Operacje na ciągach

Liczby losowe

Liczby losowe mają bardzo szerokie zastosowanie w programowaniu serwisów www. Przydają się przy generowaniu losowych haseł, tymczasowych kodów weryfikujących lub linków aktywacyjnych. Wyświetlanie różnych cytatów, czy losowych produktów w sklepie również opiera się o liczby losowe.
Tyle słowem wstępu, czas na bliższe przyjrzenie się generowaniu liczb losowych:

```
<?php
	echo rand();
?>
```

Powyższy skrypt spowoduje wyświetlenie się na ekranie liczby z przedziału od 0 do **getrandmax()** – zdefiniowanego maksimum. Dla systemów z rodziny Windows wynosi ono 32768, natomiast w systemach Unixowych 2147483647.

Wyznaczanie zakresu liczb

W większości przypadków chcemy wyznaczyć przedział, z którego ma zostać wylosowana liczba. PHP udostępnia nam taką możliwość. Wystarczy podać dwa argumenty w wywołaniu funkcji **rand(),** minimalną i maksymalną wartość.
Zobaczmy, jak wygląda to w praktyce:

```
<?php
	$i = rand(15,30);
	echo "Wylosowana liczba z zakresu (15 - 30) to $i";
?>
```

PHP łączenie stringów

Programując w PHP, bardzo często zachodzi potrzeba połączenia ze sobą dwóch lub więcej, ciągów znaków. Można to zrobić na różne sposoby. Jednym z nich jest zastosowanie operatora łączenia. Poniżej przykład:

```
<?php
	$wyswietl = "Bardzo";
	$wyswietl .= " lubię";
	$wyswietl .= " placki.";

	// polecenie wyświetli napis:
	// "Bardzo lubię placki."
	echo $wyswietl;

	// wyświetli napis:
	// Moje motto to: "Bardzo lubię placki."
	echo 'Moje motto to: "'.$wyswietl.'"';
?>
```

Działanie operatora łączenia jest bardzo oczywiste. Po prostu łączy ciąg z prawej oraz lewej strony w jeden.
Można również łączyć wiele stringów w jednej linijce i przypisać je do jednej zmiennej. Tak jak tutaj:

```
<?php
	$string1 = " z sosem.";
	$wyswietl = "Bardzo" . " lubię" . " placki" . $string1;

	// polecenie wyświetli napis:
	// "Bardzo lubię placki z sosem."
	echo $wyswietl;

	// wyświetli napis:
	// Moje motto to: "Bardzo lubię placki."
	echo 'Moje motto to: "'.$wyswietl.'"';
?>
```

Istnieje jeszcze inna metoda w PHP na połączenie ciągów…

Funkcja implode()

Funkcja implode łączy zbiór ciągów, przekazany w postaci tablicy PHP, w jeden, podzielony łącznikiem. Przyjmuje dwa argumenty, w tym jeden opcjonalny. Jako pierwszy podajemy element łączący – domyślnie jest to pusty ciąg. Drugim argumentem jest tablica ciągów, którą chcemy połączyć. Wartość zwracana to ciąg znaków składający się z elementów tablicy, przedzielonych łącznikiem.
Zobaczmy poniższy listing:

```
<?php
	$data[0] = "18";
	$data[1] = "07";
	$data[2] = "1976";
	$wyswietl_date = implode(" - ", $data);

	// wyświetli "18 - 07 - 1976"
	echo $wyswietl_date;
?>
```

W przypadku pominięcia pierwszego argumentu zwrócony zostałby ciąg „18071976”. Funkcja **implode** jest bardzo użyteczna, gdy mamy do czynienia z wieloma zmiennymi formularza. Możemy je wtedy przedzielić, np. przecinkiem, po czym zapisać do jednej zmiennej.
Implode w PHP stosujemy zazwyczaj, gdy zmiennych mamy bardzo dużo i łączenie ich operatorem łączenia (kropką) byłoby bardzo mało czytelne.

Szukanie podciągu

Bardzo ważny element, przydatny podczas tworzenia różnego rodzaju ksiąg gości oraz for internetowych. Niezbędny szczególnie wtedy, gdy wpisy nie są moderowane i od razu wyświetlają się na stronie. Najczęściej chcemy sprawdzić, czy dany wyraz występuje wewnątrz ciągu znaków. Przykładowo, gdy chcemy wprowadzić zakaz wpisywania wulgaryzmów.

