

Temat: Instrukcje sterujące

PHP – instrukcje sterujące

Język PHP zawiera kilka instrukcji sterujących:

- ✓ if,
- ✓ switch,
- ✓ break,
- ✓ for,
- ✓ while,
- ✓ do-while,
- ✓ foreach,
- ✓ continue ,
- ✓ return (przy funkcjach).

Do grupowania instrukcji służą nawiasy klamrowe { oraz }.

Kryteria:

1. Wyjaśniam konstrukcję i działanie instrukcji warunkowej
-

PHP – instrukcja warunkowa

Podczas pisania programu często zachodzi konieczność zaprogramowania różnego jego działania w zależności od jakiegoś warunku.

Taką możliwość zapewnia w PHP ***instrukcja warunkowa***.

Składnia instrukcji if ma postać:

```
if (warunek) instrukcja;
```

Poprawnym użyciem będzie więc:

```
if ($a < 15) echo $a;
```

PHP – instrukcja warunkowa

Stosując nawiasy klamrowe, możemy w miejsce jednej instrukcji echo wykonać kilka instrukcji (blok instrukcji):

```
if ($a < 15) {  
    echo $a;  
    echo "\n";  
    $a = $a * 1;  
}
```

Warunek jest oceniany przez wartość logiczną. Jeśli wyrażenie zostanie ocenione na TRUE, PHP wykona instrukcję , a jeśli oceni na FALSE- zignoruje.

PHP – instrukcja warunkowa

Instrukcja if przyjmuje jedną z dwóch form.

Krótką:

```
if (warunek) {  
    instrukcja;  
}
```

lub **długą:**

```
if (warunek) {  
    instrukcja1;  
} else {  
    instrukcja2;  
}
```

Instrukcja **else** jest wykonywana tylko wtedy, gdy wyrażenie **if** zostało poddane ocenie **FALSE**

PHP – instrukcja warunkowa

Słowo kluczowe elseif pozwala na dodanie kolejnych warunków:

```
if (war1) {  
    instr1;  
} elseif (war2) {  
    instr2;  
} elseif (war3) {  
    instr3;  
} elseif (war4) {  
    instr4;  
}
```

PHP – instrukcja warunkowa

- ✓ w PHP można również napisać "**else if**" (w dwóch słowach), a zachowanie będzie identyczne z zachowaniem "**elseif**" (w jednym słowie). Znaczenie syntaktyczne jest nieco inne ale obydwa skutkują dokładnie tym samym zachowaniem.
- ✓ instrukcje **if** można zagnieżdżać w nieskończoność w innych instrukcjach **if** , co zapewnia pełną elastyczność warunkowego wykonywania różnych części programu.

PHP – instrukcja warunkowa

```
<?php
    $i=7;
    print "i = ".$i."<br>";
    if ($i>0) {
        print ('zmienna i jest większa od 0');
    } else if ($i == 0) {
        print ('zmienna i jest równa 0');
    } else {
        print ('zmienna i jest mniejsza od 0');
    }
?>
```

PHP – instrukcja warunkowa

Złożone warunki

```
<?php
    $orderValue = 350;
    $productCategory = 'Tel';
    if ($orderValue >= 300 && $productCategory == 'Tablety'){
        echo 'Rabat 15%.';
    }
    elseif($orderValue >= 300 || $productCategory == 'Tablety'){
        echo 'Rabat 10%.';
    }
    else{
        echo 'Brak rabatu.';
    }
?>
```

PHP – instrukcja warunkowa

Złożone warunki

```
<?php
    $liczba = "4";
    if (isset($liczba) and is_numeric($liczba)){
        /*Funkcja isset zwróci wartość prawda jeżeli podana
        zmienna istnieje.
        Funkcja is_numeric zwróci wartość prawda jeżeli podana
        zmienna przechowuje wartość liczbową
        */
        echo 'Zmienna liczba przechowuje wartość liczbową';
    }
    elseif(isset($liczba) and !is_numeric($liczba)){
        echo 'Zmienna liczba nie przechowuje liczby';
    }
    else{
        echo 'Zmienna liczba nie istnieje';
    }
?>
```

