# **PHP**

PHP (Hypertext Preprocessor) to skryptowy język programowania najczęściej wykorzystywany do tworzenia aplikacji internetowych. Skrypty pisane w PHP są umieszczane w kodzie HTML i wykonywane po stronie serwera. Klient otrzymuje tylko wynik wykonania skryptu.

PHP umożliwia przetwarzanie danych z formularzy, dynamiczne generowanie zawartości stron internetowych, wysyłanie i odbieranie cookies. Pozwala na dynamiczne tworzenie obrazów, dokumentów PDF, animacji Flash.

PHP jest rozpowszechniany na zasadach open source, w związku z tym dostępny jest jego pełny kod źródłowy. Istnieją również liczne rozszerzenia i narzędzia, z których można korzystać, zwiększając jego możliwości.

Wyróżnia się trzy główne obszary, w których używany jest język PHP.

* Skrypty po stronie serwera – główne zastosowanie PHP. Aby tworzyć skrypty potrzebny jest praser PHP, serwer WWW oraz przeglądarka internetowa.
* Skrypty wywołane z wiersza poleceń – skrypty są tworzone z wykorzystaniem prasera PHP i wywołane w linii poleceń. Mogą być wykorzystane również do prostego przetwarzania tekstu.
* Aplikacje po stronie klienta – zaawansowane funkcje PHP umożliwiające pisanie aplikacji desktopowych.

PHP jest skryptowym językiem programowania, za pomocą którego można tworzyć aplikacje WWW. Używając PHP, można:

* Generować dynamicznie zawartość strony internetowej,
* Tworzyć i edytować pliki na serwerze,
* Pobierać dane z formularzy,
* Odbierać i wysyłać cookies,
* Wykonywać operacje na bazie danych,
* Sterować dostępem użytkowników do stron witryny internetowej,
* Szyfrować dane

Blok PHP w kodzie HTML

|  |
| --- |
| <!DOCTYPE HTML>  **<?php ……..?>**  <html><head>  <title>Skrypt PHP</title>  <meta http-equiv=”Content-Type” content=”text/html; charset=UTF-8”/>  <link ……. **<?php …….. ?>**……../>  </head>  <body>  <p> **<?php**  **echo ” Mój pierwszy skrypt PHP”;**  **?>** </p>  </body></html> |

# **Struktura języka PHP**

**Skrypty PHP są plikami tekstowymi**, dlatego do ich pisania można wykorzystać dowolny edytor tekstu. Można używać również edytorów tekstu przeznaczonych do pisania skryptów PHP. Aby pliki tekstowe były prawidłowo rozpoznawane przez serwer WWW, muszą mieć rozszerzenie **.php**.

**Podstawową zasadą podczas tworzenia skryptów PHP jest oddzielanie kolejnych instrukcji znakiem średnika**. Instrukcje mogą być pisane pojedynczo w jednym wierszu lub w kilku.

**W kodzie wyświetlanej strony nie ma znaczników PHP, pozostaje jedynie kod HTML.**

Tak działają języki skryptowe wykonywane po stronie serwera. Ich zadaniem jest wygenerowanie kodu, który będzie mogła zinterpretować przeglądarka internetowa. Kod został wykonany na serwerze, a jego wynik został zwrócony do przeglądarki jako zwykły HTML.

**Znaczniki kodu PHP: <?php ……. ?>,**

po konfiguracji serwera również: <? ….. ?>, <% …. %>

# **Zmienne**

**Przykłady**

* $zmienna = ”Hello world!”; // zadeklarowana zmienna + inicjalizacja
* $Zmienna= ’10’; // zadeklarowana zmienna + inicjalizacja
* $ZMIENNA = 10; // zadeklarowana zmienna + inicjalizacja

## **Zmienne lokalne**

Zmienne w programowaniu służą do przechowywania danych i wyników wykonywanych operacji. Wartością zmiennej może być liczba, ciąg znaków, tablica, obiekt. Zmienna musi posiadać nazwę, przez którą można odwoływać się do niej w skrypcie. Nazwa może być dowolna, ale musi spełniać warunki:

* **Musi zaczynać się od litery lub znaku podkreślenia**
* **Może składać się jedynie z liter, cyfr i znaku podkreślenia**
* **W nazwach rozróżniane są małe i duże litery**
* **W nazwach można stosować polskie litery**
* **Przed nazwą zmiennej należy umieścić znak $.**

Zmienne deklarowane wewnątrz skryptu lub funkcji są zmiennymi lokalnymi i istnieją tylko w obrębie danego skryptu lub funkcji. Nie można się nimi odwoływać w innych skryptach lub funkcjach.

