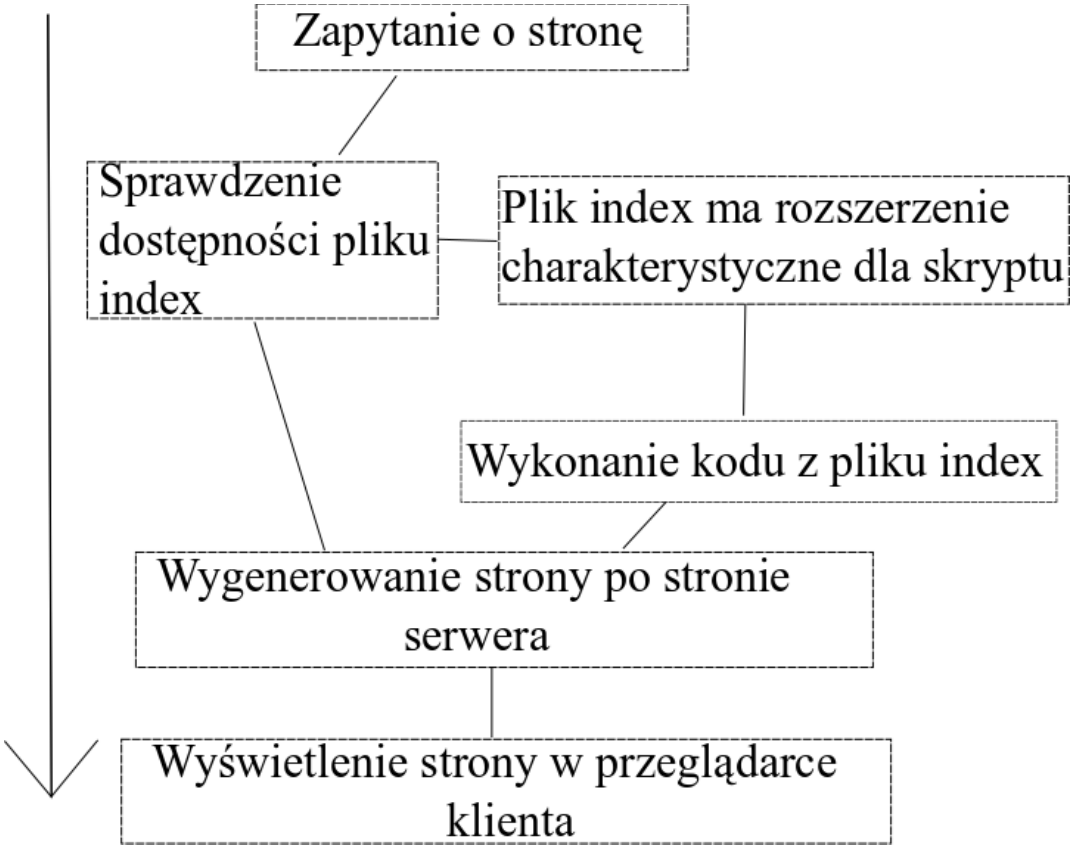
Instrukcja 1

1. Zasada działania skryptów po stronie serwera.

Niektóre zadania, jak pobieranie informacji z baz SQL, przetwarzanie informacji, przesyłanie informacji pomiędzy podstronami czy generowanie całych stron/grafiki dla stron WWW niemożliwe jest tylko przy wykorzystywaniu skryptów po stronie klienta. O ile na chwilę obecną skrypty JavaScript umożliwiają tworzenie grafiki w czasie rzeczywistym, o tyle działania takie jak przetwarzanie informacji, sprawdzanie haseł bądź odczyt/zapis w bazach danych poprzez skrypty po stronie klienta byłyby co najmniej nierozsądne ze względów bezpieczeństwa. Należy bowiem pamiętać, że skrypt musi „znać" hasło chroniące dostęp do baz danych; skrypt po stronie klienta musiałby to hasło czytać – mieć je umieszczone w kodzie bądź pobierać z pliku na serwerze. Każda z tych opcji zagrożona jest „przeczytaniem" tego hasła przez osoby niepowołane. Inną kwestią jest zablokowany przeważnie dostęp do bazy danych od strony sieci (dostęp do niej mają jedynie aplikacje po stronie lokalnej). Ponadto skrypty serwerowe przenoszą ciężar przetwarzania informacji na sprzęt serwera – nawet klienci ze słabym sprzętem mogą bezproblemowo korzystać z tak przygotowanej aplikacji internetowej.