Najczęściej stosowana funkcja, spełniająca powyższe zadanie, to **strpos().** Przyjmuje ona dwa argumenty, ciąg szukany oraz ciąg, w którym szukamy. W przypadku znalezienia wyrazu wewnątrz szukanego stringa, funkcja zwróci pozycję występowania wyrazu. Jeżeli natomiast podciąg nie zostanie znaleziony, funkcja zwróci wartość logiczną FALSE.
Zobaczmy jej zastosowanie w praktyce:

```
<?php
	$wpisany_tekst = "Rozwijam swoją wiedzę w dziedzinie PHP.";
	$czy = strpos($wpisany_tekst, "cholera");

	if ($czy == FALSE) // nie znaleziono słowa cholera
		echo "Można wyświetlić: $wpisany_tekst.";
	else // znaleziono szukany wyraz
		echo "Tekst zawiera wulgarne słownictwo.";
?>
```

3 przykłady funkcji PHP explode [wraz z kodem źródłowym]

Czasami samo określenie, czy dany podciąg znajduje się w stringu nie wystarczy.
Często potrzebujemy określić, ile razy występuje on w tekście. Jedną z możliwości, udostępnianą przez PHP, jest funkcja **explode**.
Explode pozwala nam podzielić stringa, przypisując każdy rozdzielony element do kolejnego indeksu w tablicy.
Jednym z najczęstszych użyć funkcji **explode** podczas programowania w PHP jest podzielenie całej zawartości pliku wg znaku końca linii. Wtedy otrzymamy każdą linijkę pliku jako osobną zmienną.
Na to jednak przyjdzie pora.
Zobaczmy jak wygląda jej podstawowe użycie:

```
<?php
	// string do rozdzielenia
	$dane = "Marcin,Wesel,Bielsko-Biała";

	// wykorzystanie funkcji explode, wg przecinka
	$dane_osobowe = explode(",", $dane);

	// wyświetlenie otrzymanej tablicy
	echo $dane_osobowe[0]."<br/>";
	echo $dane_osobowe[1]."<br/>";
	echo $dane_osobowe[2]."<br/>";
?>
```

Funkcja **explode** biegnie po kolei przez zmienną podaną w drugim argumencie i szuka wystąpienia pierwszego argumentu (w naszym przypadku był to przecinek). W momencie, gdy wystąpienie znajdzie, całą zawartość do tej pory wrzuca jako pierwszy element tablicy, wyrzuca przecinek i biegnie dalej, aż do kolejnego wystąpienia. Jak znajdzie kolejne, wrzuca zawartość do drugiego elementu, wyrzuca separator i tak aż do końca zmiennej.
Co się stanie, jeśli wykonamy funkcję **explode** na stringu, który nie posiada ani jednego wystąpienia? Zupełnie nic, w rezultacie dostaniemy tego samego stringa, gdyż funkcja nie była w stanie go podzielić.

W ten sposób otrzymaliśmy tablicę $dane_osobowe, która zawiera w sobie każdą część z rozbijanego stringa.

Dalsze zastosowania funkcji explode

Wykorzystując funkcję **explode**, możemy w łatwy sposób policzyć np. ilość słów w tekście. Wystarczy rozdzielić ciąg według spacji, a następnie zliczyć ilość indeksów w tablicy. Pomoże nam w tym metoda **count().**
Poniżej prezentacja:

```
<?php
	$tekst = "Liwto, ojczyzno moja, Ty jesteś jak zdrowie.";
	$wyrazy = explode(" ", $tekst);

	// wyświetli ilość wyrazów w zmiennej $tekst
	echo count($wyrazy);
?>
```

Jeszcze obiecane liczenie linijek. Nie potrafisz jeszcze przeczytać zawartości pliku, ale możesz podzielić zawartość każdym wystąpieniem znaku nowej linii.

```
<?php
	$tekst = "Litwo
	Ojczyzno
	Moja";
	$linie= explode("\n", $tekst);

	// wyświetli ilość linijek w zmiennej $tekst, zakładając, że entery są poprawnie wstawione
	echo count($linie);
?>
```

Słowo wyjaśnienia odnośnie znaku nowej linii – to znak specjalny zależny od systemu operacyjnego. Z racji, że PHP w ogromnej większości przypadków, uruchamiany jest na systemach Linux/Unix, ten znak to „\n”. W Windowsie byłby to znak „\r\n”. PHP jest w stanie samodzielnie rozpoznać, jaki znak końca linii ma wstawić. Wystarczy użyć zarezerwowanej stałej PHP_EOL, która w efekcie zamieni się na znak końca linii.
Poczytaj więcej o zarezerwowanych stałych w oficjalnej dokumentacji PHP: http://php.net/manual/en/reserved.constants.php
Link do dokumentacji explode() na znajdziesz tutaj: http://php.net/manual/en/function.explode.php
To tylko niektóre z zastosowań tej funkcji. W przygodzie z programowaniem na pewno natkniesz się na wiele więcej.