## **Zmienne predefiniowane**

Zmienne predefiniowane nazywane superglobalnymi. Przechowują one informację dotyczące konfiguracji bieżącego skryptu i najczęściej mają postać tablicy. Ich wartość jest ustalana na podstawie zmiennych środowiskowych serwera WWW. Są dostępne we wszystkich skryptach. Należą do nich:

* $\_GET – tablica zawierająca zmienne przesyłane do skryptu za pomocą metody GET.
* $\_POST – tablica zawierająca zmienne przesyłane do skryptu za pomocą metody POST.
* $\_COOKIE – tablica zawierająca zmienne przekazane z serwera do skryptu za pomocą cookies.
* $\_FILES – tablica zawierająca zmienne przekazane do skryptu podczas przesyłania plików do serwera.
* $\_SERVER - tablica zawierająca zmienne przekazane do skryptu przez serwer WWW. Są to dane takie jak wersja serwera, ścieżka do pliku, adres skryptu, wysłane nagłówki.
* $\_ENV - tablica zawierająca wartości zmiennych środowiskowych serwera.
* **$\_REQUEST - tablica zawierająca zmienne przekazane do skryptu przez użytkownika. Zawiera dane z $\_GET, $\_POST, $\_COOKIE , $\_FILES.**
* $\_SESSION - tablica zawierająca zmienne zarejestrowane w bieżącej sesji.
* $GLOBALS - tablica zawierająca odniesienie do każdej zmiennej utworzonej przez użytkownika, która ma zasięg globalny dla danego skryptu.

# **Typy danych**

**Typy skalarne, czyli proste**. Należą do nich:

* Typ boolean
* Typ integer
* Typ float lub double
* Typ string

**Typy złożone**. Należą do nich:

* Typ array (tablicowy)
* Typ object (obiektowy)

**Typy specjalne.** Należą do nich:

* Typ resource
* Typ null

# **Zmiana typu zmiennej**

W języku PHP podczas przypisywania zmiennej nowej wartości ponownie ustalany jest jej typ. Czasami zachodzi jednak konieczność zmiany typu danych. Można tego dokonać za pomocą **rzutowania (cast)** – wtedy efekt jest jednorazowy - lub za pomocą **funkcji settype() –** wtedy efekt jest trwały.

Rzutowanie typów odbywa się przez podanie nowego typu w nawiasie przed zmienną lub wartością, której typ należy zmienić. Dozwolone są określone typy rzutowań. Są to:

* integer – rzutowanie do typu całkowitego
* float (double) – rzutowanie do typu rzeczywistego
* string – rzutowanie do ciągu tekstowego
* array – rzutowanie do tablicy
* object – rzutowanie do obiektu.

# **Operatory**

Operatory służą do wykonywania działań na zmiennych. W języku PHP można je podzielić na operatory:

* Arytmetyczne
* Porównania
* Bitowe
* Logiczne
* Przypisania
* Inkrementacji i dekrementacji
* Pozostałe

## **Operatory arytmetyczne**

|  |  |  |
| --- | --- | --- |
| Operator | Działanie | Przykład |
| + | Dodawanie | $a + $b |
| - | Odejmowanie | $a - $b |
| \* | Mnożenie | $a \* $b |
| / | Dzielenie | $a / $b |
| % | Dzielenie modulo (z resztą) | $a % $b |

