[**Site Didactic**](https://web.ceiti.md/index.php)

<https://web.ceiti.md/lesson.php?id=7>

* [ACASĂ](https://web.ceiti.md/index.php)
* [LECŢII](https://web.ceiti.md/lectii.php)
* [LUCRĂRI](https://web.ceiti.md/lucrari.php)
* [TESTE](https://web.ceiti.md/test.php)
* [REFERINŢE](https://web.ceiti.md/referinte.php)
* [DESCARCĂ](https://web.ceiti.md/descarca.php)
* [ELABORĂRILE ELEVILOR](https://web.ceiti.md/elaborare.php)

Unități de învățare

* [Limbaje de programare pentru server](https://web.ceiti.md/lesson.php?id=7#c1-)
* [Dezvoltarea limbajului PHP](https://web.ceiti.md/lesson.php?id=7#c1)
* [Server web local](https://web.ceiti.md/lesson.php?id=7#c2)
* [Editoare de cod PHP](https://web.ceiti.md/lesson.php?id=7#c3)
* [Inițiere în PHP](https://web.ceiti.md/lesson.php?id=7#c4)
* [Variabile](https://web.ceiti.md/lesson.php?id=7#c5)
* [Constante](https://web.ceiti.md/lesson.php?id=7#c6)
* [Operatori](https://web.ceiti.md/lesson.php?id=7#c7)
* [Structuri condiționale](https://web.ceiti.md/lesson.php?id=7#c8)
* [Structuri repetitive](https://web.ceiti.md/lesson.php?id=7#c8-1)
* [Tipul tablou](https://web.ceiti.md/lesson.php?id=7#c8-12)
* [Tipul șir de caractere, în PHP](https://web.ceiti.md/lesson.php?id=7#c8-13)
* [Funcții în PHP](https://web.ceiti.md/lesson.php?id=7#c9)
* [Funții predefinite în PHP](https://web.ceiti.md/lesson.php?id=7#c10)
* [Prelucrarea datei și orei în PHP](https://web.ceiti.md/lesson.php?id=7#c10-1)
* [Clase și obiecte în PHP](https://web.ceiti.md/lesson.php?id=7#c11)
* [Construiește un site cu PHP](https://web.ceiti.md/lesson.php?id=7#c12)
* [Securitatea paginilor web](https://web.ceiti.md/lesson.php?id=7#c13)
* [Preluarea datelor din formulare](https://web.ceiti.md/lesson.php?id=7#c15)
* [Validarea datelor trimise serverului](https://web.ceiti.md/lesson.php?id=7#c15-1)
* [Baze de date în aplicații web](https://web.ceiti.md/lesson.php?id=7#c14)
  + [- MySQLi procedural](https://web.ceiti.md/lesson.php?id=7#s1)
  + [- MySQLi obiect orientat](https://web.ceiti.md/lesson.php?id=7#s2)
* [Citirea conținutului unui director](https://web.ceiti.md/lesson.php?id=7#c16)
* [Lucrul cu fișiere](https://web.ceiti.md/lesson.php?id=7#c17)
* [Creare imagini cu PHP](https://web.ceiti.md/lesson.php?id=7#c18)
* [PHP - GD Library](https://web.ceiti.md/lesson.php?id=7#c19)
* [Ce este World Wide Web?](https://web.ceiti.md/lesson.php?id=7#c20)
* [Greșeli de web design](https://web.ceiti.md/lesson.php?id=7#c21)

**Limbaje de programare pentru server**

Tehnologiile Server Side Scripting permit crearea de aplicații Web complexe prin procesarea datelor pe server și generarea unor pagini în mod dinamic. În felul acesta se pot interfața aplicațiile Web cu servere de baze de date sau servere de poștă electronică. Limbajele Server Side Script au posibilitatea de a accesa datele citite în formulare HTML și implementează biblioteci de acces la resurse externe cum ar fi o bază de date. Spre deosebire de tehnologii mai vechi care realizau același deziderat (Ex. CGI) server side scripting asigura o viteză și o securitate sporită. În general interpretoarele limbajelor SSS sunt integrate sub formă de module chiar in serverul de Web. Exemple de limbaje SSS: ASP, ColdFusion Markup Language, JSP, PHP, Perl, Phyton, Rubi etc.

**Dezvoltarea limbajului PHP. Caracteristicele de bază.**

PHP, înseamnă "PHP: Hypertext Preprocessor" este un limbaj de scripting de uz general, cu cod-sursă deschis (Open Source), utilizat pe scară largă, și care este potrivit în special pentru dezvoltarea aplicațiilor Web și poate fi integrat în HTML. Sintaxa sa provine din C, Java și Perl și este ușor de învățat. Scopul principal al limbajului este acela de a permite programatorilor web să creeze rapid pagini web generate dinamic, însă cu PHP puteți realiza mult mai multe.

**PHP/FI**

PHP succede un produs mai vechi, numit PHP/FI. PHP/FI a fost creat de către **Rasmus Lerdorf** în 1995, inițial doar ca un simplu set de scripturi Perl pentru a urmări accesele la rezumatul său online. El a denumit acest set de scripturi 'Personal Home Page Tools — Instrumente pentru paginile personale'. Deoarece era necesară mai multă funcționalitate, Rasmus a scris o implementare mult mai largă în C, care era în stare să comunice cu baze de date și dădea posibilitate utilizatorilor să dezvolte simple aplicații Web dinamice. Rasmus a hotărît să lanseze codurile-sursă ale PHP/FI pentru ca oricine să le poată vedea, utiliza, să poată corecta erorile din ele și să îmbunătățească codul-sursă.

PHP/FI, care se descifra ca Personal Home Page / Forms Interpreter — Interpretor al paginilor personale / al formularelor, includea unele funcționalități de bază ale PHP pe care îl cunoaștem astăzi. El poseda variabile asemănătoare cu cele din Perl, interpretare automată a variabilelor din formulare și sintaxă inclusă în HTML. Insăși sintaxa era similară cu cea din Perl, cu toate că era mult mai limitată, simplă și într-un anumit fel inconsistentă.

În 1997, PHP/FI 2.0, a doua re-scriere a implementării în C, avea un cult din cîțiva mii de utilizatori din lume (estimat), cu aproximativ 50,000 domenii, care raportau că îl au instalat, ceea ce reprezenta aproximativ 1% din toate domeniile din Internet. Cu toate că erau cîteva persoane, care contribuiau porțiuni de coduri-sursă în acest proiect, el era totuși în mare măsură un proiect al unui singur om.

PHP/FI 2.0 a fost oficial lansat doar în noiembrie 1997, după ce și-a petrecut o bună parte a vieții în versiuni beta. În scurt timp el a fost înlocuit de versiunile alfa ale PHP 3.0.

**PHP 3**

PHP 3.0 a fost prima versiune care se aseamănă mult cu PHP, așa cum îl cunoaștem astăzi. El a fost creat de către Andi Gutmans și Zeev Suraski în 1997 ca o re-scriere completă, după ce ei au descoperit că PHP/FI 2.0 avea o capacitate foarte redusă pentru a dezvolta o aplicație de comerț electronic, asupra cărei ei lucrau într-un proiect al Universității. Într-un efort de a coopera și a continua dezvoltarea PHP/FI cu baza existentă de utilizatori, Andi, Rasmus și Zeev au decis să coopereze și au anunțat PHP 3.0 în calitate de succesor oficial al PHP/FI 2.0, iar dezvoltarea PHP/FI 2.0 a fost în mare măsură stopată.

Unul din cele mai mari avantaje ale PHP 3.0 erau facilitățile sale largi de extindere. În afară de faptul că oferea utilizatorilor finali o infrastructură solidă pentru o multitudine de baze de date, protocoale și interfețe API, facilitățile de extindere ale PHP 3.0 au impulsionat zeci de programatori să se alăture și să înainteze noi module de extindere. Discutabil, aceasta a fost cheia succesului enorm al PHP 3.0. Alte facilități de bază introduse în PHP 3.0 au fost susținerea sintaxei orientate pe obiecte și o sintaxă a limbajului mult mai puternică și consistentă.

Noul limbaj a fost lansat sub un nou nume, care excludea implicarea unei utilizări personale limitate, pe care o avea numele PHP/FI 2.0. El a fost numit pur și simplu 'PHP', sensul fiind un acronim recursiv - PHP: Hypertext Preprocessor — Preprocesor al Hipertextului.

Spre sfîrșitul anului 1998, PHP a atins un număr de zeci de mii de utilizatori (estimat) și sute de mii de site-uri Web raportau că îl au instalat. În punctul său de vîrf, PHP 3.0 a fost instalat aproximativ pe 10% din toate serverele Web din Internet.

PHP 3.0 a fost lansat oficial în iunie 1998, după ce a petrecut aproximativ 9 luni în faza de testare publică.

**PHP 4**

Către iarna anului 1998, la scurt timp după ce PHP 3.0 a fost lansat oficial, Andi Gutmans și Zeev Suraski au început lucrul asupra rescrierii nucleului PHP. Scopurile reproiectării erau de a îmbunătăți performanța aplicațiilor complexe și de a îmbunătăți modularitatea bazei de coduri-sursă ale PHP. Astfel de aplicații puteau fi realizate datorită noilor facilități din PHP 3.0, precum și a susținerii unei game largi de baze de date ți interfețe API ale terților producători, însă PHP 3.0 nu fusese proiectat să lucreze cu astfel de aplicații complexe în mod eficient.

Noul nucleu, denumit 'Zend Engine' (compus din numele autorilor, Zeev și Andi), a făcut față cerințelor de proiect cu succes, și a fost prezentat pentru prima dată la mijlocul anului 1999. PHP 4.0, bazat pe acest nucleu și în cuplu cu o gamă largă de facilități noi a fost lansat oficial în mai 2000, aproape doi ani după predecesorul său, PHP 3.0. Suplimentar la performanța îmbunătățită substanțial în această versiune, PHP 4.0 a inclus alte facilități de bază, cum ar fi susținerea mult mai multor servere Web, sesiunilor HTTP, buferizarea ieșirii, metode mai securizate de gestionare a datelor introduse de utilizatori și cîteva construcții noi ale limbajului.

Astăzi PHP este utilizat de sute de mii de programatori (estimat) și cîteva milioane de site-uri raportează că îl au instalat, ceea ce alcătuiește peste 20% din toate domeniile din Internet.

Echipa de dezvoltare a PHP include zeci de programatori, precum și alte zeci de persoane ce lucrează asupra proiectelor asociate cu PHP, cum ar fi PEAR și proiectul de documentare.

**PHP 5**

Pe 13 iulie 2004 a fost lansat PHP 5, cu Zend Engine II, ce a adus și o orientare obiect mai pronunțată și suportînd mai multe caracteristici ale acestui tip de programare.

PHP 5 aduce mai multe noutăți față de versiunea 4:

* Suport îmbunătățit pentru OOP
* Introduce extensia PDO - PHP Data Objects, care definește o modalitate facilă și consistentă de accesare a diferitelor baze de date
* Imbunătățiri de performanță
* Suport îmbunătățit pentru MySQL ș MSSQL
* Suport nativ pentru SQLite
* Suport SOAP integrat
* Iteratori pentru date
* Controlul erorilor prin tratarea de excepții

La sfîrșitul lui 2007 doar versiunea 5.x mai era întreținută, deoarece în data de 13 iulie 2007 (exact la 3 ani după lansarea PHP5), PHP Group a anunțat că PHP4 va fi scos din uz pe 31 decembrie 2007 , deși prognozează că anumite upgrade-uri de securitate se vor oferi pînă pe 8 august.

**PHP 6**

La versiunea PHP 6 s-a lucrat mai bine de 5 ani cu rezultate modeste și după tot acest efort s-a renunțat la această versiune în 2010. Acest efort considerabil nu a rămas fără urmări iar diferite caracteristici se regăsesc în versiunile PHP 5.3 și PHP 5.4. Versiunile PHP 5.3 și PHP 5.4 au venit cu diferite caracteristici foarte impresionante și din acest motiv mulți programatori considera că versiunea PHP 5.4 ar fi trebuit să devină versiunea PHP 6.0 a limbajului însă nu a fost cazul.

Din cauza că diferite caracteristici ce trebuiau implementate în versiunea PHP 6 au fost implementate în PHP 5.3 și PHP 5.4 și întreaga documentație cu tot ce urma să vină în PHP 6 a fost publica timp de 5 ani atunci s-a luat decizia să se renunțe la versiunea PHP 6 și să fie omisă pentru a nu exista nici-o confuzie în legătură cu noile caracteristici ale limbajului PHP.

**PHP 7**

PHP 7 reprezintă singura lansare a acestui limbaj din ultimii 11 ani și înainte de a fi lansat a stat o perioada de 2 ani în care a fost dezvoltat. PHP 7 vine cu o nouă versiune a motorului ZEND, ceea ce îl face să fie de doua ori mai rapid folosind mai puțină memorie. PHP 7 vine cu un suport considerabil pentru platformele ce rulează pe 64 de biți.

PHP 7 a fost lansat în data de 3 Decembrie 2015.

PHP 7 este de doua ori mai rapid decât orice versiune anterioară. PHP 7 convertește mult mai rapid liniile noastre de cod în format binar și folosește mult mai putină memorie pentru acest proces. Pentru a înțelege mai bine uitați-vă la următoarele date ce reprezintă cererile pe care PHP 5.6 și PHP 7.0 le poate procesa într-o secundă pe diferite platforme.

|  |  |  |
| --- | --- | --- |
|  | **PHP 5.6** | **PHP 7** |
| WordPress | 191 cereri/secundă | 374 cereri/secundă |
| Drupal | 182 cereri/secundă | 316 cereri/secundă |
| Magento | 41 cereri/secundă | 69 cereri/secundă |
| Laravel | 256 cereri/secundă | 485 cereri/secundă |
| Zend Framework | 250 cereri/secundă | 489 cereri/secundă |

PHP 7 este proiectat și refactored având în vedere volumul de lucru de astăzi.

**Popularitate:**

PHP-ul este unul din cele mai folosite limbaje de programare server-side, conform unui studiu efectuat de Netcraf în aprilie 2002, apărînd pe 9 din cele 37 milioane de domenii cercetate în studiu. De asemenea, există un grafic al creșterii folosirii PHP-ului pe site-ul oficial. Popularitatea de care se bucură acest limbaj de programare se datorează următoarelor caracteristici:

* **Familiaritatea**: sintaxa limbajului este foarte ușoară combinînd sintaxele unora din cele mai populare limbaje Perl sau C
* **Simplitatea**: sintaxa limbajului este destul de liberă. Nu este nevoie de includere de biblioteci sau de directive de compilare, codul PHP inclus într-un document executîndu-se între marcajele speciale;
* **Eficiența**: PHP-ul se folosește de mecanisme de alocare a resurselor, foarte necesare unui mediu multiutilizator, așa cum este web-ul;
* **Securitate**: PHP-ul pune la dispoziția programatorului un set flexibil și eficient de măsuri de siguranță;
* **Flexibilitate**: fiind apărut din necesitatea dezvoltării web-ului, PHP a fost modularizat pentru a bine pasul cu dezvoltarea diferitelor tehnologii. Nefiind legat de un anumit server web, PHP-ul a fost integrat pentru numeroasele servere web existente: Apache,IIS,Zeus,server, etc.
* **Gratuitate** : este probabil cea mai importantă caracteristică a PHP-ului. Dezvoltarea PHP-ului sub licența open-source a determinat adaptarea rapidă a PHP-ului la nevoile web-ului, eficientizarea și securizarea codului.

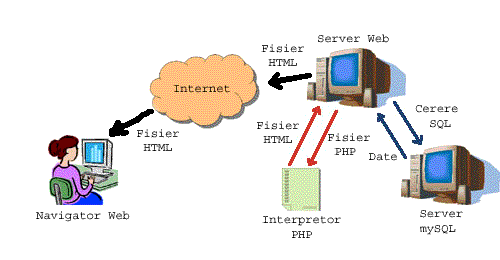
**Utilizare:**

PHP este simplu de utilizat, fiind un limbaj de programare structurat, ca și C-ul ,Perl-ul sau începînd de la versiunea 5 chiar Java , sintaxa limbajului fiind o combinație a celor trei. Datorită modularității sale poate fi folosit și pentru a dezvolta aplicații de sine stătătorare, de exemplu în combinație cu PHP-GTK sau poate fi folosit ca Perl sau Python în linia de comandă. Probabil una din cele mai importante facilități ale limbajului este conlucrarea cu majoritatea bazelor de date relaționale, de la MySQL și pînă la Oracle, trecînd prin MS Sql Server, PostgreSQL, sau DB2.

**PHP** poate rula pe majoritatea sistemelor de operare, de la UNIX ,Windows, sau Mac OS X și poate interacționa cu majoritatea serverelor web. Codul dumneavoastră PHP este interpretat de serverul WEB și generează un cod HTML care va fi văzut de utilizator (clientului -browserului- fiindu-i transmis numai cod HTML).

Arhitectura tip LAMP a devenit populară în industria web ca modalitate rapidă, gratuită și integrată de dezvoltare a aplicațiilor. Alături de Linux, Apache și Mysql, PHP reprezintă litera P, deși uneori aceasta se referă la Python sau Perl. Linux ocupă rolul de sistem de operare pentru toate celelalte aplicații, MySQL gestionează bazele de date, Apache are rol de server web, iar PHP are rol de interpretator și comunicator între acestea.

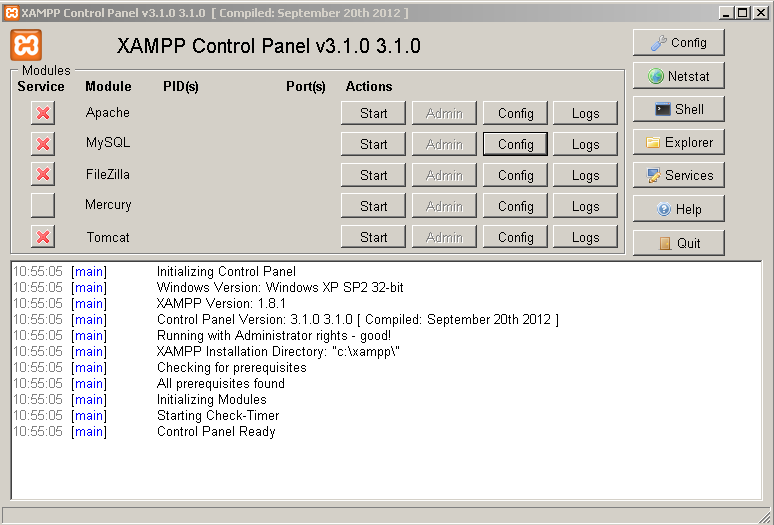
PHP folosește extensii speci fice pentru fișierele sale: .php, .php3, .ph3, .php4, .inc, .phtml. Aceste fișiere sunt interpretate de catre serverul web iar rezultatul este trimis în formă de text sau cod HTML către browser-ul clientului.



Serverul Web încarcă pagina din sistemul de fișiere propriu și, constatând că este vorba de o pagină PHP o trimite interpretorului PHP. Interpretorul PHP parcurge pagina și execută instrucțiunile PHP întâlnite. Instrucțiunile PHP au ca și scop principal acela de a genera cod HTML. De asemenea, codul HTML din fișierul original este copiat pur și simplu la ieșire. Interpretorul PHP transmite înapoi către server fișierul transformat prin interpretarea codul PHP. Acest fișier va conține doar cod HTML (eventual si Client Side Script etc.) dar nu și cod PHP.

