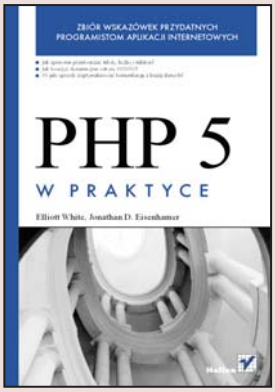
PHP 5 w praktyce



Autorzy: Elliott White, Jonathan D. Eisenhamer
Tłumaczenie: Piotr Gaczkowski,
Radosław Meryk, Piotr Pietrzak
ISBN: 978-83-246-0814-0
Tytuł oryginału: PHP 5 in Practice
Format: B5, stron: 432

Poznaj potęgę języka PHP 5

Chcesz poprawić działanie swoich aplikacji internetowych, jednocześnie wykorzystując pełnię możliwości oferowanych przez PHP 5? A może w projekcie natknąłeś się na problem i szukasz rozwiązania? Najnowsza wersja PHP to w pełni obiektowy język programowania, umożliwiający tworzenie rozbudowanych i stabilnych aplikacji internetowych. Dostępny bezpłatnie PHP 5 jest podstawowym narzędziem wielu programistów na całym świecie, ceniących jego prostą składnię, ogromny zakres zastosowań i szybkość działania.

„PHP 5 w praktyce" to zbiór porad, dzięki którym utworzysz ciekawe, wydajne i szybko działające aplikacje internetowe i witryny WWW. Znajdziesz tu rozwiązania najczęściej spotykanych zadań programistycznych oraz wskazówki dotyczące stosowania poszczególnych elementów języka PHP 5. Przeczytasz o przetwarzaniu tekstów, liczb i dat, programowaniu obiektowym, połączeniach z bazami danych, tworzeniu elementów interfejsu użytkownika oraz operacjach na plikach i katalogach. Dowiesz się także, jak korzystać z plików XML, przetwarzać grafikę i usuwać błędy z aplikacji PHP.

Dzięki książce poznasz:

- Przetwarzanie danych tekstowych i liczbowych
- Operacje na tablicach
- Stosowanie funkcji
- Praca z systemem plików
- Tworzenie elementów witryn WWW
- Obsługa formularzy HTML
- Sesje i pliki cookie
- Wysyłanie poczty elektronicznej
- Komunikacja z bazami danych
- Przetwarzanie plików XML
- Generowanie grafiki
- Szyfrowanie danych

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

3

Godziny i daty

APLIKACJE, KTÓRE WYKONUJĄ operacje związane z czasem, są szczególnie często używane w internecie. Godziny przesłania formularza, dane wprowadzane przez użytkowników, na przykład data urodzenia, a także aktualizacja i usuwanie stron w przypadku, gdy są przestarzałe — to zaledwie kilka przykładów. Chociaż obliczenia dotyczące dat w PHP mogą wydawać się skomplikowane, w rzeczywistości są dość proste.

W tym rozdziale głównie opisano podstawowe funkcje przetwarzania dat i godzin. Funkcje te zapewniają wykonywanie wszystkich podstawowych operacji na datach i godzinach, takie jak pobieranie, formatowanie i konwersja dat. W PHP jest dostępne także inne rozszerzenie — *Calendar*, które służy do konwersji między różnymi systemami kalendarzy, na przykład między kalendarzem gregoriańskim i żydowskim. Są to na przykład funkcje `cal_to_jd()` i `cal_from_jd()` opisane w punkcie „Szybkie wskazówki".

Zagadnienie planowania zadań — na przykład tworzenie harmonogramu spotkań i zdarzeń — wykracza poza zakres tego rozdziału. Własności te są dostępne dzięki rozszerzeniu *MCAL*. Więcej informacji na ten temat można znaleźć pod adresem *http://php.net/mcal*.

Aby w pełni wykorzystać możliwości przetwarzania dat i godzin w PHP, trzeba wiedzieć, w jaki sposób w środowisku PHP mierzy się czas. Interpreter PHP odlicza czas w postaci liczby sekund, jakie upłynęły od północy 1 stycznia 1970 roku — tzw. epoki Unix. W związku z tym każdy moment w czasie jest zapamiętywany w postaci prostej liczby typu `integer`. Czas zapisany w ten sposób to tzw. **znacznik czasu** (ang. *time stamp*). Jak się przekonamy, wiele funkcji albo zwraca znacznik czasu albo wykorzystuje go jako argument. Należy jednak zachować ostrożność w przypadku czytania dokumentacji lub komentarzy w kodzie. W zależności od kontekstu, termin *znacznik czasu* może być używany także w sensie bardziej ogólnym: jako czas, kiedy coś się wydarzyło. Gdy mowa na przykład o plikach dzienników serwera WWW, terminem *znacznik czasu* określa się czas, w którym zarejestrowano zdarzenie.

Szybkie wskazówki

- **Odczytanie znacznika czasu dla bieżącej godziny:**

  ```
  $timestamp = time();
  ```

  Funkcja nie pobiera parametrów. Zwraca bieżący czas, tzn. czas, w którym wywołano funkcję `time()`.

  Pełna dokumentacja: *http://php.net/time*.

