Pętle w PHP

W tej lekcji omówimy rodzaje pętli (*ang. loop = pętla*) dostępnych w języku PHP. Instrukcje iteracyjne (*iteracja = jedno wykonanie kodu pętli*) – umożliwiają nam wykonywanie tych samych instrukcji (bloku kodu) wielokrotnie, co przydaje się w wielu zastosowaniach – np. przy wypisywaniu wyjętych z bazy danych produktów z określonej kategorii w sklepie internetowym czy ostatnich wpisów opublikowanych na blogu. Podobnie w wielu algorytmach pętle okażą się kluczowe do poprawnego przeprowadzenia różnorakich obliczeń. Oczywiście zrozumienie działania pętli to zawsze krok w stronę bardziej profesjonalnego programowania oraz nieoceniona pomoc na egzaminie, gdzie pętle chociażby *fetchujące* (wyjmujące rekordy z bazy) spotkamy w większości zadań INF.03 dotyczących PHP.

Jeśli chodzi o wiedzę na temat działania pętli, to w lekcji: Pętle w JS wyjaśniliśmy już wiele fundamentalnych zasad logicznych dotyczących iterowania oraz ukazaliśmy klasyczne pętle w akcji, stąd gorąco zalecam zapoznać się najpierw właśnie z tamtą lekcją z rozdziału o JavaScript. Natomiast niniejszy artykuł – o pętlach w PHP – potraktujmy bardziej jako przedstawienie różnic składniowych pomiędzy PHP a JavaScript – nie będziemy zatem po raz drugi tak bardzo szczegółowo wyjaśniać tych samych zasad logicznych. Do dzieła!

Klasyczne rodzaje pętli w PHP

W PHP mamy do dyspozycji cztery fundamentalne rodzaje pętli:

- `for` – gdy znamy z góry liczbę iteracji do wykonania.
- `while` – gdy wykonujemy kod dopóki warunek pozostanie spełniony.
- `do..while` – podobna do `while`, lecz chcemy aby kod w środku pętli wykonał się na pewno co najmniej jeden raz.
- `foreach` – najwygodniejsza forma iterowania po elementach tablicy.

Pętla for

Pętlę

`for` zastosujemy, gdy wiemy ile dokładnie razy ma się wykonać dana instrukcja (w końcu stosujemy w niej licznik). Rozważmy dość podstawowy przykład – zsumowanie wszystkich liczb z przedziału od 1 do 100 w PHP:

```
$suma = 0;
for ($i = 1; $i <= 100; $i++) {
    $suma += $i;
}
echo "Suma liczb od 1 do 100 wynosi: " . $suma;
```

Zasada działania samej pętli

`for` jest analogiczna jak w JavaScript – jedynie zmienne zyskały operator `$` – jak to w PHP. W nagłówku pętli najpierw pojawia się wartość startowa `$i = 1`, potem warunek trwania pętli `$i <= 100` – jeżeli chociaż raz okaże się nieprawdziwy, to pętla zakończy swoje działanie. A trzecim zabiegiem jest inkrementacja licznika `$i++` pętli.

Pętla while

Używamy jej, gdy nie znamy z góry liczby powtórzeń pętli – liczba iteracji może na przykład zależeć od tego, ile rekordów zostało wyjętych z bazy danych, od wartości podanej przez użytkownika albo wartości wyznaczonej w danym kroku algorytmu.

Wyobraźmy sobie, że chcemy sumować kolejne liczby naturalne począwszy od 1, dopóki łączna suma tych liczb *nie przekroczy* pewnej zadanej wartości, np. 1000. To taki prosty i klasyczny przykład "logicznego" (czyli zależnego od sprawdzenia warunku) zatrzymania pętli – nie wiemy dokładnie ile iteracji się wykona zanim zsumowane liczby dadzą 1000 lub więcej, ale wiemy, że musimy przestać działać gdy suma przekroczy ten ustalony próg:

```php
<?php
$licznik = 1;
$suma = 0;
$limit = 1000;
while ($suma + $licznik <= $limit) {
    $suma += $licznik;
    $licznik++;
}
echo "Suma najbliższa limitowi wynosi: " . $suma . "<br>";
echo "Ostatnia dodana liczba to: " . ($licznik - 1);
?>
```