Rys 1. Uproszczony schemat żądania strony WWW.

Z powyższego schematu łatwo jest wywnioskować, że tak zabezpieczony kod nie jest widoczny (w przeciwieństwie do kodu znaczników lub skryptów JS) dla użytkownika docelowego. Można zatem wprowadzać do niego pewne informacje, których nie powinna poznać żadna z korzystających osób. W związku z tym tak pisane aplikacje mogą „posiąść" wiedzę o loginach i hasłach np. do kont pocztowych (w celu automatycznego wysyłania poczty na wskazany adres), baz danych, a także łączyć się z innymi usługami dostępnych jedynie na serwerze (nieudostępnionych dla sieci Internet).

2. Zadania i funkcje języków skryptowych po stronie serwera.

- nadzorowanie transakcji z serwerami baz danych

Dostęp do serwerów baz danych musi być jak najbardziej ograniczany. W dzisiejszych czasach, niezależnie od typu samej bazy (SQL, XML, JSON), niemal wszystkie dane przechowuje się właśnie w bazach danych. Na chwilę obecną możliwe jest nawet umieszczanie w nich plików graficznych bądź wideo, co zresztą bardzo chętnie wykorzystują programiści – tego typu działanie potrafi znacznie zmniejszyć zapotrzebowanie na przestrzeń dyskową na serwerze. Jednak najbardziej krytycznymi, pod względem bezpieczeństwa, są informacje osobiste o użytkownikach aplikacji, jak dane personalne, loginy oraz hasła. Dlatego same bazy danych nie powinny być widoczne dla kogokolwiek w sieci Internet – stanowiłyby „łakomy" kąsek dla wielu przestępców (którzy mogliby te dane wykorzystywać do podszywania się pod daną osobę). Skrypty serwerowe pełnią zatem rolę „pomostu" pomiędzy stroną graficzną aplikacji (w naszym wypadku strony WWW), a zatem samym użytkownikiem, a bazą informacyjną tejże aplikacji (w naszym wypadku znajdującej się właśnie po stronie serwera). Przeważnie większość języków posiada wbudowaną obsługę poleceń większości popularnych magazynów danych. Coraz częściej różnice pomiędzy poszczególnymi typami baz mogą być dla programisty niezauważalne – funkcje połączeń oraz zapytań realizowane są „wysokopoziomowo". Oznacza to, iż podczas łączenia programista musi podać jedynie nazwę bazy danych, a funkcje same „przetłumaczą" zapytania na odpowiedni język dla konkretnego systemu przechowywania danych.

- generowanie dynamicznych treści stron WWW

Chociaż coraz częściej do generowania treści strony można wykorzystać język JavaScript (generowanie następuje w czasie rzeczywistym – bez przeładowania przeglądanej strony) to należy pamiętać, iż jest to zarezerwowane tylko do prostych zastosowań, jak np. zmiana imienia osoby odwiedzającej stronę albo kwot wydanych na zakupy w sklepie internetowym. Jeżeli natomiast potrzebne jest wygenerowanie całego dokumentu strony (wraz z całym kodem HTML) to skrypty po stronie serwera nadają się do tego idealnie. Mogą one bowiem łączyć ze sobą zdobyte informacje np. z pól formularzy WWW (które niedawno wprowadził użytkownik) z informacjami zapisanymi po stronie serwera (takich jak chociażby pliki xml). Ponadto mogą korzystać z gotowych wzorców stron znajdujących się w odpowiednich plikach bądź wygenerowanej dynamicznie grafiki (opisana poniżej).

- generowanie grafiki

Chociaż HTML5 posiada możliwość generowania w czasie rzeczywistym grafiki na stronę WWW to należy pamiętać iż w przeważającej części tego typu generowanie nie jest trwałe – tak stworzona grafika może zostać zapisana jedynie po stronie klienta (zapis na serwer jest znacznie ograniczony). Natomiast języki serwerowe mają możliwość generowania pożądanej grafiki (np. dodanie znaków wodnych do istniejących już zdjęć) i zapisania ich na serwerze. Dzięki temu raz utworzona grafika może być później wykorzystywana wielokrotnie, a użytkownik/użytkownicy nie muszą tracić czasu w oczekiwaniu na końcowy efekt (w JavaScript przy każdym odwiedzeniu strony z danym zdjęciem wymagane byłoby ponowne nałożenie znaku wodnego; ponadto nieuczciwe osoby mogłyby zablokować nakładanie go poprzez odpowiednie modyfikacje kodu).