- **Odczytanie bieżącego znacznika czasu z dokładnością do mikrosekund:**

  ```
  $result = microtime($format);
  ```

  Funkcja zwraca uniksowy znacznik czasu z dokładnością do mikrosekund. Jeśli argument `$format` ma wartość `false` (domyślnie), funkcja zwraca czas w postaci ciągu znaków `"sekundy mikrosekundy"`, gdzie sekundy mają tę samą wartość, co zwrócona przez funkcję `time()`, a mikrosekundy oznaczają ułamek sekund. Jeśli argument `$format` ma wartość `true`, funkcja zwraca czas w postaci liczby zmiennoprzecinkowej.

  Pełna dokumentacja: *http://php.net/microtime*.

- **Utworzenie znacznika czasu dla określonej daty i godziny:**

  ```
  $timestamp = mktime($godzina, $minuta, $sekunda, $miesiac, $dzien,
  $rok);
  ```

  Zwraca znacznik czasu odpowiadający określonej dacie i godzinie.

  Pełna dokumentacja: *http://php.net/mktime*.

- **Formatowanie określonego czasu:**

  ```
  $string = date($format, $timestamp);
  ```

  Zwraca znakową reprezentację podanego znacznika czasu z wykorzystaniem zdefiniowanego formatu. Jeśli nie określi się argumentu `$timestamp`, funkcja wykorzystuje bieżący czas.

  Pełna dokumentacja: *http://php.net/date*.

- **Sformatowanie daty i godziny. Dostosowanie wyników do formatu Greenwich Mean Time (GMT):**

  ```
  $string = gmdate ($format, $timestamp);
  ```

  Funkcja równoważna do `date()`. Jednak czas jest przedstawiany jako reprezentacja GMT podanej godziny.

  Pełna dokumentacja: *http://php.net/gmdate*.

- **Odczytanie różnych informacji na temat podanego czasu:**

  ```
  $time_array = getdate($timestamp);
  ```

  Zwraca tablicę różnych wartości dotyczących podanego znacznika czasu. Indeksami tablicy są `seconds`, `minutes`, `hours`, `mday`, `wday`, `mon`, `year`, `yday`, `weekday` i `month`.

  Pełna dokumentacja: *http://php.net/getdate*.

- **Odczytywanie i ustawianie strefy czasowej używanej w funkcjach czasowych:**

  ```
  $string = date_default_timezone_get();
  date_default_timezone_set($string);
  ```

Powyższa para funkcji służy do odczytywania (ustawiania) identyfikatora strefy czasowej.

Pełna dokumentacja: *http://php.net/date_default_timezone_set* i *http://php.net/date_default_timezone_get*.

- **Zamiana czasu z postaci tekstowej w języku angielskim na znacznik czasu:**

```
$timestamp = strtotime($english_time);
```

Funkcja pobiera tekstowy opis daty i godziny w języku angielskim i zwraca odpowiadający jej znacznik czasu.

Pełna dokumentacja: *http://php.net/strtotime*.

- **Sprawdzenie poprawności daty według kalendarza gregoriańskiego:**

```
$isvalid = checkdate($month, $day, $year);
```

Zwraca `true`, jeśli określoną datę można znaleźć w kalendarzu gregoriańskim. Przydaje się do sprawdzania, czy czas wprowadzany przez użytkownika bądź czas wyliczony są poprawne.

Pełna dokumentacja: *http://php.net/checkdate*.

- **Godzina wschodu i zachodu słońca we wskazanej lokalizacji:**

```
$time_rise = date_sunrise($day_ts, SUNFUNCS_RET_TIMESTAMP, $latitude,
$longitude);
$time_set = date_sunset($day_ts, SUNFUNCS_RET_TIMESTAMP, $latitude,
$longitude);
```

Powyższa para funkcji zwraca godziny wschodu i zachodu słońca dla wskazanej lokalizacji. Użycie stałej `SUNFUNCS_RET_TIMESTAMP` powoduje, że funkcja zwraca godzinę w postaci znacznika czasu.

Pełna dokumentacja: *http://php.net/date.sunrise* i *http://php.net/date.sunset*.

- **Konwersja daty między różnymi kalendarzami a kalendarzem juliańskim:**

```
$jday count = cal_to_jd($calendar, $rnonth, $day, $year);
$date_array = cal_from_jd($jday_count, $calendar);
```

Powyższa para funkcji umożliwia konwersję daty wyrażonej w określonym systemie kalendarza na kalendarz juliański — i odwrotnie. Funkcje obsługują następujące typy kalendarzy: `CAL_GREGORIAN`, `CAL_JULIAN`, `CAL_JEWISH` oraz `CAL_FRENCH`.

Pełna dokumentacja: *http://php.net/cal_to_jd* i *http://php.net/cal_from_jd*.

3.1. Obliczanie różnicy czasu między dwiema datami

W programach bardzo często trzeba znaleźć liczbę tygodni, dni, a nawet sekund dzielących dwie daty. W niektórych systemach baz danych są dostępne procedury umożliwiające porównywanie dat i wykonywanie działań arytmetycznych na datach. Jednak w PHP takie operacje trzeba wykonywać samodzielnie.