**Server web local**

Pentru a putea rula scenarii php(scripturi) avem nevoie să programăm calculatorul în calitate de server Web, server de aplicații PHP și nu în ultimul rînd server de BD MySQL. Aceste componente software de bază necesare (*MySQL*, *Apache* și *PHP*) sunt gratuite (din categoria "*open source*") și pot fi descărcate de pe site-ul oficial al prodducătorului. După instalarea fiecăruia din aceste aplicații apare necesitatea configurării lor, care uneori este dificilă pentru începători. Pentru a evita problemele configurării, putem instala un pachet complet ce conține Apache + PHP + MySQL + PHPMyAdmin care se numește **XAMPP (PHPTriad, EasyPhp, Base, WampServer, Denver, ...).**Aplicația PHPMyAdmin inclusă în pachetul**XAMPP** permite administrarea comodă a BD MySQL. Înainte de a rula scripturi PHP vom lansa aplicația **XAMPP** și anume din *Start/All Programs/Apache Friends/XAMPP/XAMPP Control Panel*, după care se va lansa panoul de control al aplicației **XAMPP**(*vezi imaginea*).



Pentru a lansa serverul Apache acționăm butonul **start**la**modulul Apache**și respectiv dacă lucrăm cu BD acționăm butonul **start**la**la modulul MySql.** Pentru a închide serverele deschise vom acționa respectiv pe butoanele **stop**iar pentru a închide aplicația **XAMPP**vom acționa **exit.**

Pentru a verifica dacă funcționează aplicațiile instalate, vom lansa browserul iar în cîmpul de adrese scriem:  
***http://localhost/xampp*** în loc de *localhost* se poate de scris și *IP* calculatorului sau *127.0.0.1*.Pentru a rula scripturile ele trebuiesc salvate(copiate) în *C:\xampp\htdocs.*

**Editoare de cod**

Întrucît sintaxa limbajului PHP este relativ simplă, iar structura codului sursă nu este una impusă, codul PHP poate fi scris cu orice editor text, cum ar fi Notepad. În realitate, acest lucru este posibil doar pentru scripturile de cîteva linii de cod; pentru fișiere mari și complexe, un editor PHP specializat este aproape indispensabil.

Exista mai multe editoare de cod-sursă ce pot fi folosite pentru PHP, de la IDE-uri complexe pînă la editoare simple. Toate oferă ***highlight*** pentru cuvintele cheie, completare automată a instrucțiunilor, facilități avansate de căutare și altele. Puteți folosi pe oricare dintre ele, în funcție de nevoile voastre.

**1. Dreamweaver** - un program fascinant pentru creare de pagini web. Potențialul său este mai larg decît celelalte editoare de pagini web. Dreamweaver vă permite să creați obiecte și pagini web folosind DHTML, sprijină Cascading Style Sheets, precum și straturi și JavaScript.

**2. PHP Designer** - un mediu de dezvoltare pentru limbaje ca PHP, HTML, XHTML, CSS și SQL (suporta Ingres, Interbase, MSSQL, MySQL, Oracle si Sybase). Pachetul mai include un browser, un client FTP, un file manager, biblioteci de coduri (pentru PHP, SQL si HTML) și un modul pentru conversia documentelor în formate ca LaTex, HTML si RTF.

**3. Notepad ++** - un editor de coduri sursă, care indeplinește foarte bine și funcțiile "Notepad" din windows. Limbajele de programare suportate sunt foarte variate ( ASP, Ada, ASCII art, Assembly, AutoIt, CSS, doxygen, FORTRAN, HTML, Haskell, Java, JavaScript, BAT, C, C#, C++, Caml, KiXtart, Lisp, Lua, makefile, Matlab, Objective-C, Pascal, Perl, PHP, Smalltalk, SQL, Tcl, TeX, Verilog, VHDL, VB/VBScript, XML, PostScript, Python, Ruby, Scheme, Unix Shell Script), suportă "drag and drop" și poate fi folosit în locul clasicului "Notepad" pentru editarea de texte fără nici un fel de problemă. Programul nu necesita resurse hardware mai mari decat "Notepad-ul" din Windows.

Un plus în ceea ce privește editarea codurilor sursă este faptul că programul poate modifica (edita) același fișier în mai multe formate de vizualizare avînd loc sincronizarea fișierului în ambele formate și de asemenea se pot tipări codurile sursă pe hîrtie.

Notepad++ - un notepad mult mai complex. Proiectul include foarte multe plug-in-uri ajutatoare pentru anumite editari complexe. Softul este gratuit (licenta GPL).

**4. ConTEXT** - un editor mic, rapid și puternic de text freeware, dezvoltat pentru a servi ca un instrument secundar pentru dezvoltatorii de software.

**5. Zend Studio** - un mediu de dezvoltare pentru limbajul de programare PHP, care include optiuni pentru editare cod, diagnosticare și testare.

**6. Eclipse** - o platformă integrată de dezvoltare (IDE) extensibilă, ce dispune de o multitudine de funcționalități. Folosită inițial pentru dezvoltarea în limbajul Java, a fost ulterior extinsă și la alte limbaje de programare.

Eclipse PDT (Eclipse PHP Development Tools) reprezinta la ora actuala cel mai avansat editor PHP. Este gratuit și poate rula pe orice sistem de operare. Mai multe informații pot fi gasite pe site-ul proiectului [Eclipse PDT](http://wiki.eclipse.org/PDT).

**7. Netbeans** - vine din zona programării în Java. Este un IDE complet ce a fost extins recent pentru lucrul în PHP. Ca și Eclipse poate fi folosit pe orice sistem de operare întrucît rulează pe platforma Java. Poate fi descărcat gratuit de la <http://netbeans.org/downloads/> (alegeti pachetul PHP).

Modul de lucru în Netbeans este similar cu cel din Eclipse. Avem și aici conceptul de **proiect** iar pentru a putea crea scripturi PHP este nevoie să creați în prealabil un proiect de tip PHP. Toate celelalte observații făcute la editorul Eclipse rămîn valabile și aici.

**8. Sublime Text** - Sublime Text 3 este un editor de text care e la modă de ceva timp, datorită faptului că reușește să îmbine o mulțime de aspecte pe care ți le-ai dori de la un editor într-un program mic, dar foarte puternic. Sublime Text 3 folosește un sistem de API Python pentru pluginuri și teme, care sunt incredibil de folositoare.

**9. Atom** - Atom este un editor gratuit și open-source dezvoltat de cei de la GitHub ca răspuns la o serie de editoare extrem de populare cum ar fi Sublime Text, TextWrangler, TextMate si Textastic, preluând o parte din feature-urile acestora. Spre diferență de acestea însă, Atom oferă posibilitatea de a modifica aproape toate aspectele editorului de către utilizator.

**Inițiere în PHP**

**Taguri PHP, instrucțiuni, punct și virgulă**

Codul PHP trebuie inclus în fișierele ce vor fi deservite utilizatorilor finali și trebuie delimitat de restul conținutului prin tagul de început **<?php** și tagul de sfîrșit **?>**. Practic tot ce se află în interiorul acestor etichete va fi interpretat ca fiind cod-sursa PHP (cod ce va fi executat de interpretorul PHP). Textul din afara celor 2 taguri este lăsat neschimbat, fiind ulterior interpretat de browser. Se pot folosi și alte etichete în funcție de configurația serverului web, dar acestea nu sunt recomandate.

|  |  |
| --- | --- |
| 1  2  3 | <?php  /\* forma recomandata \*/  ?> |
| 1  2  3 | <script language="php">  /\* forma disponibila oricand, putin folosita \*/  </script> |

|  |  |
| --- | --- |
| 1  2  3 | <?  /\* taguri scurte, acceptarea lor de catre web-server depinde de configurare \*/  ?> |
| 1  2  3 | <%  /\* taguri in stilul asp, acceptarea lor de catre web-server depinde de configurare \*/  %> |

Codul PHP este format din **instrucțiuni** - comenzi date către interpretor, în urma cărora se execută ceva. Așa cum s-a mai spus, PHP este folosit în principal pentru a genera cod HTML, așa că de cele mai multe ori instrucțiunile folosite sunt cele de afișare. În aplicații mai complexe se pot însă folosi și instrucțiuni de conectare la bazele de date, de citire/scriere/manipulare fișiere, instrucțiuni de trimitere email-uri și altele. Instrucțiunile se pot grupa în **blocuri** delimitate de acolade { }. Aceste blocuri, numite și "instrucțiuni complexe" se comportă ca și cum ar fi o singură instrucțiune (vezi mai jos exemple de blocuri).

Toate instructiunile (in afară de blocuri) trebuie să fie terminate cu **punct si virgulă** ( ; ). Lipsa acestuia generează o eroare.

**Intrucțiuni uzuale în PHP: echo, print, printf**

În multe din scripturile scrise de incepători (și nu numai) conținutul rezultat în urma execuției este un text (care poate fi sau nu cod HTML). Pentru a obține acest text, secvențele de cod trebuie să conțină instrucțiuni explicite care să "spună" intrepretorului ce anume trebuie afișat.

Instrucțiunile de afișare sunt **print**, **echo** și **printf**. Există și alte modalitați dar aceste 3 instrucțiuni sunt cele mai uzuale și usor de folosit. Diferențe între ele există, dar ca idee generală **print** și **echo** fac același lucru și poate fi folosită oricare dintre ele. **printf** este folosita mai rar datorită sintaxei oarecum greoaie.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?php  # cele 2 instructiuni de mai jos sunt echivalente  print "Text";  echo "Text";    # o instructiune echo poate primi mai multi parametrii  echo "Afisez", " un text din ", 4, " bucati";    # o singura intstructiune print poate primi doar unul  print "Afisez";  print " un text din ";  print 4;  print  " bucati";    # printf este folosita pentru a formata continutul, la fel ca in C/C++  printf( "Am %4.2f lei", 102.1234 ); // afiseaza Am 102.12 lei  ?> |

În cazul paginilor statice create cu HTML utilizatorul poate vedea doar imagini și text. Folosind PHP însă vom putea crea așa numitele pagini dinamice în care apare elementul de interactivitate: se pot prelua date din formulare realizate în PHP, citi, prelucra și afișa. De asemenea se poate face interacțiunea cu o bază de date creată pe un server.

**Crearea primului script php**

|  |  |
| --- | --- |
| 1  2  3 | <?php   echo "Primul script php";  ?> |

**Obs:** echo - afișează un mesaj

**Combinarea codului HTML cu cod PHP**

Acest lucru se întîmplă foarte des. În interiorul unui script HTML poate exista script PHP delimitat de etichetele ***<?php***și ***?>***însă fișierul final trebuie să fie PHP.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <html>  <body>   Text combinat HTML cu PHP  <?php   echo "Text scris cu PHP";   ?>   </body>   </html> |

Comentarii în scripturi:  
# - un tip de comentariu pentru un rînd;  
// - un alt tip de comentariu pentru un rînd;  
/\* și \*/ - pentru mai multe rînduri.

**Variabile PHP**

Variabilele sunt mărimi care pot lua diferite valori. Numele dat unei variabile începe întotdeauna cu simbolul $ urmat de o literă sau caracterul "\_". Nu poate fi precedat niciodată de o cifră. Restul caracterelor care formează numele variabilei pot fi litere, cifre sau liniuțe de subliniere. În PHP, sunt considerate litere toate caracterele cuprinse între '**a**' și '**z**', cele cuprinse între '**A**' și '**Z**', precum și cele care au codul ASCII cuprins între 127 și 255.

Definim o variabilă astfel:  
*$<variabila>=<valoare>;*

Exemplu :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | // variabile numerice  $variabila = 1;  $numar = 0;    // variabile text (stringuri)  $text = "Salut";  $text2 = 'Ce faci?';    // variabile logice (boolean)  $stiuPHP = true;  $uitRepede = false;    // vector  $vectorGol = array();  $vectorS = array( 10,11,12,13 ); // vector simplu    // vector asociativ  $vector = array(  'luni' => 'Monday',  'marti' => 'Tuesday',  'miercuri' => 'Wednesday'  );    // definirea unei noi componente a unui vector asociativ  $vector[ 'joi' ] = 'Thursday';    // vector simplu definit ca asociativ, echivalentul lui $vectorS  $vectorS2 = array(  0 => 10,  1 => 11,  2 => 12,  3 => 13  ); |
| 1  2  3  4  5  6  7  8 | <?php   $a = "PHP5";   $b = &$a;   $b = "PHP e foarte  bun";   echo $a;   echo "<br>";   echo $b;   ?> |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php   $nume = "Ciobanu";   $prenume =  "Andrei";;   $virsta = "22 ani";>   $tara = "Moldova";   echo "Ma numesc  ".$nume.$prenume.", am ".$virsta.", si locuiesc in ".$tara;   ?> |

Atentie: PHP face distinctie între literele mari și mici. Astfel, cele 3 variabile de mai jos sunt TOTAL diferite:

|  |  |
| --- | --- |
| 1  2  3 | $variabila = "negru";  $vaRiabila = "alb";  $Variabila = 1; |

**Atribuirea de valori**

Se realizează la fel ca și declararea prin intermediul instrucțiunii de atribuire.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # atribuirea unei valori anume  $variabila = 2;    # copierea valorii de la o alta variabila  $text = $variabila;    # copierea valorii functioneaza si in cazul vectorilor  # in urma copierii cei 2 vectori vor fi identici  $vectorS = $vectorS2; # toate componentele lui $vectorS2 sunt copiate in $vectorS    # in cazul vectorilor se pot atribui valori separat fiecarei componente  $vectorS[ 0 ] = 100;  $vectorS[ 3 ] = 10;    $vector[ 'luni' ] = 'Lundi';  $vector[ 'joi' ] = 'Jeudi'; |

**Stergerea variabilelor**

De obicei nu e nevoie că variabilele să fie șterse după ce au fost folosite. Totuși, lucrul aceste se poate face în următoarele moduri:

|  |  |
| --- | --- |
| 1  2 | unset( $variabila );  $variabila = null; |

**Afișarea variabilelor**

Afișarea se realizează utilizînd oricare din instrucțiunile de afișare:

|  |  |
| --- | --- |
| 1  2  3  4  5 | $variabila = "Text";  $randNou = "<br>";    print $variabila; print $randNou;  echo $variabila, $randNou; |

**Afișarea tipului de date a variabilei**

Afișarea tipului de date a variabilei se realizează utilizînd funcția **gettype($var)**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php  echo gettype(102).'<br>';  echo gettype(true).'<br>';  echo gettype(' ').'<br>';  echo gettype(null).'<br>';  echo gettype(array()).'<br>';  echo gettype(new stdclass());  ?> |

**Output :**  
integer  
boolean  
string  
NULL  
array  
object

**Variabile globale**

**$GLOBALS** - pot fi accesate toate variabilele globale care sunt accesibile script-ului PHP curent; acest vector este indexat chiar prin numele variabilelor globale. Această variabilă superglobală este accesibilă începînd cu versiunea 3.0.0

**$\_SERVER** - conține o serie de variabile ale căror valori sunt setate de server-ul web; majoritatea valorilor variabilelor din acest vector depind de mediul de execuție al script-ului curent. A fost introdusă începînd de la versiunea 4.1.0. În versiunile anterioare se folosea **$HTTP\_SERVER\_VAR** .  
Este o variabilă care conține informații cum ar fi headere, locația scriptului, sau informații despre vizitator și browser. Este posibil ca în funcție de setările fiecărui server și browser să nu fie active toate informațiile specificate de această variabilă. Această variabilă este de asemenea globală, adică se poate folosi oriunde în script, fără apelarea ei ca funcție globală (adică nu trebuie să o cereți cu **global $\_SERVER** în funcțiile dvs.) În schimb *$HTTP\_SERVER\_VARS* conține aceleași informații dar nu este o funcție globală (atenție **$HTTP\_SERVER\_VARS**și **$\_SERVER** sunt două variabile diferite, chiar dacă aparent fac aceleași lucruri) Dacă directiva *register\_globals* este ON atunci aceste variabile vor deveni la rîndul lor variabile globale în cadrul scriptului. Elementele acestor variabile sunt cele de mai jos (atenție: în funcție de setările serverului și browserului unele elemente pot lipsi)

**' PHP\_SELF '**- conține numele fișierului unde se execută scriptul curent, relativ la rădacină (document root) De exemplu, *$\_SERVER['PHP\_SELF']* într-un script la adresa *http://example.com/test.php/foo.bar* va fi */test.php/foo.bar* . Constanta **\_\_FILE\_\_** conține toată calea și numele fișierului curent.

**' argv**' - Un array al argumentului trimis către script. Cînd scriptul rulează în linie de comandă, va da acces C-style la parametrii liniei de comanda. Cînd este accesat prin metoda GET, acesta va conține un sir de interogare.

**' argc '** - Conține un număr al parametrilor liniei de comandă trimise către script (daca este rulat în linie de comanda).

**' GATEWAY\_INTERFACE '** - Returneaza versiunea CGI utilizată pe server (exemplu ' CGI/1.1 ').

**' SERVER\_NAME '** - Numele server-ului de hosting unde se execută scriptul curent. Dacă scriptul rulează pe un host virtual, aceasta va conține valoarea definită pentru acel host virtual.

**' SERVER\_SOFTWARE '** - Returnează un șir de identificare al server-ului dat în header ca raspuns la cerere.

**' SERVER\_PROTOCOL '** - Numele și versiunea protocolului prin care este cerută pagina (exemplu ' HTTP/1.0 ')

**'REQUEST\_METHOD'** - returnează metoda utilizată pentru accesarea paginii (exemplu ' GET ', ' HEAD ', ' POST ', ' PUT ')

**' REQUEST\_TIME'** - Valoarea *timestamp* dată la pornirea cererii. Este valabilă începînd de la versiunea PHP 5.1.0.

' **QUERY\_STRING** ' - Sirul de interogare, dacă este, de la pagina care a fost accesată.

' **DOCUMENT\_ROOT** ' - Directorul radăcină unde este rulat scriptul curent, așa cum este definit în fișierul de configurație al serverului.

' **HTTP\_ACCEPT** ' - Conținutul header-ului acceptat de la cererea curentă, dacă este unul.

' **HTTP\_ACCEPT\_CHARSET** ' - Conține setul de caractere acceptate și care este trimis de header-ul cererii respective, dacă există. Exemplu: ' iso-8859-1,\*,utf-8 '.

' **HTTP\_ACCEPT\_ENCODING** ' - Conține tipul de encodare trimis de cerera curentă prin header, dacă există.. Exemplu: ' gzip '.

' **HTTP\_ACCEPT\_LANGUAGE** ' - Conține limbajul trimis de cererea curentă prin header, dacă exista. Exemplu: ' en '.

' **HTTP\_CONNECTION** ' - Conține informații despre conecsiiunea trimisă de cererea curentă prin header, dacă există. Exemplu: ' Keep-Alive '.

' **HTTP\_HOST** ' - Conține informații despre host, header-ul de la cererea curentă, dacă există.

' **HTTP\_REFERER** ' - Adresa paginii (dacă este) de unde a venit userul în pagina curentă. Aceasta este setată de către browser. Nu toate browserele setează această valoare, și chiar unele au posibilitatea de a modifica *HTTP\_REFERER* așa cum vor. Pe scurt, nu este de încredere.

