
```

Zwróci **true** (5 jest różne od 9)

```
$A <> $B;
```

Zwróci **true** (5 jest równe zarówno 5 jak i 5.0)

```
$A == $C;
```

```
$A == $D;
```

Zwróci **true** (5 jest równe 5)

```
$A === $C;
```

```
$A === $D;
```

Zwróci **false** (5 jest różne od 5.0 – inne typy)

```
$A !== $D
```

Zwróci **true** (5 jest różne od 5.0 – inne typy)

```
?>
```

Luźna i ścisła równość

Luźna równość:

Luźna równość jest porównaniem przy użyciu operatora `==`. Porównuje ona tylko wartość zmiennej a nie tylko jej typ.

Zmienna może być dowolnie rzutowana (silnik PHP będzie zmieniał jej typ próbując dopasować dane).

`<?php`

`$A = 5;` ← Integer
`$B = "5";` ← String
`$C = 5.0;` ← Float

`$A == $B;`
`$A == $C;` ← Zwróci **true**
`?>`

Ścisła równość:

Ścisłą równość jest porównaniem przy użyciu operatora `===`. Porównuje ona zarówno wartość zmiennej jak jej typ.

Więcej możesz przeczytać tutaj:

<http://php.net/manual/en/types.comparisons.php>

`<?php`

`$A = 5;` ← Integer
`$B = "5";` ← String
`$C = 5.0;` ← Float

`$A === $B;`
`$A === $C;` ← Zwróci **false** (inne typy)
`?>`

Operatory porównania

- Służą do określenia wzajemnej zależności między wartościami zmiennych.
- Powinny być stosowane wyłącznie do porównywania wartości liczbowych.
- Dostępne są następujące operatory:
<, <=, >, >=.
- W PHP nie ma operatorów porównania które biorą pod uwagę typ zmiennej. W tych przypadkach zmienna będzie zawsze rzutowana!

```
<?php
```

```
$A = 5;
```

```
$B = 9.0;
```

```
$C = 5;
```

```
$A < $B;
```

```
$A > $B;
```

```
$A <= $B;
```

```
$A >= $C;
```

```
?>
```

Operatory inkrementacji/dekrementacji

- Inkrementacja i dekrementacje to skrócony zapis dodawania (**inkrementacja**, **++**) lub odejmowania (**dekrementacja**, **--**) liczby 1 od wartości zmiennej.
 - Występują w postaciach przedrostkowej (**preinkrementacja** i **predekrementacja**) oraz przyrostkowej (**postinkrementacja** i **postdekrementacja**).
- | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none">➤ Preinkrementacja – wartość zmiennej jest najpierw zwiększana o 1, a potem używana w wyrażeniu.➤ Predekrementacja - wartość zmiennej jest najpierw zmniejszana o 1, a potem używana w wyrażeniu. | <ul style="list-style-type: none">➤ Postinkrementacja – wartość zmiennej jest najpierw używana w wyrażeniu a dopiero pod koniec zwiększana o 1.➤ Postdekrementacja - wartość zmiennej jest najpierw używana w wyrażeniu a dopiero pod koniec zmniejszana o 1 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Operatory inkrementacji/dekrementacji

<?php

`$zmiennaA = 5;`

`echo($zmiennaA++);`

`echo($zmiennaA);`

`echo($zmiennaA--);`

`echo($zmiennaA);`

Wartość zostanie zwiększona dopiero po wykonaniu wszystkich innych operacji w tej linii kodu.

`echo(++$zmiennaA);`

`echo($zmiennaA);`

`echo(--$zmiennaA);`

`echo($zmiennaA)`

Wartość zostanie zmniejszona przed wykonaniem wszystkich innych operacji w tej linii kodu.

?>

Operatory logiczne

Służą do tworzenia wyrażeń logicznych z wyrażeń mających wartości typu logicznego (**true** lub **false**). Funkcjonują zgodnie z zasadami logiki matematycznej.

Dostępne są operatory:

- **&& (AND)** – prawda jeżeli obie strony są prawdą
- **|| (OR)** – prawda jeżeli chociaż jedna z stron jest prawdą
- **! (NOT)** – zaprzeczenie, zmienia wartość logiczną na przeciwną
- **XOR** – (exclusive or) prawda tylko jeżeli jedna z stron jest prawdą a druga fałszem

```
<?php
```

```
$a = true;
```

```
$b = false
```

Zwróci fałsz (prawda i fałsz = fałsz)

```
$a && $b;
```

```
$a AND $b;
```

Zwróci prawdę (prawda i fałsz = prawda)

```
$a || $b;
```

```
$a OR $b;
```

Zwróci fałsz (zaprzecza prawdzie)

```
!a;
```

```
NOT $a;
```

```
$a XOR $b;
```

Zwróci prawdę

```
?>
```


Kolejność operatorów

- W każdym języku programowania operatory mają swoją kolejność wykonywania (podobnie jak w matematyce).
- O tej kolejności musimy pamiętać pisząc bardziej skomplikowane wyrażenia.
- Kolejność operatorów możemy też wymuszać przy użyciu nawiasów okrągłych
- **Nawiasy powinniśmy też stosować do zwiększenia czytelności naszych wyrażeń.**
- Dokładna kolejność operatorów jest wypisana tutaj: <http://php.net/manual/en/language.operators.precedence.php>