- przetwarzanie informacji z formularzy

JavaScript potrafi przechwycić dane wprowadzone przez użytkownika do pól formularzy. Jej problem polega jednak na tym iż danych tych nie jest w stanie umieścić na serwerze ani w bazie danych po stronie serwera (może jedynie zapisać je na maszynie klienta). Z kolei języki skryptowe

osadzone na serwerze bez problemu mogą dane te przechwytywać i przechowywać zarówno w pamięci podręcznej hosta, jak i w plikach bądź bazach danych. Ponadto dane te mogą być porównywane z już istniejącymi, łącznone, dodawane bądź usuwane.

- generowanie dokumentów HTML bez możliwości ich modyfikacji/blokowania po stronie klienta

Języki skryptowe, jak to już zostało wielokrotnie pokreślone w niniejszej instrukcji, potrafią wygenerować pełny kod żądanej witryny WWW – włączając w to znaczniki HTML, funkcje JavaScript oraz CSS. Oznacza to, że dla każdego użytkownika można stworzyć stronę spersonalizowaną pod niego, włączając w to parametry i zmienne JavaScript czy całe szablony CSS (przykładowo jeżeli użytkownik chce posiadać stronę z motywem koloru zielonego oraz wyświetlaniem samej dany, bez zegara na belce menu).

- wykonywanie dodatkowych poleceń i aplikacji niezależnych od systemu użytkownika

Programista skryptów może, znając konfigurację oraz dostępne oprogramowanie serwera stron WWW, wykonać dowolny program po stronie serwera, a jego wynik wykorzystać przy tworzeniu docelowej strony. Oczywiście należy mieć na uwadze iż niewskazane jest wykonywać aplikacji graficznych. Po pierwsze serwery stron (poza wyjątkiem – Windows Server) nie posiadają powłok graficznych – a co za tym idzie aplikacji „okienkowych”. Po drugie aplikacje graficzne nie mogą zostać wyświetlone poprzez serwer stron – aby tego dokonać programista musiałby pisać tzw. aplikację proxy, przekazującą obraz tego co dzieje się na serwerze (skrypty w przeważającym stopniu nie dają możliwości napisania tak zaawansowanej aplikacji WWW). Po trzecie zaś skrypty są w stanie przechwycić jedynie komunikaty z konsoli; aplikacje graficzne natomiast nie „wypisują” przeważnie żadnych komunikatów tekstowych w konsoli (poza ewentualnymi błędami w kodzie – tzw. komunikaty debuggera).

- niezależność od wydajności sprzętu użytkownika docelowego

Projektując skrypty serwerowe programista nie musi brać pod uwagę różnorodności sprzętowej potencjalnych odbiorców jego aplikacji. W przeciwieństwie bowiem do skryptów klienckich, które nierzadko wymagają wydajnego sprzętu do komfortowego przeglądania zawartości witryny (jako przykład niech posłużą strony „multimedialne” w postaci youtube, vimeo, aplikacje flashowe oraz HTML5 takie jak projekcje 3D lub gry przeglądarkowe), opisywane skrypty wymagają jedynie dużej wydajności serwera; to właśnie na serwerze spoczywa ciężar wykonywanych obliczeń/generowania w czasie rzeczywistym odpowiedniej zawartości docelowej witryny WWW.

- możliwość tworzenia/utrzymywania sesji użytkownika

Cecha ta jest w zasadzie kluczowa dla 90% obecnych stron/aplikacji internetowych. Kto bowiem z nas wyobraża sobie dzisiaj sieć bez stron forumowych, społecznościowych, bez płatności internetowych i innych usług, do których wymagana jest autoryzacja? Na chwilę obecną tylko regularne aplikacje oraz aplikacje wykorzystujące języki skryptowe po stronie serwera umożliwiają autoryzowanie, tworzenie, utrzymywanie, zarządzanie i kasowanie sesji użytkownika, dzięki której nie musimy przykładowo po każdorazowym przeładowaniu strony podawać loginu i hasła.