Zamiana na wielkie litery

Bardzo często, podczas obsługi formularzy, zachodzi potrzeba zamiany na wielkie litery. Przykładowo, chcemy przechowywać w bazie danych loginy użytkowników. Wyczulone algorytmy sprawdzające rozróżniają małe i wielkie litery. Admin, ADMIN, admin i AdMIn to cztery różne stringi. Żeby zabezpieczyć się przed przypadkami dodania dwóch tych samych

nazw, różniących się tylko wielkością liter, musimy nauczyć się zamieniać je w jedną oraz drugą stronę.
Do zamiany wszystkich liter w ciągu na wielkie, używamy funkcji **strtoupper()**. Poniżej przykład zastosowania:

```
<?php
	// przykładowy ciąg znaków zawierający małe oraz wielkie litery
	$wyswietl = "Bardzo lubię placki";

	// użycie funkcji strtoupper()
	$duze_litery = strtoupper($wyswietl);

	// wyświetli napis:
	// BARDZO LUBIĘ PLACKI
	echo $duze_litery;
?>
```

PHP małe litery

W analogiczny sposób możemy zamienić wszystkie znaki na małe litery. Służy do tego funkcja **strtolower().** Poniżej przykład:

```
<?php
	// przykładowy ciąg znaków zawierający małe oraz wielkie litery
	$wyswietl = "BarDzo Lubię PlaCki";

	// użycie funkcji strtolower()
	$male_litery = strtolower($wyswietl);

	// wyświetli napis:
	// bardzo lubię placki
	echo $male_litery;
?>
```

Zamiana pierwszych liter wyrazów na wielkie

Czasami, w celach estetycznych, zachodzi potrzeba zamiany tylko pierwszej litery na wielką (np. wyświetlanie i zapisywanie imienia lub nazwiska). PHP udostępnia nam funkcję **ucwords(),** która realizuje to zadanie.

```
<?php
	// przykładowy ciąg znaków zawierający małe litery
	$wyswietl = "bardzo lubię placki";

	// użycie funkcji ucwords()
	$zdanie = ucwords($wyswietl);

	// wyświetli napis:
	// Bardzo Lubię Placki
```

```
	echo $zdanie;
?>
```

Sprawdzanie liczby znaków w ciągu

Często chcemy zbadać, czy wpisywany tekst (np. post na forum) ma określoną długość. Możemy to osiągnąć używając funkcji **strlen().** Poniżej przykład:

```
<?php
	// przykładowy ciąg znaków
	$wyswietl = "bardzo lubię placki";

	// użycie funkcji strlen()
	$ilosc = strlen($wyswietl);

	if ($ilosc > 20)
		echo "Treść dłuższa niż 20 znaków.";
	else
		echo "Treść zawiera 20 znaków lub mniej.";
?>
```

Usuwanie białych znaków z początku i końca ciągu

Czasem sprytniejsi internauci mogą chcieć nas oszukać, próbując wstawić sporo spacji na początku lub na końcu tekstu. Spacja zostanie policzona jako jeden znak, więc 20 spacji będzie się równało 20 znakom. Oczywiście, chcemy takiej sytuacji uniknąć. Służy do tego funkcja **trim().**
Zobaczmy przykład:

```
<?php
	// przykładowy ciąg znaków
	$wyswietl = "    bardzo lubię placki    ";

	// użycie funkcji trim()
	$zdanie = trim($wyswietl);

	// wyświetli napis:
	// "bardzo lubię placki"
	echo $zdanie;
?>
```

PHP wyrażenia regularne

Wyrażenie regularne to nic innego, jak szablon ciągu. Definiując wyrażenie możemy w dowolny sposób określić format stringa. Narzędzie bardzo przydatne w przypadku sprawdzania poprawności wpisywanych tekstów, jak np. adresu e-mail lub danych osobowych. Wykorzystując wyrażenia regularne możemy również wyszukać oraz podmienić wszystkie podciągi znajdujące się w tekście, które pasują do wpisanej formuły.