Pętla while, która *fetchuje* informacje z bazy danych

Najczęściej w praktyce rozwiązywania zadań INF.03 z pętlą

`while` w PHP spotkamy się podczas *fetchowania* danych (wypisywania informacji wyjętych z bazy). Załóżmy, że tworzymy system ogłoszeń. W bazie danych istnieje tabela `ogloszenia`, która zawiera tytuły i treści dodanych ogłoszeń – są to kolumny o nazwach: `tytul` oraz `tresc`. Chcemy te wszystkie ogłoszenia kolejno wypisać użytkownikowi na podstronie. I tutaj idealnie sprawdzi się pętla `while`, ponieważ nie wiemy ile łącznie ogłoszeń zostanie znalezionych – może 3, a może 30? Wiemy tylko jedno: pętla ma działać dopóki istnieją dane do przetworzenia:

```php
<?php
$connection = mysqli_connect("localhost", "root", "", "baza12345");
if (!$connection) {
    die("Błąd połączenia z bazą!");
}
mysqli_set_charset($connection, "utf8");
$query = "SELECT tytul, tresc FROM ogloszenia";
$wynik = mysqli_query($connection, $query);
while ($row = mysqli_fetch_assoc($wynik)) {
    echo "<h3>" . $row['tytul'] . "</h3>";
    echo "<p>" . $row['tresc'] . "</p>";
}
mysqli_close($connection);
?>
```

Tego typu pętla posiada w nagłówku przypisanie wartości z jednego, bieżącego rekordu do tablicy

`$row`, z której następnie wyjmujemy poszczególne wartości tytułów i treści ogłoszeń dla kolumn z tabeli. Ponieważ tutaj wykorzystano tzw. *fetchowanie asocjacyjne* `mysqli_fetch_assoc()`, czyli nazywanie szuflad tablicy `$row` nazwami słownymi (zamiast ich ponumerowania), to kolejno wypisujemy wartości `$row['tytul']` oraz `$row['tresc']`. Więcej o naturze *fetchowania* oraz różnych metodach jego dokonywania dowiemy się najwięcej się z dedykowanych temu tematowi lekcjach: Fetchowanie danych z użyciem mysqli oraz Fetchowanie danych z bazy w PDO – zależnie od tego, którego z rozszerzeń PHP chcemy używać do komunikacji z bazą danych.

Nie zmienia to jednak faktu, iż *fetchująca* rekordy z bazy danych pętla

`while` (w której w nagłówku jest przypisanie) to absolutny klasyk w kodzie PHP na egzaminach INF.03.

Pętla do..while

Główna różnica w stosunku do pętli

`while` polega na tym, iż warunek sprawdzany jest na końcu bloku pętli, czyli już po wykonaniu iteracji co najmniej jeden raz. Załóżmy, że chcemy napisać w PHP symulację losowania liczby z przedziału od `1` do `49` (trochę jak w polskim *Lotto*) , aż do trafienia wartości uznanej za wygrywającą – na przykład niech naszym "szczęśliwym" numerem będzie wartość `7` (*lucky number seven!*) – jeśli ją wylosujemy, to zakończymy grę:

```
<?php
$szczesliwa = 7; // nasz cel
$licznikProb = 0;
$wylosowana = 0;
do {
    $wylosowana = rand(1, 49); // losujemy liczbę z zakresu 1-49
    $licznikProb++;
    echo "Próba $licznikProb: Wylosowano $wylosowana<br>";
} while ($wylosowana != $szczesliwa);
echo "<strong>Udało się! Trafiliśmy liczbę $szczesliwa po $licznikProb próbach.
</strong>";
?>
```

Zwróćmy uwagę, iż nawet gdyby warunek trwania pętli

`$wylosowana != $szczesliwa` już w pierwszej iteracji okazał się fałszywy (bo wylosowaliśmy faktycznie od razu 7) to akt losowania dokonał się rzeczywiście przynajmniej jeden raz.

Pętla foreach

Podobnie jak to było w JavaScript, konstrukcja językowa

`foreach` świetnie nadaje się do "przeglądania" (mówiąc ogólniej – do uzyskiwania dostępu) do elementów zgromadzonych w tablicach:

```
<?php
$owoce = ["jabłko", "banan", "gruszka"];
foreach ($owoce as $owoc) {
    echo "Owoc pobrany z tablicy: $owoc <br>";
}
?>
```

Wystarczyło tylko wymyślić nazwę jednego elementu tablicy

`$owoce` – tutaj pojedynczą wartość nazwano `$owoc`. A co jeśli oprócz wartości (nazwanej przez nas) potrzebujemy też wartości indeksu elementu w tablicy? Wystarczy zastosować taką składnię:

```
<?php
$owoce = ["jabłko", "banan", "gruszka"];
foreach ($owoce as $indeks => $owoc) {
```

```
    echo "Element numer $indeks: $owoc <br>";
}
```

Oczywiście pamiętajmy, że wartości

`$indeks` rozpoczną się domyślnie od wartości zerowej (bo taki indeks zawsze posiada pierwsza z lewej szuflada tablicy).