PHP – instrukcja warunkowa

Zagnieżdżanie:

```
<?php
    $liczba = "słowo";
    if (isset($liczba)){
        if (is_numeric($liczba)){
            echo 'Zmienna liczba przechowuje wartość liczbową';
        }
        else{
            echo 'Zmienna liczba nie przechowuje liczby';
        }
    }
    else{
        echo 'Zmienna liczba nie istnieje';
    }
?>
```

PHP – operator warunkowy

Krótszy zapis instrukcji warunkowej umożliwia operator warunkowy

Składnia ma postać:

warunek ? co jeśli prawda : co jeśli fałsz;

np.:

```
$discount=($orderValue>=300) ? 10: 5;
```

PHP – operator warunkowy

```
<h3>Operator warunkowy:</h3>
```

```
<?php
```

```
    $orderValue = 140;
```

```
    /* if ($orderValue >= 300){
```

```
        $discount = 10;
```

```
    }
```

```
    else {
```

```
        $discount = 5;
```

```
    }*/
```

```
    $discount=($orderValue>=300) ? 10:5;
```

```
    echo "wartość rabatu to $discount%";
```

```
?>
```

PHP – operator warunkowy

echo i print a operator warunkowy

<h3>Operator warunkowy:</h3>

<?php

```
$a = 5; // przypisujemy wartość zmiennej $a
```

```
echo ($a>5) ? 'Większa od 5' : 'Mniejsza, bądź równa 5';
```

```
echo "<br>";
```

```
// inny sposób użycia
```

```
$a>5 ? print 'Większa od 5' : print 'Mniejsza, bądź równa 5';
```

```
echo "<br>";
```

```
/*
```

```
w wersji z echo:
```

```
$a>5 ? echo 'Większa od 5' : echo 'Mniejsza, bądź równa 5';
```

```
nie działa*/
```

?>

Kryteria:

1. Wyjaśniam konstrukcję i działanie instrukcji warunkowej
-

Pokaż światłami/kciukami jak oceniasz, na ile spełniasz kryterium

Kryteria:

2. Wyjaśniam konstrukcję i działanie instrukcji wyboru

PHP – Instrukcja wyboru (switch)

Umożliwia rozgałęzienie działania programu w zależności od wartości wyrażenia

```
switch(wyrażenie){  
    case wartość1 :  
        instrukcje1;  
        break;  
    case wartość2 :  
        instrukcje2;  
        break;  
    case wartość3 :  
        instrukcje3;  
        break;  
    default :  
        instrukcje4;  
}
```

Instrukcja **break** przerywa wykonywanie całego bloku case. Jej przypadkowe pominięcie może doprowadzić do nieoczekiwanych wyników i błędów w programie.

Blok **default** nie jest obligatoryjny i jeśli nie jest w skrypcie potrzebny, można go pominąć.

PHP – Instrukcja wyboru (switch)

```
<?php
    $liczba = 15;
    switch($liczba){
        case 10 :
            echo "Zmienna liczba = 10";
            break;
        case 20 :
            echo "Zmienna liczba = 20";
            break;
        default :
            echo "Zmienna liczba nie jest
            równa ani 10, ani 20.";
    }
?>
```

PHP – Instrukcja wyboru (switch)

```
<?php
// Polska nazwa miesiąca określana w funkcji switch
// do uzyskania dnia, miesiąca i roku używasz funkcji date.
$dzien = date("d");
$miesiac = date("m");
$rok = date("Y");
switch ($miesiac) {
    case '01': $miesiac = 'stycznia'; break;
    case '02': $miesiac = 'lutego'; break;
    case '03': $miesiac = 'marca'; break;
    case '04': $miesiac = 'kwietnia'; break;
    case '05': $miesiac = 'maja'; break;
    case '06': $miesiac = 'czerwca'; break;
    case '07': $miesiac = 'lipca'; break;
    case '08': $miesiac = 'sierpnia'; break;
    case '09': $miesiac = 'września'; break;
    case '10': $miesiac = 'października'; break;
    case '11': $miesiac = 'listopada'; break;
    case '12': $miesiac = 'grudnia'; break;
    default: $miesiac = 'niezidentyfikowany'; break;
}
print "$dzien $miesiac $rok";
```

?>

Kryteria:

2. Wyjaśniam konstrukcję i działanie instrukcji wyboru

Pokaż światłami/kciukami jak oceniasz, na ile spełniasz kryterium

PHP – Pętle

Pętle są konstrukcjami programistycznymi, które pozwalają na wykonywanie powtarzających się czynności.