Poznamy podstawy konstrukcji wyrażeń regularnych oraz funkcje sprawdzającą występowanie szablonu w ciągu znaków. Zaczniemy od prostego przykładu:

```
<?php
	// konstrukcja wyrażenia regularnego
	$wyrazenie = '/^[a-z]{1,}$/';

	// preg_match() sprawdza występowanie wyrażenia w ciągu
	if (preg_match($wyrazenie, $tekst))
		echo("Tekst zawiera tylko małe litery bez polskich znaków.");
	else
		echo("Tekst zawiera dodatkowe znaki.");
?>
```

Przeanalizujmy powyższy kod. Do zmiennej **$wyrazenie** przypisaliśmy nasz wzorzec w postaci wyrażenia regularnego (omówienie konstrukcji za chwilę). Na chwilę obecną sprawdza on jedynie, czy ciąg zawiera wyłącznie małe litery bez polskich znaków. Następnie funkcja **preg_match($wyrazenie, $tekst)** sprawdza, czy w ciągu **$tekst** znajduje się szablon **$wyrazenie**. Zwraca **TRUE**, jeżeli dopasuje szablon lub **FALSE**, jeśli nie.

Tworzenie wyrażeń regularnych

Na podstawie wyrażenia wpisanego w poprzednim przykładzie omówię ogólne zasady. Wyrażenie regularne musi znaleźć się pomiędzy ukośnikami **„/ … /„.** W przypadku przypisania do zmiennej nie możemy zapomnieć o apostrofach! Następnie każde wyrażenie regularne należy rozpocząć znakiem ^ oraz zakończyć znakiem **$**. Są to tak zwane znaki początku i końca. Między tymi granicami wpisujemy żądaną formułę. W naszym przypadku były to małe litery od a do z. By wpisać przedział znaków wykorzystujemy nawiasy kwadratowe **[ ]**. Możemy między nimi wpisać dowolne przedziały, np.: **[a-eA-E0-7]**, wtedy wzorcem będą tylko takie ciągi, które zawierają w sobie jedynie małe i wielkie litery od a do e oraz cyfry od 0 do 7. Następnie w nawiasach klamrowych podajemy liczbę wystąpień znaku **{1,}** oznacza co najmniej jedno wystąpienie.

Zestawienie wzorców wyrażeń regularnych

W poprzednim punkcie zaprezentowany został szczątkowy zarys zastosowania wyrażeń regularnych. Czas teraz, by znacznie rozszerzyć to zagadnienie.

Znaki specjalne

W tworzeniu szablonu wyrażenia używamy tzw. znaków specjalnych. Cztery z nich poznaliśmy ostatnio (^ – znak początku szablonu oraz zaprzeczenia, **[ ]** – znaki zakresu, **$** – znak końca wyrażenia, **–** – przedział). Poniżej znajdziecie zestawienie ważniejszych znaków specjalnych:

.	Zastępuje dowolny znak
/s	Spacja
/n	Znak nowej linii
/d	Cyfra
^	Początek linii lub zaprzeczenie
$	Koniec linii

|	Alternatywa
{a,b}	Ilość wystąpień danego wzorca – co najmniej a i co najwyżej b razy
{a,}	Jak wyżej, bez limitu górnego
{,b}	Jak wyżej, bez limitu dolnego
{a}	Dokładnie a wystąpień
?	Zero lub jedno wystąpienie; tak samo jak {0,1}
+	Jedno lub więcej wystąpień; tak samo jak {1,}
*	Dowolna ilość wystąpień (również zero); podobnie jak {0,}
[]	Zakres

Znak ” \ „

Patrząc na tabelę powyżej widzimy, że część znaków traktowana jest jako znaki specjalne. Co w przypadku, gdy chcemy sprawdzić, czy w stringu występuje kropka? Wpisując **/^[.]$/** interpreter potraktuje wpisaną kropkę jako dowolny znak. Możemy jednak wymusić, by kropka była sprawdzana jako znak kropki – wystarczy poprzedzić ją odwrotnym ukośnikiem. Nasz wzorzec wyglądałby wtedy następująco: **/^[\.]$/**. Idąc dalej tym tokiem myślenia, żeby sprawdzić, czy w ciągu występuje znak \, należy również poprzedzić go odwrotnym ukośnikiem, otrzymując „\\”.