Instrukcje wpływające na przebieg pętli: break i continue

Podobnie jak to było w pętlach w JavaScript, także i w PHP możemy wewnątrz pętli użyć instrukcji przerywających. Instrukcja

`break` zrywa działanie całej pętli. Wyobraźmy sobie sytuację, w której szukamy konkretnej osoby w tablicy uczniów zapisanych na egzamin – po znalezieniu kogoś nie ma już sensu kontynuować dalszego porównywania imienia tego uczestnika z pozostałymi ludźmi:

```
<?php
$uczestnicy = ["Kamil", "Anna", "Tomek", "Zofia", "Natalia", "Marek"];
$szukanyUczen = "Zofia";
$znaleziono = false;
foreach ($uczestnicy as $uczen) {
    echo "Sprawdzam: $uczen<br>";

    if ($uczen == $szukanyUczen) {
        echo "<strong>$uczen jest zapisany na egzamin!</strong><br>";
        $znaleziono = true;
        break; // przerywamy dalsze przeszukiwanie tablicy
    }
}
if (!$znaleziono) {
    echo "<strong>$szukanyUczen nie znajduje się na liście uczestników.</strong>";
}
?>
```

W skrócie można powiedzieć, iż jest to mechanika “szukania pierwszego elementu spełniającego warunek” – jedna z klasycznych sytuacji, gdy w razie sukcesu warto pętlę zerwać. Z kolei instrukcja

`continue` pomija tylko bieżącą iterację (*pojedyncze wykonanie bloku kodu*) pętli. Załóżmy, że internauta wypełnił formularz z kilkoma polami (dane kontaktowe), a my chcemy wyświetlić tylko te dane, które zostały faktycznie uzupełnione – ignorując puste:

```
<?php
$dane_uzytkownika = [
"imie" => "Kamil",
"nazwisko" => "",
"email" => "kamil@example.com",
"telefon" => "",
"adres" => "ul. Kwiatowa 12"
];
foreach (dane_uzytkownika as $pole) {
    if (empty($pole)) {
        continue; // pomiń dalsze instrukcje w iteracji, jeśli wartość pola pusta
    }
```

```
    // inne działania przed pokazaniem na ekranie, np. zapis do bazy danych
    echo $pole . "<br>";
}
?>
```

Dzięki instrukcji

`continue` puste pola (nazwisko, telefon) zostaną pominięte przy wkładaniu ich do bazy danych oraz wyświetlaniu na stronie z użyciem `echo`.

Podsumowanie

- W PHP mamy cztery podstawowe rodzaje pętli: `for`, `while`, `do..while` oraz `foreach`.
- Pętli `for` używamy, gdy znamy liczbę iteracji z góry.
- Pętla `while` wykonuje kod dopóki warunek jest spełniony – świetna do obliczeń, które mają zakończyć się po osiągnięciu jakiegoś progu lub przy fetchowaniu danych z bazy.
- Alternatywna `do..while` działa podobnie jak `while`, ale gwarantuje wykonanie kodu przynajmniej raz – przydatne np. przy losowaniu do skutku.
- Konstrukcja `foreach` to najwygodniejsza pętla do przeglądania tablic – zarówno samych wartości, jak i wartości wraz z indeksami.
- Pętle PHP bardzo często wykorzystujemy do *fetchowania* rekordów z bazy danych – klasycznie z użyciem `while ($row = mysqli_fetch_assoc(...))`.
- Instrukcja `break` przerywa całkowicie działanie pętli – np. po znalezieniu pasującego imienia w tablicy uczestników.
- Instrukcja `continue` pomija tylko bieżącą iterację pętli – np. ignorując puste pola z formularza w tablicy danych.
- Pętle to nie tylko “mechanika powtórzeń” – to fundament większości obliczeń, przetwarzań, przeszukiwań i wyświetlania danych w PHP.
- Znajomość pętli to kluczowy punkt każdego egzaminu INF.03 – szczególnie tych fragmentów, które operują na bazach danych i tablicach.
- Pętle mogą być wymagane w walidacji danych lub generowaniu dynamicznej treści (np. wiersze tabel czy *itemy* listy).
- Umiejętność łączenia pętli z warunkami `if` to absolutna podstawa.