' **HTTP\_USER\_AGENT** ' - Conține semnătura browser-ului trimisă prin header la cererea curentă, dacă există. Un exemplu tipic este: Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586) . În afară de asta, puteți utiliza **get\_browser()** pentru a vedea ce capabilitați are browserul folosit de user și astfel să returnați pagina specifică pentru acel browser.

' **HTTPS** ' - Setează o valoare dacă scriptul a fost cerut printr-un protocol securizat HTTPS

' **REMOTE\_ADDR** ' - Adresa de IP a userului care accesează pagina curentă.

' **REMOTE\_HOST** ' - Numele hostului userului care vede pagina curentă. Este inversul DNS-ului userului bazat pe *REMOTE\_ADDR*. Atenție: Serverul web trebuie configurat pentru a crea această variabila. De exemplu în Apache va trebui *HostnameLookups On* în fișierul de configurare *httpd.conf*. Pentru alte detalii vedeți și funcția **gethostbyaddr()**.

' **REMOTE\_PORT** ' - Portul prin care userul comunică cu serverul web.

' **SCRIPT\_FILENAME** ' - Calea absolută unde se execută scriptul curent. Atenție: Dacă scriptul este rulat cu CLI, ca și cale relativă, cum ar fi file.php sau ../file.php , $\_SERVER['SCRIPT\_FILENAME'] va conține calea relativă specificată de către user.

' **SERVER\_ADMIN** ' - Valoarea dată de directiva *SERVER\_ADMIN* (pentru Apache) în fișierul de configurare al web serverului. Dacă scriptul rulează pe un host virtual, acesta va conține valoarea definită pentru acel host virtual.

' **SERVER\_PORT** ' - Portul de pe server care este folosit pentru comunicare pe web. Pentru setările standard, acesta este ' 80 '; pentru SSL, de exemplu, se va schimba cu oricare port definit în transmisia securizată HTTP.

' **SERVER\_SIGNATURE** ' - Conține un șir care cuprinde versiunea serverului și numele hostului virtual dacă este.

' **PATH\_TRANSLATED** ' - Calea de bază a sistemul de fișiere (nu rădacina documentului) de la scriptul curent, după ce serverul a terminat maparea oricăror cai ( virtual-to-real mapping) Atenție: Începînd de la PHP 4.3.2, **PATH\_TRANSLATED** nu mai este setat implicit în Apache 2 SAPI în contrast cu situația din Apache 1, unde este setată aceeași valoare ca și la **SCRIPT\_FILENAME**. Această schimbare a fost făcută pentru a fi compatibilă cu specificațiile CGI în care **PATH\_TRANSLATED** trebuie să existe doar dacă **PATH\_INFO** este definit. Utilizatorii Apache 2 pot utiliza *AcceptPathInfo = On* în fișierul de configurație *httpd.conf* pentru a defini **PATH\_INFO** .

' **SCRIPT\_NAME** ' - Conține calea către scriptul curent. Acesta este util pentru pagini care trebuiesc definite către ele însuși. Constanta \_\_FILE\_\_ conține calea completă și numele de fișier al scriptului curent.

' **REQUEST\_URI** ' - URI care este dat pentru accesarea paginii curente, de exemplu ' /index.html '.

' **PHP\_AUTH\_DIGEST** ' - Cînd rulați sub Apache ca modul, acțiunea *Digest HTTP,* valoarea acestei variabile este setată în 'Authorization' header trimis către client (care ar trebui utilizat pentru a face o validare corespunzătoare).

' **PHP\_AUTH\_USER** ' - Cînd este rulat sub Apache sau IIS (ISAPI pe PHP 5) ca modul, actiunea *HTTP authentication*, această variabilă conține numele userului trimis de către user.

' **PHP\_AUTH\_PW** ' - Cînd este rulat sub Apache sau IIS (ISAPI pe PHP 5) ca modul, actiunea *HTTP authentication*, această variabilă conține parola userului trimisă de către user.

' **AUTH\_TYPE** ' - Cînd este rulat sub Apache ca modul acțiunea *HTTP authenticated* , această variabilă seteaza tipul de autentificare.

**Variabilele pentru trimiterea datelor**

**$\_GET** este o variabilă array globală. A fost introdusă în versiunea 4.1.0. Pentru versiunile anterioare se folosește **$HTTP\_GET\_VARS** care nu este variabilă globală. Puteți folosi **$\_GET** pentru a trimite variabile cu valori prin intermediul linkurilor.

**$\_POST** este de asemenea o variabilă array globală. A fost introdusa în versiunea 4.1.0. Pentru versiunile anterioare se folosește **$HTTP\_POST\_VARS** care nu este variabilă globală. Puteți folosi **$\_POST** pentru a trimite variabile cu valori prin intermediul formularelor.

**$\_GET**și **$\_POST** conțin variabile primite de script prin intermediul unor transferuri care folosesc metodele HTTP *get*, respectiv *post*. De exemplu, prin intermediul acestor vectori pot fi accesate valorile cîmpurilor dintr-un formular care a fost completat și transmis folosind una dintre cele doua metode.

**Variabilele HTTP coockies**

**$\_COOKIE** conține valorile variabilelor care conțin informații referitoare la cookie-urile păstrate pe calculatorul utilizatorului care accesează pagina web. A fost introdus în versiunea 4.1.0. În versiunile anterioare, utilizați **$HTTP\_COOKIE\_VARS** .

**Variabile pentru uploadul fișierelor**

**$\_FILES** conține variabile primite de script prin intermediul încărcărilor de fișiere prin metoda *post*. A fost introdus în versiunea 4.1.0. În versiunile anterioare se folosește **$HTTP\_POST\_FILES** care nu este variabilă globală.

**Variabilele de mediu $\_ENV**

**$\_ENV** conține variabile disponibile prin intermediul mediului în care este executat. A fost introdus în versiunea 4.1.0. În versiunile anterioare, utilizați **$HTTP\_ENV\_VARS** .

**Variabilele pentru cereri**

**$\_REQUEST** conține variabile disponibile prin intermediul oricărui tip de mecanism cu ajutorul căruia utilizatorul poate introduce date. Este de tip array și conține valorile variabilelor **$\_GET** , **$\_POST** , și **$\_COOKIE**. A fost introdusă în versiunea 4.1.0. Începînd de la versiunea 4.3.0, informațiile din **$\_FILES** au fost de asemenea incluse în **$\_REQUEST**

**Variabile de sesiune**

**$\_SESSION**conține variabile care corespund sesiunii curente a script-ului, sub formă de array. A fost introdusă în versiunea 4.1.0 și este de asemenea globală. În versiunile anterioare folosiți **$HTTP\_SESSION\_VARS**.

**Variabile de mesaje de eroare**

**$php\_errormsg** - este o variabilă ce conține ultimul mesaj generat de către PHP Această variabilă este folosită în scopul de a vedea ce eroare apare și doar dacă opțiunile de configurare a erorilor **track\_errors**este pornita (standard este pe OFF)

**Tipuri de variabile**

Variabilele pot fi de mai multe tipuri, nu doar numere. PHP are opt tipuri de variabile. Patru dintre acestea sunt tipuri scalare (*boolean, integer, float*și*string*), două sunt tipuri compuse (*array*și*object*), iar alte două sunt tipuri speciale (*resource*și*null*).

În PHP, de obicei, tipul unei variabile nu este specificat de către programator, ci este stabilit în timpul execuției în funcție de contextul în care este folosită variabila.

**Observație**: Tipul de date este specificat prin valoarea atribuită variabilei inițiale.

Exemplu :

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $a=1; // număr întreg (zecimal)  $b=3.14: // număr real  $c="exemplu de text"; //șir de caractere  $c='exemplu de text'; //șir de caractere  $matrice = array('alb','rosu','verde');  //matrice  $a=TRUE; //variabilă booleana |
| 1  2 | $a=1;  echo "Numarul ", $a,"<br>"; |

Se va afișa : Numarul 1

În cazul unui șir de caractere definit prin ghilimele duble se pot folosi următoarele secvențe:

|  |  |
| --- | --- |
| **Secvența** | **Semnificație** |
| \n | rînd nou |
| \r | sfîrșit de linie |
| \t | tab orizontal |
| \\ | backslash |
| \$ | simbol dolar |
| \" | ghilimele duble |

**Constante PHP**

O constantă stochează o valoare, cum este și o variabilă, dar a această valoare, după ce a fost stabilită nu mai paote fi modificată în script. Constantele sunt create cu ajutorul funcției *define*(), iar numele constantelor este scris cu Majuscule, aceasta optiune nu este obligatorie, însî va face codul dumneavoastră mai frumos și mai lizibil. O diferență importantă între constante și variabile, este faptul că o constantă nu are în fața ei semnul $.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4 | define("PI", 3.14);  define('NUME', "Ciobanu");  define('VARSTA', 40);  define("ADEVARAT", true); |
| 1  2  3  4  5 | <?php  define('AUTOR','Eminescu');   define('VARSTA','20 ani');   echo AUTOR.'<br>'.VARSTA;   ?> |

Este recomandat ca numele constantelor sa fie scris cu majuscule (pentru a fi mai ușor identificate) deși nu este obligatoriu. Atunci cînd se folosesc, se specifică doar numele, fără ghilimele sau $:

|  |  |
| --- | --- |
| 1  2 | print "Ma numesc " .NUME;  print " si am " .VARSTA." ani"; |

Pentru a verifica dacă o constantă este definită, se folosește funcția **defined**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | if( defined('NUME' ) ) {  print "Ma numesc " . NUME;  }  if (!defined( 'VARSTA' ) ) {  define('VARSTA', 2 );  }  print " Am ". VARSTA ." ani"; |

**Operatori PHP**

Interpretorul PHP permite folosirea a nouă tipuri diferite de operatori. Aceștia operează asupra unor expresii (una, două sau trei) și furnizează ca rezultat o altă expresie (rezultatul operației corespunzătoare).

**Operatori aritmetici**

Toți operatorii aritmetici operează asupra a două expresii (operanzi). Există cinci astfel de operatori:

|  |  |  |
| --- | --- | --- |
| + | Adunare | $a + $b |
| - | Scădere | $a - $b |
| \* | Înmulțire | $a \* $b |
| / | Împărțire | $a / $b |
| % | Modulo(rest) | $a % $b |

Aceste operații se aplică asupra unor valori care reprezintă tipuri numerice. Dacă unul dintre operanzi nu are tip numeric, atunci el va fi convertit automat la o valoare întreagă. Dacă cel puțin unul dintre operanzi (după efectuarea eventualelor conversii necesare) este un număr real, atunci rezultatul operației va fi tot un număr real, cu excepția operatorului rest; rezultatul operației de determinare a restului este întotdeauna un număr întreg. Dacă ambii operanzi sunt numere întregi, atunci rezultatul va fi un număr întreg, cu excepția împărțirii al cărei rezultat este întotdeauna un număr real. Dacă operatorul rest este aplicat asupra unor numere reale, atunci rezultatul este un număr întreg. Nu este permisă împărțirea la valoarea 0. Dacă al doilea operand asupra căruia se aplică unul dintre operatorii '**/**' sau '**%**' are valoarea 0, va fi semnalată o eroare.

Exemple:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?php     $a = '12';     $b = '8';     $rezultat = $a + $b;     echo 'Rezultatul adunarii lui '.$a.' cu  '.$b.' este '.$rezultat.' ';   ?> |

**Operatori de comparare**

Interpretorul PHP pune la dispozitie mai mulți operatori care pot fi folosiți pentru compararea a două valori. Expresiile în care apar astfel de operatori au ca rezultat valori logice (true sau false).

Cel mai des întîlnit operator este cel de atribuire, definit prin semnul **=**  
Dacă scriem *$x = 1* nu înseamnă că $x este egal cu 1 ci că i s-a acordat valoarea 1. Dacă scriem *$rezultat = $x + $y* nu înseamnă că *$rezultat* este egal cu suma celor două, ci că i-am atribuit (acordat) valoarea sumei celor două.

Alți operatori folosiți pentru compararea valorilor variabilelor sunt:

|  |  |  |  |
| --- | --- | --- | --- |
| **'=='** | Egal | $a==$b | rezultatul este **true** dacă cele două expresii au aceeași valoare |
| **'==='** | Identic | $a===$b | rezultatul este **true** dacă cele două expresii au aceeași valoare și au același tip; |
| **'!='** sau **'<>'** | Diferit | a!=$b sau $a<>$b | rezultatul este **true** dacă cele două expresii au valori diferite; |
| **'!=='** | Neidentic | **$a!==$b** | rezultatul este **true** dacă cele două expresii au valori diferite și au tipuri diferite |
| **'<'** | Mai mic | $a<$b | rezultatul este **true** dacă valoarea primei expresii este mai mică decît valoarea celei de-a doua expresii |
| **'>'** | Mai mare | $a>$b | rezultatul este **true** dacă valoarea primei expresii este mai mare decît  valoarea celei de-a doua expresii |
| **'<='** | Mai mic sau egal | $a<=$b | rezultatul este **true** dacă valoarea primei expresii este mai mică sau egalăcu valoarea celei de-a doua expresii; |
| **'>='** | Mai mare sau egal | $a>=$b | rezultatul este **true** dacă valoarea primei expresii este mai mare sau egală valoarea celei de-a doua expresii |

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php    $x=1;    $y=&7;      if ($x == $y) {  print "$x este egal cu $y";  }      if ($x != $y)  {   print "$x este diferit de $y";  }    ?> |
| 1  2  3  4  5  6  7  8  9  10 | <?php    $x=5;    $y=4;      if ($x>$y) {  print "$x este mai mare ca $y"; }    if ($x <= $y) {   print "$x este mai mic sau egal cu $y"; }    ?> |

**Operatori pentru șiruri de caractere**

Operatorii pentru stringuri sunt **'.'** (punct) pentru concaternare și **'.='**(punct și egal) pentru atribuirea concaternării. Acești operatori sunt folosiți pentru a uni stringuri, în felul urmator:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?php    // concaternarea stringurilor    $test='Am scris un '.'text'.'oarecare';    print $test;    // pe ecran va fi afisat "Am scris un text oarecare"      //concaternarea stringurilor cu  variabile    $nr=5;    print'Am scris numarul '.$nr.' si un text '.'oarecare';    // pe ecran va fi afisat "Am scris numarul 5 si un text oarecare"    // atribuirea concaternarii    $nr = 6;    $text = "Un text ";    $text.= "oarecare ";    $text.="si numarul ".$nr;    print $text;    // pe ecran va fi afisat "Un text oarecare si numarul 6"    ?> |

**Operatori pe biți și operatori logici**

Interpretorul PHP pune la dispoziție șase operatori care operează asupra biților unui număr întreg sau ai unui șir de caractere. Aceștia sunt:  
- conjuncție ('&') - SI (AND);  
- disjuncție ('|') - SAU (OR);  
- disjuncție exclusivă ('^') - SAU exclusiv (XOR);  
- negație ('~') - NU (NOT);  
- deplasare la stînga ('<<');  
- deplasare la dreapta ('>>').  
  
Operatorii logici vă vor veni la îndemînă în execuția scriptului atunci cînd aveți nevoie să lucrați cu valori de adevăr. Să presupunem că la intrarea în secțiunea de administrare avem un formular care cere numele și parola de acces în secțiune. Pentru contruirea expresiilor logice operanzii trebuie să aibă tipul boolean; dacă operatorii au alt tip, valoarea este convertită la o valoare booleană. Scriptul PHP ar putea verifica aceste informații pentru a autoriza accesul în secțiune folosind operatorii logici astfel:  
- operatorul **'xor'** (SAU exclusiv)  
Expresia în care apare operatorul **'xor'** va avea valoarea **true** dacă exact unul dintre operanzi are valuarea **true**  
  
- operatorul **'!'** (NOT)  
if (!parola\_e\_buna) ... parola nu este bună, accesul este interzis  
if (!parola\_nu\_e\_buna) ... parola e bună, accesul este permis  
Operatorul **'!'** returneaza **TRUE** dacă valoarea inițială de adevăr e **FALSE** și **FALSE** dacă valoarea inițială este **TRUE**.  
  
- operatorul **'||'** (OR)  
if (numele\_este\_valid) || parola\_este\_buna) ...  
verifică dacă numele sau parola sunt valide și dacă oricare dintre ele este, returnează valoarea de adevăr **TRUE**. În acest exemplu de pseudocod dacă numele ar fi valid dar parola nu, i-am acorda utilizatorului acces mai departe, ceea ce nu este de dorit. Trebuie să fim siguri că și numele și parola sunt valide. Operatorul **'||'** returnează **TRUE** dacă oricare din valorile verificate e **TRUE**. Returnează **FALSE** doar dacă amîndouă sunt **FALSE**.  
  
- operatorul **'&&'** (AND)  
if (numele\_este\_valid && parola\_este\_buna) ...  
dacă atît numele cît și parola sunt valide putem acorda utilizatorului acces în secțiunea de administrare. Operatorul **'&&'** returnează **TRUE** doar dacă ambele valori verificate sunt **TRUE**. El returnează **FALSE** dacă oricare dintre ele este **FALSE** (sau dacă amîndouă sunt **FALSE**).