## **Operatory porównania**

|  |  |  |
| --- | --- | --- |
| Operator | Działanie (True) | Przykład |
| == | gdy argumenty są równe. | $a == $b |
| != | gdy argumenty są różne | $a != $b |
| === | gdy argumenty są tego samego typu i są równe | $a === $b |
| !== | gdy argumenty są różne lub są różnych typów | %a !== $b |
| > | gdy argument pierwszy jest większy od drugiego | $a > $b |
| < | gdy argument pierwszy jest mniejszy od drugiego | $a < $b |
| >= | gdy argument pierwszy jest większy od drugiego lub mu równy | $a >= $b |
| <= | gdy argument pierwszy jest mniejszy od drugiego lub mu równy | $a <= $b |
| <> | gdy argumenty są różne | $a <> $b |

## **Operatory bitowe**

|  |  |  |
| --- | --- | --- |
| Operator | Działanie | Przykład |
| & | Iloczyn bitowy (AND) | $a & $b |
| | | Suma bitowa (OR) | $a | $b |
| ~ | Negacja bitowa (NOT) | ~$a |
| ^ | Bitowa różnica symetryczna | $a ^ $b |
| >> | Przesunięcie bitowe w prawo | $a >> $b |
| << | Przesunięcie bitowe w lewo | $a << $b |

## **Operatory logiczne**

|  |  |  |
| --- | --- | --- |
| Operator | Działanie | Przykład |
| and | Iloczyn logiczny | $a and $b |
| && | Iloczyn logiczny | $a && $b |
| or | Suma logiczna | $a or $b |
| || | Suma logiczna | $a || $b |
| ! | Negacja logiczna (NOT) | !$a |
| xor | Różnica asymetryczna | $a xor $b |

## **Operatory przypisania**

|  |  |  |
| --- | --- | --- |
| Operator | Przykład | Znaczenie |
| = | $x = 10 | $x = 10 |
| += | $x += 5 | $x = $x + 5  $x = 10 + 5 |
| -= | $x -= 5 | $x = 10 - 5 |
| \*= | $x \*= 5 | $x = 10 \* 5 |
| /= | $x /= 5 | $x = 10 / 5 |
| %= | $x %= 5 | $x = 10 % 5 |
| .= | $x .= ”ab” | $x = $x. ”ab” |

## **Operatory inkrementacji i dekrementacji**

Operatory inkrementacji powoduje zwiększenie wartości o jeden. Może występować w postaci przedrostkowej (++$x) lub przyrostkowej ($x++). Operacja $x++ zwiększa wartość zmiennej po jej wykorzystaniu, natomiast ++$x przed jej wykorzystaniem. Oba operatory zwiększają wartość zmiennej, ale nie są równoważne.

Operatory dekrementacji działa analogicznie, tylko zamiast zwiększać wartości zmiennych, zmniejsza je.

# **Stałe**

Wartości nie ulegające zmianie. Podobnie jak zmienne są identyfikowane w skrypcie przez nazwę. Nazwa musi spełniać podobne warunki jak nazwa zmiennej, jednak w stałych nie używa się znaku $. Nazwy pisane są wielkimi literami, zasięg stałych jest globalny i można się odwoływać się do nich w każdym miejscu skryptu. Nie można tworzyć stałych poprzez przypisanie. Do ich definiowania służy funkcja define(), która posiada dwa argumenty: nazwę stałej oraz przypisaną do niej wartość. Stałe raz zdefiniowane nie mogą być zmieniane ani usuwane. Mogą zawierać tylko w wartości skalarne.

Przykład:

|  |
| --- |
| define("CONSTANT", "Hello world."); |
| define("STALA", "Hello world."); |

# **Stałe predefiniowane**

Istnieje grupa stałych predefiniowanych, które towarzyszą każdemu uruchamianemu skryptowi. Część z nich jest dostępna dzięki różnym rozszerzeniom i można z nich korzystać, gdy te rozszerzenia zostały wczytane. Np.