- wiele innych

Ostatecznie tylko od projektanta aplikacji zależy w jaki sposób wykorzysta dostępne technologie tworzenia stron internetowych. Należy jednak pamiętać by nie wykorzystywać którejkolwiek technologii „na wyrost” jak to ostatnimi czasy sugerują niektóre firmy czy instytucje. Chociaż skupienie się tylko na jednym rozwiązaniu wydaje się być sensowne (nie trzeba poświęcać czasu na

dodatkową naukę) to jednak dobrze jest mieć na uwadze, iż dany język nie zawsze będzie najwydajniejszy do wszystkich zadań.

3. Przegląd dostępnych języków skryptowych.

Na chwilę obecną programiści mogą wybierać z co najmniej kilku rozwiniętych i dobrze udokumentowanych technologii języków skryptowych.

1) PHP – jeden z najpopularniejszych języków skryptowych wykorzystywanych przy tworzeniu stron. Posiada stosunkowo prostą i łatwą do nauki składnię. Działa niemal na każdej konfiguracji systemowej – Windows, Linux, Mac OS, FreeBSD, Solaris Novell i wielu innych. Rozwijany jest na zasadach otwartego źródła przez PHP Group. Co istotne stosowanie go przy projektach komercyjnych nie pociąga za sobą żadnych kosztów (poza kosztami utrzymania serwera). Ponadto jego skryptu z powodzeniem można wykorzystywać do zarządzania systemem operacyjnym – jest więc językiem uniwersalnym. Jedną z poważniejszych jego wad jest jego przeładowanie – kilka różnych poleceń pozwala wykonać to samo zadanie. Kolejna jest nieefektywność – napisane w nim skrypty są najczęściej zasobożerne, przez co programista wybierający go powinien dbać o maksymlane optymalizowanie swojego kodu.

2) Perl – jeden ze starszych języków skryptowych. Został napisany z myślą o zaawansowanej pracy z plikami tekstowymi. Stąd też posiada wiele funkcji ułatwiających przeszukiwanie, porządkowanie, zamienianie czy usuwanie wybranych fragmentów tekstu. Na chwile obecną stanowi jeden z języków konsoli wykorzystywanych w systemach Unix/Unix-like, rzadziej w Windows. Ponadto znalazł zastosowanie także przy pisaniu stron WWW; dzięki społeczności wydane zostały do niego specjalne biblioteki, dzięki którym za pomocą niewielkiej ilości linii kodu można zarządzać sesjami użytkowników, tworzyć połączenia do baz danych, komunikować się z aplikacjami systemowymi, aplikacjami typu serwer-klient itp. Jego zaletą jest lepsza optymalizacja na poziomie parsera oraz klarowność poszczególnych instrukcji.

3) Python – język bardzo dynamicznie rozwijany przez Python Software Foundation. Tak jak dwa wcześniej wspomniane języki również jest darmowy, posiada otwarty kod źródłowy oraz rozpowszechniany bez opłat licencyjnych. Jako jeden z nielicznych języków wysokopoziomowych może poszczycić się bardzo dobrą efektywnością i wydajnością. Dzięki prostej i intuicyjnej składni zyskuje coraz większą popularność jako język uniwersalny – pozwala on bowiem wykonywać skrypty systemowe, strony internetowe jak i pisać kompletne aplikacje graficzne (okienkowe). Na chwilę obecną coraz więcej firm hostingowych udostępnia jego parser w ramach serwera WWW. Najpopularniejszym zestawem ułatwiającym pisanie stron internetowych przy wykorzystaniu Pythona jest framework Django.

4) Ruby – kolejny z interpretowanych języków. Język ten jest w pełni obiektowy; oznacza to, iż nawet pojedyncza instrukcja wykonywana w nim należy do jakiegoś obiektu. Jego twórca, Yukihiro Matsumoto, wzorował się na wielu innych językach, takich jak CLU, Eiffel, Lisp, Perl, Python czy Smalltalk. Na chwilę obecną język ten jest coraz chętniej wybierany przez programistów ze względu na posiadanie między innymi tzw. odśmiecacza pamięci, możliwośi dynamiczniej zmiany ciała klasy działającego obiektu, modułowości, łatwego przekazywania funkcji w postaci parametrów innych funkcji czy operowaniu na liczbach całkowitych o dowolnym rozmiarze. Ponadto od czasu ukazania się frameworka Ruby on Rails poziom skomplikowania pisania w nim całych stron internetowych jest porównywalny do PHP czy JavaScript, przy czym parser Ruby jest znacznie efektywniejszy od wyżej wspomnianych. Język ten posiada bardzo duży potencjał rozwojowy, który został dostrzeżony przez wiele firm hostingowych – jego kompilator jest coraz częściej dołączany do podstawowej konfiguracji serwerów WWW.