**Operatori de atribuire**

Primul operand asupra căruia este aplicat un astfel de operator de atribuire trebuie să fie o variabilă, iar al doilea poate fi o expresie.

|  |  |  |
| --- | --- | --- |
| **Operația** | **Varianta scurtă** | **Varianta standard** |
| adunare ('+=') | $a += 2 | $a = $a + 2 |
| scădere ('-=') | $a -= 2 | $a = $a - 2 |
| înmulțire ('\*=') | $a \*= 2 | $a = $a \* 2 |
| împărțire ('/=') | $a /= 2 | $a = $a / 2 |
| rest ('%=') | $a %= 2 | $a = $a % 2 |
| conjuncție ('&=') | $a &= 2 | $a = $a & 2 |
| disjuncție ('|=') | $a |= 2 | $a = $a | 2 |
| disjuncție exclusivă ('^=') | $a ^= 2 | $a = $a ^ 2 |
| deplasare la stînga ('<<=') | $a <<= 2 | $a = $a << 2 |
| deplasare la dreapta ('>>=') | $a >>= 2 | $a = $a >> 2 |
| concaternare ('.=') | $a .= "2" | $a = $a . "2" |

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | <?php  $b = 2;  $a = 1;  echo "<TT>$a += $b = ";  echo $a += $b;  $b = 2;  $a = 1;  echo "<br>$a -= $b = ";  echo $a -= $b;  $b = 2;  $a = 1;  echo "<br>$a \*= $b = ";  echo $a \*= $b;  $b = 2;  $a = 1;  echo "<br>$a /= $b = ";  echo $a /= $b;  $b = 2;  $a = 1;  echo "<br>$a %= $b = ";  echo $a %= $b;  $b = 2;  $a = 1;  echo "<br>$a &= $b = ";  echo $a &= $b;  $b = 2;  $a = 1;  echo "<br>$a |= $b = ";  echo $a |= $b;  $b = 2;  $a = 1;  echo "<br>$a ^= $b = ";  echo $a ^= $b;  $b = 2;  $a = 1;  echo "<br>$a <<= $b = ";  echo $a <<= $b;  $b = 2;  $a = 1;  echo "<br>$a >>= $b = ";  echo $a >>= $b;  $b = 2;  $a = 1;  echo "<br>$a .= $b = ";  echo $a .= $b;  echo "</TT>";  ?> |

Rezultatul codului de mai sus este:  
  
1 += 2 = 3  
1 -= 2 = -1  
1 \*= 2 = 2  
1 /= 2 = 0.5  
1 %= 2 = 1  
1 &= 2 = 0  
1 |= 2 = 3  
1 ^= 2 = 3  
1 <<= 2 = 4  
1 >>= 2 = 0  
1 .= 2 = 12

**Operatori de incrementare și decrementare**

Operatorul de incrementare **(++)** are ca efect creșterea cu 1 a valorii unei variabile, iar cel de decrementare **(--)** are ca efect scăderea cu 1 a valorii variabilei. Tipul variabilei poate fi întreg sau real. Dacă operatorul precede variabila, atunci rezultatul expresiei este valoarea obținută după incrementare sau decrementare. Dacă variabila precede operatorul, atunci rezultatul expresiei este valoarea variabilei înaintea incrementării sau decrementării. Cu alte cuvinte, dacă operatorul precede variabila, atunci valoarea variabilei este mai întîi modificată și apoi utilizată, iar dacă variabila precede operatorul, atunci valoarea ei este mai întîi utilizată și apoi modificată.  
Acești operatori pot fi utilizați și pentru variabile care conțin șiruri de caractere. Operatorul de incrementare duce la creșterea cu 1 a codului ASCII a ultimului caracter din șir dacă acesta este o litera sau cifră. În cazul în care litera este **'z'**, respectiv **'Z'**, sau cifra este **'9'**, atunci ea devine **'a'**, respectiv **'A'**, sau cifra **'0'** și se încearcă incrementarea penultimului caracter. Dacă aceasta este tot **'9'**, **'z'** sau **'Z'** se aplică același procedeu și se trece la antepenultimul caracter. Procedeul continuă pînă în momentul în care se ajunge la un caracter care nu este **'9'**, **'z'** sau **'Z'** sau se ajunge la începutul șirului. În acest ultim caz, la începutul șirului se adaugă caracterul **'1'**, **'a'** sau **'A'**. Incrementarea nu are nici un efect pentru caracterele care nu sunt cifre sau litere.  
Decrementarea șirurilor de caractere nu are nici un efect, valorile variabilelor rămînînd nemodificate. Există o singură excepție și anume șirurile care reprezintă numere întregi sau reale. Acestea sunt incrementate sau decrementate potrivit regulilor pentru valorile numerice.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | <?php    $a=7;    echo"<tt>a=$a";    echo": </tt><i>Rezultatul operatiei </i><tt>a++</tt><i> este </i><tt>";    echo$a++;    echo "</tt>. <i>Noua valoare a variabilei </i><tt>a</tt><i> este </i><tt>$a.";    echo "<br>a = $a";    echo ": </tt><i>Rezultatul operatiei </i><tt>++a</tt><i> este </i><tt>";    echo++$a;    echo"</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i> este </i><tt>$a.";    echo"<br>a=$a";    echo": </tt><i>Rezultatul operatiei </i><tt>a--</tt><i> este </i><tt>";    echo $a--;    echo "</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i> este </i><tt>$a.";    echo "<br>a=$a";    echo ":</tt><i>Rezultatul operatiei </i><tt>--a</tt><i> este </i><tt>";    echo --$a;    echo"</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i> este </i><tt>$a.";    $a=7.5;    echo"<br>a=$a";    echo ":</tt><i>Rezultatul operatiei </i><tt>a++</tt><i> este </i><tt>";    echo $a++;    echo"</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i> este </i><tt>$a.";    echo"<br>a=$a";    echo":</tt><i>Rezultatul operatiei </i><tt>++a</tt><i> este </i><tt>";    echo++$a;    echo"</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i> este </i><tt>$a.";    echo"<br>a=$a";    echo":</tt><i>Rezultatul operatiei </i><tt>a--</tt><i> este </i><tt>";    echo$a--;    echo"</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i> este </i><tt>$a.";    echo"<br>a=$a";    echo":</tt><i>Rezultatul operatiei </i><tt>--a</tt><i> este </i><tt>";    echo--$a;    echo "</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i>este </i><tt>$a.";    $a ="PHP";    echo "<br>a =$a";    echo ":</tt><i>Rezultatul operatiei </i><tt>a++</tt><i> este </i><tt>";    echo $a++;    echo "</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i> este </i><tt>$a.";    echo "<br>a=$a";    echo ":</tt><i>Rezultatul operatiei </i><tt>++a</tt><i> este</i><tt>";    echo ++$a;    echo"</tt>. <i>Noua valoare a variabilei </i><tt>a</tt><i> este </i><tt>$a.";    echo"<br>a=$a";    echo":</tt><i>Rezultatul operatiei </i><tt>a--</tt><i>este </i><tt>";    echo$a--;    echo"</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i>este </i><tt>$a.";    echo"<br>a=$a";    echo":</tt><i>Rezultatul operatiei </i><tt>--a</tt><i>este</i><tt>";    echo --$a;    echo "</tt>.<i>Noua valoare a variabilei </i><tt>a</tt><i>este </i><tt>$a.";    echo "</tt>";    ?> |

Rezultatul codului de mai sus este:  
  
a = 7: *Rezultatul operatiei*a++*este*7. *Noua valoare a variabilei*a*este*8.  
a = 8: *Rezultatul operatiei*++a*este*9. *Noua valoare a variabilei*a*este*9.  
a = 9: *Rezultatul operatiei*a--*este*9. *Noua valoare a variabilei*a*este*8.  
a = 8: *Rezultatul operatiei*--a*este*7. *Noua valoare a variabilei*a*este*7.  
a = 7.5: *Rezultatul operatiei*a++*este*7.5. *Noua valoare a variabilei*a*este*8.5.  
a = 8.5: *Rezultatul operatiei*++a*este*9.5. *Noua valoare a variabilei*a*este*9.5.  
a = 9.5: *Rezultatul operatiei*a--*este*9.5. *Noua valoare a variabilei*a*este*8.5.  
a = 8.5: *Rezultatul operatiei*--a*este*7.5. *Noua valoare a variabilei*a*este*7.5.  
a = PHP: *Rezultatul operatiei*a++*este*PHP. *Noua valoare a variabilei*a*este*PHQ.  
a = PHQ: *Rezultatul operatiei*++a*este*PHR. *Noua valoare a variabilei*a*este*PHR.  
a = PHR: *Rezultatul operatiei*a--*este*PHR. *Noua valoare a variabilei*a*este*PHR.  
a = PHR: *Rezultatul operatiei*--a*este*PHR. *Noua valoare a variabilei*a*este*PHR.

**Operatori de execuție**

Interpretorul PHP permite executarea unor comenzi sistem prin intermediul operatorului**(` `)**. Comanda respectivă este cuprinsă între apostroafele inverse (tasta de lîngă 1 deasupra tastei tab), iar rezultatul acestei comenzi este un șir de caractere care reprezintă și rezultatul expresiei.  
Urmatorul *script* PHP determină execuția pe *server* a comenzii **help /?** și afișarea rezultatului în fereastra programului de navigare.

|  |  |
| --- | --- |
| 1  2  3 | <?php    echo "<PRE>".`help /?`."</PRE>";    ?> |

Atenție!  
Pe unele configurații de server nu se permite executarea de comenzi prin intermediul PHP (ca măsură de siguranță, altfel oricine ar putea da un format sau reboot la server prin intermediul unui script PHP).

**Operatori pentru vectori**

Singurul operator care poate fi aplicat asupra vectorilor este cel de concaternare **(+)** Rezultatul concaternării a doi vectori este un vector care conține elementele din cei doi vectori. Dacă cei doi vectori conțin elemente cu aceeași cheie, atunci este păstrat doar elementul din vectorul care reprezintă primul operînd.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?php    $x[2]=2;    $x[4]=4;    $x[5]=5;    $x[7]=7;    $y[1]=100;    $y[2]=200;    $y[3]=300;    $y[4]=400;    $y[5]=500;    $z =$x+$y;    echo"$z[1]$z[2]$z[3]$z[4]$z[5]$z[7]";    ?> |

Rezultatul codului de mai sus este:  
100 2 300 4 5 7

Exerciții:  
1. Se dau două numere reale **x**și **y**. Să se calculeze următoarele expresii:  
A=2+x-y;  
B=x\*A+y;  
C=A-2\*B+x.

**Structuri condiționale**

**Structura *if***

Una dintre cele mai importante structuri în orice limbaj este cea alternativă. În PHP sintaxa acestei structuri este următoarea:

|  |  |
| --- | --- |
| 1 | if (conditie) {instructiuni;} |

Folosirea unei astfel de structuri indică faptul că instrucțiunile se vor executa dacă și numai dacă valoarea expresiei *conditie* (eventual după conversia la tipul boolean) va fi ***true***.

Exemplu

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?php  $a = 12;  $b = 8;  $rezultat = $a + $b;  if($rezultat == 20) {echo "Rezultatul este perfect";}  ?> |

se compară două variabile şi se afişează un mesaj:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $a=4;  $b=3;  echo "$a=4<br>";  echo "$b=3<br>";  if ($a>$b) {echo "$a>$b";}  ?> |

**Construcția *else***

În foarte multe cazuri dorim să se execute alte instrucțiuni dacă valoarea expresiei condiție este ***false***. În PHP putem utiliza o construcție de tipul ***else*** în acest scop. Sintaxa este:

|  |  |
| --- | --- |
| 1 | if (conditie) {instructiuni #1;} else {instructiuni #2;} |

Exemplu

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $a = 20;  $b = 8;  $rezultat = $a + $b;  if($rezultat == 20) {echo "Rezultatul este perfect";}   else {echo "Rezultatul nu este egal cu cel din conditie";}  ?> |

**Construcția *elseif***

Uneori, suntem nevoiți să folosim structuri alternative imbricate în diferite scopuri. Folosind construcții de tipul ***if - else*** vom obține o secvență de tipul:

|  |  |
| --- | --- |
| 1  2  3  4  5 | if (conditie #1) {instructiuni #1;}  else if (conditie #2) {instructiuni #2;}  ...  else if (conditie #n) {instructiuni #n;}  else {instructiuni #n+1;} |

Limbajul PHP permite folosirea unei prescurtări și anume construcția ***elseif***. Practic, aceasta înlocuieste un ***else*** urmat de un ***if***. Folosind această structură, codul anterior poate fi scris astfel:

|  |  |
| --- | --- |
| 1  2  3  4  5 | if (conditie #1) {instructiuni #1;}  elseif (conditie #2) {instructiuni #2;}  ...  elseif (conditie #n) {instructiuni #n;}  else {instructiuni #n+1;} |

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $a = 20;  $b = 8;  if ($a > $b) {echo "a este mai mare ca b";}   elseif ($a == $b) {echo "a este egal cu b";}    else {echo "a este mai mic ca b";}  ?> |

**Structura *switch***

În anumite cazuri trebuie să efectuăm anumite operații în funcție de anumite valori ale unei expresii. O soluție ar fi folosirea unor structuri ***if*** imbricate sau a uneia singure în care apar mai multe construcții ***elseif***.  
Să presupunem că o anumită variabilă poate avea cinci valori distincte și pentru fiecare valoare trebuie executată o anumită instrucțiune. Dacă folosim structura ***if***, secvența de cod va fi următoarea:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | if ($variabila == valoare #1)  instructiune #1  elseif ($variabila == valoare #2)  instructiune #2  elseif ($variabila == valoare #3)  instructiune #3  elseif ($variabila == valoare #4)  instructiune #4  else  instructiune #5 |

Folosind structura ***switch***, secvența echivalentă este:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | switch ($variabila) {  case valoare #1: instructiune #1; break;  case valoare #2: instructiune #2; break;  case valoare #3: instructiune #3; break;  case valoare #4: instructiune #4; break;  case valoare #5: instructiune #5; break;  } |

Prezentăm în continuare sintaxa generală a structurii ***switch***:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | switch (expresie) {  case valoare #1: grup instructiuni #1; break;  case valoare #2: grup instructiuni #2; break;  ...  case valoare #n: grup instructiuni #n; break;  [default: grup instructiuni #n + 1;]  } |

Pentru început se determină valoarea expresiei și apoi se verifică dacă aceasta apare ca valoare pentru una dintre construcțiile ***case***. În caz afirmativ, toate instrucțiunile începînd cu cea corespunzătoare valorii respective (pînă la sfîrșitul blocului ***swich***) vor fi executate. Execuția acestor instrucțiuni poate fi întreruptă dacă este folosită instrucțiunea ***break***, care va fi prezentată mai jos. Dacă valoarea expresiei nu corespunde nici uneia dintre valorile corespunzătoare construcțiilor ***case***, se execută instrucțiunile corespunzătoare construcției ***default***. Dacă aceasta lipseste, atunci nu se execută nici o instrucțiune. Trebuie precizat faptul că valorile corespunzătoare construcțiilor ***case*** pot fi numere întregi, numere reale sau șiruri de caractere; nu pot fi utilizate obiecte sau vectori.

Exemplu: Transformă numărul zilei în numele ei

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?php  $ziua=2;  echo $ziua;  echo "<br>";  switch ($ziua) {  case 1: echo "Luni";  break;  case 2: echo "Marti"; break;  case 3: echo "Miercuri"; break;  case 4: echo "Joi"; break;  case 5: echo "Vineri"; break;  case 6: echo "Sambata"; break;  case 7: echo "Duminica"; break;  }  ?> |
| 1  2  3  4  5  6  7  8  9 | <?php   $i=2;   Switch($i)  {   case 0: echo "i este egal cu 0"; break;   case 1: echo "i este egal cu 1"; break;   case 2: echo "i este egal cu 2"; break;   default: echo "i nu este egal cu 0,1,2";   }  ?> |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php  if(!isset($\_GET['pag'])) $\_GET['pag'] = '';  switch($\_GET['pag'])  {  case '': echo 'Pagina  switch.php'; break;  case 'pagina1': echo 'Pagina  switch.php?pag=pagina1'; break;  case 'pagina2': echo 'Pagina  switch.php?pag=pagina2'; break;  }  ?> |

Exerciții:  
1. Să se afle minimum și maximum din trei numere date **a, b, c,**cu ajutorul instrucțiunii if, else.  
2. Sunt date trei numere **a, b, c**. De afișat numerele în ordine crescătoare(descrescătoare).

**Structuri repetitive**

**Structura *while***

Majoritatea script-urilor PHP vor conține cicluri. Pentru a folosi o buclă anterior condiționată vom utiliza structura ***while***. Sintaxa acestei structuri este:

|  |  |
| --- | --- |
| 1 | while (conditie) {instructiuni;} |

Efectul acestei structuri este executarea instrucțiunii atît timp cît valoarea expresiei *conditie* (eventual după conversia la tipul *boolean*) este *true*. Trebuie observat faptul că este posibil ca instrucțiunile din acoladă să nu fie executată niciodată.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $numar=1;  while ($numar<=7) {  echo $numar."<br>";  $numar++;  }  ?> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | $brush\_price = 5;   $counter = 10;   echo "<table border=\"1\"  align=\"center\">";    echo  "<tr><th>Quantity</th>";    echo "<th>Price</th></tr>";    while ( $counter <= 100 ) {    echo  "<tr><td>";   $counter;   echo  "</td><td>";   echo  $brush\_price \* $counter;   echo  "</td></tr>";   $counter  = $counter + 10;   }   echo "</table>"; |

**Structura *do - while***

Uneori, dorim să folosim bucle posterior condiționate. În acest scop, în PHP avem la dispoziție structura ***do - while*** a cărei sintaxă este:

|  |  |
| --- | --- |
| 1 | do {instructiuni;} while (conditie); |

Singura diferență față de structura while este faptul că valoarea condiției este determinată, de fiecare dată, după executarea instrucțiunii. Ca urmare, instrucțiunea va fi executată cel puțin o dată. Nu există o sintaxă alternativă pentru această structură.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php  $i=1;  do {  echo $i."<br>";  $i++;  }  while ($i<=5)  ?> |

Exercițiu:  
1. Se dă șirul 1..20, utilizînd operatorul *while*. Să se afișeze numerele pare apoi și numerele impare din șirul dat.

**Structura*for***

Sintaxa este foarte asemănătoare cu cea din limbajele C/C++ și Java și anume:

|  |  |
| --- | --- |
| 1  2  3 | for (expresie #1; conditie; expresie #2) {   instructiuni;   } |

Prima expresie este evaluată o singură dată, înainte de începerea execuției ciclului. Expresia *conditie* este testată înaintea fiecarei repetari a buclei. Dacă expresia returneaza valoarea **false**, repetarea se oprește.  *expresie #2* este executata la fiecare repetare a buclei.  
*instructiuni* se execută la fiecare repetare a buclei.  
Oricare dintre cele trei expresii poate lipsi. În cazul în care o expresie lipsește, se consideră că ea are valoarea *true*.  
Bucla WHILE și FOR sunt identice din punct de vedere funcțional însă bucla FOR este puțin mai complexă.

Exemple:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <?php  for ($i=1; $i<=10; $i++) {  echo $i."<br>";  }  ?> |
| 1  2  3  4  5 | <?php  for($i=10; $i>=1; $i--) {  echo $i."<br>";  }  ?> |

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?php  for($a=1; $a<=10; $a++) {  $r=$a%2;  if($r==1) echo "$a - impar" else echo "$a - par";  }  ?> |
| 1  2  3  4  5  6  7  8  9 | <?php  $i = 0;  $s = 0;  do {   $s = $s +$i;   $i++;  } while ($i <= 3);  echo "Suma este $s";  ?> |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php  echo"<table  border=\"1\">\n<tr><td>Celula</td></tr>\n";  $culoare="yellow";  for($i=1; $i<=10; $i++){  if($culoare=="yellow") $culoare="red";  else $culoare="yellow";  echo "<tr><td  bgcolor=".$culoare.">".$i."</td></tr>\n";  }  echo"</table>";  ?> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?php  for ($i=1; $i<=15; $i++):  echo"<b><font size =3 ";  echo"color=";  if(!($i%2))  echo"red";  elseif(!($i%3))  echo"green";  else  echo"blue";  echo">$i</font></b>";  endfor;  ?> |

Exerciții:  
1. Sunt date numerele **K**și **N**(N>0). De afișat de **N**ori numărul **K**;  
2. De elaborat un script ce va afișa tabla înmulțirii;  
3. De elaborat un script ce va afișa tabla de șah;  
4. De elaborat un script care va calcula numărul exact de ani, luni și zile dela ziua de naștere.  
5. Să se scrie un program care să afișeze următoarele valori:  
   1\*9 + 2 = ?  
   12\*9 + 3 = ?  
   123\*9 + 4 = ?  
   .................  
   123456789\*9 + 10 = ?  
6. Să se scrie un program care să afișeze următoarea "piramidă" de numere:  
               1  
          1 2 3  
     1 2 3 4 5  
 ..............  
 1 ......(2n-1)

**Tipul tablou(matrice) în PHP**

**1.Generalitati: definitie, forme**

Variabilele matrice (array, tablou), sunt acel tip de variabile, care pot sa stocheze o matrice de valori (mai multe valori). Variabilele matrice sunt variabilele care pot stoca colectii de valori, organizat ca seturi pereche, **cheie - valoare**.

Tipul acestor valori, poate fi de acelasi tip, dar poate fi si diferita. Variabilele matrice suporta orice tip de date suportate de PHP, inclusiv "tipul de date" matrice, cea ce face posibila crearea de variabile matrice multidimensionale.