```
<?php
```

```
if($A&&$B||$C&&!$D||!$E&&$F){
```

```
...
```

```
}
```

Zapis mało czytelny, bez dokładnej znajomości kolejności wykonania operatorów może nie robić tego co chcemy

```
if(($A && $B ) || ( $C && !$D ) || ( !$E && $F )){
```

```
...
```

```
}
```

```
?>
```

Dodanie spacji i nawiasów powoduje zwiększenie czytelności kodu.

Czas na zadania

- Przeróbcie ćwiczenia z katalogu „Operatory”.
Pierwsze ćwiczenie zróbcie z wykładowcą.

Kontrola przepływu programu

Kontrola przepływu programu

- Standardowo każdy program (skrypt w PHP) wykonuje instrukcję po instrukcji, w kolejności w jakiej owe instrukcje zostały zapisane przez programistę.
- Czasem zachodzi potrzeba, aby pewne instrukcje zostały wykonane tylko wtedy, gdy zachodzi jakiś określony warunek.
- Jeśli zachodzi warunek, to wykonaj pewne instrukcje, w przeciwnym wypadku wykonaj inne instrukcje.
- Służą do tego konstrukcje **if**, **if...elseif...else**, **switch**.

Często chcemy, aby skrypt powtórzył kilkakrotnie wykonywanie pewnego fragmentu instrukcji, zmieniając przy każdym powtórzeniu wartości pewnych zmiennych (lub wykonując inne operacje).

Do tego używamy następujących pętli:

- **while**,
- **for**,
- **foreach**.

if oraz if razem z else

Tę instrukcję warunkową można porównać do obecnego w języku naturalnym stwierdzenia:

jeśli (**if**)..., to zrób....,

jeśli nie poprzednie ale (**else if**) ..., to zrób ...

jeśli nie wszystkie poprzednie (**else**), to zrób....

Część **else if** i **else** jest opcjonalna.

```
if(wyrażenie_warunkowe) {  
    //instrukcja wykonywana,  
    //jeśli spełniony zostanie warunek  
}  
else if(inne_wyrażenie_warunkowe) {  
    //instrukcje wykonywana, jeśli spełniony  
    //zostanie drugi warunek, a pierwszy nie  
}  
else {  
    //instrukcja wykonywana, jeśli nie zostanie  
    //spełniony żaden z poprzednich warunków  
}
```

Operator trójargumentowy

- Operuje na trzech argumentach:
 - **wyrażeniu**, którego wartość logiczna jest sprawdzana,
 - **wartości**, która będzie wynikiem, jeśli sprawdzane wyrażenie jest prawdziwe,
 - **wartości**, która będzie wynikiem, jeśli sprawdzanie wyrażenie nie jest prawdziwe.
- Po wyrażeniu, którego wartość sprawdzamy, następuje znak **?**,
- Po znaku **?** następuje wartość, która będzie wynikiem, jeśli sprawdzane wyrażenie jest prawdziwe,
- Następnie znak **:** oraz wartość, która będzie wynikiem, jeśli sprawdzane wyrażenie nie jest prawdziwe.

```
<?php
$a = 5;
$b = 7;

$result = ($a == $b) ? 9 : 91;

var_dump($result);
?>
```

Wartość zwracana jeżeli wyrażenie ewaluuje się do prawdy

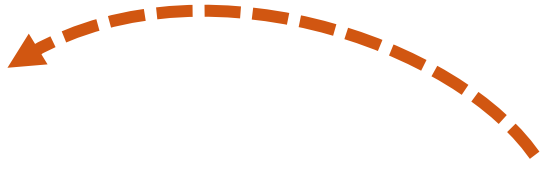
Wartość zwracana jeżeli wyrażenie ewaluuje się do fałszu

Wyrażenie które zdecyduje jaka wartość zostanie zwrócona z operatora.

Instrukcja wyboru switch

- Instrukcja **switch** umożliwia wykonanie określonej instrukcji (lub zbioru instrukcji) w zależności od wartości, jaką wygeneruje badane wyrażenie.
- Jest przydatna, gdy trzeba zbadać dokładną wartość jakiegoś wyrażenia (bez porównywania typu, operator **==**) i w zależności od tej wartości wykonać jakieś instrukcje.
- W wyrażeniu **switch** nie zadajemy pytań, możemy sprawdzić tylko wartość naszej zmiennej względem napisanych przez nas wartości.