4. Podstawowe cechy języka PHP

Jak to zostało już wcześniej wspomniane język PHP to wciąż jeden z najpopularniejszych języków skryptowych wykorzystywanych do tworzenia stron WWW. Pomimo jego wad i słabej optymalizacji nadal jest najchętniej wybieraną technologią. Zawdzięcza to przede wszystkim swojej prostocie, mnogości aktualnie istniejących już aplikacji (m. in. do obsługi baz danych przez przeglądarkę) i frameworks (PEAR, Zend), łatwości nauki (zawiera cechy kilku innych popularnych języków programowania, przez co programistom łatwo jest się w nim „odnaleźć"). Oczywiście nic nie stałoby na przeszkodzie pozostawić obecne aplikacje napisane w PHP i doinstalować obsługę innego języka (jest to jak najbardziej technologicznie możliwe). Niestety administratorzy serwerów (przeważnie tych niezależnych bądź dedykowanych) boją się (i nie są to obawy bezpodstawne) dodawać nowe składniki serwera ponieważ mogłoby to zdestabilizować ich działanie bądź umożliwić atakującemu włamanie (każdy nowy składnik systemu serwerowego wymaga dodatkowych zabezpieczeń – ZAWSZE).

Podstawowe właściwości języka PHP:
- język proceduralny – PHP nawet w chwili obecnej umożliwia pisanie programów bez wykorzystywania obiektów (bez tworzenia klas; w kodzie mogą znaleźć się same stałe, zmienne oraz funkcje wykonywane rozkaz po rozkazie)
- język obiektowy – nowsze wersje PHP umożliwiają pisanie własnych klas, dzięki czemu kod pisanej aplikacji staje się bardziej przejrzysty i czytelny dla zespołu programistycznego; poza tym na dzień dzisiejszy niewiele aplikacji pisanych w językach wysokiego poziomu kodowanych jest wedle stylu proceduralnego
- dynamiczne typy zmiennych – PHP, podobnie jak JavaScript, nie wymaga od programisty podawania typu dla zmiennej; sam dba o interpretowanie typu na podstawie jej zawartości
- tworzenie stałych – prócz zmiennych PHP posiada możliwość tworzenia stałych; stałe przez cały okres działania programu posiadają jedną stałą przypisaną na sztywno wartość. Stałe są szczególnie przydatne w dużych projektach, w których wielokrotnie trzeba wstawiać konkretną wartość
- pętle warunkowe – pętle te działają identycznie jak w przypadku skryptów JavaScript
- operacje logiczne – operacje logiczne działają identycznie jak w przypadku JavaScript
- pętle – pętle (zarówno for jak i while, do...while) działają identycznie jak w przypadku JavaScript
- zmienne środowiskowe – PHP umożliwia dostęp do wielu tzw. środowiskowych zmiennych, zawierających zawórno informacje o serwerze, na którym pracuje (data, wolne zasoby, typ serwera itp.) jak i o użytkowniku docelowym (adresy IP, wykorzystywana przeglądarka, system itp.) Ponadto do zmiennych środowiskowych można także zaliczyć zmienne przechowujące dane z formularzy (metody POST, GET)
- sterowniki baz danych – są to zestawy gotowych funkcji (bądź klas) ułatwiające obsługę wielu typów baz danych (niemal wszystkich); na chwilę PHP posiada sterowniki uniwersalne – są to klasy uniwersalnych metod pozwalających operować na wielu typach baz danych przy użyciu tych samych metod (tzw. warstwa abstrakcyjna). Programistę nie interesuję „tłumaczenie" funkcji, które napisał na polecenia zrozumiałe dla poszczególnych typów baz danych
- modularność – każda instalacja PHP pozwala w dowolnym momencie dołączać (instalować) do siebie (bądź częściej włączać/wyłączać) nowe moduły i funkcje parsera. Przykładowo modułem w PHP jest obsługa bazy danych (i poszczególnych typów – osobny moduł odpowiada za połączenia z MySQl, inny MongoDB itd.), wysyłanie poczty (najczęściej wyłączana funkcja mail() ), obsługa wzorców i wyrażeń regularnych itd.