În funcție de tipul, natura, cheii folosite, sunt două forme de variabile matrice:

* **indexate** - cheile acestor variabile matrice sunt numere intregi, incepand cu 0
* **asociative** - cheile acestor variabile matrice sunt siruri de caractere

In realitate, PHP trateaza toate variabilele matrice, ca variabile matrice asociative, cu diferenta ca cheile matricii indexate, sunt numere intregi. In ambele cazuri, cheile sunt unice, prin urmare nu putem sa avem doua elemente cu aceasi cheie. Reciproca nu este valabila, in sensul ca doua elemente cu chei diferite pot sa aiba aceasi valoare.

Variabilele matrice in PHP, au o ordine interna a elementelor lor, care este independenta de chei sau de valori. Ordinea normala este data de ordinea in care s-au initializat elementele, dar functiile de sortare despre care vom vorbi mai tarziu, dau posibilitatea schimbarii ordinii interne in functie de chei, de valori, sau oricum doriti.

**2.Initializarea variabilelor matrice**

Initializarea variabilelor matrice, se poate realiza simplu, initializand fiecare element separat, ca o variabila simpla, sau folosind unele functi interne ale PHP, dedicate pentru acest lucru, cum este **array ()**.

**2.1.Initializarea separata a variabilelor matrice**

Pentru initializarea variabilelor matrice, cel mai la indemana, este initializarea fiecarui element separat:

|  |  |
| --- | --- |
| Initializarea variabilelor matrice - fiecare element separat | |
| **Matrice indexata** | **Matrice asociativa** |
| $a[0]="Vasile";  $a[1]="Ion";  $a[2]="Gheorghe"; | $a["Popescu"]="Vasile";  $a["Dascalu"]="Ion";  $a["Popa"]="Gheorghe"; |

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <?php  /\*  variabile matrice - initializare separata \*/  $a1[0]="Vasile";  $a1[1]="Ion";  $a1[2]="Gheorghe";  echo "<h3>Initializarea separata a variabilelor matrice</h3>";  for ($ind=0; $ind<3; $ind++)     { echo $a1[$ind]."<br />";}     print\_r ($a1);     echo "<hr />";  $a2["Popescu"]="Vasile";  $a2["Dascalu"]="Ion";  $a2["Popa"]="Gheorghe";     echo $a2["Popescu"]."<br />";     echo $a2["Dascalu"]."<br />";     echo $a2["Popa"]."<br />";     print\_r ($a2);  ?> |

**2.2. Initializarea variabilelor matrice cu functia array**

Variabilele de tip tablou se mai pot initializa si prin constructia sau declaratia **array()**, care este, cel putin in cazul matricilor unidimensionale, mai usoara:

|  |  |
| --- | --- |
| Initializarea variabilelor matrice cu constructia **array()** | |
| **Matrice indexata** | **Matrice asociativa** |
| $a=array("Vasile", "Ion", "Gheorghe"); | $a=array("Popescu"=> "Vasile", "Dascalu"=> "Ion", "Popa"=> "Gheorghe"); |

Valorile (in cazul variabilelor cu index), sau perechile cheie - valoare (in cazul tablourilor asociative) sunt separate prin virgula.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <?php  /\*  variabile matrice - initializare cu array \*/  echo "<h3>Initializarea cu array a variabilelor matrice</h3>";  $a1=array ("Vasile", "Ion", "Gheorghe");    for ($ind=0; $ind<3; $ind++)     { echo $a1[$ind]."<br />";}       print\_r ($a1);       echo "<hr />";  $a2=array    ("Popescu"   => "Vasile",     "Dascalu"=> "Ion",     "Popa"      => "Gheorghe");     echo $a2["Popescu"]."<br />";     echo $a2["Dascalu"]."<br />";     echo $a2["Popa"]."<br />";     print\_r ($a2);     echo "<hr />";;  ?> |

Cu ajutorul constructiei **array ()**, variabilele matrice se mai pot initializa si dupa cum urmeaza:

* se defineste o variabila matrice fara valori: **$var=array();**
* se specifica o cheie de pornire, cu **=>**, si apoi o lista de valori; valorile sunt inserate in matrice, incepand cu cheia specificata

In cazul tablourilor indexate, urmatoarele chei vor fi in ordine crescatoare fata de cheia initiala

In cazul tablourilor a caror cheie de pornire este un sir, cheile urmatoare vor fi numere intregi, prima valoare fiind 0.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <?php  /\* variabile matrice - initializare cu array \*/  echo "<h3>Initializarea cu array a variabilelor matrice</h3>";  $a1=array();  $a1=array (1=>"Vasile", "Ion", "Gheorghe");    for ($ind=1; $ind<4; $ind++)     {echo $a1[$ind]."<br />";}      print\_r($a1);      echo "<hr />";  $a2=array ("Popescu"   => "Vasile", "Ion", "Gheorghe");  $key=array ("Popescu","0","1");      echo $a2["Popescu"]."<br />";      echo $a2["0"]."<br />";      echo $a2["1"]."<br />";      print\_r($a2);      echo "<br />";    for ($ind=0; $ind<3; $ind++)     {echo $key[$ind]." => ".$a2[$key[$ind]]."<br />";}      echo "<hr />";  ?> |

**2.3. Adaugarea de valori suplimentare, ulterior initializarii**

Daca cunoastem indicele ultimei valori a variabilei, putem adauga noi elemente prin indicarea indexului:  
**$var[4] = "Nat";**

Totusi, nu intodeauna cunostem cate elemente are o variabila, sau care este cheia ultimului element. PHP va furnizeaza o cale simpla, pentru a adauga variabilei un nou element:

**$var[] = "val\_sup";**

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?php  /\*  variabile matrice - elemente suplimentare \*/  echo "<h3>Setarea cu noi valori a variabilelor matrice</h3>";  $a=array ("Vasile", "Ion", "Gheorghe");  print\_r ($a);  echo "<hr />";#################################################  $a [3]="Dinu";  print\_r ($a);  echo "<hr />";#################################################  $a []="Stefan";  $a []="Petrache";  print\_r($a);  echo "<hr />";  ?> |

**3.Identificarea elementelor unei variabile matrice**

Pentru a accesa valoarea unui anumit element dintr-o variabila matrice, folosim numele variabilei, urmat de cheia elementului intre paranteze drepte:

**$nume\_var[numar]  
$nume\_var["sir\_cheie"]**

Cheia, **sir\_cheie** poate sa fie, un sir sau un numar intreg. Cand este un numar intreg, este tratat ca atare. Astfel, **$var[3]** si **$var["3"]** sunt acelasi element, dar **$var["03"]** este alt element. Numerele negative sunt chei valide, si ele nu specifica pozitia la sfarsitul tabloului ca in Perl.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?php  /\* variabile matrice - identificare \*/  $a=array ("Vasile", "Ion", "Gheorghe");  echo $a[1]."<br />";  $a=array    ("0"   => "Vasile",     "-1"  => "Ion",     "02"  => "Gheorghe");  echo $a["-1"]."<br />";  print\_r($a);  ?> |

Sirurile dintr-un singur cuvant, nu este obligatoriu sa fie puse intre ghilimele. De exemplu, **$var[Popescu]** este la fel ca**$var["Popescu"]**. Totusi, se considera ca o buna practica PHP, sa folositi intodeauna ghilimelele, deoarece cheile fara ghilimele, nu se pot deosebi de constantele introduse ca index:

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php  /\*  variabile matrice - identificare \*/  define ("IDX", 1);   //definim o constanta  $a=array    ("IDX"   => "Vasile",      IDX    => "Ion", "Gheorghe");  echo $a[IDX]."<br />";  echo $a["IDX"]."<br />";  print\_r($a);  ?> |

Cand aveti nevoie sa recuperati elemente prin indexul matricii (matrice asociativa cu index), trebuie sa folositi ghilimelele duble, pentru ca PHP sa recunoasca variabila indice.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?php  /\* variabile matrice - identificare \*/  $a['ind1']="Vasile";  $a['ind2']="Ion";  $a['ind3']="Gheorghe";  for ($i=1; $i<=3; $i++)    {echo $a["ind$i"]."<br />";}  print\_r($a);  ?> |

**4.Variabilele matrice multi-dimensionale**

Vom vorbii in particular despre matricile (tablourile) bidimensionale, deoarece sunt mai usor de vizualizat, atat vizual, cat si mental. Vom lua un tabel, valorile sale le vom introduce intr-o variabila matrice, ca sa avem o asociere buna.

|  |  |  |  |
| --- | --- | --- | --- |
| Tabel exemplu | | | |
| **-** | **Nume** | **Prenume** | **An nastere** |
| Rand/Col | 0 | 1 | 2 |
| 0 | Popescu | Vasile | 1964 |
| 1 | Dascalu | Ion | 1982 |
| 2 | Popa | Gheorghe | 1955 |

Lasand la o parte capul de tabel, precum si randul, respectiv coloana, pentru numerotarea randurilor si coloanelor, avem un tabel cu trei randuri si trei coloane. Valorile acestui tabel se pot stoca intr-o variabila matrice bidimensionala in trei modalitati:

* prin initializarea separata a fiecareui element

**$a[0][0]="Popescu";  
$a[0][1]="Vasile";  
$a[0][2]="1964";  
$a[1][0]="Dascalu";  
$a[1][1]="Ion";  
etc.**

* prin initializarea mixta, fiecare rand cu constructia **array**

**$a[0]=array("Popescu", "Vasile", "1964");  
$a[1]=array("Dascalu", "Ion", "1982");  
$a[2]=array("Popa", "Gheorghe", "1955");**

* doar cu constructia **array**

**$a=array(array("Popescu", "Vasile", "1964"), array("Dascalu", "Ion", "1982"), array("Popa", "Gheorghe", "1955") );**

Observam ca matricile bidimensionale, au evident doi indici. Prin extrapolare, putem spune ca o matrice **n** dimensionala trebuie sa aiba**n** indici. Nu am vazut nicaieri vreo restrictie privind dimensiunea unei variabile matrice, eu am folosit in acest curs matrice cu 4 dimensiuni, dar nu este bine sa exagerati.

Recuperarea valorilor, elementelor unei matrici bidimensionale, este identic procedural ca a matricilor unidimensionale, adica prin indicarea cheilor variabilei, deci trebuie sa cunoastem cheile elementului a carei valoare vrem s-o recuperam:

**$nume\_var[cheie\_1][cheie\_2]**

Deoarece PHP trateaza cele doua tipuri de variabila matrice, ca variabile asociative, putem sa spunem ca si variabilele asociative se initializeaza si recupereaza similar.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <?php  /\* variabile matrice bidimensionale \*/  $a[0]=array("Nume"      =>"Popescu",               "Prenume"   =>"Vasile",               "An\_nastere"=>"1964");  $a[1]=array("Nume"      =>"Dascalu",               "Prenume"   =>"Ion",               "An\_nastere"=>"1982");  $a[2]=array("Nume"      =>"Popa",               "Prenume"   =>"Gheorghe",               "An\_nastere"=>"1955");  for ($i=0; $i<=2; $i++)    {echo $a["$i"]["Nume"]." - ".          $a["$i"]["Prenume"]." - ".          $a["$i"]["An\_nastere"]."<br />";}  ?> |

**5.Functiile specifice variabilelor matrice**

Limbajul PHP chiar daca are un numar restrins de instructiuni, are in schimb un numar impresionant de funtii interne, care dau puterea acestui limbaj de programare.

Cele mai importante domenii privind variabilele matrice sunt legate de initializarea (crearea) lor, precum si identificarea elementelor sale. Alte domenii cel putin la fel de importante sunt modificarea si efectuarea de operatii cu elementele variabilelor matrice. Un alt domeniu interesant, este domeniul verificarilor. Functiile interne furnizate de PHP, incearca sa acopere aceste domenii, unele din ele, interferand chiar doua domenii.

**5.1.Functii pentru crearea variabilelor matrice**

Functiile doar pentru crearea variabilelor matrice sunt:

|  |  |
| --- | --- |
| Functii pentru crearea de noi variabile matrice | |
| **Funcția** | **Descriere** |
| array() | Crează o variabilă matrice. |
| range() | Întoarce o variabilă matrice, a carui elemente au valoarile ordonate de la o valoare minima la una maxima, cu diferenta intre valorile elementelor consecutive egala cu un pas - ordonarea poate fi si descrescatoare. |

**5.1.1.functia *array()***

Aceasta functie am vazut-o in actiune, asa ca vom vorbi foarte pe scurt despre ea. In primul rand este principala metoda pentru crearea variabilelor matrice.

**Sintaxa:**

**$nume\_var = array ("cheie\_1" => "val\_1", "cheie\_2" => "val\_2", ... );**

Aceasta este sintaxa generala. Dupa cum am vazut sunt cazuri particulare.

* **cheie\_1, cheie\_2** - sunt valorile cheilor viitoarei variabile matrice - sunt optionale
* **val\_1, val\_2** - sunt valorile elementelor viitoarei variabile matrice, corespunzator cheii - si ele sunt optionale, daca sunt omise si ele, se defineste o variabila matrice fara valori
* **=>** - defineste corespondenta cheie - valoare - optional, in cazul in care cheile nu sunt definite, chiar trebuie sa lipseasca.

Cheile **cheie\_1, cheie\_2** pot fi, fie numere intregi, fie sir de caractere. Cand este omisa atunci functia genereaza automat chei numerice incepand cu 0, cu rata de crestere 1.

**Descriere si functionare**

Aceasta functie creaza (intoarce) o variabila matrice, cu perechi chei valori, conform argumentelor setate.

**5.1.2. functia *range()***

**Sintaxa:**

**$nume\_var = range (val\_min, val\_max, pas);**

|  |  |
| --- | --- |
| Argumentele functiei *range()* | |
| **Argument** | **Descriere** |
| val\_min | Necesar. Specifica valoarea minima a valorilor elementelor variabilei matrice, care in general se aloca elementului cu indicele 0. |
| val\_max | Necesar. Specifica valoarea maxima (cea mai mare) a valorilor elementelor variabilei matrice, care in general se aloca elementului cu indicele cel mai mare. |
| pas | Optional. Specifica pasul cu care cresc valorile elementelor variabilei matrice. Predefinit (deci daca nu setam acest argument) este 1. |

**Descriere si functionare**

Aceasta functie creaza o variabila matrice, care contine o clasa de elemente, ale carei valori sunt de la **val\_min** pana la inclusiv **val\_max**, cu pasul intre valori determinat de **pas**. Altfel spus, intoarce o variabila matrice, a carui elemente au valoarile ordonate de la **val\_min** la **val\_max**, cu diferenta intre valorile elementelor consecutive egala cu **pas**.

**Nota:** Daca parametrul **val\_min** este mai mare decat **val\_max** atunci ordonarea se va face de la mai mare spre mai mic.

**Exemplu:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?php  /\*  functia range \*/  $a = range(1,4);  print\_r ($a);  echo "<hr />";  ##################  $a = range(1,12,3);  print\_r ($a);  echo "<hr />";  ##################  $a = range("e","s",2);  print\_r ($a);  echo "<hr />";  ?> |

**5.2.Functii pentru crearea variabilelor matrice prin procesarea altor variabile**

Pentru acest caz, PHP furnizeaza mai multe functii, dintre care am selectat:

|  |  |
| --- | --- |
| Functii pentru crearea de noi variabile matrice | |
| **Functia** | **Descriere** |
| array\_pad() | Intoarce o variabila matrice care are inserate un numar specificat de elemente, cu valoare definita, pe langa elementele matricii care se prelucreaza. |
| array\_slice() | Intoarce o variabila matrice, care are ca elemente, unele din elementele selectate ale variabilei matrice argument, fara a o afecta pe aceasta. |
| array\_chunk() | Intoarce o variabila matrice bidimensionala, prin impartirea variabilei matrice argument. |
| array\_keys() | Intoarce o variabila matrice, care are ca elemente, cheile altei variabile matrice. |
| array\_values() | Intoarce o variabila matrice, insa indexata, care are ca elemente, valorile altei variabile matrice, asociative. |
| explode() | Intoarce o variabila matrice, dintr-un sir de caractere, prin impartirea sirului de caractere, in functie de un separator. |
| compact() | Intoarce o variabila de matrice, cu valorile unor variabilelor simple. |
| array\_reverse() | Intoarce o variabila matrice, cu valorile elementelor in ordine inversata fata de variabila matrice argument. |
| array\_flip() | Intoarce o variabila matrice, care are ca chei, valorile variabilei matrice argument, iar ca valori, cheile acestei variabile. |
| array\_keys() | Intoarce o variabila matrice care contine valorile tuturor variabilelor matrice care sunt argumentele functiei. |
| array\_chunk() | Aceasta functie compara doua sau mai multe variabile matrice, si intoarce o alta variabila matrice, cu elementele primei variabile (cheile si valorile sale), din care se extrag elementele a caror valoare se gaseste in una din variabilele urmatoare. |
| array\_unique() | Aceasta functie intoarce o variabila matrice, care are valorile elementelor matricei argument, dintre care se inlatura valorile duplicate. Daca doua sau mai multe valori sunt la fel, prima aparitie va fi pastrata, iar celelalte vor fi inlaturate. |
| array\_intersect() | Aceasta functie compara doua sau mai multe variabile matrici, si intoarce o matrice cu cheile si valorile primei matrice, dar doar daca valorile sunt si in celelalte matrice. Daca mai multe chei au aceeasi valoare, prima cheie va fi pastrata. |
| array\_filter() | Aceasta functie trece fiecare valoare a variabilei, unei functi utilizator, de tip comparator, care intoarce fie TRUE fie FALSE. Functia intoarce o variabila matrice doar cu valorile care au intors TRUE. |

**5.2.1. functia *array\_pad()***

**Sintaxa**

**$nume\_var = array\_pad ($array, marime, valoare);**

|  |  |
| --- | --- |
| Argumentele functiei *array\_pad()* | |
| **Argument** | **Descriere** |
| $array | Necesar. Specifica matricea care se prelucreaza. |
| marime | Necesar. Specifica numarul de elemente a noii variabile matrice intoarsa de functie. |
| valoare | Necesar. Specifica valoarea noilor elemente din matrice intoarse de functie. |

**Descriere si functionare**

Aceasta functie, intoarce o variabila matrice, care are ca elemente, elementele variabilei matrice argument, pe langa care sunt inserate un numar specificat de elemente (numarul de elemente este determinat de argumentul **marime**), cu o valoare specificata, (determinata de argumentul **valoare**).

