Podstawowy skrypt – zmienne, operatory, wypisywanie

Jak zapewne wiemy, języki opisowe: HTML (definiowanie zawartości strony) oraz CSS (opis jej wyglądu) pozwalają na stworzenie tzw. strony statycznej, czyli takiej która zawsze wyświetla tę samą zawartość każdemu użytkownikowi. Może to być np. blog osobisty, wizytówka firmy, portal informacyjny etc.

Lecz co w sytuacji, gdy chcemy stworzyć stronę dynamiczną, czyli w praktyce tzw. aplikację webową – pozwalającą wyświetlić inną zawartość w zależności od tego, kto się do niej zalogował. Jako przykład niech posłużą serwisy *social-media*, fora dyskusyjne, sklepy internetowe, serwisy aukcyjne itd.

Aby móc zrealizować witrynę dynamiczną potrzebujemy między innymi poprawnej obsługi formularzy (możliwość publikowania treści, logowania się, dokonywania zamówień) oraz oczywiście współpracy z bazą danych. I właśnie tutaj do akcji wkraczają języki tzw. *back-endowe* (wykonywane po stronie serwera), a w przypadku egzaminu INF.03 w technikum jest to język programowania PHP.

W niniejszej, pierwszej lekcji przypomnimy fundamenty tworzenia i uruchamiania skryptów po stronie serwerowej.

PHP to język interpretowany po stronie serwera

Jak wspominaliśmy, PHP to język *back-endowy*, co w praktyce oznacza iż:

- kod napisany w języku PHP będzie wykonany nie przez przeglądarkę klienta, lecz przez specjalny program (*interpreter PHP*) zainstalowany na serwerze – współcześnie, w realnej witrynie serwerem HTTP będzie najczęściej Apache lub nginx. W przypadku egzaminu INF.03 oraz lokalnych prac developerskich przed oficjalną publikacją w internecie, kod wykona dla nas interpreter PHP znajdujący się w zainstalowanym pakiecie XAMPP – domyślnie interpreterem PHP jest to w Windows program `php.exe` umieszczony w katalogu `C:\xampp\php`.
- wszystkie linie znajdujące się pomiędzy tagami `<?php` oraz `?>` jako wykonywane przez serwer stanowią "tajemnicę" autora witryny – użytkownik końcowy nie może kodu PHP wprost oglądać w przeglądarce, tak jak może zobaczyć HTML, CSS czy JS. Jedyne co zobaczy internauta odwiedzający witrynę to np. dane wyjęte z bazy danych (imię, nazwisko) – nie dowie się natomiast jakie linie PHP spowodowały nawiązanie połączenia, wykonanie kwerendy czy jakie były *credentials* (login, hasło, nazwa bazy) do usługi MariaDB. Wniosek: kody źródłowe *back-endowe* pozostają (z punktu widzenia klienta serwisu) tajne. Dzięki utajnieniu serwerowych kodów źródłowych nie można więc skopiować całego serwisu internetowego wraz z mechaniką jego działania (co najwyżej jedynie wygląd) – w końcu mechanika działania serwisu to rezultat ciężkiej pracy autorów witryny.

Proces instalacji, konfiguracji oraz instrukcję użycia pakietu XAMPP pobranego z witryny fundacji Apache Friends przedstawiono w tym tutorialu video – na stanowisku egzaminacyjnym oczywiście pakiet XAMPP będzie dla Ciebie już zainstalowany i gotowy do pacy.

Podstawowy skrypt PHP

Na poziomie technikum, kody źródłowe skryptów PHP "mieszają się" z zapisami HTML – jedynie plik przyjmuje rozszerzenie `.php` zamiast zwyczajowego `.html` lub `.htm`. Aby skutecznie powiedzieć interpreterowi PHP które linie należy wykonać (mówiąc kolokwialnie: "odtąd – dotąd") stosujemy tagi: otwierający `<?php` oraz zamykający `?>` – to są wyraźne granice występowania kodu PHP pośród zapisów HTML:

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <title>PHP w HTML</title>
</head>
<body>
    <h1>Witaj na stronie!</h1>
```