W PHP występują 4 rodzaje pętli:

- ✓ typu for
- ✓ typu while
- ✓ typu do...while
- ✓ typu foreach

Kryteria:

3. Wyjaśniam konstrukcję i działanie pętli for

PHP – Pętla for

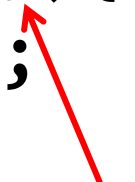
Pętla typu for ma ogólną postać:

```
for (ustalenie_początku; warunek_pętli;  
zwiększenie_licznika)  
{  
    blok instrukcji  
}
```


PHP – Pętla for

Istnieje duża dowolność w umieszczaniu wyrażeń. Na przykład **zwiększenie_licznika**, które najczęściej jest wykorzystywane do modyfikacji zmiennej iteracyjnej, można umieścić wewnątrz samej pętli:


```
for (ustalenie_początku; warunek_pętli;){  
    blok instrukcji;    zwiększenie_licznika;  
}
```



zwróć uwagę na znak średnika, występującego po wyrażeniu warunkowym. Mimo że wyrażenie modyfikujące znalazło się wewnątrz bloku pętli, **średnik ten jest niezbędny.**

PHP – Pętla for

Podobnych przenosin można dokonać z **ustaleniem początku**. Taka konstrukcja wygląda następująco:

```
ustalenie_początku;  
for (; warunek_pętli;){  
    blok instrukcji;  
    zwiększenie_licznika;}  

```

zwróć uwagę na umiejscowienie średników pętli for. Oba są niezbędne do prawidłowego działania kodu.

PHP – Pętla for

```
<p>1 seria liczb</p>
<?php
for ($i = 1; $i < 7; $i++) {
    ..... echo $i . "<br>";
}
?>

<p>2 seria liczb</p>
<ul>
<?php
$i = 1;
for (; $i < 7; $i++) {
    ..... echo "<li>$i</li>";
}
?>
</ul>
```

1 seria liczb

1
2
3
4
5
6

2 seria liczb

- 1
- 2
- 3
- 4
- 5
- 6

PHP – Pętla for

Istnieje również możliwość przeniesienia wyrażenia warunkowego do wnętrza pętli, wymaga to jednak zastosowania instrukcji **break**.

```
<p>3 seria liczb</p>
<table>
<?php
for ($i = 1; ; $i++) {
    if($i >= 7){
        break;
    }
    echo "<tr><td>$i</td></tr>";
}
?>
</table>
```

3 seria liczb

1
2
3
4
5
6

PHP – Pętla for

Wszystkie wyrażenia przeniesione poza nawias okrągły:

```
<p>3 seria liczb</p>
<table>
<?php
$i = 1;
for (; ; ) {
    if($i >= 7) {
        break;
    }
    echo "<tr><td>$i</td></tr>";
    $i++;
}
?>
</table>
```

3 seria liczb

1
2
3
4
5
6

PHP – Pętla for

Zagnieżdżanie pętli

Jaki będzie wynik działania skryptu?

```
<table>
<tr>
|   <th></th>
<?php
$row = 3;
$col = 6;
for ($i = 1; $i <= $col; $i++) {
|   echo "<th>$i</th>";
}
?>
</tr>
<?php
for ($i = 1; $i <= $row; $i++) {
|   echo '<tr>';
|   echo "<th>$i</th>";
|   for ($j = 1; $j <= $col; $j++) {
|       |   echo '<td>';
|       |   echo $j * $i;
|       |   echo '</td>';
|       }
|   echo "</tr>";
}
?>
</table>
```

PHP – Pętla for

Zagnieżdżanie pętli

	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	8	10	12
3	3	6	9	12	15	18

```
<table>
<tr>
    <th></th>
<?php
$row = 3;
$col = 6;
for ($i = 1; $i <= $col; $i++) {
    echo "<th>$i</th>";
}
?>
</tr>
<?php
for ($i = 1; $i <= $row; $i++) {
    echo '<tr>';
    echo "<th>$i</th>";
    for ($j = 1; $j <= $col; $j++) {
        echo '<td>';
        echo $j * $i;
        echo '</td>';
    }
    echo "</tr>";
}
?>
</table>
```