* TRUE – stała zawierająca logiczną wartość prawdy
* FALSE – stała zawierająca logiczną wartość fałszu
* PHP\_VERSION – stała reprezentująca aktualnie wersję praser PHP
* PHP\_OS – stała zawierająca nazwę systemu operacyjnego, na którym uruchamiany jest praser PHP

Istnieje grupa tak zwanych magicznych stałych, które zmieniają się w zależności od miejsca ich użycia. Należą do nich:

* \_\_FILE\_\_ - stała zawierająca nazwę pliku ze skryptem, który jest aktualnie przetwarzany
* \_\_LINE\_\_ - stała zawierająca numer linii w skrypcie, która aktualnie jest przetwarzana
* \_\_DIR\_\_ - stała zawierająca nazwę katalogu pliku
* \_\_FUNCTION\_\_ - stała zawierająca nazwę funkcji
* \_\_CLASS\_\_ - stała zawierająca nazwę klasy
* \_\_METHOD\_\_ - stała zawierająca nazwę metody.

# **Obsługa plików**

## **Otwieranie**

Zanim zaczniemy coś zapisywać, bądź też odczytywać z pliku, musimy go najpierw otworzyć, oraz skojarzyć z jakąś zmienną, z której będziemy później korzystać przy odwoływaniu się do niego. Do otwarcia pliku służy funkcja fopen(). Posiada dwa argumenty. Pierwszy to nazwa pliku, natomiast drugi, to tryb w jakim plik ma być otworzony. To co funkcja zwraca przypisujemy do naszej zmiennej $plik. Czyli np.:

$plik = fopen("plik.txt","w+");

Teraz troszkę o drugim parametrze, może on przyjmować następujące wartości:

|  |  |
| --- | --- |
| a | Otwiera plik do dopisywania danych. Dane będą dodawane na końcu pliku. |
| a+ | Otwiera plik do odczytu i dopisywania danych. Dane będą dodawane do końca pliku. |
| r | Otwiera plik tylko do odczytu (również plików za pomocą protokołu HTTP). |
| r+ | Otwiera plik do odczytu i zapisu. Dane będą dodawane na początku pliku. |
| w | Otwiera plik tylko do zapisu. Jeżeli plik istnieje wszystkie dane zostaną skasowane, jeżeli nie, PHP spróbuje go utworzyć. |
| w+ | Otwiera plik do odczytu i zapisu. Jeżeli plik istnieje wszystkie dane zostaną skasowane, jeżeli nie, PHP spróbuje go utworzyć. |

Możliwe też jest otwarcie pliku z innej strony HTTP,

chyba że dostęp jest zablokowany przez serwer:

$plik = fopen("http://www.example.com/plik.txt", "r");

WIP

<https://www.php.net/manual/en/function.fopen.php>

## **Odczytywanie**

Jeżeli mamy już otworzony plik możemy coś odczytać np. dwuargumentową funkcją fread(). Pierwszy argument to nasza zmienna skojarzona z plikiem, natomiast drugi to liczba bajtów które chcemy odczytać. Funkcja zwraca to, co odczytała. Przykład:

$tresc = fread($plik, filesize('plik.txt'));

$tresc = fread($plik, 100);

## **Zapisywanie**

Podczas zapisywanie trzeba pamiętać, żeby skrypt miał odpowiednie uprawnienia (zwłaszcza w systemach typu Linux). Do zapisu korzystamy z funkcji fputs() lub fwrite() (pierwsza jest aliasem tej drugiej). Pierwszy argument to nasza ukochana zmienną, do której przypisany jest plik, natomiast drugim jest oczywiście to, co chcemy zapisać:

fputs($plik,"A kot ma już dość Ali\n\r");

## **Zamykanie**

Po skończeniu pracy z plikiem możemy go zamknąć:

fclose($plik);

## **Załączanie zewnętrznych plików**

Cały skrypt nie musi znajdować się w jednym pliku. Skrypt można podzielić na części, które będziemy wczytywać, gdy będą nam potrzebne. Służą nam do tego dwie funkcje: **include() i require()** Działanie wydają się mieć identyczne, ale jest między nimi kilka różnic:

* Błąd w include() można wyciszyć stosując @ przed funkcją, require() generuje *fatal error* zatrzymując działanie całego skryptu.
* include() jest wczytywany i załączany, gdy wykonywana jest linia z instrukcją, require() natomiast od razu próbuje wczytać żądany plik, nawet jeśli instrukcja znajduje się w instrukcji warunkowej i może zostać nie wykonana.