**Nota:**

* nu taie din elementele matricii, cand parametrul **marime**, este mai mic decat numarul original de elemente.
* daca setam **marime** cu un numar negativ, functia va insera elementele noi in fata elementelor originale.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <?php  /\*  functia array\_pad() \*/  $a=array("Nume"      =>"Popescu",            "Prenume"   =>"Vasile",            "An\_nastere"=>"1964");     echo "variabila care se proceseaza este:<br />";     print\_r ($a);     echo "<hr />";    $pad= array\_pad($a,5,"element nou" );     echo "noua variabila este:<br />";     print\_r ($pad);     echo "<hr />";    echo "noua variabila este:<br />";     print\_r (array\_pad($a,1,"element nou"));     echo "<hr />";    echo "noua variabila este:<br />";     print\_r (array\_pad($a,-5,"element nou"));     echo "<hr />";    echo "variabila care s-a procesat a ramas:<br />";     print\_r ($a);  ?> |

**5.2.2. functia *array\_slice()***

**Sintaxa:**

**$nume\_var = array\_slice ($array, start, lungime, pastrare);**

|  |  |
| --- | --- |
| Argumentele functiei *array\_slice ()* | |
| **Argument** | **Descriere** |
| $array | Necesar. Specifica variabila matrice care se prelucreaza. |
| start | Necesar. Este o valoare numerica intraga. Specifica de la al catelea element al variabilei matrice argument (**$array**) incepe functia sa preia elemente. 0 - este incepand cu primul element. Daca acest parametru este setat cu un numar negativ, functia va incepe numararea elementelor, de la ultimul element. (-2 inseamna ca va incepe cu penultimul element) |
| lungime | Optional. Este o valoare numerica intreaga. Specifica numarul de elemente a variabilei matricei intoarse. Daca acest parametru este setat cu un numar negativ, functia nu va mai numara elementele, ci va lua ca ultim element intors, elementul anterior elementului cu numarul setat, dar numarand de la ultimul element (vezi exemplul 5.13). Predefinit functia intoarce toate elementele, incepand de la pozitia setata prin argumentul **start**. |
| pastrare | Optional. Valori posibile: true - pastreaza cheile variabilelor matrice indexate conform variabilei argument (**$array**) false - predefinit, reface cheile variabilelor matrice indexate, incepand cu 0. |

**Descriere si functionare**

Aceasta functie creaza o variabila matrice, taiand o parte dintr-o alta variabila matrice. Cu alte cuvinte, seteaza cheile si valorile unei noi variabile cu o parte din cheile si valorile corespunzatoare ale matricii argument. Partea cu care este setata noua variabila este determinata de argumentele functiei.

* cheia de la care se incepe "taierea" este data de argumentul **start** - de la al **start**-lea element
* cate elemente se transfera, este data de argumentul **lungime**

**Nota:**  
Predefinit, variabilele matrice asociative, raman variabile matrice asociative.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <?php  /\*  functia array\_slice() \*/  $a=array("Popescu", "Vasile", "1964", "Chisinau");  $var=array\_slice ($a, 2);     print\_r ($a);            // $a nu se modifica     echo "<hr />";            #####################     echo "1<br />";print\_r ($var);   # 1     echo "<hr />";            #####################  $var=array\_slice ($a, -3);         # 2     echo "2<br />";print\_r ($var);     echo "<hr />";            #####################  $var=array\_slice ($a, 2, 2);       # 3     echo "3<br />";print\_r ($var);     echo "<hr />";            #####################  $var=array\_slice ($a, 2, 2, true); # 4     echo "4<br />";print\_r ($var);     echo "<hr />";            #####################  $var=array\_slice ($a, 0, 2);       # 5     echo "5<br />";print\_r ($var);     echo "<hr />";            #####################  $var=array\_slice ($a, -3, 2);      # 6     echo "6<br />";print\_r ($var);     echo "<hr />";            #####################  ?> |

Exemplu si cu functionarea valorii negative a argumentului **lungime**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?php  /\* functia array\_slice () \*/  $a=array("Popescu", "Vasile", "1964", "Chisinau");  $var=array\_slice ($a, 1, -2);        # 1     echo "1<br />";print\_r ($var);     echo "<hr />";                   ###########  $var=array\_slice ($a, 1, -1);        # 2     echo "2<br />";print\_r ($var);     echo "<hr />";                   ###########  $var=array\_slice ($a, 0, -2);        # 3     echo "3<br />";print\_r ($var);     echo "<hr />";                   ###########  ?> |

Exemplu cu variabile matrice asociative:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <?php  /\*  functia array\_slice () \*/  $a=array("Nume"         => "Dascalu",            "Prenume"     => "Ion",            "An\_nastere"  => "1982",            "Localitatea" => "Ialoveni");  $var=array\_slice ($a, 1);    print\_r ($a);    echo "<hr />";          ###################  $i=0;$i++;echo "$i<br />";    print\_r ($var);    echo "<hr />";          ###################    print\_r (array\_slice ($a, 1));    echo "<hr />";          ###################  $i++;echo "$i<br />";  print\_r (array\_slice ($a, 0, 2));    echo "<hr />";          ###################  $i++;echo "$i<br />";    print\_r (array\_slice ($a, -2));    echo "<hr />";          ###################  $i++;echo "$i<br />";    print\_r (array\_slice ($a, 0, -2));    echo "<hr />";          ###################  $i++;echo "$i<br />";    print\_r (array\_slice ($a,0,2,true));    echo "<hr />";          ###################  ?> |

**5.2.3. functia *array\_chunk()***

**Sintaxa**

**$nume\_var = array\_chunk ($array, marime, pastrare);**

|  |  |
| --- | --- |
| Argumentele functiei *array\_chunk ()* | |
| **Argument** | **Descriere** |
| $array | Necesar. Specifica variabila matrice care se prelucreaza. |
| marime | Necesar. Specifica cate elemente va contine primul element al noi variabile matrice bidimensionale. |
| pastrare | Optional. Valori posibile: true - pastreaza cheile variabilelor matrice indexate conform variabilei argument false - predefinit, reface cheile variabilelor matrice indexate, incepand cu 0. |

**Descriere si functionare**

Aceasta functie, intoarce o variabila matrice bidimensionala, din elementele altei variabile matrice. Noua variabila matrice va avea primul element cu un numar de **marime** elemente, restul elementelor variabilei matrice argument, vor face parte din al doilea element a variabilei.

Daca argumentul **marime**, are valoarea mai mare decat numarul de elemente a variabilei originale, atunci toate elementele acestei variabile, vor fi alocate primului element a noi variabile.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <?php  /\* functia array\_chunk () \*/  $a=array("Popescu", "Vasile", "1964", "Chisinau");  $var=array\_chunk ($a, 2);     print\_r ($a);          # $a nu se modifica     echo "<hr />";          #######################     echo "1<br />";print\_r ($var);   # 1     echo "<hr />";          #######################  $var=array\_chunk ($a, 3);          # 2     echo "2<br />";print\_r ($var);     echo "<hr />";          #######################  $var=array\_chunk ($a, 5);          # 3     echo "3<br />";print\_r ($var);     echo "<hr />";          #######################  $var=array\_chunk ($a, 2, true);    # 4     echo "4<br />";print\_r ($var);     echo "<hr />";  ?> |

**5.2.4. functia *array\_keys()***

**Sintaxa**

**$nume\_var = array\_keys ($array, val\_key, strict);**

|  |  |
| --- | --- |
| Argumentele functiei *array\_keys()* | |
| **Argument** | **Descriere** |
| $array | Necesar. Specifica variabila matrice argument, cea a carei chei le va prelua noua variabila, intoarsa de functie. |
| valoare | Optional. Puteti specifica o valoare a matricii origine, astfel va fi intoarsa doar cheia cu aceasta valoare. |
| strict | Optional. Folosit impreuna cu parametrul valoare. Valori posibile: true - Intoarce cheia cu valoarea specificata, in functie de tip: numarul 5 nu este la fel cu sirul "5". false - Valoarea predefinita. Nu depinde de tip, numarul 5 este la fel ca sirul "5". |

**Descriere si functionare**

Aceasta functie, intoarce o variabila matrice, ale carei elemente vor fi cheile variabilei matrice argument **$array**.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <?php  /\* functia array\_keys () \*/  $a=array("Nume"        => "Popescu",            "Prenume"     => "Vasile",            "An\_nastere"  => "1964",            "Localitatea" => "Chisinau");  $var=array\_keys ($a);     print\_r ($a);          #   $a nu se modifica     echo "<hr />";          #######################     echo "1<br />";print\_r ($var);     # 1     echo "<hr />";          #######################  $var=array\_keys ($a, 1964);          # 2     echo "2<br />";print\_r ($var);     echo "<hr />";          #######################  $var=array\_keys ($a, 1964, true);    # 3     echo "3<br />";print\_r ($var);     echo "<hr />";          #######################  $var=array\_keys ($a, "1964", true);  # 4     echo "4<br />";print\_r ($var);     echo "<hr />";  ?> |

**5.2.5. functia *array\_values()***

**Sintaxa**

**$nume\_var = array\_values ($array);**

**Descriere si functionare**

Aceasta functie, intoarce o variabila matrice indexata, ale carei elemente vor avea valorile variabilei matrice argument **$array**, care macar pentru logica, trebuie sa fie asociativa.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?php  /\*  functia array\_values () \*/  $a=array("Nume"        => "Popescu",            "Prenume"     => "Vasile",            "An\_nastere"  => "1964",            "Localitatea" => "Chisinau");  $var=array\_values ($a);     print\_r ($a);          // $a nu se modifica     echo "<hr />";  print\_r ($var);     echo "<hr />";  ?> |

**5.2.6. functia *explode ()***

**Sintaxa**

**$nume\_var = explode (separator, sir, limita);**

|  |  |
| --- | --- |
| Argumentele functiei *explode ()* | |
| **Argument** | **Descriere** |
| separator | Necesar. Specifica separatorul care indica unde se intrerupe sirul. |
| sir | Necesar. Este sirul care se imparte, pentru a devenii variabila matrice. |
| limita | Optional. Specifica numarul maxim de elemente a variabilei matrice care va fi intoarsa. |

**Descriere si functionare**

Converteste un sir de caractere (string) intr-o variabila matrice, prin divizarea cu un separator specificat. Intoarce variabila matrice cu subsiruri, creat prin impartirea sirului, ori de cate ori intalneste separatorul.

**Nota:** Separatorul nu poate fi un sir gol.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?php  /\* functia explode () \*/  $str = "Salut lume. Este o zi minunata, ca fiecare zi, de altfel.";  echo "ex 1 <br />";  $var=explode(" ",$str);  print\_r ($var);                  ##   1  echo "<hr />";                   ##############  echo "ex 2 <br />";  print\_r (explode(".",$str));     ##   2  echo "<hr />";                   ##############  echo "ex 2 <br />";  print\_r (explode(",",$str, 2));  ##   3  echo "<hr />";  ?> |

**5.2.7. functia *compact ()***

**Sintaxa**

**$nume\_var = compact (var1, var2,...);**

**Descriere si functionare**

Aceasta functie creaza o variabila matrice, din mai multe variabile simple a caror nume sunt trecute ca argumente ale functiei (**var1, var2, ...**), fara caracterul $. Argumentele functiei (numele functiilor simple, fara caracterul $), devin cheile noi variabile matrice, intoarse de functie, iar valorile variabilelor simple, devin valorile corespunzatoare ale noi variabile.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?php  /\*  functia compact () \*/  $a1 = "Salut lume.";  $a2="Este o zi minunata";  $a3="Ca fiecare zi, de altfel.";  $comp=compact ("a1", "a2", "a3");      echo "$a1<br />";      echo "$a2<br />";      echo "$a3<br />";  echo "<hr />";          ######################      print\_r ($comp);  echo "<hr />";  ?> |

**5.2.8. functia *array\_reverse ()***

**Sintaxa**

**$nume\_var = array\_reverse ($array, pastrare);**

|  |  |
| --- | --- |
| Argumentele functiei *array\_reverse ()* | |
| **Argument** | **Descriere** |
| $array | Necesar. Specifica variabila matrice care se proceseaza. |
| pastrare | Optional. Specifica daca functia isi pastreaza cheile matricii sau nu. Valori posibile: - true - pastreaza cheile variabilei de origine - false - predefinit - aloca chei noi |

**Descriere si functionare**

Acesta funcție întoarce o nouă variabilă matrice, cu valorile elementelor in ordine inversa decat a variabilei matrice care se prelucreaza.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?php  /\*  functia array\_reverse () \*/  $var\_ar=array("cap1", "cap2", "cap3");     print\_r ($var\_ar);     echo "<hr /> ex1 <br />";       ############  $b=array\_reverse ($var\_ar);     print\_r ($b);                         #  1     echo "<hr /> ex2 <br />";     ###############  print\_r(array\_reverse ($var\_ar, true));  #  2     echo "<hr />";                   ############  print\_r ($var\_ar);     echo "<hr />";  ?> |

**5.2.9. functia *array\_flip ()***

**Sintaxa**

**$nume\_var = array\_flip ($array);**

**Descriere si functionare**

Aceasta functie creaza o variabila matrice, care preia valorile matricii care se proceseaza ca chei, iar cheile ca valori.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?php  /\*  functia array\_flip () \*/  $var\_ar=array("cap1", "cap2", "cap3");  print\_r ($var\_ar);  echo "<hr />";  ###############################  $b=array\_flip ($var\_ar);  print\_r ($b);  echo "<hr />";  ###############################  print\_r ($var\_ar);  echo "<hr />";  ?> |

**5.2.10. functia *array\_merge ()***

**Sintaxa**

**$nume\_var = array\_merge ($array1, $array2, ...);**

**Descriere si functionare**

Aceasta functie uneste una **$array1** (necesar), sau mai multe **$array2, ...** (optionale) variabile matrice intr-o noua variabila matrice **$nume\_var**.

**Nota**: Daca doua sau mai multe elemente a variabilelor matrice au aceasi cheie (in cazul variabilelor asociative), cheia ultimei variabile, se suprapune peste celelalte, rezultand un numar de elemente mai redus.

Daca atribuiti o singura variabila functiei si cheile sunt numere intregi, functia va intoarce o noua variabila cu cheie numere intregi pornind de la 0 si crescand cu 1.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <?php  /\*  functia array\_flip () \*/  $var\_ar=array("cap1", "cap2", "cap3");  print\_r ($var\_ar);  echo "<hr />";  ###############################  $b=array\_flip ($var\_ar);  print\_r ($b);  echo "<hr />";  ###############################  print\_r ($var\_ar);  echo "<hr />";  ?> |

**5.2.11. functia *array\_diff ()***

**Sintaxa**

**$nume\_var = array\_diff ($array1, $array2, ...);**

**Descriere si functionare**

Aceasta functie intoarce o variabila matrice, care are ca elemente, elementele (chei si valori) matricii argument **$array1**, din care se retrag elementele a caror valoare se regaseste in una din variabilele matrice argument **$array2, ...**. Atentie, se compara doar valorile, nu si cheile variabilelor.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | <?php  /\*  functia array\_diff () \*/  echo "<h3>functia array\_diff ()</h3>";  $var1=array("Popescu", "Nicolae", "Cimislia");  $var2=array("Popescu", "Vasile", "Chisinau");     echo "<h4>var1=</h4>";       print\_r ($var1);     echo "<h4>var2=</h4>";       print\_r ($var2);     echo "<h4>array diff=</h4>";       $b=array\_diff ($var1, $var2);       print\_r ($b);  echo "<hr />";  ##############################################  $var1=array("nume"        => "Popescu",              "prenume"     => "Nicolae",              "localitatea" => "Cimislia");  $var2=array("nume"        => "Popescu",              "prenume"     => "Vasile",              "an nastere" => "1959");     echo "<h4>var1=</h4>";       print\_r ($var1);     echo "<h4>var2=</h4>";       print\_r ($var2);     echo "<h4>array diff=</h4>";       $b=array\_diff ($var1, $var2);       print\_r ($b);  ?> |

**5.2.12. functia *array\_unique ()***

**Sintaxa**

**$nume\_var = array\_unique ($array);**

**Descriere si functionare**

Aceasta functie intoarce o variabila matrice, care are ca elemente, elementele matricii argument **$array**, dar pastreaza o singura valoare in cazul in care sunt mai multe elemente cu aceasi valoare.

**Nota :** Variabila intoarsa va pastra cheia primului element duplicat, al variabilei matrice argument.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <?php  /\* functia array\_unique () \*/  echo "<h3>functia array\_unique ()</h3>";  $var=array("nume"      => "Popescu",             "prenume"   => "Nicolae",             "loc\_domic" => "Cimislia",             "loc\_nast"  => "Cimislia");     echo "<h4>var=</h4>";        print\_r ($var);     echo "<h4>array unique=</h4>";        $b=array\_unique ($var);        print\_r ($b);  echo "<hr />";  ###############################  $var=array("Popescu", "Nicolae", "Chisinau", "Chisinau");     echo "<h4>var=</h4>";       print\_r ($var);     echo "<h4>array unique=</h4>";       $b=array\_unique ($var);       print\_r ($b);  echo "<hr />";  ?> |

**5.2.13. functia *array\_intersect ()***

**Sintaxa**

**$nume\_var = array\_intersect ($array1, $array2, ...);**

**Descriere si functionare**

Aceasta functie intoarce o variabila matrice, care are ca elemente, elementele matricii argument **$array1** (necesar), dar doar valorile care se regasesc si in celelalte variabile matrice argument **$array2, ...**(necesar $array2, celelalte optionale).

Variabila intoarsa, va avea cheile si valorile primei variabile matrice argument, dar doar daca valorile se regasesc si in celelalte variabile matrice. Daca mai multe chei au aceeasi valoare, prima cheie va fi pastrata.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?php  /\*  functia array\_intersect () \*/  echo "<h3>functia array\_intersect ()</h3>";  $var1=array("trotineta", "bicicleta", "motoreta", "motocicleta");  $var2=array(2=>"camion", "autoturism", "motoreta", "motocicleta");     echo "<h4>var1=</h4>";       print\_r ($var1);     echo "<h4>var2=</h4>";       print\_r ($var2);     echo "<h4>array intersect=</h4>";       $b=array\_intersect ($var1, $var2);       print\_r ($b);  echo "<hr />";  ?> |

**5.2.14. functia *array\_filter ()***

**Sintaxa**

**$nume\_var = array\_filter ($array, "nume\_functie");**

**Descriere si functionare**

Aceasta functie intoarce o variabila matrice **$nume\_var**, din elementele care intorc TRUE ale unei variabile matrice **$array** atunci cant sunt aplicate unei functii utilizator de tip comparativ **nume\_functie**, asociata functiei **array\_filter**, ca argument.