PHP – Pętla for

Zagnieżdżanie pętli

Jaki będzie wynik działania skryptu?

```
<pre>
<?php
$heigh = 16;
for ($i = 0; $i <= $heigh; $i++) {
    for ($j = 1; $j <= $heigh - $i; $j++) {
        echo ' ';
    }
    for ($j = 1; $j <= 2 * $i - 1; $j++) {
        echo '*';
    }
    echo "<br>";
}
?>
</pre>
```


<http://localhost/zsk/l5p11.php>

[illegible]

PHP – Pętla for

Zagnieżdżanie pętli

Jaki będzie wynik działania skryptu?

```
<?php
    define('X', 11);
    echo '<pre>';
    for ($w = 1; $w <= X; $w++){
        for ($k = 1; $k <= X + $w; $k++)
        {
            if ($k == X + $w) echo "\n"; else
            if ($k > X - $w && $k < X + $w) echo '*';
            else echo ' ';
        }
    }
    echo '</pre>';
?>
```


Kryteria:

3. Wyjaśniam konstrukcję i działanie pętli for

Pokaż światłami/kciukami jak oceniasz, na ile spełniasz kryterium

Kryteria:

4. Wyjaśniam konstrukcję i działanie pętli
while

Pokaż światłami/kciukami jak oceniasz, na ile spełniasz
kryterium

PHP – Pętla while

- ✓ Pętla typu **while**, podobnie jak pętla for, służy do wykonywania powtarzających się czynności.
- ✓ pętlę **for** najczęściej wykorzystuje się wówczas, gdy liczba powtarzanych operacji jest znana przed wejściem do pętli
- ✓ pętlę **while** — gdy liczby powtórzeń nie znamy (np. jest ona wynikiem działania pewnej funkcji).
- ✓ podział jest umowny, gdyż oba typy pętli można zapisać w taki sposób, aby były swoimi funkcjonalnymi odpowiednikami.

PHP – Pętla while

Ogólna postać pętli while :

```
while (wyrażenie warunkowe){instrukcje;
```

Instrukcje są wykonywane dopóty, dopóki wyrażenie warunkowe jest prawdziwe.

Wykonanie instrukcji **while** przebiega następująco:

- ✓ najpierw sprawdzany jest warunek *wyrażenie warunkowe*,
- ✓ jeśli warunek przyjmuje wartość logiczną true, instrukcja jest wykonywana.
- ✓ po wykonaniu instrukcji ponownie sprawdzany jest warunek, jeśli warunek przyjmuje wartość false, wykonanie instrukcji while jest zakończone.

PHP – Pętla while

W pętli `while` **sprawdzanie warunku następuje przed pierwszym wykonaniem instrukcji**. Może się więc tak zdarzyć, że instrukcja nie zostanie wykonana ani razu.

PHP – Pętla while

```
<?php
    $k=8;
    $i=$k;
    $n = 12;
    $sum = 0;
    while ($k<=$n) {
        $sum += $k;
        $k+=2;
    }
    print ("suma liczb parzystych od $i do $n wynosi $sum");
?>
```

suma liczb parzystych od 8 do 12 wynosi 30

PHP – Pętla while

Wyrażenie warunkowe może być jednocześnie wyrażeniem modyfikującym:

```
<p>seria liczb</p>
<ul>
<?php
    $i = 0;
    $n = 10;
    while ($i++<$n) {
        echo "<li>$i</li>";
    }
?>
</ul>
```

seria liczb

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Kryteria:

4. Wyjaśniam konstrukcję i działanie pętli
while

Pokaż światłami/kciukami jak oceniasz, na ile spełniasz
kryterium

Kryteria:

5. Wyjaśniam konstrukcję i działanie pętli
do....while

PHP – Pętla do...while

Ogólna postać pętli **do...while** wygląda następująco:

```
do {instrukcje;}while(warunek);
```

Konstrukcję tę należy rozumieć następująco:

"Wykonuj *instrukcje*, dopóki *warunek* jest prawdziwy".

To odwrotność pętli typu **while**, którą rozumie się jako:

"Dopóki *warunek* jest prawdziwy, wykonuj instrukcje".