Jak powiedziano we wprowadzeniu, w PHP daty są reprezentowane w formacie epoki UNIX. Oznacza to, że są zapisywane w postaci liczby sekund, jaka upłynęła od północy 1 stycznia 1970 roku. Dzięki temu można z łatwością obliczyć liczbę sekund, jaka upłynęła między dwoma punktami w czasie. Wystarczy odjąć od siebie dwa znaczniki czasu. Wykonanie popularniejszego działania — obliczenie liczby dni — wymaga jednak dodatkowych czynności.

Podczas działań arytmetycznych na datach trzeba pamiętać o latach przestępnych, miesiącach o różnej liczbie dni, miesiącach obejmujących pięć tygodni, latach z 53. tygodniem, zmianie czasu z letniego na zimowy i innych tego rodzaju zagadnieniach. W przeciwnym razie algorytm będzie obsługiwał jedynie większość przypadków — nie wszystkie.

Rozpocznijmy od omówienia częstego problemu wyznaczenia liczby dni dzielących dwie daty. Zwróćmy uwagę, że w kodzie z listingu 3.1.1 do utworzenia znaczników czasu wykorzystano funkcję `strtotime()`. Funkcja umożliwia pobranie wielu różnych postaci tekstowej reprezentacji daty i przekształcenie jej na znacznik czasu.

> **Uwaga**
>
> Począwszy od wydania PHP w wersji 5.1, przed wywołaniem jakiejkolwiek funkcji operacji na datach trzeba wywołać funkcję `date_default_timezone_set()`, która służy do określenia strefy czasowej, dla jakiej mają być wykonane obliczenia. W innym przypadku generowany jest błąd trybu `STRICT`, ponieważ interpreter próbuje odgadnąć strefę czasową. Więcej informacji na temat stref czasowych można znaleźć w podrozdziale 3.4, „Obsługa stref czasowych" w dalszej części tego rozdziału. Jeśli tworzony kod ma działać zarówno w środowisku PHP 5.0, jak i PHP 5.1, powstaje problem, ponieważ w PHP 5.0 funkcja `date_default_Timezone_set()` jest niedostępna. Z tego względu można użyć funkcji `function_exists()` i za jej pomocą sprawdzić, czy funkcja `date_default_Timezone_set()` istnieje. Jeśli tak, można ją wywołać. Alternatywnie można ustawić domyślną strefę czasową w pliku konfiguracyjnym na serwerze PHP 5.1. W takiej sytuacji nie trzeba się martwić ustawieniami strefy czasowej.

Listing 3.1.1. **Liczba dni między dwoma datami**

```
<?php
// Ustawienie domyślnej strefy czasowej na Europe/Warsaw.
date_default_timezone_set('Europe/Warsaw');

// Funkcja zwraca liczbę dni między dwoma przekazanymi datami.
function count_days($a, $b) {
    // Najpierw należy przekształcić daty na części składowe:
    $a_dt = getdate($a);
    $b_dt = getdate($b);
```

```
    // Odtworzenie znaczników czasu z wykorzystaniem południa każdego dnia.
    // Dokładny czas nie ma znaczenia, ale musi to być ta sama godzina w każdym dniu.
    $a_new = mktime(12, 0, 0, $a_dt['mon'], $a_dt['mday'], $a_dt['year']);
    $b_new = mktime(12, 0, 0, $b_dt['mon'], $b_dt['mday'], $b_dt['year']);

    // Odjęcie dwóch liczb i podzielenie przez liczbę sekund przypadającą na dzień.
    // Zaokrąglenie wyniku, ponieważ przekroczenie granicy zmiany czasu z letniego na zimowy (lub odwrotnie)
    // spowoduje różnicę w czasie o godzinę lub dwie.
    return round(abs($a_new - $b_new) / 86400);
}

// Przygotowanie kilku dat
$date1 = strtotime('12/3/1973 8:13am');
$date2 = strtotime('1/15/1974 10:15pm');
$date3 = strtotime('2/14/2005 1:32pm');

// Obliczenie różnic. Powinniśmy uzyskać wyniki 43 i 11353
echo "<p>Upłynęło ", count_days($date1, $date2), " dni.</p>";
echo "<p>Upłynęło ", count_days($date2, $date3), " dni.</p>";
?>
```

Problem z pozoru wydaje się prosty: odjęcie dwóch liczb i podzielenie przez liczbę sekund przypadającą na dzień. To prawda i w ten sposób uzyskamy prawidłowe wyniki, jeśli interesuje nas ściśle matematyczna definicja dnia. Weźmy jednak za przykład dwie daty: 11 listopada, godz. 23.00 i 12 listopada, godz. 1.00. Zgodnie z ogólnie przyjętą interpretacją dat, różni je jeden dzień. Jednak jeśli odejmiemy od siebie znaczniki czasowe odpowiadające obu datom, okaże się, że wymienione daty dzielą tylko dwie godziny.