**Exemplu**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?php  /\*  functia array\_intersect () \*/  echo "<h3>functia array\_intersect ()</h3>";  $var1=array("trotineta", "bicicleta", "motoreta", "motocicleta");  $var2=array(2=>"camion", "autoturism", "motoreta", "motocicleta");     echo "<h4>var1=</h4>";       print\_r ($var1);     echo "<h4>var2=</h4>";       print\_r ($var2);     echo "<h4>array intersect=</h4>";       $b=array\_intersect ($var1, $var2);       print\_r ($b);  echo "<hr />";  ?> |

**Structura *foreach***

Această structură poate fi folosită pentru a realiza o iterație printre toate elementele unui vector. Așadar, ea nu poate fi folosită decît împreună cu vectorii; utilizarea sa asupra unei variabile de alt tip duce la apariția de erori. Există două sintaxe acceptate pentru această structură și anume:

|  |  |
| --- | --- |
| 1  2  3 | foreach (expresie\_vectoriala as $valoare){    instructiune;    } |

sau

|  |  |
| --- | --- |
| 1  2  3 | foreach (expresie\_vectoriala as $cheie => $valoare){  instructiune;   } |

Dacă se utilizează prima variantă, atunci la fiecare iterație valoarea elementului curent este atribuită variabilei *$valoare* și apoi se trece la elementul următor (a cărui valoare va fi atribuită variabilei la următoarea iterație). Execuția ciclului se încheie în momentul în care nu mai există alte elemente în vector. Singura diferența care apare în cazul utilizării celei de-a doua variante este faptul că la fiecare iterație valoarea cheii elementului curent este atribuită variabilei *$cheie*.  
În continuare este un exemplu de folosire a celor doua sintaxe ale structurii **foreach**.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?php   $sir=array("unu", "doi", "trei","patru","cinci");   foreach($sir as $valoare) {   echo "Valoare:".$valoare."<br>\n";   }  ?> |
| 1  2  3  4  5  6 | <?php  $sir=array("unu", "doi", "trei","patru","cinci");  foreach($sir as $cheie=> $valoare) {  echo "Cheie:".$cheie."Valoare:".$valoare."<br>\n";  }  ?> |

Ordonarea a 3 numere în ordine crescătoare.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php   $n["d"]=7;   $n["b"]=45;   $n["c"]=3;   asort($n);   foreach($n as $key=> $vall}   {echo"$vall";}  ?> |

**Instrucțiunea *break***

Această instrucțiune poate fi folosită pentru a întrerupe forțat execuția unui ciclu sau a secvenței de instrucțiuni corespunzătoare unei structuri ***switch***. Instrucțiunea poate fi urmată de un argument care indică numărul de structuri îmbricate a căror execuție se încheie. Valoarea implicită este 1, deci se întrerupe execuția unei singure structuri. Următoarea secvență de cod PHP realizează parcurgerea elementelor unui vector de numere întregi pînă în momentul în care se întîlnește un număr negativ.

|  |  |
| --- | --- |
| 1  2  3  4  5 | <?php  foreach ($a as $v)  if($v<0)  break;  ?> |

Mai departe aveți cazul în care este întreruptă execuția mai multor cicluri; vom considera că parcurgem elementele unei matrice pătratice cu n elemente și n coloane pînă în momentul în care întîlnim o valoare nulă.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?php  for ($i = 0; $i < $n; $i++)  for($j=0;$j<$n;$j++)  if(!$a[$i][$j])  break 2;  ?> |

Instrucțiunea ***break*** poate fi utilizată pentru întreruperea execuției secvențelor de instrucțiuni corespunzătoare structurilor ***for***, ***foreach***, ***while***, ***do - while*** și ***switch***.

**Instrucțiunea *continue***

Această instrucțiune este folosită pentru a întrerupe execuția secvenței de instrucțiuni din interiorul unui ciclu și trecerea la următoarea iterație. În cazul instrucțiunii ***for***, înainte de următoarea iterație se evaluează (execută) expresia de incrementare (expresia #3 din sintaxa generală). La fel ca și în cazul instrucțiunii ***break***, poate apărea un argument care indică numărul structurilor îmbricate asupra cărora are efect. Exemplul următor realizează afișarea elementelor unui șir de numere întregi care sunt mai mari decît 1000.

Cod

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php    foreach($a as $v)    {   if($v <=1000)   continue;   echo $v;    }    ?> |

Următorul exemplu ilustrează efectul folosirii argumentelor pentru instrucțiunea ***continue***.

Cod:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <?php    $i = 0;    while($i++ <5)    {    echo "Ciclul #1 <br>\n";    while(1)    {    echo "??Ciclul #2 <br>\n";    while(1)    {    echo "??Ciclul #3<br>\n";    continue 3;    }    echo "Acest mesaj nu va fi afisat niciodata.<br>\n";    }    echo "Nici acest mesaj nu va fi afisat niciodata.<br>\n";    }    ?> |

Rezultatul codului de mai sus este:  
Ciclul #1  
Ciclul #2  
Ciclul #3  
Ciclul #1  
Ciclul #2  
Ciclul #3  
Ciclul #1  
Ciclul #2  
Ciclul #3  
Ciclul #1  
Ciclul #2  
Ciclul #3  
Ciclul #1  
Ciclul #2  
Ciclul #3

**Exerciții:**

1. 1. Scrieți codul PHP care să sorteze crescător/decrescător un tablou de valori cu ajutorul algoritmului de sortare numit "metoda bulelor"(BubbleSort).
2. 2. Este dat un tablou cu numere întregi. De calculat:
   1. a) suma elementelor;
   2. b) elementul maxim/minim și numărul lor;
   3. c) numarul de elemente prime.
3. 3. Este dat un tablou cu numere întregi. Scrieți codul PHP care verifica dacă cel puțin 3 numere dintr-un tablou sunt consecutive. Ex. (0,2,3,4,5) => 3,4,5 - sunt 3 numere consecutive
4. 4. Se consideră un tablou bidimensional a[1..n, 1..n] format din numere întregi.Elaboraţi un script care determină o permutare a coloanelor tabloului astfel încît suma componentelor de pe diagonala principală să fi e minimă.

**Tipul șir de caractere, în PHP**

Șirurile de caractere sunt bucati de text, bine delimitate, folosite in codul-sursa pentru diferite scopuri. PHP prezinta particularitati in modul in care sunt folosite string-urile, dupa cum se poate vedea in exemplele de mai jos.

**Important!** Sirurile de caractere sunt expresii (entitati ce au si returneaza o valoare). Asadar, un string poate fi folosit, pe langa, afisare, in atribuiri, la verificari, etc. In exemplele ce urmeaza s-a optat pentru afisarea sirurilor.

**Siruri de caractere delimitate de ghilimele duble**

Sirurile delimitate prin ghilimele duble au particularitatea ca pot interpreta variabilele si caracterele speciale din interiorul lor. Astfel, la evaluarea textului, variabilele existente in interiorul lui sunt inlocuite cu valoarea lor iar rezultatul final este returnat.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $a = "valoarea variabilei"; // definesc o variabila  print "Textul definit cu ghilimele contine $a";    // va afisa  // Textul definit cu ghilimele contine valoarea variabilei  ?> |

O alta particularitate a acestor siruri o reprezinta folosirea backslash-ului (caracterul \). Acesta are o functie speciala de marcare a anumitor caractere care nu pot fi incluse in mod normal intr-un text (din cauza ca sunt ele insele caractere speciale). Din acest motiv backslash-ul poarta numele de escape character.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?php  print "Ma numesc \"Ion\"!"; // sir ce contine ghilimele, se foloseste \" pentru a afisa ghilimele    print "Text afisat pe 3 linii: \n linia 2 si \n linia3";  // atentie, textul e afisat pe 2 linii in sursa HTML cu ajutorul caracterului \n  (new-line)  // pentru verificare vizualizati sursa paginii (alegeti "View Source" din browser)    print "Am castigat \$30 :D"; // caracterul dolar are un statut aparte (marcheaza variabilele)  // pentru a afisa semnul $ asa cum este se foloseste \$    print "Text cu \\backslash\\."; // intrucat backslash \ este un caracter special el  // poate fi afisat ca atare intr-un text doar daca este dublat    // daca e folosit doar un singur \ PHP ar trata textul diferit:  print "Text fara \backslash\- \nu se afiseaza corec\t.";    ?> |

**Siruri de caractere delimitate de ghilimele simple**

Sirurile delimitate prin ghilimele simple nu permit interpretarea variabilelor continute si nici a caracterelor speciale cu exceptia \' si \\

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?php  $a = 'valoarea variabilei'; // definesc o variabila  print 'Textul definit cu ghilimele contine $a'; // Textul definit cu ghilimele contine $a    print 'Porecla ta e Kelu\' ?'; // sir delimitat de ghilimele simple ce contine un apostrof  print 'Text cu backslash \\'; // sir delimitat de ghilimele simple ce contine un backslash    print 'Alte escape chars nu sunt interpretate \n \t \b'; // textul este afisat ca atare  ?> |

**Siruri de caractere delimitate cu notatia speciala <<<**

Aceste siruri de caractere au avantajul ca nu necesita marcarea (escaparea) delimitatorilor prin \' sau \". In rest, aceste stringuri sunt tratate in acelasi mod ca cele delimitate de ghilimele duble, in sensul ca permit interpretarea variabilelor si a altor caractere speciale.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?php  $a = 'valorii lor'; // definesc o variabila    print <<<TXT  Text pe mai multe linii. <br />  Delimitatorii pot avea orice nume: TXT, START, etc, cu urmatoarele conditii: <br />  - ambii delimitatori (de inceput si sfarsit) trebuie sa aiba acelasi nume  - inainte de primul delimitator se foloseste <<<  - delimitatorul de inceput nu trebuie sa fie urmat de spatiu sau alt caracter (important!)  - delimitatorul de final sa fie la inceputul liniei (fara spatii inainte)  - dupa delimitatorul final se pune punct si virgula ;  - pot contine ghilimele " sau apostrof ' fara nevoia de a le escapa  - permit interpretarea variabilelor si afisarea $a  TXT;  ?> |

In versiunile mai noi de PHP (de la 5.3.0) a fost introdusa posibilitatea de a defini siruri prin notatia speciala <<< fara a interpreta variabilele continute. Exemplul de mai jos foloseste un sir astfel definit (vezi diferenta la delimitatorul de inceput).

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?php  $a = 'valoarea'; // definesc o variabila    print <<<'TXT'  Incepand cu versiunea 5.3.0 exista un alt mod de delimitare a sirurilor  in care variabilele si caracterele speciale nu sunt interpretate. <br />  Asadar $a si \n raman asa cum sunt.  TXT;  ?> |

**Operatii cu siruri de caractere**

Mai jos sunt prezentate operatiile uzuale cu siruri de caractere si functiile oferite de limbajul PHP pentru realizarea lor.

**Lungimea sirului**

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5 | $s = "acesta este un text";  $sir = "stiu PHP stiu HTML stiu CSS";    # ce lungime are sirul? (numarul de caractere)  print strlen( $s ); // 19 |

**Cautarea unei secvente**

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # verific daca un cuvant sau text (in cazul de fata cuvantul 'PHP') apare in  # sirul exprimat prin variabila $sir  if( strstr( $sir, 'PHP' ) !== false ) print 'gasit';  else print "nu am gasit";    # pentru a nu tine cont de litere mari/mici se foloseste stristr  if( stristr( $sir, 'phP' ) !== false ) print 'gasit'; |

**Afisarea unui subsir**

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # afisez o sectiune din sir  print substr( $sir, 0, 4); // stiu  print substr( $sir, 5 ); // PHP stiu HTML stiu CSS  print substr( $sir, 5, -3 ); // PHP stiu HTML stiu  print substr( $sir, -3 ); // CSS    # returnez doar un caracter din string  print $sir{5}; // P  print $sir{ strlen($sir)-1 }; // S |

**Transformarea sirului**

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # inlocuirea unor secvente  print str\_replace( "stiu", "invat", $sir); // invat PHP invat HTML invat CSS    # schimb tipul literelor (mari, mici)  print strtoupper( $s ); // ACESTA ESTE UN TEXT  print strtolower( $sir ); // stiu php stiu html stiu css  print ucfirst( $s ); // Acesta este un text  print ucwords( $s ); // Acesta Este Un Text    # sterg spatiile de la inceput si sfarsit: trim, ltrim, rtrim  print trim('      ok        '); // ok    # caractere "enter" transformate in <br />  print nl2br( "acesta e afisat pe \n 2 linii" ); // acesta e afisat pe <br /> 2 linii |

**Impartirea (spargerea) sirului**

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | # impart sirul dupa un caracter, cuvant sau un alt sir  $output1 = explode( "stiu ", $sir ); // impart dupa stiu<spatiu>  /\*  Array (      [0] => PHP      [1] => HTML      [2] => CSS  )  \*/    # impart sirul dupa o expresie regulata (regex)  $output2 = preg\_split( '/ /', $s ); // impart dupa spatiu  /\*  Array (      [0] => acesta      [1] => este      [2] => un      [3] => text  )  \*/    # operatia inversa impartirii unui sir:  $a = implode( 'invat ', $output1 ); // invat PHP invat HTML invat CSS  $b = join( '-', $output2 ); // acesta-este-un-text |

**Notă:** implode si join sunt echivalente (nu exista nici o diferenta intre ele), pe cand explode si preg\_split sunt diferite.

**Concatenarea (legarea) sirurilor**

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4 | print 'Text 1' . " legat de " . 'text 2' . "\n"; // Text 1 legat de text 2    // se pot concatena siruri rezultate din alte functii sau din variabile  print ucfirst($sir) . '!!! ' . $s; |

**Nota:** prin concatenarea sirurilor se obtine in final un singur sir, care este tratat ca atare, de sine statator. Altfel spus, prin legarea mai multor siruri se obtine o singura entitate (o singura expresie). Aceasta poate fi transmisa ca parametru unor functii sau instructiuni precum print care accepta un singur argument.

**Interpretarea sirului**

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # parsez un Query String  $str = "first=value&arr[]=foo+bar&arr[]=baz";  parse\_str($str);  print $first;  // value  print $arr[0]; // foo bar  print $arr[1]; // baz    parse\_str($str, $output);  print $output['first'];  // value  print $output['arr'][0]; // foo bar  print $output['arr'][1]; // baz |

**Masuri de siguranta**

In cazul in care textul provine din surse nesigure (cum ar fi un formular de comentarii), atunci este indicat sa fie "sterilizat" (sanitized), prin eliminarea elementelor ce pot fi daunatoare (tag-uri HTML, caractere speciale, etc).

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | print addslashes( "Baiatu' ia vino-ncoa'!" ); # Baiatu\' ia vino-ncoa\'!  # functia inversa este stripslashes();    print htmlspecialchars("<a href='test'>Test</a>", ENT\_QUOTES);  # afiseaza <a href='test'>Test</a>  # functia inversa este htmlspecialchars\_decode()    print strip\_tags( "<p>E <b>bold</b></p>" ); // E bold  print strip\_tags( "<p>E <b>bold</b></p>", '<b>' ); // E <b>bold</b> |

**Exerciții:**

1. 1. Dintr-o listă ce conţine **n** candidaţi trebuie alese **m** persoane care vor fi incluse în echipa de fotbal a unui raion. Elaboraţi un script care afișează la ecran toate modalităţile de selecţie a celor **m** persoane.
2. 2. Se dă șirul **$sir = "Ion Vasile Petru Vitalie Dumitru";**. De transformat șirul dat într-o matrice folosind funcția **explode()**.

**Funcții în PHP**

În limbajul PHP, funcțiile pot fi definite de către utilizator folosind următoarea sintaxă:

|  |  |
| --- | --- |
| 1  2  3 | function numef ($param1, $param2, ...,  $paramN) {    // instructiuni    } |

În PHP o funcție poate fi definită oriunde în cadrul script-ului și în interiorul unei funcții poate să apară orice secvență validă de cod care include definirea de alte funcții și definiții de clase. Argumentele unei funcții trebuie separate prin virgulă, și, implicit, acestea sunt transmise prin valoare. Pentru ca funcția să returneze un rezultat se folosește construcția **return** care primește ca parametru o expresie care reprezintă valoarea funcției. În momentul în care este întîlnită construcția **return**, execuția funcției se încheie. În exemplul următor se calculează cu ajutorul unei funcții PHP, pătratul unui număr.

Cod:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  function patrat($n)  {  return $n\*$n;  }  echo "4^2=<b>".patrat(4)."</b>";  ?> |

Rezultatul codului va fi:  
4^2=16

**Transmiterea parametrilor prin referință**

Pentru a transmite parametri unei funcții prin referință, fapt care implică modificarea valorii parametrilor și păstrarea noii valori după ce execuția funcției s-a încheiat, se folosește operatorul '*&*' înaintea numelui parametrului formal, în momentul definirii funcției. Următorul exemplu indică modul în care se modifică valoarea unei variabile în interiorul unie funcții și modul în care această modificare este resimțită în exteriorul acesteia:

Cod:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?php  function modificInt ($s)  {  $s.="prima functie.";  echo "<b>In modificInt:</b>".$s."<br>";  }  function modificExt(&$s)  {  $s.="a doua functie.";  echo "<b>In modificExt: </b>".$s."<br>";  }  <br />  $s="Iesire din ";  <br />  echo "<b>In script: </b>".$s."<br>";  modificInt ($s)  echo "<b>In script: </b>".$s."<br>";  modificExt ($s);  echo "<b>In script:</b>".$s."<br>";  ?> |

Rezultat:

**In script**: Iesire din  
**In modificInt:** Iesire din prima functie.  
**In script:** Iesire din  
**In modificExt:** Iesire din a doua functie.  
**In script:** Iesire din a doua functie.

Nu există posibilitatea de supraîncărcare a unei funcții, de redefinire a ei după ce aceasta a fost definită în cadrul scriptului respectiv și nu există nici posibilitatea de anulare a unei funcții.

**Parametri cu valori implicite**

În PHP parametrii formali pot avea valori implicite, și, în cazul în care parametrul actual lipsește, atunci se va considera că are valoarea implicită. Următorul exemplu ilustrează modul de folosire al funcțiilor cînd acestea au parametri formali cu valori implicite:

Cod:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?php  function comanda($s='cafea')  {  return"Ati comandat ;".$s.".";  }  echo comanda();  echo "<br>";  echo comanda("suc");  ?> |

Rezultat:

Ati comandat cafea.  
Ati comandat suc.

În cazul în care se folosesc parametri cu valori implicite este necesar ca orice parametru care are o valoare implicită să se afle în partea dreaptă a tuturor parametrilor pentru care nu se folosesc valori implicite, în caz contrar interpretorul PHP nu poate să decidă cărui parametru să-i atribuie valoarea de pe o anumită poziție din lista de parametri. De exemplu, dacă avem o funcție a cărei declarație este

|  |  |
| --- | --- |
| 1 | function transform ($baza=10, $nr) |

si care returnează rezultatul transformării lui *$nr* din baza 16 în baza *$baza*, a cărei valoare implicită este 10, dacă se apeleaza *transform (50)*, interpretorul nu atribuie valoarea 50 parametrului *$nr*, ci parametrului *$baza* și generează o eroare deoarece lipseste valoarea parametrului *$nr*.

**Funcții cu număr variabil de parametri**

O altă facilitate a limbajului PHP este aceea că oferă programatorului posibilitatea de a utiliza funcții care au un număr nedeterminat de parametri. Funcțiile care folosesc un număr variabil de parametri nu au nici o particularitate în ceea ce privește definirea lor. Aceste funcții se definesc la fel ca cele prezentate anterior, dar pentru a putea accesa parametri se vor folosi următoarele funcții predefinite:

- ***func\_num\_args( )*** - această funcție returnează numărul parametrilor funcției care a apelat-o; dacă această funcție este apelată din exteriorul unei funcții definite de utilizator se va genera un mesaj de avertizare;  
- ***func\_get\_arg(arg\_num)*** - returnează valoarea parametrului care se află pe pozitia ***arg\_num*** în lista de parametri; primul parametru are numărul de ordine 0; dacă este apelată din exteriorul unei funcții definite de utilizator se va genera un mesaj de avertizare;  
- ***func\_get\_args( )*** - returnează un tablou unidimensional care conține valorile parametrilor pe care funcția apelantă i-a primit; dacă această funcție este apelată din exteriorul unei funcții definite de utilizator se va genera un mesaj de avertizare.

În continuare aveți 2 exemple de utilizare a acestor funcții. Primul exemplu afișează lista parametrilor funcției folosind funcția ***func\_num\_args***și ***func\_get\_arg***, iar al doilea exemplu afișează aceeași listă folosind numai funcția ***func\_get\_args***.