Przykład zastosowania wyrażeń regularnych

Na początek zobaczmy prosty przykład, który sprawdza, czy prawidłowo wpisano imię oraz nazwisko:

```
<?php
	$imie = $_POST['imie'];
	$nazwisko = $_POST['nazwisko'];

	// konstrukcja wyrażenia regularnego
	// poprawność imienia oraz nazwiska
	$sprawdz = '/^[A-ZŁŚ]{1}+[a-ząęółśżźćń]+$/';

	if(preg_match ($sprawdz, $imie))
		{
			if(preg_match($sprawdz, $nazwisko))
				echo "Podano poprawne dane.";
			else
				echo "Błędne nazwisko.";
		}
	else
		echo "Błędne imię.";
?>
```

Przeanalizujmy konstrukcję sprawdzającą imię oraz nazwisko. Nasz ciąg musi zaczynać się od wielkiej litery – stąd przedział wszystkich liter od A do Z (alfabetu angielskiego). Dodatkowo należy dodać dwie polskie litery, na które może zacząć się imię – Ł oraz Ś. Nawias klamrowy z jedynką w środku oznacza, że oczekujemy na początku tylko jednego wystąpienia

wielkiej litery. Następnie spodziewamy się ciągu małych liter, wraz ze wszystkimi polskimi znakami.
Wiesz już jak można sprawdzić, czy użytkownik wpisał poprawnie swoje dane. W podobny sposób można przetestować wpisywaną miejscowość, datę czy nr telefonu. Wystarczy jedynie zmodyfikować wyrażenie regularne.

Walidacja e-mail w PHP

Jak można sprawdzić, czy użytkownik podał poprawny adres e-mail.
Niestety nigdy nie mamy pewności, że podany adres jej prawdziwy, możemy jednak kierować się pewnymi wytycznymi:

- W adresie musi znaleźć się znak małpy @, dodatkowo może pojawić się tylko jeden raz
- Mogą występować tylko litery (duże i małe), cyfry, znak podkreślnika, myślnik oraz kropka
- Kropka musi pojawić się przynajmniej raz, później niż znak @
- Końcówka domeny musi zawierać się w przedziale od 2 (np. „PL") do 4 (np. „INFO") znaków

Wiedząc to, możemy spróbować utworzyć takie wyrażenie regularne, które wyłapie nam wszystkie adresy pasujące do tego wzorca:

```
<?php
	// przypisanie adresu e-mail do zmiennej
	$email = $_POST['email'];

	// formuła prawidłowego adresu e-mail
	$sprawdz = '/^[a-zA-Z0-9.\-_]+@[a-zA-Z0-9\-.]+\.[a-zA-Z]{2,4}$/';

	if(preg_match($sprawdz, $email))
		echo 'Podano prawidłowy adres e-mail';
	else
		 echo 'Adres e-mail nieprawidłowy';
?>
```

Wyżej wypisane wyrażenie regularne musi zostać spełnione, jeśli adres jest poprawny. Są jednak takie adresy, które również będą pasować, będąc jednocześnie złe. Dodatkowo nie mamy pewności, czy wpisana domena istnieje. Jest to jednak dobry sposób, by wstępnie zweryfikować podawane dane. Należy mieć też na uwadze, że adres e-mail może być wpisany poprawnie, ale z poziomu PHP nie mamy możliwości sprawdzić, czy istnieje (nawet jeśli domena jest zarejestrowana, adres może nie być utworzony).

Podsumowanie

Przyszedł czas na podsumowanie nabytej wiedzy z zakresu liczb losowych, łączenia i dzielenia ciągów oraz wyrażeń regularnych.

Napiszemy obsługę formularza kontaktowego, który sprawdzi poprawność wpisywanych danych. Tego typu formularze można znaleźć w większości stron. Aby uniknąć niechcianego spamu, musimy nauczyć się poprawnie je obsłużyć i oddzielić wartościowe wiadomości od śmieci. Formularz będzie się składał z imienia, numeru telefonu, adresu e-mail oraz treści wiadomości.