5. Przykłady wykorzystania skryptów PHP.

W przeciwieństwie do statycznych stron pisanych w HTML, skryptów PHP nie można bezpośrednio uruchomić przez przeglądarkę. Gdy spróbujemy w ten sposób wywołać napisany przez nas kod w dowolnej przeglądarce to po prostu zobaczymy wyświetlony kod PHP -

przeglądarka potraktuje <?php ?> jako nieznane znaczniki HTML. Jak już wielokrotnie zostało to wspomniane, uruchomienie tak napisanego skryptu wymaga umieszczenia go na serwerze (na wcześniejszych zajęciach przedstawiony został serwer XAMPP dla Windows/Linux). Po umieszczeniu skryptu w odpowiednim katalogu i uruchomieniu serwera można sprawdzić efekt jego działania wystarczy w dowolnej przeglądarce wpisać adres docelowy według schematu:

localhost/nazwa_katalogu/nazwa_skryptu.php

Localhost to nazwa własna komputera (każdy komputer ma taką własną, zastrzeżoną nazwę); nazwa_katalogu to katalog, do którego skopiowaliśmy nasz skrypt (jeżeli skopiowaliśmy go do głównego katalogu htdocs to pomijamy ten człon adresu).

UWAGA 1: jeżeli nasz skrypt będzie nazywał się: index.html, index.htm, index.php, index.php4, index.php5, index.xhtml itd. to serwer wykona go automatycznie gdy podamy tylko nazwę katalogu! Nie potrzeba wtedy w pasku adresu podawać nazwy skryptu.

UWAGA 2: skrypt może się wykonać a my możemy ujrzeć tylko białą, pustą stronę. Stanie się tak wtedy kiedy skrypt nie posiada żadnej funkcji wypisującej tekst. Nie musi to oznaczać błędu!

Najprostszym skryptem, który serwer wykona będzie:

```
<?php
echo 'Tekst sparsowany przez PHP';
```

Plik trzeba zaopatrzyć w rozszerzenie php; w przeciwnym wypadku serwer nawet nie „przejrzy” pliku na potencjalne wystąpienia wstawek php.

Przykład kodu z dodanymi komentarzami:

```
<html>
<?php
/*To jest komentarz wieloliniowy
Tego typu komentarze umieszcza się przeważnie na początku pliku ze
skryptem – opisuje się jego działanie, przykładowe wyniki działania,
często też podaje się autora skryptu wraz z licencją, na jakiej
upublicznia się zawarty niżej kod */

//komentarz jednoliniowy – poniżej funkcja echo wypisze nam na ekranie 'Domyślna treść'
echo 'Domyślna treść';

$a = 15; //tak również można dodać komentarz w jednej linii

$b += $a * 4; #jaki będzie wynik tego wyrażenia? (komentarz w stylu Perla)

echo $b;
?>
<p>Kod PHP zakończył się!</p> //tutaj już komentarz PHP nie zadziała i zostanie wyświetlony

</html>
```

Instrukcje warunkowe:

```
if ($a == $c) {
//tutaj będzie kod odpowiedzialny wykonanie warunku gdy $a będzie równe $c
}
else {
	if ($c == $b) {
	//warunki można zagnieżdżać teraz ten warunek się wykona gdy $a nie będzie równe $c
	}
	elseif($b == $a) {
	//jeżeli nie tamten warunek to może ten?
	}
	elseif($a > $c) {
	//.....
	}
	else {
	//można by znowu zagnieździć warunek; jednak to może spowodować nieczytelność kodu
	//poza tym z każdym zapętleniem program stanie się coraz wolniejszy!
	//dlatego zawsze warto rozważyć na ile możemy uprościć strukturę warunkową!
	}
}
//dalszy kod

switch($a) {
case 0: //można też, zamiast używania struktur if...else zastosować strukturę switch
	break;
case 1: //chociaż bardziej przejrzysta to należy pamiętać, że jest ona bardziej złożona obliczeniowo
	break;
case 2: //trzeba w niej również pamiętać o tym, by po kodzie który ma się wykonać dla danego
warunku
case 3: //podać rozkaz break; inaczej, gdy np. wartość $a będzie równa 2,a nie będzie
wspomnianego
//rozkazu to wykona się także kod dla wartości 3 – bo pętla warunkowa nie będzie wiedzieć iż to
jest nowy warunek!!
	break;
case 4: //tym sposobem można oprogramować dogodne nam przypadki
	break;
default: //jeżeli chcemy akcji domyślnej to podajemy ją właśnie tutaj, na końcu przy default
}

($a >= $b) ? /*$a większe od bądź równe $b*/ : /*jednaj jest mniejsze...*/
```