PHP – Pętla do...while

Na wykonanie instrukcji do-while składa się kilka etapów:

- ✓ najpierw wykonywana jest instrukcja ,
- ✓ następnie sprawdzany jest warunek ,
- ✓ jeśli warunek przyjmuje wartość true, to następuje ponowne wykonanie instrukcji , a po nim — ponowne sprawdzanie warunku,
- ✓ jeśli warunek przyjmuje wartość false, to wykonanie instrukcji do-while jest zakończone.

PHP – Pętla while a do...while

W pętli **while** najpierw jest sprawdzany warunek, a dopiero później są wykonywane instrukcje.

W przypadku pętli **do...while** jest odwrotnie, najpierw są wykonywane instrukcje, a dopiero potem jest sprawdzany warunek.

Ta cecha pętli **do...while** jest bardzo ważna, oznacza bowiem, że **pętla tego typu jest wykonywana co najmniej raz, nawet jeśli warunek jest fałszywy.**

PHP – Pętla do...while

1 seria:

Pętla do...while [i = 0]
Pętla do...while [i = 1]
Pętla do...while [i = 2]
Pętla do...while [i = 3]
Pętla do...while [i = 4]
Pętla do...while [i = 5]
Pętla do...while [i = 6]
Pętla do...while [i = 7]
Pętla do...while [i = 8]
Pętla do...while [i = 9]

2 seria:

Pętla do...while [i = 0]

```
<p>1 seria:</p>
```

```
<?php
```

```
    $i = 0;
```

```
    do{
```

```
        echo "Pętla do...while [i = $i]";
```

```
        echo "<br />";
```

```
        $i++;
```

```
    }
```

```
    while($i < 10);
```

```
?>
```

```
<p>2 seria:</p>
```

```
<?php
```

```
    $i = 0;
```

```
    do{
```

```
        echo "Pętla do...while [i = $i]";
```

```
        echo "<br />";
```

```
        $i++;
```

```
    }
```

```
    while($i < 0); //niespełniony warunek
```

```
?>
```


Kryteria:

5. Wyjaśniam konstrukcję i działanie pętli
do....while

Pokaż światłami/kciukami jak oceniasz, na ile spełniasz
kryterium

Kryteria:

6. Wyjaśniam konstrukcję i działanie pętli
foreach

PHP – Pętla foreach

- ✓ Ma specyficzne zastosowanie, służy wyłącznie do przeglądania zawartości typów złożonych: tablic oraz obiektów.
- ✓ spowoduje błąd podczas próby użycia go w zmiennej o innym typie danych lub niezainicjowanej zmiennej
- ✓ pętla ta przechodzi przez każdy element tablicy pobierając jego wartość .Pobraną wartość zapisuje w tymczasowej zmiennej

PHP – Pętla foreach

Dwie składnie.

Pierwsza:

```
foreach( $przegladanyObiekt as tymczasowaWartosc )  
{ // ciało pętli }
```

Dla każdej iteracji pętli wartość bieżącego elementu tablicy jest przypisywana do **tymczasowaWartosc**, a wskaźnik tablicy jest przesuwany o jeden, aż dotrze do ostatniego elementu tablicy.

✓ **tymczasowaWartosc** może mieć dowolną nazw

PHP – Pętla foreach

Nie musimy określać jak duża jest tablica, pętla automatycznie ma tyle przebiegów ile tablica ma elementów

```
<?php
$frameworks = ['Laravel', 'Zend', 'Symfony', 'CakePHP'];
?>
<h3>Popularne frameworki PHP</h3>
<ul>
    <?php
        foreach ($frameworks as $fraem){
            echo "<li>$fraem</li>";
        }
    ?>
</ul>
```

Popularne frameworki PHP

- Laravel
- Zend
- Symfony
- CakePHP

PHP – Pętla foreach a for

Aby wypisać wszystkie elementy tablicy przy pomocy pętli for konieczna jest wiedza, ile ich jest lub użycie odpowiedniej funkcji wyznaczającej ilość elementów tablicy

```
<?php
$frameworks = ['Laravel', 'Zend', 'Symfony', 'CakePHP'];
?>
<h3>Popularne frameworki PHP</h3>
<ul>
    <?php
        for($i = 0; $i < count($frameworks); $i++){
            echo "<li>$frameworks[$i]</li>";
        }
    ?>
</ul>
```

Popularne frameworki PHP

- Laravel
- Zend
- Symfony
- CakePHP

PHP – Pętla foreach

W przypadku tablic asocjacyjnych korzystając z pierwszej składni uzyskamy jednak jedynie wartości kolejnych kluczy, nie zaś nazwy kluczy. Jeśli również ta informacja jest potrzebna, trzeba zastosować drugą wersję pętli foreach.

