# Aplikacje internetowe

## *Laboratorium 7*

**Cele**:

- Podstawy języka PHP

***Tutoriale***
**http://php.net/**
**http://phpkurs.pl/**

**Zadanie 1**

Chcąc połączyć kod PHP z kodem HTML, należy bezwzględnie pamiętać, że musi on być oddzielony od HTML za pomocą znaczników '<?php ?>' np. <?php //kod ?>. Prostym przykładem może być skrypt pokazujący informacje o konfiguracji PHP na serwerze za pomocą funkcji phpinfo().

```
przykład 1. (phpinfo.php)
<HTML>
      <HEAD>
            <TITLE> PHP INFO </TITLE>
      </HEAD>
      <BODY>
            <?php phpinfo(); ?>
      </BODY>
</HTML>
```

**Zadanie 2**

```
przykład 2. Łączenie PHP i HTML
<?php
      $tekst = ”to jest przykładowy tekst”;
      echo ”to jest tekst przykładowy nr2”;
      echo $tekst;
?>
<HTML>
      <HEAD>
            <TITLE> Łączenie kodu PHP i HTML </TITLE>
      </HEAD>
      <BODY>
      </BODY>
</HTML>
```

Lub osadzony w kodzie HTML jak w przykładzie 3.

```
przykład 3. Łączenie PHP i HTML (zagnieżdżanie)
<HTML>
<HEAD>
      <TITLE> Laczenie kodu PHP i HTML </TITLE>
</HEAD>
<BODY>
<?php
      $tekst = ”to jest przykładowy tekst”;
      echo ”to jest tekst przykładowy nr2”;
      echo $tekst;
?>
</BODY></HTML>
```

Kody z obu przykładów są równoważne, to znaczy, że wynik ich działania jest dla oglądającego taki sam.

Bardzo ważną sprawą oprócz kończenia poleceń średnikiem jest odwoływanie za pomocą znaków ucieczki takich znaków jak cudzysłowy. Gdy znaki cudzysłowu znajdują się wewnątrz innej pary znaków cudzysłowu, to parę wewnętrzną należy oddzielić od zewnętrznej znakiem ucieczki (\).

```
<?php echo ”<p> W epopei ”Pan Tadeusz” Adam Mickiewicz …”; - źle
<?php echo ”<p> W epopei \”Pan Tadeusz\” Adam Mickiewicz …”; - dobrze
```

**Zadanie 3**

Wstawianie plików poleceniem **include**.

Polecenie **include** pozwala na wstawienie (dosłowne) w danym miejscu pliku innego pliku wskazanego ścieżką (w lokalnym systemie plików). Polecenie to pozwala na tworzenie dynamicznej zawartości pliku wynikowego w zależności od wyboru użytkownika, co można wykorzystać przy tworzeniu systemów portalowych (patrz **przykład 3**). Stosowanie tego polecenia pozwala zmniejszyć ilość kodu strony oraz przyspiesza jej tworzenie.

**przykład 3.**

```
<!—plik index.php -->
<HTML>
<BODY>
      <?php include(”inc/menu.html”); // wstawienie pliku z zawartością menu ?>
</BODY>
</HTML>

<!—plik menu.html w katalogu inc-->
<HTML>
<BODY>
      <a href=”index.php?page=cv”> moje cv </a>
      <a href=”http://moja.strona.pl/info/ankieta.html”> ankieta </a>
</BODY>
</HTML>
```

**Zadanie 4**

Wykorzystanie zmiennych i operatorów.

Istnieje możliwość przekazywania zmiennych do skryptów (ustawiania tych zmiennych przy wywoływaniu skryptu). Daje to możliwość dynamicznego sterowania zawartością strony. Zilustruje to poniższy przykład.

**przykład 4.** (wykorzystanie zmiennych)

```
<!—plik menu.html -->
<HTML>
<BODY>
      <A href=”index.php?id=buty”> Buty </A><br>
      <A href=”index.php?id=kurtki”> Kurtki </A><br>
      <A href=”index.php?id=spodnie”> Spodnie </A><br>
</BODY>
</HTML>

<!-- plik index.php -->
<HTML>
<BODY>
<?php
```

```
include(”menu.html”);
      if (isset($id)) echo ”<br><strong>$id</strong>”;
      else echo “<br> <font color=\”red\”> Wybierz coś ! </font>”;
?>
<center><a href="index.php"> powrót do strony głównej </a></center>
</BODY>
</HTML>
```

W powyższym przykładzie wybór opcji w pliku menu powoduje odmienne zachowanie się skryptu w pliku **index.php**. W instrukcji **if** wykorzystano wewnętrzną funkcję PHP **isset($zmienna)** w celu określenia czy zmienna **$id** występuje w skrypcie. Brak takiej zmiennej świadczy o wejściu na stronę **index.php** bez przejścia przez **menu.html**. W przypadku wejścia przez menu w skrypcie **index.php** zmienna **$id** będzie już zdeklarowana i będzie miała jedną z trzech wartości. Dzięki podaniu adresu w postaci **index.php?id=buty** automatycznie w skrypcie **index.php** będzie widoczna zmienna id przechowująca wartość łańcucha „buty”. Połączenie tego mechanizmu z poleceniem **include** oraz instrukcją **switch** lub **if ... elseif ... else** umożliwia stworzenie elastycznego serwisu www z przełączanymi podstronami.