# **Instrukcje sterujące**

Instrukcjami sterującymi nazywamy wszystkie polecenia, które pozwalają nam na sterowanie przepływem danych w naszym skrypcie.

Do podstawowych instrukcji zaliczamy:

* Warunkową IF oraz operator warunkowy „?”;
* Switch;
* Pętle FOR, WHILE i DO…WHILE (a także ich pomocnicze BREAK i CONTINUE) oraz FOREACH do przeszukiwania tablic (list itp.);
* Wszelkie funkcje oraz ich składową RETURN;
* Polecenia REQUIRE() i INCLUDE();

## **Warunki**

Często potrzeba wykonać jedną instrukcję programu, gdy pewien warunek

jest spełniony lub inną, gdy ten sam warunek nie jest spełniony.

Służy do tego instrukcja IF ... ELSEIF ... ELSE.

|  |
| --- |
| if($zmienna == 15) ----- Jeśli zmienna jest równa 15… { echo ’Zmienna $zmienna jest równa 15’; ----- …wyświetl dany tekst.  }  else ----- Jeżeli jednak nie jest… { echo ’Zmienna $zmienna nie jest równa 15’; ----- …wyświetl inny tekst. } |

W instrukcji ELSEIF możemy dodać dodatkowy warunek, w którym możemy sprawdzić, czy zmienna równa/nie równa się danej wartości.

|  |
| --- |
| if($zmienna < 15) ----- Jeśli zmienna jest mniejsza od 15… { echo ’Zmienna $zmienna jest mniejsza niż 15’; ----- …wyświetl dany tekst.  }  elseif($zmienna == 15) ----- Jeśli zmienna jest równa 15… { echo ’Zmienna $zmienna jest równa 15’; ----- …wyświetl dany tekst.  }  else ----- Jeżeli jednak jest większa… { echo ’Zmienna $zmienna jest większa niż 15’; ----- …wyświetl inny tekst. |

Można użyć kilku instrukcji ELSEIF w jednym bloku instrukcji IF. Wykonany będzie ten blok, który pierwszy będzie miał wartość TRUE.

## **Operator warunkowy**

Szybkim operatorem warunkowym jest znak zapytania „?”. Jest on dokładnie tym samym co polecenie IF jednak w znacznie uproszczonym stopniu. Bardzo przydatny przy budowaniu krótkich instrukcji, mając proste warunki:

|  |
| --- |
| $a = 5; $odpowiedz = ($a>5) ? ’Większa od 5’ : ’Mniejsza, bądź równa 5’;  echo $odpowiedz; |

$odpowiedz jest zmienną, do której przypiszemy wynik zwracany przez operator „?”. Wyrażenie w nawiasie oznacza nasz warunek. Jeżeli jest TRUE, $odpowiedz przyjmuje wartość pierwszą, czyli ‚Większa od 5′. Jeśli natomiast $a nie będzie większa od 5, $odpowiedz przyjmie wartość drugą (po dwukropku).

## **Instrukcja switch**

W wielu przypadkach istnieje potrzeba porównania jednej zmiennej (lub wyrażenia) z wieloma różnymi wartościami i wykonania różnych fragmentów kodu, w zależności od wyniku porównania tej zmiennej z różnymi wartościami. Do tego właśnie służy instrukcja SWITCH.

Poniżej znajdują się dwa przykłady wykonujące dokładnie to samo, przy czym jeden z nich wykorzystuje serię instrukcji if, zaś drugi używa instrukcji SWITCH:

|  |
| --- |
| switch ($i)     {   case 0:             echo "i jest równe 0";             break;         case 1:             echo "i jest równe 1";             break;         case 2:             echo "i jest równe 2";             break;     } |

=

|  |
| --- |
| if ($i == 0) {     {    echo "i jest równe 0";     }     elseif ($i == 1)     {    echo "i jest równe 1";     }     elseif ($i == 2)     {    echo "i jest równe 2";     } |

## **Pętle**

Pętla WHILE jest najprostszym typem pętli w PHP. Zachowuje się ona identycznie jak jej odpowiednik z języka C.