Rozwiązanie polega na sprowadzeniu dat do wspólnej godziny — na przykład południa. Teraz można odjąć od siebie znaczniki czasu. Aby kolejność dat nie miała znaczenia, można posłużyć się wartością bezwzględną. Aby otrzymać liczbę dni, uzyskany wynik należy podzielić przez liczbę sekund przypadającą na dzień (86 400). Byłoby to niemal doskonałe rozwiązanie, gdyby nie to, że nie uwzględnia zmiany czasu z letniego na zimowy bądź odwrotnie. Jeśli między datami, dla których wykonujemy obliczenia, nastąpiła zmiana czasu, dokładny wynik może się różnić o jedną lub dwie godziny. Aby pozbyć się błędu spowodowanego zmianą czasu, wystarczy zaokrąglić wynik do najbliższej liczby całkowitej.

Podobną strategię można również wykorzystać do obliczenia liczby tygodni, miesięcy i lat dzielących dwie daty. Spotyka się jednak dwa różne punkty widzenia na te większe jednostki czasu. Na przykład mówimy czasami, że coś miało miejsce tydzień temu, mając na myśli siedem dni. W takim przypadku, by obliczyć liczbę tygodni między dwiema datami, wystarczy skorzystać z utworzonej przed chwilą funkcji `count_days()` i podzielić uzyskany wynik przez 7. Innym razem pod tym samym pojęciem rozumie się zdarzenie, które miało miejsce w ubiegłym tygodniu. W takiej sytuacji należy skorzystać z podobnego mechanizmu, jak w przypadku obliczania różnicy w liczbie dni, kiedy sprowadziliśmy oba dni do tej samej godziny. Można sprowadzić daty do niedzieli danego tygodnia i wtedy wykonać obliczenia. Przykład podobnej operacji zamieszczono w podrozdziale 3.6 „Wyznaczanie liczby dni roboczych”, w dalszej części tego rozdziału.

3.2. Wyznaczanie ostatniego dnia podanego miesiąca

Czasami trzeba sprawdzić, jaki jest ostatni dzień określonego miesiąca. Do sprawdzenia dnia można wykorzystać tabelę przeglądową (choć w dalszym ciągu trzeba uwzględnić lata przestępne dla lutego), ale często potrzebny jest znacznik czasu szukanego dnia. Na szczęście dzięki funkcji `mktime()` uzyskanie znacznika czasu nie jest trudne. Funkcja pobiera argumenty określające kolejno godzinę, minutę, sekundę, miesiąc, dzień i rok. Zaletą funkcji `mktime()` jest automatyczna prawidłowa obsługa argumentów ujemnych lub zerowych.

Tak więc, jeśli wprowadzimy argument dla miesiąca równy 3 (marzec), a dla dnia równy 1, uzyskamy prawidłowy znacznik czasu dla 1 marca. W przypadku wprowadzenia parametru dnia równego 0 otrzymamy znacznik czasu poprzedniego dnia, czyli ostatni dzień lutego. Na listingu 3.2.1 pokazano, jak można wykorzystać funkcję `mktime()` do obliczenia znaczników czasu ostatniego dnia miesiąca.

Listing 3.2.1. **Wyznaczanie znacznika czasu ostatniego dnia miesiąca**

```
<?php
// Ustawienie domyślnej strefy czasowej na Europe/Warsaw.
date_default_timezone_set('Europe/Warsaw');

// Funkcja zwraca znacznik czasu ostatniego dnia miesiąca podanego roku
function last_day($month, $year) {
    // Wykorzystanie funkcji mktime do utworzenia znacznika czasu dla następnego miesiąca, ale o jeden dzien
    // wcześniej.
    //  Ustawienie godziny na bliską północy, tak by
    //  uzyskany znacznik można było wykorzystać jako granicę miesiąca.
    return mktime(23, 59, 59, $month + 1, 0, $year);
}

// Wyznaczenie znacznika czasu dla ostatniego dnia lutego 2006 roku.
$stamp = last_day(2, 2006);

// Wyświetlenie wyników: 28
echo '<p>Ostatni dzień lutego 2006 roku przypada na: ', date('d', $stamp) ,'</p>';
?>
```

3.3. Sprawdzenie, czy rok jest przestępny

Często wykonywanym zadaniem, dla którego w PHP nie ma wbudowanej funkcji, jest sprawdzenie, czy określony rok jest przestępny. Choć można to wyliczyć z prostego wzoru, wiele osób zapomina o niektórych jego elementach. Rok jest przestępny tylko wtedy, jeśli dzieli się przez 4, ale nie dzieli się przez 100, chyba że dzieli się przez 400. Funkcję sprawdzającą, czy określony rok jest przestępny, zaimplementowano na listingu 3.3.1.

Listing 3.3.1. **Sprawdzanie, czy rok jest przestępny**

```
<?php
// Funkcja sprawdza, czy określony rok jest przestępny, czy nie:
function is_leap_year($y) {
    // Jeśli rok dzieli się przez 4, ale nie dzieli się przez 100 lub dzieli się przez 400:
    return ((($y % 4) == 0) && ((($y % 100) != 0) || (($y % 400) == 0)));
}

// Sprawdzenie, czy lata z podanego zakresu są przestępne:
foreach (range(1999, 2009) as $year) {
    // Wyświetlenie wyników:
    echo "<p>{$year} = ", is_leap_year($year) ? 'Rok przestępny' : 'rok zwykły',
'</p>';
}
?>
```

3.4. Obsługa stref czasowych

Obsługa stref czasowych może sprawiać wiele problemów. Na szczęście w PHP dostępne są mechanizmy, które ją ułatwiają. Wiele z nich wykorzystuje własność środowiska PHP w wersji 5.1 polegającą na wykorzystaniu funkcji `date_default_timezone_get()` w celu określenia strefy czasowej, dla której są wykonywane obliczenia. W tym punkcie omówimy kilka innych metod. Każda z nich doskonale nadaje się do wykonywania różnych zadań.