PHP – Pętla foreach

Druga składnia:

```
foreach( $przebadanyObiekt as tymczasowyKlucz =>  
tymczasowaWartosc )  
{ // ciało pętli }
```

✓ **tymczasowaWartosc** i **tymczasowyKlucz** mogą mieć dowolne nazwy

PHP – Pętla foreach

```
<?php
$user= array(
    "imię"=>"Adam",
    "nazwisko"=>"Małolepszy",
    "klasa"=>"kl.4",
    "szkoła"=>"ZSK",
    "wiek"=>19,
    "średnia"=>4.5);
echo "<h3>Wyświetlenie tablicy asocjacyjnej, 1sposób:</h3>";
foreach ($user as $person){
    echo $person." ";
}
echo "<h3>Wyświetlenie tablicy asocjacyjnej, 2sposób:</h3>";
foreach ($user as $key=>$person){
    echo $key." : ".$person."<br>";
}
?>
```

Wyświetlenie tablicy asocjacyjnej, 1sposób:

Adam Małolepszy kl.4 ZSK 19 4.5

Wyświetlenie tablicy asocjacyjnej, 2sposób:

imię: Adam
nazwisko: Małolepszy
klasa: kl.4
szkoła: ZSK
wiek: 19
średnia: 4.5

PHP – Pętla foreach

- ✓ Tworzone przez foreach zmienne są jedynie kopiami oryginalnych wartości, dlatego próba ich modyfikacji wewnątrz pętli w żaden sposób nie wpłynie na zawartość tablicy.
- ✓ Aby dokonać zmian w oryginale należy użyć referencji. (więcej na kolejnych lekcjach)

PHP – Pętla foreach

```
<?php
    $konta = array(200, 4000, 600, 700, 340);
    echo '<h3>Konta przed operacja</h3>';
    print_r($konta);
    echo '<h3>Szanowni klienci, pobieramy od kazdego
    konta prowizje w wysokości 20 zl</h3>';
    foreach($konta as &$daneKonto){
        // pobieramy za pomoca referencji
        $daneKonto -= 20;
    }
    print_r($konta);
?>
```

Konta przed operacja

Array ([0] => 200 [1] => 4000 [2] => 600 [3] => 700 [4] => 340)

Szanowni klienci, pobieramy od kazdego konta prowizje w wysokości 20 zl

Array ([0] => 180 [1] => 3980 [2] => 580 [3] => 680 [4] => 320)

Kryteria:

6. Wyjaśniam konstrukcję i działanie pętli
foreach

Pokaż światłami/kciukami jak oceniasz, na ile spełniasz
kryterium

Kryteria:

7. Wyjaśniam konstrukcję i działanie instrukcji
break i continue

PHP – break i continue

- ✓ Przy okazji omawiania instrukcji switch poznaliśmy komendę **break**. Ma ona bardzo duże zastosowanie przy pętlach, które pozwala przerywać.
- ✓ Istnieje także kolejne polecenie: **continue**. Przerywa aktualny cykl pętli i powoduje rozpoczęcie następnego.
- ✓ Pozwalają na większe możliwości sterowania pętlami

PHP – break i continue

- ✓ instrukcja **break**; powoduje natychmiastowe wyjście z pętli i rozpoczęcie wykonywania pierwszej instrukcji znajdującej się po pętli (tak jak w instrukcji switch...);
- ✓ Wykorzystywana w *for*, *foreach*, *while*, *do-while* oraz *switch*
- ✓ instrukcja **continue**; powoduje pominięcie pozostałej części iteracji pętli i przeskoczenie do następnego obiegu pętli;