Znaczenie instrukcji WHILE jest bardzo proste. Nakazuje ona PHP tak długo wykonywać określone instrukcje, jak długo wyrażenie po słowie WHILE ma wartość TRUE. Wartość tego wyrażenia jest sprawdzana za każdym razem na początku wykonywania nowej iteracji pętli, więc jeśli jego wartość zmieni się w trakcie wykonywania instrukcji, wykonanie całej pętli nie skończy się do momentu zakończenia całej iteracji. Jedna iteracja jest to jednokrotne wykonanie wszystkich instrukcji w pętli. Jeżeli wyrażenie logiczne ma wartość FALSE już na samym początku, instrukcje wewnątrz pętli nie będą w ogóle wykonane.

Poniższy przykład wyświetla 10 znaków #:

|  |
| --- |
| $i = 1;     while ($i <= 10)     { echo ’#’;  $i++;      } |

Pętla DO…WHILE zachowuje się bardzo podobnie do pętli WHILE, z wyjątkiem tego, że wartość wyrażenia logicznego sprawdzana jest na końcu iteracji, a nie na początku. Wynikającą z tego główną różnicą jest to, że pierwsza iteracja w pętli DO…WHILE na pewno zostanie wykonana (gdyż wyrażenie logiczne będzie sprawdzone dopiero na koniec iteracji). Natomiast w pętli WHILE, gdzie wyrażenie logiczne jest sprawdzane na początku iteracji, może dojść do sytuacji, że pętla w ogóle nie zostanie wykonana, jeśli to wyrażenie będzie miało wartość FALSE od początku, np:

Pętla wykona się tylko raz

|  |
| --- |
| $i = 0;     do {        echo $i;     } while ($i>0); |

Pętla nie wykona się ani raz

|  |
| --- |
| $i = 0;    while ($i>0)  { echo $i;  } |

Pętla for jest najbardziej skomplikowanym rodzajem pętli w PHP. Zachowuje się identycznie jak jej odpowiedniki z C++. Jej składnia wygląda następująco:

|  |
| --- |
| for (wyrażenie1; warunek; wyrażenie2) {instrukcje} |

wyrażenie1 jest wykonywane tylko raz, na początku pętli.

Na początku każdej nowej iteracji, sprawdzany jest warunek. Jeśli jego wartość jest TRUE – pętla wykonuje się dalej, jeżeli FALSE – pętla zostaje przerwana.

Na końcu każdej iteracji zostaje wykonane wyrażenie2.

Ten przykład wyświetla 10 znaków #.

|  |
| --- |
| for ($i = 1; $i <= 10; $i++)     {   echo ’#’;     } |

BREAK przerywa wykonywanie aktualnej instrukcji FOR, FOREACH, WHILE, DO…WHILE, SWITCH.

CONTINUE używany jest wewnątrz instrukcji pętli do przerwania wykonywania danej iteracji pętli i rozpoczęcia kolejnej iteracji.

# Funkcje

Funkcje w PHP są częściowo podobne do funkcji matematycznych. Mogą przyjmować argumenty oraz zwracać wartości. Nie jest to jednak wymogiem. Funkcją nazywamy napisany przez nas kod, zamknięty w nawiasy klamrowe, poprzedzony słowem kluczowym FUNCTION oraz unikatową nazwą. Nazwy funkcji nie mogą się powtarzać. Nie można również używać nazw istniejących już poleceń, takich jak WHILE, IF, ECHO itp.

Funkcje możemy podzielić na cztery kategorie:

* Bezargumentowa, niezwracająca wartości,
* Bezargumentowa, zwracająca pewną wartość,
* Przyjmująca argumenty, niezwracająca wartości,
* Przyjmująca argumenty oraz zwracająca wartość.