Najpierw plik **kontakt.html**, który wyświetli formularz kontaktowy:

```
<html>
<head>
  <title>Formularz kontaktowy</title>
</head>
<body>
  <form action="sprawdz.php" method="post" >
    Imię: <input type="text" name="imie" /><br/>
    Telefon: <input type="text" name="telefon" /><br/>
    E-mail: <input type="text" name="email" /><br/>
    Treść wiadomości:<br/>
    <textarea name="tresc">TUTAJ WPISZ TREŚĆ</textarea>
    <input type="submit" value="OK" /><br/>
  </form>
</body>
</html>
```

Nie dzieje się tutaj nic nadzwyczajnego. Zwykły formularz wysyłający dane dalej. Teraz kolej na plik **sprawdz.php**:

```
<?php
function sprawdz_email($email)
{
	$spr = '/^[a-zA-Z0-9.\-_]+@[a-zA-Z0-9\-.]+\.[a-zA-Z]{2,4}$/';
	if(preg_match($spr, $email))
		return true;
	else
		return false;
}

function sprawdz_imie($imie)
{
	$sprawdz = '/^[a-zA-ZŁŚĆŹŻĄĘÓŃąęółśżźćń]+$/';
	if(preg_match($sprawdz, $imie))
	{
		$imie = ucwords(strtolower($imie));
		return $imie;
	}
	 else
		return false;
	}

function sprawdz_telefon($telefon)
{
	$sprawdz = '/^[0-9]{9}$/';
		if(preg_match($sprawdz, $telefon))
			return true;
		else
			return false;
}
```

```
function sprawdz_tresc($tresc)
{
	$tresc = trim($tresc);
		 if(strlen($tresc) < 30)
			return false;
		else
			return $tresc;
}

$email = $_POST['email'];
$imie = $_POST['imie'];
$tel = $_POST['telefon'];
$tresc = $_POST['tresc'];
$blad_danych = false;

if (!sprawdz_email($email))
{
	echo "Adres e-mail niepoprawny";
	$blad_danych = true;
}
$imie = sprawdz_imie($imie);
if (!$imie)
{
	echo "Imię wpisano niepoprawnie";
	$blad_danych = true;
 }
if (!sprawdz_telefon($tel))
{
	echo "Błędny format telefonu";
	$blad_danych = true;
}
$tresc = sprawdz_tresc($tresc);
if (!$tresc)
{
	echo "Niepoprawna treść";
	$blad_danych = true;
}
if ($blad_danych)
{
	echo "Wystąpił jeden lub więcej błędów podczas";
	echo "przetwarzania danych.";
}
else
{
	echo "Imię klienta: $imie;";
	echo "Adres e-mail: $email;";
	echo "Numer telefonu: $tel;";
	echo "Treść: $tresc;";
 }
```

```
?>
```

Przegląd rozwiązania

Przeanalizujmy rozwiązanie tego przykładowego problemu. Każda funkcja sprawdza oddzielną daną, która jest przesyłana z formularza. W przypadku **telefonu** oraz **e-maila** zwraca **true**, jeżeli jest poprawny, lub **false**, jeżeli nie jest. Funkcje sprawdzające imię oraz treść zawierają wewnętrzną korekcję danych.

Imię zamienia najpierw wszystkie znaki na małe litery, by później powiększyć jedynie pierwszy znak. **Treść** natomiast usuwa wszystkie białe znaki z początku i końca tekstu, po czym sprawdza długość ciągu. Obie zwracają poprawionego stringa, jeżeli był prawidłowy lub **false**, jeżeli nie był. Po definicjach funkcji następuje przypisanie zmiennych formularza do nowych, krótkich nazw.

Wprowadzamy zmienną pomocniczą, nazwaną **$blad_danych**. Domyślnie jest jej przypisana wartość **false**. W przypadku, gdy choć jedna z wysyłanych danych jest niepoprawna, zmienna przyjmuje wartość **true**. Jeżeli na końcu jej wartość wynosi **true**, wyświetlany jest komunikat o błędzie. Jeśli jednak nie wystąpił żaden błąd, na ekranie wyświetlane są informacje o wysyłanych danych.

Ćwiczenia

- Dopisz funkcję **sprawdz_domene(),** która sprawdzi, w jakiej domenie znajduje się podawany adres e-mail. Zastosuj funkcję **explode**, by rozdzielić adres względem znaku **@.**
- Dopisz sprawdzanie treści, szukając wulgaryzmów i wyświetlając odpowiednie komunikaty.
- Połącz przesłane dane w jeden ciąg. Możesz najpierw stworzyć tablice danych, a następnie połączyć je funkcją **implode**.