```
switch( $zmienna ) {  
    case wartość1:  
        instrukcja1;  
        break;  
    case wartość2:  
        instrukcja2;  
        break;  
    ...  
    default:  
        instrukcjaN;  
}
```



Jeżeli **\$zmienna** będzie miała dokładnie wartość podaną po **case** to ten kod zostanie wywołany

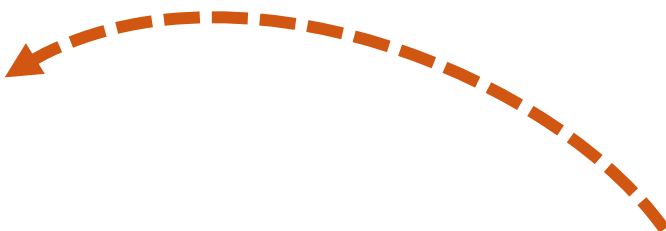
Instrukcja wyboru switch

- Każda z wartości występujących po słowie kluczowym **case** porównywana jest kolejno z wartością obliczoną na podstawie badanego wyrażenia zawartego w nawiasach okrągłych po słowie kluczowym **switch**.
- Jeżeli wartość badanego wyrażenia i wartość występująca po słowie kluczowym **case** będą równe, wykonane będą instrukcje występujące w tej sekcji **case**.
- Instrukcja **break** powoduje opuszczenie całego bloku instrukcji **switch**.
- Jeśli żadna z wartości po słowach **case** nie będzie pasowała do wartości obliczonej w badanym wyrażeniu, wykonana zostanie instrukcja umieszczona po słowie kluczowym **default**.
- Sekcja **default** nie jest obowiązkowa (ale dobra praktyka mówi, żebyśmy ją zawsze dodawali). Zazwyczaj umieszcza się ją po wszystkich sekcjach **case**.

Po co nam break?

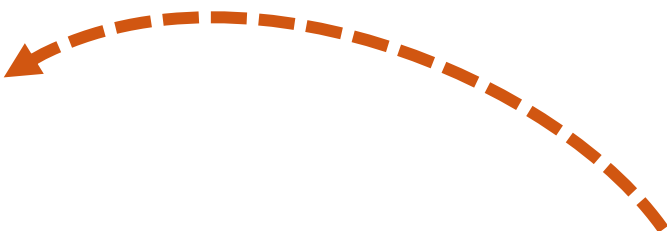
- **break** jest specjalnym słowem kluczowym które powoduje przerwanie konstrukcji w której został użyty (np. konstrukcji **switch**, lub **pętli**). W switch jest o tyle ważny że jeżeli go nie użyjemy to program przejdzie nam do następnego wyrażenia case.

```
$numer = 1;  
switch( $number ) {  
  case 1:  
    echo("Wartość 1 <br>");  
    break;  
  case 2:  
    echo("Wartość 2 <br>");  
    break;  
  case 3:  
    echo("Wartość 3 <br>");  
    break;  
  default:  
    echo("Wartość default <br>");  
}
```



Wypisze na ekranie:
Wartość 1

```
$numer = 1;  
switch( $number ) {  
  case 1:  
    echo("Wartość 1 <br>");  
  case 2:  
    echo("Wartość 2 <br>");  
  case 3:  
    echo("Wartość 3 <br>");  
  default:  
    echo("Wartość default <br>");  
}
```



Wypisze na ekranie:
Wartość 1
Wartość 2
Wartość 3
Wartość default

Pętla while

- Na początku sprawdzana jest wartość wyrażenia (**wyrażenie**) stanowiącego warunek. Jeżeli wyrażenie to ma wartość **true**, wykonywane są instrukcje zawarte w ciele pętli (zawarte w nawiasach klamrowych).
- Po każdorazowym wykonaniu instrukcji z ciała pętli sprawdzana jest wartość wyrażenia stanowiącego warunek – jeżeli wyrażenie zwróci wartość **false**, pętla zostaje zakończona.
- W ciele pętli **while** powinna być umieszczona instrukcja modyfikująca wartość wyrażenia warunkowego. W przeciwnym wypadku pętla nigdy się nie zatrzyma – pętla nieskończona.

```
while(wyrażenie) {  
    instrukcja1;  
    instrukcja2;  
    ...  
    instrukcjaN;  
}
```

Pętla for

Przed rozpoczęciem wykonywania kodu pętli wartościowane jest **wyrażenie_inicjujące**.

- Następnie sprawdzana jest wartość **wyrażenia_warunkowego**. Jeżeli jest to **true**, pętla może być kontynuowana – wykonywane są instrukcje w ciele pętli. W przeciwnym razie (**false**) wykonanie pętli zostaje zakończone.
- Kolejnym krokiem jest wartościowanie **wyrażenia_iteracyjnego** (najczęściej – modyfikacja licznika pętli) i znowu obliczane jest **wyrażenie_warunkowe** decydujące, czy pętla będzie kontynuowała swoje działanie.
- **Wyrażenie_inicjujące** wartościowane jest tylko raz, przed wykonaniem pętli.

```
for(wyrażenie_inicjujące;  
wyrażenie_warunkowe; wyrażenie_iteracyjne) {  
    instrukcja1;  
    instrukcja2;  
    ...  
    instrukcjaN;  
}
```

Czas na zadania

- Przeróbcie ćwiczenia z katalogu „Kontrola przepływu programu”.
Pierwsze ćwiczenia zróbcie z wykładowcą.

KONIEC