Funkcja zwracająca wartość:

Żeby zrozumieć zasadę działania funkcji zwracającej wartość, wyobraźmy sobie, że funkcja to taka zmienna, której wartość zmienia się dynamicznie. Co za tym idzie, funkcję możemy przypisać zmiennej. Dodatkowo możemy sprawdzić, czy funkcja jest mniejsza lub większa od pewnej liczby.

Deklaracja Funkcji

|  |
| --- |
| function dodawanie()  {  $a=(5+6);  return $a;  } |

Wywołanie funkcji w skrypcie

|  |
| --- |
| echo dodawanie(); |

Wynik wykonania

|  |
| --- |
| 11 |

# **Tablice**

Tablice są bardzo specyficznym typem zmiennych – są to, najprościej mówiąc, zmienne zawierające w sobie uporządkowany zbiór zmiennych. Do zmiennych tych uzyskuje się dostęp przez liczbę w nawiasie kwadratowym podane bezpośrednio po nazwie zmiennej – tablicy. Liczba ta to tak zwany indeks – numer kolejny zmiennej w tablicy. Tak samo przypisuje się wartość do tablicy.

Deklaracja tablicy

|  |
| --- |
| $tablica[0] = „Pierwsza”;  $tablica[1] = „Druga”;  $tablica[2] = „Trzecia”; |

Wywołanie w skrypcie

|  |
| --- |
| echo $tablica[1].”wartość”; |

Wynik wywołania

|  |
| --- |
| Druga wartość |

Deklarować tablice możemy również w taki sposób:

|  |
| --- |
| $tablica = array („Pierwsza”, „Druga”, „Trzecia”, 4,5); |

Deklaracja tablicy

|  |
| --- |
| $tablica[0] = „Pierwsza”;  $tablica[1] = „Druga”;  $tablica[2] = „Trzecia”; |

Wywołanie w skrypcie

|  |
| --- |
| print\_r($tablica); |

Wynik wywołania

|  |
| --- |
| [0] => Pierwsza  [1] => Druga  [2] => Trzecia |

Do wyświetlania tablic możemy użyć polecenia PRINT\_R()

Pokaże nam ono tablicę w sposób „ładny” dla oka.

Tablica w PHP nie jest tablicą asocjacyjną, a więc zestawem par „klucz-wartość” czyli taką nieskończoną tabelą o dwóch kolumnach.

|  |  |
| --- | --- |
| Klucz | Wartość |
| żółty | banan |
| czerwona | truskawka |
| zielone | kiwi |

Możemy indeksować ją w następujący sposób:

|  |
| --- |
| $x = $tablica[’czerwona’]; ----- $x ma wartość ’truskawka’ |

Można też tradycyjnie:

|  |
| --- |
| $x = $tablica[0]; ----- $x ma wartość ’banan’ |

PHP ofertuje cały zestaw funkcji służących do sortowania tablic. Są to: asort(), arsort(), ksort(), rsort(), sort(), uasort(), sort(), i uksort().

* asort() - sortuje tablice asocjacyjne zachowując przypisanie kluczy do wartości.
* arsort() - sortuje w odwrotnej kolejności tablice asocjacyjne zachowując przypisanie kluczy do wartości. Funkcja prawie identyczna jak poprzednia, tyle że dane sortowane są "od tyłu".
* ksort() - sortuje tablice asocjacyjne według kluczy.
* rsort() - sortuje zwykłe tablice (nie asocjacyjne) w odwróconej kolejności
* sort() - sortuje zwykłe tablice (nie asocjacyjne) w kolejności alfabetycznej
* uasort() - funkcja sortująca tablice asocjacyjne za pomocą zdefiniowanej przez użytkownika funkcji porównującej elementy (nazwa funkcji jest podawana za pomocą drugiego parametru
* usort() - funkcja sortująca zwykłe tablice za pomocą funkcji zdefiniowanej przez użytkownika
* uksort() - funkcja sortująca tablice asocjacyjne według klucza za pomocą funkcji zdefiniowanej przez użytkownika.