We wszystkich tych przykładach założymy, że bieżąca strefa czasowa to *Europe/Warsaw* (GMT+1:00), a strefa czasowa, dla której wykonujemy obliczenia, to *Asia/Tokyo* (GMT+9:00).

Aby dowiedzieć się, jaki jest odpowiednik określonego czasu w innej strefie czasowej, można skorzystać z kombinacji funkcji ustawiającej domyślną strefę czasową i funkcji `mktime()`, która umożliwia wygenerowanie czasu w innej strefie czasowej i wyświetlenie go w strefie domyślnej (listing 3.4.1).

Listing 3.4.1. **Obliczanie czasu w różnych strefach czasowych**

```
<?php
// Sprawdzenie, która godzina jest w Warszawie w czasie, gdy w Tokio jest 8 rano.

// Ustawienie domyślnej strefy czasowej na czas tokijski.
date_default_timezone_set('Asia/Tokyo');

// Wygenerowanie znacznika czasu w tej strefie czasowej dla 1 stycznia 2000 roku.
$stamp = mktime(8, 0, 0, 1, 1, 2000);

// Ustawienie domyślnej strefy czasowej z powrotem na Europe/Warsaw.
date_default_timezone_set('Europe/Warsaw');

// Wyświetlenie daty w formacie standardowym (RFC1123). Funkcja wyświetli następujący ciąg:
// Fri, 31 Dec 1999 18:00:00 EST
echo '<p>', date(DATE_RFC1123, $stamp) ,'</p>';
?>
```

Także funkcja `strtotime()` ułatwia wykonywanie konwersji między strefami czasowymi. W ciągach znaków opisujących czas można wprowadzić modyfikatory stref czasowych. Funkcja `strtotime()` analizuje skróty stref czasowych, na przykład *EST* (ang. *Eastern Standard Time*). Nie zaleca się jednak ich stosowania, ponieważ mogą wystąpić trudności z ich interpretacją w różnych językach. Akceptuje również przesunięcie strefy względem czasu UTC, na przykład `"+0100"` dla strefy *Europe/Warsaw* oraz `"+0900"` dla strefy *Asya/Tokyo*. Przykład użycia tych mechanizmów do wykonywania operacji na strefach czasowych pokazano na listingu 3.4.2.

Listing 3.4.2. **Wyznaczanie czasu w różnych strefach czasowych z wykorzystaniem funkcji strtotime()**

```
<?php
// Ustawienie domyślnej strefy czasowej na Europe/Warsaw.
date_default_timezone_set('Europe/Warsaw');

// Wygenerowanie znacznika czasu dla 1 stycznia 2000 o 8 rano w Tokio.
// Użycie notacji ze skrótem 'JST' (Japan Standard Time)
$stamp1 = strtotime('1/1/2000 8am JST');
// Użycie modyfikatora określającego relację względem GMT: +9 godzin w stosunku do czasu UTC.
$stamp2 = strtotime('Jan 1 2000 08:00 +0900');

// Wyświetlenie daty w formacie standardowym (RFC1123). Funkcja wyświetli następujący ciąg:
// Fri, 31 Dec 1999 18:00:00 EST
echo '<p>', date(DATE_RFC1123, $stamp1) ,'</p>';
echo '<p>', date(DATE_RFC1123, $stamp2) ,'</p>';
?>
```

Wykonywanie konwersji w ten sposób może okazać się kłopotliwe. Często się zdarza, że ktoś przechowuje dane w postaci lokalnego czasu i chce wyświetlić go w innych strefach czasowych. Aby to zrobić, wystarczy skorzystać z tabeli konwersji zawierającej właściwe przeliczniki. Takie tabele nie uwzględniają jednak wielu nietypowych sytuacji, takich jak zmiana czasu z letniego na zimowy, która w różnych miejscach występuje w różnym czasie. Na listingu 3.4.3 do zmiany stref czasowych wykorzystano tabelę przeglądową.