Instrukcje sterujące:

UWAGA! Duże litery (A, B itd.) oznaczają DOWOLNĄ wartość (czyli trzeba za nie podstawić jakąś konkretną wartość, np. 12, 3 itd.)

```
$c = A;
for ($i = B; $i < $c; $i++) {
//kod zawarty w tej pętli będzie wykonywał się do czasu, gdy $i będzie mniejsze od $c. Gdyby np.
$i było 0, a $c 5 to kod wykonałby się dokładnie 5 razy (0..4), po czym program zakończy pętlę
}
```

```
for ($i = C; $i > $c; $i—) {
//tutaj pętla odwrotna; $i będzie zmniejszane aż będzie mniejsze od $c;
}

while ($c > 0) {
//kod który ma się powtarzać do czasu, gdy $c będzie większe od zera
//WAŻNE! Jeżeli w funkcji while nie będzie instrukcji zmniejszania wartości $c stanie się ona funkcją zapętloną! Będzie się wykonywać w nieskończoność!
}

do {
//tutaj warunek do wykonywania
} while ($c > 0);
```

Instrukcja foreach:

```php
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$value) {
    $value = $value * 2;
}
// $arr is now array(2, 4, 6, 8)
unset($value); // usunięcie odniesienia do ostatniego elementu tablicy

//nie zadziała
foreach (array(1, 2, 3, 4) as &$value) {
    $value = $value * 2;
?>
```

```php
<?php
$array = [
    [1, 2],
    [3, 4],
];

foreach ($array as list($a, $b)) {
    // $a contains the first element of the nested array,
    // and $b contains the second element.
    echo "A: $a; B: $b\n";
}
?>
//A: 1; B: 2
//A: 3; B: 4
```

Podobnie jak w przypadku JavaScript bądź CSS kod PHP można rozłożyć na kilka plików. Plików tych nie podłącza się jednak przy pomocy znaczników HTML, a odpowiednich instrukcji PHP. Poniższy przykład pokazuje jak podłączyć dodatkowe pliki do głównego skryptu:

vars.php

```php
<?php

$color = 'green';
$fruit = 'apple';

?>
```

test.php

```php
<?php

echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

```php
<?php

function foo()
{
    global $color;

    include 'vars.php';

    echo "A $color $fruit";
}

/* vars.php is in the scope of foo() so     *
* $fruit is NOT available outside of this  *
* scope.  $color is because we declared it *
* as global.                               */

foo();                    // A green apple
echo "A $color $fruit";   // A green

?>
```

```php
<?php

/* This example assumes that www.example.com is configured to parse .php
* files and not .txt files. Also, 'Works' here means that the variables
* $foo and $bar are available within the included file. */

// Won't work; file.txt wasn't handled by www.example.com as PHP
include 'http://www.example.com/file.txt?foo=1&bar=2';

// Won't work; looks for a file named 'file.php?foo=1&bar=2' on the
// local filesystem.
include 'file.php?foo=1&bar=2';

// Works.
include 'http://www.example.com/file.php?foo=1&bar=2';

$foo = 1;
$bar = 2;
include 'file.txt';  // Works.
include 'file.php';  // Works.