UWAGA na kod ASCI najpierw są znaki specjalne, cyfry, duże litery a następnie małe litery

# **Czas**

WIP

formatowanie daty/czasu

http://www.php.net/manual/pl/function.date.php

date("Y-m-d"); // rok-miesiąc-dzień date("d-m-Y");

time ();

Zwraca aktualny czas, podawany jako liczba sekund, które upłynęły od uniksowej Epoki (1 stycznia 1970 00:00:00 GMT).

<?php

$data=date("Y-m-d");

$czas=date("H:i");

$datapl=date("d-m-Y");

echo "Stronę wyświetlono dnia $data, po polsku: $datapl o godzinie $czas<br>";

echo "jest czas:".time ()."<br>";

?>

# **Cookie**

Aby wysłać cookie lub zainicjować sesje musisz to zrobić zanim wyślesz nagłówki serwisu, a zatem przed znacznikiem <html> i <head> w dokumencie.

Cookies i sesje to bardzo często stosowane elementy języka PHP, które pomagają przechowywać dane podczas wizyty użytkownika na stronie, a także po zakończeniu takiej wizyty. Dzięki nim działają systemy logowania, liczniki itp.

Setcookie() może przyjąć 6 argumentów. Jednak wymagane jest podanie tylko pierwszego, którym jest nazwa ciasteczka.

// utworzenie ciasteczka

setcookie($nazwa, $wartosc, $koniec, $sciezka, $domena, $bezpieczne);

setcookie('ciasteczko', 'wartość ciasteczka', time()+3600, '', <http://loclhost/>');

funkcja ta posiada sześć argumentów - nie wszystkie są obowiązkowe Pierwszą wartością jaką podajemy jest nazwa ciasteczka, czyli zmiennej,

Następnym elementem jest wartość ciasteczka, czyli zawartość zmiennej.

Trzecim argumentem jest tzw. data ważności ciasteczka.

Czwartym argumentem jest ścieżka, do której odwołuje się cookie.

Ostatnim opisywanym przeze mnie argumentem będzie domena, czyli dla jakiej domeny będzie działało cookie. Jest to zabezpieczenie w przypadku, gdy ktoś z innej domeny chce pobrać dane z twojego serwera.

Bardzo ważne jest także to, że ciasteczko widziane jest przez serwer dopiero po przeładowaniu przeglądarki, więc w różnych sytuacjach okaże się, że wartość cookie będziesz musiał najpierw zapisać w zmiennej i wysłać ciasteczko o takiej samej nazwie, a wartość podać poprzez zmienną, np.:

$ciasteczko = 'jakas wartosc';

setcookie('ciasteczko', $ciasteczko);

Każde ciasteczko jest przechowywane w tablicy globalnej $\_COOKIES.

# **Sesje**

Zasada działania sesji jest bardzo podobna do cookie.

Największa różnica polega na tym, że sesja przechowuje dane na serwerze, a cookie na komputerze klienta.

Aby zarejestrować daną sesyjną będziemy się posługiwać zmienną $\_SESSION,

Dodatkowo będziemy korzystać jeszcze z funkcji session\_start(), session\_unset(), session\_destroy(), których całkowite działanie opsiano w dokumentacji PHP.

* session\_start() inicjalizuje sesję
* session\_unset() usuwa wszystkie zmienne sesyjne
* session\_destroy() niszczy wszystkie dane sesyjne

Nową rzeczą jakiej nie było w przypadku cookie jest zmienna SID, która zawiera ID bieżącej sesji. Gdyby jej nie było, to po każdym przeładowaniu przeglądarki użytkownik otrzymywał by nową sesję i ich stosowanie okazało by się niepotrzebne.

Musisz zapamiętać jednak jeszcze jedną bardzo ważną związaną z tym rzecz. A mianowicie przy inicjalizacji sesji zmienna SID jest zawsze dołączana do adresu metodą GET i automatycznie jest generowane ciasteczko z tą zmienną. Jeśli jednak użytkownik ma wyłączoną obsługę cookie to zmienna ta przepada. W celu utrzymania w dalszym ciągu takiej sesji, należałoby dołączyć do każdego hiperłącza i formularza na stronie taką właśnie zmienną.