**Zadanie 5**

Struktury sterujące

**Konstrukcja if … elseif … else**

Konstrukcja ta wykonuje instrukcje na podstawie wartości testowanego wyrażenia.

**przykład 5a.** (instrukcja warunkowa)

```
<?php
if ( $a == ”10” )
      {
            // TRUE wykonaj ten kod
            echo ”a= 10”;
      }// ewentualnie
      else {
            // FALSE wykonaj ten kod
            echo ”a nie równe 10 ”;
      }
?>
```

Po sprawdzaniu zmiennej **$a**, jeśli będzie ona miała wartość **10** (warunek będzie miał wartość **TRUE**) zostanie wykonany kod **TRUE** w przeciwnym wypadku kod **FALSE**. Blok **else** jest opcjonalny, natomiast w warunku możemy określić dowolną kombinację logiczną różnych warunków. Drugim wariantem instrukcji **if** jest instrukcja **if … elseif … else**, której przykład przedstawiono poniżej.

**przykład 5b.**

```
<?php
if ( $a == ”10” )
      {
            // TRUE wykonaj ten kod
            echo ”a = 10”;
      }// ewentualnie
elseif ( $b == ”8” ) {
            // TRUE drugi warunek, wykonaj ten kod
            echo ”b = 8”;
      }else{
```

```
            // FALSE drugi warunek.
            echo ”a nie równe 10 i b nie równe 8”;
}
?>
```

**Pętla while**

Inaczej niż w przypadku instrukcji **if … elseif … else**, w której każde wyrażenie jest oceniane raz i na jego podstawie wykonywana jest akcja, instrukcja **while** kontynuuje pętlę aż warunek osiągnie wartość **FALSE**. Innymi słowy pętla jest kontynuowana dopóki warunek ma wartość **TRUE**.

**przykład 5c.** (pętla while)

```
<?php
$a = 0;
while ( $a <=10 )
      {
            echo ”a równa się $a<br>”;
            $a++;
      }
?>
```

W powyższym przykładzie zmienna **$a** wyświetlana jest na ekranie do momentu, kiedy jej wartość nie przekroczy wartości **10**. Warto jednak pamiętać, że przypadkowy błąd może spowodować nieskończone działanie pętli. (co stanie się gdy zamiast **$a++** napiszemy **$a--** ?).

**Petla for**

Pętla **for** tym różni się od pętli **while** że musimy znać dokładną ilość wykonań pętli. Poniżej przedstawiono przykładową składnię takiej pętli.

**przykład 5d.** (pętla for)

```
<?php
$max_i = 25;
for ($i=0;$i<$max_i;$i++)
      {
            echo ”i ma wartość $i”;
      }
?>
```

Zamiast **$i++**, możemy wpisać **$i+=1** lub **$i+=2** itd.

**Instrukcja Switch**

Instrukcja **switch** pozwala na sterowanie przebiegiem programu poprzez wybór właściwego fragmentu kodu.

**przykład 5e.** (instrukcja switch)

```
<?php
$wariant = 0; //(lub 1,2 itd.) // gdzieś w programie, najczęściej wybór użytkownika
switch ( $wariant )
      {
      case 0:
      echo ”wybrałeś wariant $wariant”;
      break;
      case 1:
      echo ”wybrałeś wariant $wariant”;
      break;
      ….
      default: // wariant domyślny gdy wartość zmiennej nie ma swojego case’a
      echo ”użyto wariantu domyślnego”;
```

```
      break;
      }
?>
```

**Zadanie 6**

Kolejnym krokiem będzie przejęcie danych z prostego formularza i wyliczenie sumy liczb. Utwórz plik **zad06.php** i umieść w nim poniższy kod.