PHP – break

Instrukcja **break**; w pętlach najczęściej wykorzystywana jest w połączeniu z instrukcją **if**

```
1
2
3
4
5
one
two
three
four
```

```
<?php
    $i=0;
    while ($i<10){
        $i++;echo $i."<br>";
        if ($i==5){
            break;
        }
    }

    $arr = array('one', 'two', 'three',
        'four', 'stop', 'five');
    foreach ($arr as $val) {
        if ($val == 'stop') {
            break;
        }
    }
    echo "$val<br>";
}
```

?>

PHP –continue

```
<?php
for ($i = 1; $i <= 20; $i++){
    if ($i % 2 == 0) echo $i;
    else continue;
    if ($i < 20) echo ', ';
}
?>
```

2, 4, 6, 8, 10, 12, 14, 16, 18, 20

Jeżeli licznik $\$i$ jest parzysty (reszta z dzielenia równa zero), to jest wyświetlany. Jest również pokazywany (dla $\$i < 20$) przecinek. Dla nieparzystych liczników instrukcja wyświetlająca przecinki nie jest wykonywana, ponieważ po **continue** następuje przeskok do następnego obiegu pętli.

Kryteria:

7. Wyjaśniam konstrukcję i działanie instrukcji
break i continue

Pokaż światłami/kciukami jak oceniasz, na ile spełniasz
kryterium

Kryteria:

8. Wyjaśniam alternatywne składowe struktur sterujących

PHP – składnie alternatywne

Podczas pisania kodu PHP bardzo często występują sytuacje, kiedy musimy ten kod przeplatać z kodem HTML. Stosowanie nawiasów { i } oraz odpowiednich wcięć, czasem psuje przejrzystości kodu.

W takich sytuacjach możemy zamiennie zastosować składnię w której:

- ✓ nawias otwierający { zastąpimy dwukropkiem :,
- ✓ natomiast nawias zamykający } odpowiednio endif, endswitch, endwhile, endfor lub endforeach.

Dla pętli do...while... nie ma składni alternatywnej.

PHP –składnie alternatywne

```
1  
2  
3  
4  
5  
6  
7  
2, 4, 6, 8, 10, 12, 14, 16, 18, 20
```

```
<?php  
    $j=0;  
    while($j<7):  
        $j++;  
        echo $j."<br>";  
    endwhile;  
  
    for ($i = 1; $i <= 20; $i++):  
        if ($i % 2 == 0) echo $i;  
        else continue;  
        if ($i < 20) echo ', ';  
    endfor;  
?>
```

Kryteria:

8. Wyjaśniam alternatywne składowe struktur sterujących

Pokaż światłami/kciukami jak oceniasz, na ile spełniasz kryterium

PHP –podsumowanie

Popraw kod, by działał zgodnie z oczekiwaniami:

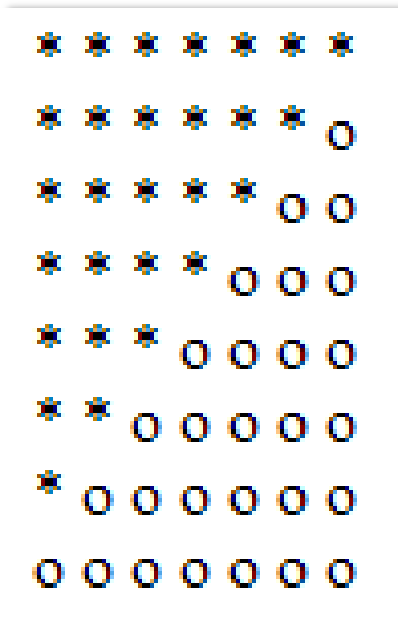
```
<?php
    $k=6;
    $n = 10;
    $sum = 1;
    do {
        $sum += $k;
    }
    while ($k++<=$n);
    print ("suma liczb parzystych od $k do $n wynosi $sum");
?>
```

Jest: suma liczb parzystych od 12 do 10 wynosi 52

Powinno być: suma liczb parzystych od 6 do 10 wynosi 24

PHP –podsumowanie

Popraw kod, by działał zgodnie z oczekiwaniami:



```
<?php
$height= 7;
while ( $i <= $height ) {
    $j = 1;
    while ( $j <=$height-$i ) {
        echo '* ';
        $j++;
    }
    $k = 1;
    while ( $k <=$i ) {
        echo 'o ';
    }
    echo "<br>";
    $i++;
}
?>
```