?>
```

Prócz podstawowej funkcji include PHP umożliwia dodawanie plików źródłowych (z kodem) za pomocą 3 innych funkcji:
• require działa niemal identycznie jak include; różnica polega na tym, że gdy dołączanego pliku nie będzie to zostanie zgłoszony błąd PHP (skrypt się nie wykona)
• include_once załącza plik tylko jeden raz. Gdy kolejnym razem nastąpi żądanie dołączenia pliku, a ten został wcześniej dołączony, linia z dołączeniem zostanie pominięta.
• require_once identycznie do include_once; w przypadku braku pliku zgłasza błąd jak przy require

6. Stałe w PHP.
Najprościej rzecz ujmując są to wartości, które przez cały czas funkcjonowania skryptu można przywołać poprzez nazwę, z którą się je wiąże. Najczęściej stosowane są w projektach wielojęzycznych, gdzie zamiast tekstów w poszczególnych częściach kodu używa się właśnie nazw stałych, a dopiero od załadowanego języka zależy jaki napis zostanie wyświetlony docelowemu użytkownikowi. Innym zastosowaniem są fragmenty tekstowe/długie teksty statyczne wielokrotnie pojawiające się w kodzie – o wiele prościej jest użyć krótkiej nazwy (np. „tekst1”) zamiast ciągle powtarzać długi tekst (np. „To jest właśnie nasz bardzo długi tekst; łatwo przy kolejnym wpisywaniu o błąd/ złe skopiowanie <ucięcie liter>;”). Ponadto ewentualna poprawa tegoż tekstu staje się o wiele prostsza – wystarczy zmienić go na samym początku pliku z kodem (zamiast przeszukiwać cały skrypt).

UWAGA: Chociaż nie ma to znaczenia dla kompilatora to w dobrym tonie programistycznym jest pisać nazwy stałych tylko dużymi LITERAMI, a nazwy zmiennych $małymi.

Przykład.

define(„ZMIENNA”, „Wartość naszej zmiennej”);
define(„X”, „indeks tablicy”);
define(„LICZBA”, 14);

$tab[X] = „Indeks”;

echo ZMIENNA . „ ” . LICZBA;
print $tab[X];

Stałe „magiczne”
PHP posiada tzw. stałe zdefiniowane. Programista może korzystać z nich zamiast np. samemu ustalać pewne wartości (np. w celu testowania skryptu: sprawdzić ilość linii w pliku, nazwę wykonywanej funkcji/metody, sprawdzić katalog docelowy itp.). Poza tym w stałych zapisane są specjalne dane: rozmiar liczb całkowitych (w bitach), na jakim systemie skrypt jest wykonywany, jaka jest wersja parsera PHP czy też ostatnio wyświetlane błędy, ostrzeżenia, uwagi.

Zadania do wykonania:

1. Proszę znaleźć w dokumentacji PHP (http://www.php.net) bądź innej stronie internetowej w jaki sposób można poprzez php uzyskać aktualną datę (miesiąc, dzień, rok). W następnej kolejności należy stworzyć dwa pliki CSS – jeden z dominującym motywem koloru niebieskiego, drugi zielonego (tło i/lub czcionka). Jeżeli dzień będzie parzysty (podzielny przez 2) należy użyć stylu niebieskiego (i analogicznie zielonego). Oznacza to iż to z poziomu PHP trzeba wstawić nazwę odpowiedniego pliku CSS do załadowania (proszę zauważyć iż w taki wypadku deklaracje poszczególnych sekcji CSS muszą posiadać takie same nazwy). Można wykorzystać istniejące już projekty.
2. Sprawdzić co zawierają następujące stałe: __FILE__, __DIR__, __FUNCTION__, PHP_MANDIR, PHP_OS, PHP_VERSION, PHP_VERSION_ID, DATE_ATOM, PHP_DATADIR, PHP_LIBDIR. Znaleźć informacje jakie jeszcze stałe zostały predefiniowane w PHP (dokumentacja PHP).
3. Proszę napisać następujące funkcje:
a) sumującą podane dwa parametry
b) wyliczającą silnię podanej liczby
c) wypisującą losową radę co odświeżenie strony (ok. 20 porad w postaci np. stałych; proszę wykorzystać metodę rand() - opis dostępny w dokumentacji PHP).
d) funkcję tworzącą tabelę HTML; pa posiadać 10 wierszy (nie licząc wiersza tytułowego). Najważniejszą istotą ma być automatyczne wstawienie kolejnych numerów wierszy (kolumna lp.); pozostałe kolumny mogą zawierać dowolną acz losowo generowaną treść.
e) przenieść napisane funkcje do osobnego pliku; dołączyć plik do głównego pliku skryptu i wywołać je.
f) Utworzyć zmienną imiona wg poniższego przykładu

$imiona = array(„imie1”, „imie2” … „imie10”);

następnie wstawić ją w funkcję, która jako parametr wejściowy będzie przyjmować zmienną z imieniem; zadaniem funkcji jest, za pomocą pętli foreach, „przejrzeć” czy podane jako parametr imię znajduje się w tablicy (w zależności od wyniku ma zwrócić wartość true lub false). Na stronie natomiast ma pojawić się napis (w zależności od wyniku) „podane imię znajduje się w bazie” lub „podane imię nie występuje w bazie” (może być inny, dowolny komunikat).s