Listing 3.4.3. **Wyznaczanie czasu w różnych strefach czasowych z wykorzystaniem tabeli przeglądowej**

```
<?php
// Ustawienie domyślnej strefy czasowej na Europe/Warsaw.
date_default_timezone_set('Europe/Warsaw');

// Utworzenie tabeli przeglądowej względem strefy Europe/Warsaw.
$locals = array('Seattle' => -7, 'Londyn' => 1, 'Tokyo' => 10);

// Wygenerowanie daty i godziny we wskazanych lokalizacjach na podstawie czasu lokalnego.
$now = time();
foreach ($locals as $place => $offset) {
    // Wyświetlenie daty i godziny po konwersji.
    echo "<p>Czas w miejscowości {$place} to: ", date('m/d/Y H:i',
        $now + 3600 * $offset) ,'</p>';
}
?>
```

3.5. Obsługa znaczników czasu w bazach danych lub plikach

Znaczniki czasu często zapisuje się w plikach. Ich format jest poza naszą kontrolą, jednak trzeba je odczytać w środowisku PHP i zinterpretować jako daty. Daty można przetwarzać za pomocą wyrażeń regularnych, jednak w wielu przypadkach format czasu powinien być zgodny z funkcją `strtotime()`. TakZdarza się tak bardzo często, ponieważ funkcja `strtotime()` obsługuje wiele formatów czasu. Na listingu 3.5.1 zamieszczono przykład je użycia do przetwarzania daty odczytanej z pliku dziennika serwera Apache:

```
127.0.0.1 - - [02/Nov/2005:22:04:41 -0500] "GET / HTTP/1.1" 200 41228
```

Listing 3.5.1. **Przetwarzanie wierszy z pliku dziennika serwera Apache za pomocą funkcji strtotime()**

```
<?php
// Ustawienie domyślnej strefy czasowej na Europe/Warsaw.
date_default_timezone_set('Europe/Warsaw');

// Symulacja odczytu wiersza z pliku dziennika serwera Apache.
$logline =
   '127.0.0.1 - - [02/Nov/2005:22:04:41 -0500] "GET / HTTP/1.1" 200 41228';

// Odczytanie fragmentu dotyczącego daty za pomocą wyrażenia regularnego.
$matches = array();
preg_match('/\[(.*?)\]/', $logline, $matches);

// Pobranie daty i jej konwersja.
$timestamp = strtotime($matches[1]);

// Ponowne wyświetlenie daty w celu potwierdzenia jej prawidłowego odczytania:
echo date(DATE_RFC1123, $timestamp);
?>
```

Podobnym problemem, z jakim często spotykają się programiści, jest zapisywanie dat w bazach danych i odczytywanie ich z baz danych. Każda baza danych charakteryzuje się własną preferowaną metodą prezentacji dat.

Na szczęście większość formatów dat można przetworzyć za pomocą funkcji `strtotime()` (w bazach danych występują również specyficzne komendy pozwalające na zwracanie dat w formatach wskazanych przez użytkowników). Format dat różni się w zależności od baz danych. W związku z tym szczegółowych informacji dotyczących formatu baz danych należy szukać w dokumentacji określonej bazy danych.

Prawdziwy problem występuje podczas zapisywania dat ze środowiska PHP do bazy danych. W każdej bazie danych obowiązuje specyficzny format dat. Należy jednak zwrócić uwagę, że niemal we wszystkich typach baz danych domyślny format daty można zmienić na serwerze. Z tego względu bezpieczniej korzystać z własnych funkcji konwersji ciągów znaków na daty. W ten sposób zyskuje się pewność, że za każdym razem zapiszemy do bazy danych prawidłową wartość daty. Poniżej wyszczególniono kilka przykładów instrukcji `date()` w PHP, które formatują ciągi znaków na domyślny format dat w różnych typach baz danych:

- Oracle — `date('d-M-Y H:i', $timestamp);`
- MySQL — `date('Y-m-d H:i', $timestamp);`
- Sybase and SQL Server — `date('m/d/Y H:i', $timestamp).`

3.6. Wyznaczanie liczby dni roboczych

Choć w podrozdziale 3.1 „Obliczanie różnicy czasu między dwiema datami" zaprezentowano sposób obliczania liczby dni między dwiema datami, skrypt ten na niewiele się zda w świecie biznesu, w którym ważne są dni robocze — w większości miejsc na świecie są to dni od poniedziałku do piątku. Dni robocze czasami uwzględniają święta, a jeśli ich nie uwzględniają, szczegółowe zasady są różne dla różnych firm. Z tego względu skoncentrujemy się na podstawowej definicji.

Jednym ze sposobów obliczania liczby dni roboczych między dwiema datami jest przetwarzanie w pętli wszystkich dni między datami i sprawdzanie, czy określony dzień nie należy do weekendu. Metoda ta jest jednak nieefektywna. Istnieją lepsze rozwiązania, na przykład to, które zaimplementowano w listingu 3.6.1. Wiemy, że tydzień roboczy ma pięć dni. W celu obliczenia liczby dni roboczych wystarczy obliczyć liczbę pełnych tygodni między dwiema datami i pomnożyć przez 5. Następnie należy dodać liczbę dni w pierwszym i ostatnim tygodniu.