**przykład 6.**

```
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
</head>
<body>
 <form action="zad06.php" method="GET">
 Podaj wartość x: <input type="text" name="x"/>
 <input type="submit" value="wyślij"/>
 <input type="reset" value="wyczyść"/>
 </form>
 <?php
 $x = $_GET['x'];
 $suma = 0;
 for ($i=0; $i<$x; $i++)
 $suma += $i;
 echo "suma liczb od 1 do ".$x." wynosi: ".$suma;
 ?>
</body>
</html>
```

Kod w pliku **zad06.php** jest niepoprawny, ponieważ wykorzystuje wartość zmiennej przekazanej przez użytkownika bez jakiejkolwiek walidacji poprawności. Zmodyfikuj główny fragment skryptu PHP w poniższy sposób i przetestuj poprawność rozwiązania. Czy potrafisz wykonać to samo obliczenie bez użycia pętli?

```
<?php
 $x = $_GET['x'];
 if (is_numeric($x))
 {
 $suma = 0;
 for ($i=0; $i<$x; $i++)
 $suma += $i;
 echo "suma liczb od 1 do ".$x." wynosi: ".$suma;
 }
 else
 echo "<div style=\"color: red\">proszę podać poprawną liczbę</div>";
?>
```

W następnym kroku zaimplementujemy algorytm Euklidesa i wykorzystamy do walidacji poprawności danych mechanizm filtrów. Zwróć uwagę na sposób walidacji. Utwórz plik **zad06a.php** i umieść w nim poniższy kod

**przykład 6a**

```
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
</head>
<body>
```

```
 <form action="zad06a.php" method="GET">
 Podaj wartość x: <input type="text" name="x"/>
 Podaj wartość y: <input type="text" name="y"/>
 <input type="submit" value="wyślij"/>
 <input type="reset" value="wyczyść"/>
 </form>
 <?php
 $int_options = array("options" =>
 array("min_range" => 1,"max_range" => 1000000));

 $x = filter_var($_GET['x'], FILTER_VALIDATE_INT, $int_options);
 $y = filter_var($_GET['y'], FILTER_VALIDATE_INT, $int_options);
 if ($x && $y)
 {
 while ($x != $y)
 {
 if ($x > $y)
 $x -= $y;
 else
 $y -= $x;
 }
 echo "największy wspólny dzielnik tych liczb to ".$x;
 }
 else
 echo "<div style=\"color: red\">podaj poprawne liczby</div>";
 ?>
</body>
</html>
```

Podobnie jak w przypadku języka JavaScript, jednym z najwygodniejszych mechanizmów walidacji danych wejściowych są wyrażenia regularne. Poniższy przykład pokazuje, w jaki sposób można wykorzystać język wyrażeń regularnych do przeprowadzenia walidacji formatu karty kredytowej. Numer karty kredytowej to cztery grupy czterocyfrowe opcjonalnie rozdzielone spacją lub myślnikiem, a data ważności to dwie grupy dwucyfrowe rozdzielone znakiem '/'. Utwórz plik **`zad06b.php`** i umieść w nim poniższy kod.

**przykład 6b.**

```
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
</head>
<body>
 <form action="zad06b.php" method="GET">
 Podaj numer karty:
 <input type="text" maxlength="19" size="19" name="numer_karty" />
 Podaj datę ważności:
 <input type="text" maxlength="5" size="5" name="data_waznosci" />
 <input type="submit" value="wyślij"/>
 <input type="reset" value="wyczyść"/>
 </form>
 <?php
 $numer_karty = $_GET['numer_karty'];
 $data_waznosci = $_GET['data_waznosci'];
 if (!preg_match("/^(\d{4}[\s\-]?){4}$/",$numer_karty))
 echo "<div style=\"color: red\">podaj poprawny numer karty</div>";
 elseif (!preg_match("/^\d{2}\/\d{2}$/",$data_waznosci))
```

```
 echo "<div style=\"color: red\">podaj poprawną datę</div>";
 else
 echo "podano poprawny numer karty kredytowej";
 ?>
</body>
</html>
```

Kod w pliku **zad06b.php** ma dwie wady. Po pierwsze, wyrażenie regularne zaakceptuje łańcuch 16 cyfr zakończony spacją. Zmodyfikuj wyrażenie regularne w taki sposób, aby na końcu numeru karty kredytowej nie mogła się pojawić spacja. Po drugie, w przypadku popełnienia błędu zawartość formularza jest tracona – stanowi to bardzo poważną niedogodność dla użytkownika. Zmodyfikuj kod tworzący formularz HTML w taki sposób, aby nie tracić wartości wprowadzanych do pól formularza.

```
<form action="zad06b.php" method="GET">
 Podaj numer karty:
 <input type="text" maxlength="19" size="19" name="numer_karty"
 value="<?php echo $_GET['numer_karty']; ?>"/>
 Podaj datę ważności:
 <input type="text" maxlength="5" size="5" name="data_waznosci"
 value="<?php echo $_GET['data_waznosci']; ?>"/>
 <input type="submit" value="wyślij"/>
 <input type="reset" value="wyczyść"/>
</form>
```