```
    <p>
        <?php
            echo "Witaj w świecie PHP!";
        ?>
    </p>
</body>
</html>
```

Instrukcja

`echo` służy w PHP rzecz jasna do wypisywania zawartości na stronie.

Uruchamianie skryptów PHP w XAMPP

W przypadku witryn statycznych, czyli realizowanych tylko w HTML, CSS i JavaScript po stronie klienta (bez PHP i baz danych), wystarczyło jedynie w folderze projektu kliknąć dwukrotnie w plik o nazwie

`index.html` i już można było obserwować stworzoną właśnie stronę główną w przeglądarce.

Natomiast ponieważ do działania PHP potrzebujemy jeszcze interpretera PHP, a nie jedynie samej przeglądarki, to skrypt tym razem już posiadający nazwę

`index.php` trzeba nam na egzaminie “przepuścić” przez serwer Apache. Katalog projektu umieszczamy w specjalnym folderze pakietu XAMPP o nazwie `htdocs` – domyślna lokalizacja to `C:\xampp\htdocs` – są to *ang. hypertext documents*, czyli właśnie dokumenty hipertekstowe przeznaczone do publikacji na tym “symulowanym” (lokalnym) serwerze Apache.

Nazwa katalogu z projektem jednocześnie stanowić będzie część jego adresu w przeglądarce – np. jeśli nasz katalog z projektem to

`C:\xampp\htdocs\sklep` to wówczas w przeglądarce nasz `index.php` znajdujący się przecież w tym katalogu wyświetlimy pod adresem:

```
http://localhost/sklep/index.php
```

Ewentualnie, wyjątkowo – bo dla strony głównej, czyli pliku o nazwie

`index.php` można pominąć nazwę pliku i wpisać jedynie:

```
http://localhost/sklep
```

Klasycznym objawem braku uruchomienia witryny poprzez serwer i katalog

`C:\xampp\htdocs` będzie przeglądarka internetowa wyświetlająca zamiast samej witryny jej kod źródłowy (i to nawet tagi HTML wyświetlą się jako tekst). Dzieje się tak dlatego, iż przeglądarka nigdy nie wykonuje skryptów PHP i stąd otrzymując do otwarcia plik z rozszerzeniem `.php` potraktuje go jako plik tekstowy.

A zatem jeśli na egzaminie widzimy w przeglądarce nasz kod źródłowy (w tym także “tajny” PHP), to zapewne w pasku adresu zamiast zapisu:

```
http://localhost/sklep
```

Umieszczony jest adres automatycznie generowany po otwarciu pliku w przeglądarce dwukrotnym kliknięciem:

```
file:///C:/xampp/htdocs/index.php
```

Otwarliśmy ten konkretny plik (podgląd jego zawartości), a nie naszą stronę wymagającą przecież przejścia przez serwer Apache, czyli w praktyce także interpreter PHP.

Zmienne w PHP, operator $

Język programowania PHP został stworzony przez ***Rasmusa Lerdorfa***. Twórca ten wykorzystał w jego składni mnóstwo elementów znanych już wcześniej w językach *C/C++* oraz *BASH* (skrypty powłoki systemowej Unix/Linux). Co ciekawe – to właśnie zapożyczenia z języka shellowego *BASH* sprawiły, iż w zmiennych tworzonych w PHP pojawia się charakterystyczny operator

`$` zawsze poprzedzający nazwę pojemnika na dane:

```
<?php
    $imie = "Rasmus";
    $nazwisko = "Lerdorf";
    $rok_urodzenia = 1965;
    echo "Imię: " . $imie . "<br>";
    echo "Wiek: " . $wiek . "<br>";
    echo "Rok urodzenia " . $rok_urodzenia . "<br>";
?>
```

W powyższym kodzie źródłowym pojawiło się także od razu wypisywanie zawartości zmiennych – raczej nie stanowi to już dla nas problemu po zajęciach szkolnych, ale przyjrzyjmy się i temu zagadnieniu w powtórce fundamentów, w ramach pierwszej lekcji tego segmentu kursu.

Wypisywanie zmiennych – echo, print, wieloliniowe echo

W kodzie PHP bardzo często musimy wypisać coś na stronie – czy to tekst, wynik działania zmiennej, informacje z bazy danych czy nawet całe fragmenty kodu HTML. W tym celu korzystamy najczęściej z konstrukcji

`echo` albo czasem także `print` . Choć oba zapisy wydają się robić to samo, to istnieją między sobą subtelne różnice, które warto rozumieć!

W praktyce zdecydowanie częściej używamy

`echo`, bo jest krótsze w zapisie, nieco szybsze i pozwala wypisywać więcej niż jeden element naraz:

```
<?php
    echo "Witaj " . $imie . " - twój rok urodzenia: " . $rok_urodzenia . "r." ;
?>
```

Operator kropki oznacza konkatenację, czyli “klejenie” (łączenie ze sobą) łańcuchów. Alternatywnie można też użyć przecinka:

```
<?php
    echo "Witaj " , $imie , " - twój rok urodzenia: " , $rok_urodzenia , "r." ;
?>
```

Natomiast instrukcja

`print` przydaje się rzadziej – głównie tam, gdzie potrzebujemy wypis i wartość zwrotną jednocześnie. O co tutaj chodzi? Otóż `print` zwraca dodatkowo wartość `1`, więc możemy go użyć np. w warunkach:

```php
<?php
    $czyPokazac = true; //wartość zależna np. od udanego zalogowania
    if ($czyPokazac && print("Witaj $imie")) {
        echo " - Udało się poprawnie zalogować i przywitać użytkownika";
    }
?>
```

Mówiąc najprościej – udane wypisanie tekstu komendą

`print` dodatkowo zwraca wartość `1`, co odpowiada też wartości `true` – stąd można to sprawdzać. Oczywiście to nie jest codzienna praktyka, a raczej podręczna sztuczka podczas debugowania czy sprawdzenia działania instrukcji warunkowej czy iteracji w pętli.

Natomiast bardzo często w zadaniach egzaminacyjnych lub w realizowanych projektach webowych musimy wypisać tekst obudowany znacznikami HTML z poziomu PHP – i to zazwyczaj mieści się w co najmniej kilku linijkach. Najprościej użyć wówczas tzw. wieloliniowego

`echo` – lub inaczej można powiedzieć ładnie: `echo` *z* wykorzystaniem ***heredoc***. Etykieta – tutaj o nazwie `LISTA` (może to być też inna nazwa) ładnie ukazuje “odkąd – dokąd” obowiązuje zakres wypisywania:

```php
<?php
$technologia1 = 'HTML';
$technologia2 = 'CSS';
$technologia3 = 'JavaScript';
echo <<<LISTA
<ul>
   <li>$technologia1</li>
   <li>$technologia2</li>
   <li>$technologia3</li>
</ul>
LISTA
?>
```

Kod źródłowy staje się po prostu czytelniejszy, aniżeli zrealizowany pojedynczymi instrukcjami

`echo` – porównajmy:

```php
<?php
$technologia1 = 'HTML';
$technologia2 = 'CSS';
$technologia3 = 'JavaScript';
echo "<ul>";
echo "<li>$technologia1</li>";
echo "<li>$technologia2</li>";
echo "<li>$technologia3</li>";
echo "</ul>";
?>
```

Nie widać już tak pięknie “czystego” HTML oraz łatwiej nam popełnić literówkę np. poprzez brak domknięcia cudzysłowu albo przeoczenie któregoś średnika.

Podejrzenie tablicy lub obiektu – rekursywny print_r()

Przydatną, roboczą sztuczką podczas kodowania w PHP jest użycie specjalnej funkcji

`print_r()`. Gdy chcemy szybko zobaczyć (podejrzeć w ramach debugowania), co tak naprawdę zawiera nasza tablica lub obiekt, to z pomocą przychodzi bardzo przydatna funkcja diagnostyczna:

```
<?php
    $owoce = ["jabłko", "banan", "gruszka"];
    print_r($owoce);
?>
```

Ta funkcja pozwala nam w czytelny sposób wyświetlić zawartość złożoną – działa w końcu rekursywnie, czyli w sposób powtarzalny. Wynikiem działania w przeglądarce będzie:

```
Array ( [0] => jabłko [1] => banan [2] => gruszka )
```

Od razu sprawdzamy, czy nasza tablica rzeczywiście poprawnie przechowuje wartości – może się to przydać np. w sytuacji gdy tablicę taką wypełniamy informacjami wyjętymi z bazy danych – jedna linijka PHP szybko pozwoli nam ustalić czy dane trafiły do tablicy.

Podstawowe operatory i funkcja sqrt()

Przypomnijmy teraz jeszcze gwoli formalności najważniejsze, podstawowe operatory i operacje matematyczne bardzo często wykorzystywane w PHP:

```
<?php
$a = 10;
$b = 5;
echo "Dodawanie: " . ($a + $b) . "<br>";
echo "Odejmowanie: " . ($a - $b) . "<br>";
echo "Mnożenie: " . ($a * $b) . "<br>";
echo "Dzielenie: " . ($a / $b) . "<br>";
echo "Reszta z dzielenia: " . ($a % $b) . "<br>";
echo "Potęgowanie: " . ($a ** $b) . "<br>";
echo "Pierwiastek kwadratowy z a: " . sqrt($a) . "<br>";
// Konkatenacja (łączenie ciągów tekstowych)
$imie = "Rasmus";
$nazwisko = "Lerdorf";
echo "Pełne imię i nazwisko: " . $imie . " " . $nazwisko . "<br>";
?>
```

Dlaczego średniki są w PHP koniecznie potrzebne?

Gdy programujemy w języku PHP, każde polecenie koniecznie musi kończyć się średnikiem

`;`. Jest to jedna z podstawowych reguł składni tego języka, którą po prostu musimy zapamiętać i stosować. A czasem – zwłaszcza po dłuższym czasie spędzonym w JavaScript, gdzie średniki nie są koniecznie potrzebne do poprawnego wykonania kodu – możemy zwyczajnie o tym zapominać.

Średnik mówi interpreterowi PHP coś w stylu: *To polecenie zostało zakończone – teraz możesz przejść do następnego*. Jeśli więc zapomnimy postawić średnik, to PHP nie będzie wiedziało, gdzie kończy się jedna instrukcja, a zaczyna następna i zobaczymy błąd składni (*ang. syntax error*). Na przykład w tej sytuacji:

```
<?php
```

```
    $imie = "Kasia"
    echo "Witaj, $imie!";
?>
```

Otrzymamy błąd: *Parse error: syntax error, unexpected 'echo'…* PHP nie wie, że poprzednia linia już się zakończyła – i wciąż "myśli", że wstawione

`echo` to ciąg dalszy wcześniejszego polecenia definiowania zmiennej, co prowadzi do błędu składni z komunikatem: *nieoczekiwane 'echo'…*

Podsumowanie – co zapamiętać?

Powtórka absolutnych fundamentów zakończona! Pora na kilka wniosków:

- PHP pozwala na generowanie tzw. dynamicznych stron (aplikacji) internetowych.
- Kod PHP umieszczamy wewnątrz `<?php ... ?>` bo inaczej interpreter nie będzie mógł stwierdzić gdzie zaczyna i kończy się kod PHP.
- Zmienne zaczynamy od operatora `$` (tradycja z języka BASH).
- Dane wyświetlamy najczęściej za pomocą `echo`, rzadziej `print` oraz w przypadku konieczności obudowania tekstów znacznikami HTML także wieloliniowej wersji `echo`.
- Przydatną, roboczą sztuczką służącą do szybkiego wyświetlenia tablicy czy obiektu jest użycie funkcji `print_r()`.
- Klasyczne operatory pozwalają wykonywać obliczenia oraz łączyć łańcuchy (tzw. konkatenacja). Operatorem konkatenacji jest w PHP kropka `.`, a nie tak jak to było w JavaScript znak plusa `+`.
- Średniki są koniecznie potrzebne w PHP – ich brak spowoduje błąd składni (*ang. syntax error*).