Listing 3.6.1. **Obliczanie liczby dni między dwiema datami**

```
<?php
// Ustawienie domyślnej strefy czasowej na Europe/Warsaw.
date_default_timezone_set('Europe/Warsaw');

// Funkcja zwracająca liczbę dni roboczych między dwiema datami.
function count_business_days($a, $b) {
    // Sortowanie dat.  Trzeba się dowiedzieć, która z dat przypada wcześniej.
    if ($a < $b) {
        $first = $a;
        $second = $b;
    } else {
        $first = $b;
        $second = $a;
    }

    // Rozłożenie znaczników czasu na składowe.
    $f = getdate($first);
    $s = getdate($second);

    // Obliczenie liczby pozostałych dni roboczych w początkowym tygodniu.
    // W tym celu należy odjąć numer dnia w tygodniu od pięciu.
    $f_days = 5 - $f['wday'];
    // W przypadku soboty lub niedzieli uzyskamy wynik  -1 lub 5, ale wtedy przyjmujemy do obliczeń wartość 0.
    if (($f_days == 5) || ($f_days < 0)) { $f_days = 0; }

    // Wykonanie podobnej operacji dla końcowego tygodnia, z tą różnicą, że liczymy od
    // początku tygodnia.  Trzeba się tylko upewnić, że sobota liczy się jako 5.
    $s_days = ($s['wday'] > 5) ? 5 : $s['wday'];
    // Obliczenia znacznika czasu dla południa w niedzielę początkowego tygodnia.
```

```
    $f_sunday = mktime(12, 0, 0, $f['mon'],
                    $f['mday'] + ((7 - $f['wday']) % 7), $f['year']);

    // Obliczenia znacznika czasu dla południa w niedzielę poprzedzającą datę końcową.
    $s_sunday = mktime(12, 0, 0, $s['mon'],
                    $s['mday'] - $s['wday'], $s['year']);

    // Obliczenie liczby pełnych tygodni między dwiema datami poprzez odjęcie od siebie
    // znaczników czasu i podzielenie przez liczbę sekund w tygodniu. Uzyskany wynik należy
    //  zaokrąglić po to, by zawsze uzyskać liczbę całkowitą. W innym przypadku z powodu zmiany czasu
    // z letniego
    //  na zimowy (lub odwrotnie) uzyskany wynik może być niedokładny.
    $weeks = round(($s_sunday - $f_sunday) / (3600*24*7));

    // Zwrócenie liczby dni poprzez pomnożenie liczby tygodni przez 5 i dodanie
    // dni roboczych z początkowego i końcowego tygodnia.
    return ($weeks * 5) + $f_days + $s_days;
}

// Kilka przykładów:
$date1 = strtotime('12/3/1973 8:13am');
$date2 = strtotime('1/15/1974 10:15pm');
$date3 = strtotime('2/14/2005 1:32pm');

// Obliczenie liczby dni roboczych. Prawidłowe wyniki to: 31 & 8109.
echo "<p>Daty dzieli ", count_business_days($date1, $date2), " dni roboczych.</p>";
echo "<p>Daty dzieli ", count_business_days($date2, $date3), " dni roboczych.</p>";
?>
```

Jak widać z powyższego listingu, najpierw wyznaczono liczbę dni między niedzielami przypadającymi po dacie początkowej i przed datą końcową. Na tej podstawie obliczono liczbę pełnych tygodni między dwiema datami. Aby to zrobić, wystarczy podzielić liczbę sekund, jakie dzielą obie niedziele, przez liczbę sekund w tygodniu. Wynik należy zaokrąglić do najbliższej liczby całkowitej. Zaokrąglenie jest potrzebne, jeśli między datą początkową a końcową wystąpiła zmiana czasu z zimowego na letni lub odwrotnie. Jeśli nie przeprowadzono by operacji zaokrąglania, wystąpiłby błąd godziny lub dwóch.

Na koniec należy dodać dni robocze z pierwszego i ostatniego tygodnia i otrzymujemy wynik. Wzór jest poprawny nawet wtedy, gdy oba dni należą do tego samego tygodnia. Co prawda, najpierw są liczone dni pozostałe do końca tygodnia i dni od początku tygodnia, co powoduje liczenie niektórych dni kilka razy. W związku z tym wynik zawsze jest większy o 5 dni. Jednak z obliczeń liczby pełnych tygodni uzyskuje się –1. W związku z tym odejmujemy nadmiarowe 5 dni, uzyskując poprawny wynik.

Do zaimplementowanego algorytmu można wprowadzić modyfikację polegającą na uwzględnieniu dni świątecznych. Należałoby za pomocą odpowiedniego algorytmu obliczyć liczbę dni świątecznych między podanymi datami, a następnie odjąć ją od końcowego wyniku.

3.7. Generowanie kalendarza dla określonego miesiąca

Daty są cześcią naszego życia, a z nimi nierozerwalnie wiążą się kalendarze. Ludzie cenią sobie możliwość patrzenia na kalendarz, który umożliwia wizualną ocenę relacji między określonymi datami. Z tego powodu kalendarz jest dostępny w wielu aplikacjach internetowych. Funkcja zamieszczona na listingu 3.7.1 pobiera parametry w postaci miesiąca i roku i wyświetla kalendarz w formacie prostej tabeli HTML.

Zamieszczony algorytm można łatwo zmodyfikować w celu uwzględnienia specyficznych wymagań. W obecnej postaci wprowadzono jedną dyrektywę konfiguracyjną umożliwiającą wskazanie numeru dnia, od którego rozpocząć tabelę. Jej domyślna wartość wynosi 0 (niedziela).

Listing 3.7.1. **Generowanie kalendarza w formacie HTML**

```
<?php
// Funkcja wyświetla kalendarz w formacie HTML dla wskazanego miesiąca i roku.
function print_calendar($month, $year, $weekdaytostart = 0) {
    // O miesiącu, dla którego generujemy kalendarz, potrzebne są pewne informacje, na przykład znacznik
    // czasu ostatniego dnia.
    $last = idate('d', last_day($month, $year));

    // Potrzebna jest również informacja o tym, w jaki dzień tygodnia przypada pierwszy dzień miesiąca.
    // Nazwę miesiąca uzyskamy automatycznie.
    $firstdaystamp = mktime(0, 0, 0, $month, 1, $year);
    $firstwday = idate('w', $firstdaystamp);
    $name = date('F', $firstdaystamp);

    // Aby ułatwić realizację funkcji 'rozpoczęcie od dowolnego dnia tygodnia', potrzebna jest
    //  tablica numerów dni tygodnia w używanym porządku drukowania.
    $weekorder = array();
    for ($wo = $weekdaytostart; $wo < $weekdaytostart + 7; $wo++) {
        $weekorder[] = $wo % 7;
    }

    // Początek tabeli HTML.
    echo "<table><tr><th colspan=\"7\">{$name} {$year}</th></tr>\n";

    // Wyświetlenie wiersza z nazwami dni.
    // Wykorzystanie systemu do uzyskania nazw dni tygodnia.
    echo '<tr>';
    // Przetwarzanie w pętli pełnego tygodnia począwszy od dnia numer 1.
    foreach ($weekorder as $w) {
        $dayname = date('D',
            mktime(0, 0, 0, $month, 1 - $firstwday + $w, $year));
        echo "<th>{$dayname}</th>";
    }
    echo "</tr>\n";

    // Zainicjowanie liczników i wykonanie obliczeń.
    $onday = 0;
    $started = false;

    // Pętla do ostatniego dnia miesiąca.
```

```
    while ($onday <= $last) {
        // Rozpoczęcie następnego wiersza w tabeli.
        echo '<tr>';

        // Pętla od 0 do 6 dla dni tygodnia, ale w wybranym porządku;
        //  do tego celu wykorzystano operator mod.
        foreach ($weekorder as $d) {
            // Jeśli jeszcze nie rozpoczęto:
            if (!($started)) {
                // Czy dzień bieżący odpowiada pierwszemu dniu tygodnia, od którego należy rozpocząć?
                if ($d == $firstwday) {
                    // Ustawienie zmiennej wskazującej na fakt rozpoczęcia generowania kalendarza
                    // i inkrementacja licznika.
                    $started = true;
                    $onday++;
                }
            }

            // Wyświetlenie pustej komórki, jeśli numer dnia wynosi zero lub
            //  jest większy od numeru ostatniego dnia.
            if (($onday == 0) || ($onday > $last)) {
                echo '<td> </td>';
            } else {
                // W innym przypadku wyświetlenie numeru dnia i inkrementacja licznika.
                echo "<td>{$onday}</td>";
                $onday++;
            }
        }

        // Zakończenie wiersza tabeli.
        echo "</tr>\n";
    }

    // Zakonczenie tabeli.
    echo '</table>';
}

// Wygenerowanie dyrektyw formatujących.
echo '<style>table, td, th { border: 1px solid black; }</style>';

// Wygenerowanie kilku przykładowych tabel.
// Listopad 2005 z niedzielą jako pierwszym dniem.
print_calendar(11, 2005);
echo '<br />';

// Utworzenie całorocznego kalendarza dla roku 2006 z poniedziałkiem jako pierwszym dniem.
foreach(range(1, 12) as $m) {
    print_calendar($m, 2006, 1);
    echo '<br />';
}
?>
```

Skrypt rozpoczyna się od pobrania potrzebnych informacji. Do wyznaczenia ostatniego dnia miesiąca wykorzystano funkcję `last_day()` utworzoną na listingu 3.2.1. Potrzebne są również informacje na temat dnia tygodnia pierwszego dnia miesiąca.

Ponieważ funkcja ma przetwarzać w pętli dni tygodnia, potrzebny jest łatwy sposób wykonywania takiej operacji. Gdyby funkcja zawsze przetwarzała dni od niedzieli do so-

boty (0 do 6), nie byłoby problemu. Ponieważ jednak kalendarz można rozpocząć od dowolnego dnia tygodnia, funkcja musi określić porządek przetwarzania pętli. Na przykład, jeśli kalendarz ma się zacząć od środy, dni tygodnia powinny być przetwarzane w następującej kolejności: 3, 4, 5, 6, 0, 1, 2.

Aby to było możliwe, funkcja tworzy w pętli tablicę zawierającą te liczby, począwszy od wskazanego dnia początkowego. Pętla wykonuje się 7 razy. W każdej iteracji wykonywana jest operacja licznika pętli modulo 7.

Do odczytania nazw miesięcy i dni tygodnia wykorzystano wbudowane funkcje. Dzięki temu funkcja zawsze drukuje kalendarz we właściwej wersji językowej. Pozostała jeszcze pętla przetwarzająca tygodnie i wyświetlająca kolejne dni.

Trzeba zadbać o to, by pierwszy dzień wyświetlić na właściwej pozycji. Wcześniej wyświetlamy puste komórki. Odpowiednią liczbę pustych komórek trzeba także wyświetlić w ostatnim wierszu.