Cod:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?php  function lista\_parametri()  {  for ($i=0;$i<func\_num\_args();$i++)  {  print\_r(func\_get\_arg($i));  echo "<br>";  }  }  echo lista\_parametri("Comanda:",1,"calculator",2,"procesoare","configuratie",array("local",2,3));  ?> |

Cod:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?php  function lista\_parametrii()  {  foreach(func\_get\_args()as $i)  {  print\_r($i);  echo "<br>";  }  }  echo lista\_parametrii ("Comanda:",1,"calculator",2,"procesoare","configuratie",array("local",2,3));  ?> |

Rezultat:

Comanda:  
1  
calculator  
2  
procesoare  
configuratie  
Array ( [0] => local [1] => 2 [2] => 3 )

**Valorile returnate de funcții**

Rezultatul obținut după apelarea unei funcții poate avea orice tip. O funcție poate să returneze chiar și liste sau obiecte. În PHP există un caz special de rezultat numit referință. Pentru ca o funcție să poată returna o referință, aceasta trebuie declarată folosindu-se operatorul '*&*' înaintea numelui funcției. Acest operator trebuie să apară înaintea numelui funcției și în momentul cînd o variabilă primește ca valoare referința rezultată din apelul funcției. În exemplul următor se definește o funcție al cărui rezultat îl constituie o referință:

Cod:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php  function&refer()  {  global $s;  return $s;  }  $s="Acesta este continutul variabilei referite cu ajutorul functiei.";  $z=&refer();  echo $z;  ?> |

Rezultat:  
Acesta este continutul variabilei referite cu ajutorul functiei.

Spre deosebire de majoritatea limbajelor de programare moderne, o funcție PHP poate să returneze o referință la o variabilă care a fost declarată în interiorul funcției, însă acest lucru nu este indicat deoarece, în anumite cazuri, poate duce la efecte neașteptate ale executării unui script PHP. În alte limbaje de programare efectele devin uneori fatale.

**Variabilele de tip funcție**

O altă facilitate a limbajului PHP în ceea ce privește funcțiile este aceea că suportă variabile de tip funcție. Acest lucru este util atunci cînd se folosesc liste de funcții pentru prelucrarea anumitor tipuri de date. Pentru a atribui un nume de funcție unei variabile în PHP se folosește aceeași construcție ca în cazul atribuirii unui șir de caractere, și anume, o variabilă va primi ca valoare numele funcției scris între ghilimele simple sau duble. În cazul în care interpretorul PHP găsește un nume de variabilă urmată de o listă de parametri, acesta caută funcția pe care variabila o referă și în cazul în care există, o execută. Variabilele de tip funcție nu funcționeaza cu construcții ale limbajului ca ***echo***, ***unset***, ***isset***, ***empty***, ***include*** etc. Mai jos aveți un exemplu care ilustrează modul de lucru cu variabilele de tip funcție.

Cod:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <?php  function produs($a,$b)  {  return $a\*$b;  }  function suma($a,$b)  {  return $a+$b;  }  $operatie='produs';  $rez=$operatie(4,5);  echo "4\*5=<b>".$rez."</b><br>";  $operatie='suma';  $rez=$operatie(4,5);  echo"4+5=;<b>".$rez."</b><br>";  ?> |

Rezultat:  
4 \* 5 =20  
4 + 5 =9

**Exerciții:**

1. De elaborat 3 funcții care va calcula numărul permutărilor Pn, numărul aranjamentelor Amn și numărul combinărilor Cmn.
2. De elaborat funcția care calculează **n!**. Scriptul PHP va apela funcția dată pentru a calcula suma S=1!+3!+...+(2\*n-1)! .
3. Se dă vectorul a[1..n] de numere întregi. De elaborat o funcție care calculează media componentelor unui vector.
4. De elaborat o funcție care calculează maximul dintre două numere x și y. De elaborat un program care afișează maximul dintre numerele a, b, c, d, e.
5. De elaborat o funcție care convertește literele minuscule în majuscule și viceversa.

**Funcțiile predefinite în PHP**

**Funcțiile predefinite** sunt funcțiile care aparțin **limbajului PHP**.  
Aici sunt prezentate, cîteva din cele mai utilizate funcții PHP ce trebuie cunoscute de către cei care sunt pasionați de acest limbaj al programarii web sau lucrează în acest domeniu.

**Funcții pentru Array**

Aici gasiti cateva din cele mai utile functii PHP pentru Array (matrice).  
1.**array\_change\_key\_case(array, case)**  
- Transforma cheile din primul parametru (array) in majuscule sau litere mici, dupa cum e specificat la "case" (**CASE\_UPPER** sau **CASE\_LOWER**). Default e CASE\_LOWER. Daca "array" nu e o matrice, returneaza FALSE.  
2.**array\_chunk(array, nr)**  
- Imparte "array" intr-un numar de sub-matrici, precizat la "nr", indexate de la 0. Daca "array" nu are elemente, returneaza NULL.  
3.**array\_combine(array\_k, array\_v)**  
- Creaza un array folosind valorile din "array\_k" pentru chei, iar elementele din "array\_v" pentru valorile lor. Numarul de elementele din cele 2 matrici trebuie sa fie egal, in caz contrar returneaza FALSE.  
4.**array\_count\_values(array)**  
- Returneaza o matrice cu numarul de repetari a fiecarui element din "array".  
5.**array\_diff(array1, array2)**  
- Returneaza o matrice cu elementele din "array1" care nu se gasesc in "array2".  
6.**array\_flip(array)**  
- Schimba valorile si cheile intre ele. Din valorile care se repeta o pastreaza pe ultima.  
7.**array\_intersect(array1, array2, ...)**  
- Returneaza o matrice cu toate valorile din "array1" care se gasesc in toati ceilalti "array" dati ca parametri.  
8.**array\_key\_exists(cheie, array)**  
- Returneaza TRUE daca "cheie" este o cheie (index) in matricea "array", in caz contrar returneaza FALSE.  
9.**array\_keys(array, val)**  
- Returneaza o matrice cu toate cheile din "array". Parametrul "val" e optional, daca e specificat, va returna doar cheile cu valoarea "val".  
10.**array\_map("functie", array)**  
- Aplica functia pentru fiecare element din "array"  
11.**array\_merge($array1, $array2, ...)**  
- Uneste mai multe array in unul singur. Daca au cheie de tip sir identice, va fi pastrata valoarea ultimei chei.  
12.**array\_pop(array)**  
- Elimina ultimul element din "array", care va fi transferat ca valoarea returnata de "array\_pop()". Daca parametru nu e matrice sau e gol, returneaza NULL.  
13.**array\_product(array)**  
- Returneaza produsul elementelor din "array"  
14**array\_rand(aray, nr)**  
- Preia aleator un "nr" de chei din "array", respectand ordinea lor. Daca "nr" e 1, va returna o cheie aleasa aleator. Altfel, returneaza o matrice cu acele chei preluate aleator.  
15.**array\_reverse(array)**  
- Returneaza "array" cu ordinea elementelor inversata.  
16.**array\_search("val", array)**  
- Returneaza cheia din "array" care are valoarea "val", sau FALSE daca nu exista. Cautarea e "case-sensitive"  
17.**array\_shift(array)**  
- Elimina primul element din "array", care va fi transferat ca valoarea returnata de "array\_shift()". Cheile vor fi renumerotate de la 0. Daca parametru nu e matrice sau e gol, returneaza NULL.  
18.**array\_slice(array, start, nr)**  
- Returneaza un "nr" din elementele lui "array', incepand de la elementul numarul "start" (*primul e 0*), cate elemente arata "nr". Daca "nr" nu e specificat, va fi considerat pana la ultimul.  
19.**array\_sum(array)**  
- Returneaza suma elementelor din "array"  
20.**array\_unique(array)**  
- Returneaza matricea "array" fara valorile duplicate, acestea fiind luate o singura data.  
21.**array\_values(array)**  
- Returneaza o matrice cu toate valorile din "array", ordonate numeric de la 0.  
22.**arsort(array)**  
- Sorteaza un "array" cu valorile ordonate in ordine inversa alfabetic, pastrand corelatia dintre chei si valori. In caz de succes returneaza TRUE, altfel, FALSE.  
23.**asort(array)**  
- Sorteaza un "array" cu valorile ordonate in ordine alfabetica, pastrand corelatia dintre chei si valori. In caz de succes returneaza TRUE, altfel, FALSE.  
24.**count(array)**  
- Returneaza numarul de elemente din "array"  
25.**current(array);**  
- Returneaza valoarea elementului curent din "array", daca nu are elemente returneaza FALSE. Fiecare matrice are un pointer intern, care prima data e setat la primul element.  
26.**end(array);**  
- Muta pointerul din "array" la ultimul element si returneaza valoarea lui, daca nu are elemente returneaza FALSE.  
27.**implode("separator", array);**  
- Transforma "array" in sir (string), adaugand valorile intr-un sir, separate de ceea ce e specificat la "separator".  
28.**in\_array("val", array)**  
- Verifica daca o matrice contine o anumita valoare. Returneaza TRUE daca vreunul din elementele din "array" are valoarea "val", in caz contrar FALSE.  
29.**key(array)**  
- Returneaza cheia elementului curent la care se afla pointerul  
30.**krsort(array)**  
- Sorteaza un "array" dupa chei ordonate in ordine inversa alfabetic, pastrand corelatia dintre chei si valori. In caz de succes returneaza TRUE, altfel, FALSE.  
31.**ksort(array)**  
- Sorteaza un "array" dupa chei ordonate in ordine alfabetica, pastrand corelatia dintre chei si valori. In caz de succes returneaza TRUE, altfel, FALSE.  
32.**natcasesort(array)**  
- Sorteaza un "array" in ordine alfabetica a valorilor, case-insensitive, folosind ordinea naturala a numerelor. In caz de succes returneaza TRUE, altfel, FALSE.  
33.**next(array);**  
- Avanseaza pointerul din "array" inainte cu un element si returneaza valoarea, daca nu are elemente returneaza FALSE.  
34.**prev(array);**  
- Muta pointerul din "array" inapoi cu un element si returneaza valoarea, daca nu are elemente returneaza FALSE.  
35.**reset(array);**  
- Readuce pointerul din "array" la primul element si returneaza valoarea lui, daca nu are elemente returneaza FALSE.  
36.**rsort(array)**  
- Sorteaza un "array" in ordine inversa alfabetic a valorilor (literele mici inaintea celor mari), stergand cheile initiale, devenind un Array asociativ cu chei de la 0. In caz de succes returneaza TRUE, altfel, FALSE.  
37.**sort(array)**  
- Sorteaza un "array" in ordine alfabetica a valorilor (literele mari inaintea celor mici), stergand cheile initiale, devenind un Array asociativ cu chei de la 0. In caz de succes returneaza TRUE, altfel, FALSE.  
38.**shuffle(array)**  
- Amesteca aleator elementele din "array". Sterge cheile initiale, devenind un Array asociativ cu chei de la 0. In caz de succes returneaza TRUE, altfel, FALSE.

**Functii matematice**

Aici gasiti functii PHP matematice, in special pentru lucrul cu numere.  
1.**abs(nr)**  
- Returneaza valoarea absoluta (*cu +*) a lui "nr".  
2.**base\_convert(nr, baza1, baza2)**  
- Converteste un numar, "nr", din baza1 in baza2. Rezultatul returnat este un sir cu "nr" convertit in baza2.  
3.**bindec('sir\_binar')**  
- Returneaza echivalentul numeric (decimal) al sirului binar din parametrul "sir\_binar".  
4.**ceil(nr)**  
- Returneaza valoarea rotunjita a lui "nr" la urmatorul (mai mare) intreg.  
5.**cos(radian)**  
- Returneaza valoarea cosinus a gradelor de cerc specificate la "radian".  
6.**decbin(nr)**  
- Returneaza un sir cu valoarea binara a numarului "nr".  
7.**dechex(nr)**  
- Returneaza un sir cu valoarea hexazecimala a numarului "nr".  
8.**decoct(nr)**  
- Returneaza un sir cu valoarea octala a numarului "nr".  
9.**floor(nr)**  
- Returneaza valoarea rotunjita a lui "nr" la anteriorul (mai mic) intreg.  
10.**fmod(x, y)**  
- Returneaza valoarea ramasa a lui "x" impartit la "y", care nu mai e multiplu de "y".  
11.**hexdec('hex')**  
- Returneaza valoarea zecimala a sirului hexazecimal "hex".  
12.**hypot(c1, c2)**  
- Returneaza lungimea ipotenozei unui triunghi dreptunghic care are lungimea catetelor "c1" si "c2".  
13.**is\_infinite(val)**  
- Returneaza TRUE daca valoarea lui "val" este un numar infinit (*precum log(0)*), altfel, returneaza FALSE.  
14.**lcg\_value()**  
- Returneaza un numar aleator intre 0 si 1.  
15.**log10(nr)**  
- Returneaza logaritmul in baza 10 a lui "nr"..  
16.**log(nr, baza)**  
- Returneaza logaritm de "nr" in baza "baza". Daca al doilea parametru nu e specificat, va returna logaritmul natural (in baza e).  
17.**max(nr1, nr2, ...)**  
- Returneaza cel mai mare numar dintre cele date la parametri.  
18.**min(nr1, nr2, ...)**  
- Returneaza cel mai mic numar dintre cele date la parametri.  
19.**mt\_rand(min, max)**  
- Returneaza un numar aleator cuprins intre numerele "min" si "max". Similara este si functia **rand()**, dar "mt\_rand()" este de 4 ori mai rapida.  
20.**octdec('oct')**  
- Returneaza valoarea zecimala a sirului octal "oct".  
21.**pi()**  
- Returneaza valoarea aproximativa a lui PI. La fel ca si constanta **M\_PI**.  
22.**pow(nr, putere)**  
- Returneaza valoarea obtinuta din "nr" la puterea "putere".  
23.**round(nr)**  
- Returneaza valoarea rotunjita a lui numar la intregul ce mai apropiat.  
24.**sin(radian)**  
- Returneaza valoarea sinus a gradelor de cerc specificate la "radian".  
25.**sqrt(nr)**  
- Returneaza radical din "nr".  
26.**tan(radian)**  
- Returneaza valoarea tangenta a gradelor de cerc specificate la "radian".  
27.**is\_numeric(val)**  
- Returneaza TRUE daca "val" e o valoare numerica, altfel, FALSE.

**Funcții pentru lucru cu functii**

Aici gasiti functii PHP ce pot fi folosite in lucrul cu functii create /definite in script.  
1.**call\_user\_func("func", arg)**  
- Apeleaza functia "func" dandu-i valoarea argumentului "arg". Returneaza rezultatul functiei apelate, sau FALSE.  
2.**call\_user\_func\_array("func", array\_arg)**  
- Apeleaza o functie ("func") dandu-i valoarea argumentelor dintr-un Array (array\_arg). Returneaza rezultatul functiei apelate, sau FALSE.  
3.**create\_function("argumente", "cod\_functie")**  
- Creaza dinamic o functie ce va avea argumentele "argumente" si codul ce-l executa "cod\_functie". Returneaza functia creata (*poate fi asociata cu un nume de variabila*), sau FALSE. (*"argumente" si "cod\_functie" trebuie sa fie siruri*)  
4.**func\_num\_args()**  
- Returneaza numarul de argumente primite de o functie. Se foloseste in interiorul acelei functii.  
5.**func\_get\_arg(nr)**  
- Returneaza argumentul cu index-ul "nr" din cele primite de o functie (*primul argument are index 0*). Se foloseste in interiorul acelei functii. Daca numarul de argumente e mai mic decat "nr", returneaza FALSE.  
6.**func\_get\_args()**  
- Returneaza o matrice cu argumentele primite de o functie. Se foloseste in interiorul acelei functii..  
7.**function\_exists("function")**  
- Returneaza TRUE daca functia "function" (*numele ei intre ghilimele*) exista, altfel, returneaza FALSE.  
8.**register\_shutdown\_function("func", param1, param2, ...)**  
- Inregisteaza functia "func" sa fie executata la terminarea scriptului (dupa executarea comenzilor din el), sau dupa apelarea lui "exit()". "param1", "param2" sunt optionali si reprezinta parametri ce vor fi transferati la "func".

**Functii pentru siruri (strings)**

Aici gasiti cateva din cele mai utile functii PHP pentru siruri (strings).  
1.**addslashes("sir")**  
- Adauga caracterul \ (backslash) inaintea ghilimelelor simple si duble, a lui backslash si NULL (NULL byte). Ca pretectie, de exemplu cand sirul e adaugat intr-o baza de date.  
2.**bin2hex('sir')**  
- Returneaza forma hexadecimala a lui 'sir'  
3.**chr(ascii)**  
- Returneaza caracterul reprezentat de codul ASCII din parametru.  
4.**chunk\_split("sir", nr, 'sep')**  
- Imparte "sir" in mai multe sub-siruri, avand lungimea de caractere "nr" si separate prin "sep".  
5.**echo 'sir'**  
- Afiseaza in browser continutul 'sir'.  
6.**explode('sep', "sir")**  
- Creaza un Array (matrice) cu partile din "sir" separate prin "sep". Fiecare sub-sir delimitat de "sep" devine un element in matrice. *Este inversul lui "implode()"*.  
7.**html\_entity\_decode("sir")**  
- Decodeaza entitatile html din "sir". E opusul lui "htmlentities()"  
8.**htmlentities("sir")**  
- Converteste toate caracterele specifice din sir in entitati HTML.  
9.**htmlspecialchars\_decode("sir")**  
- Converteste entitatile HTML, ce pot fi codificate cu "htmlspecialchars()", inapoi la forma de caracter.  
10.**htmlspecialchars("sir")**  
- Converteste doar caracterele ghilimele, & < si > din "sir" in entitati HTML.  
11.**md5("sir")**  
- Codifica sirul cu algoritmul MD5, returnand un sir codat cu 32 de caractere.  
12.**rtrim("sir", 'caracters')**  
- Sterge din partea dreapta a sirului caracterele specificate la "caracters", care poate fi un singur caracter sau un sir. Daca al doilea parametru nu e specificat, va sterge spatiile libere din dreappta.  
13.**sha1("sir")**  
- Similar cu "md5()", aceasta functie codifica sirul cu algoritmul SHA1 (*US Secure Hash Algorithm 1*), returnand un sir codat cu 40 de caractere.  
14.**str\_ireplace("old", "new", "sir")**  
- Inlocuieste in "sir" caracterele specificate la "old" cu cele de la "new", fara a tine cont de majuscule sau litere mici. "old" si "new" pot fi si de tip Array.  
14.**str\_replace("old", "new", "sir")**  
- Similar cu "str\_ireplace()", inlocuieste in "sir" caracterele specificate la "old" cu cele de la "new", dar este case-sensitive (tine cont de majuscule si litere mici). "old" si "new" pot fi si de tip Array.  
15.**str\_repeat("sir", nr)**  
- Returneaza un sir in cara parametrul "sir" e repetat de "nr" ori.  
16.**str\_shuffle("sir")**  
- Amesteca aleator caracterele din "sir".  
17.strong>str\_split("sir", nr)  
- Transforma "sir" in Array, impartind pentru fiecare element sub-siruri cu lungimea de "nr" caractere. Daca "nr" nu e specifcat, fiecare caracter din sir va deveni un element in matrice.  
18.**str\_word\_count("sir")**  
- Returneaza numarul de cuvinte din sir.  
19.**strcasecmp("sir1", "sir2")**  
- Compara cele doua siruri date ca parametri, fara diferenta intre majuscule si litere mici. Returneaza 0 daca sirurile sunt egale, < 0 daca "sir1" e mai mic decat "sir2" sau > 0 daca "sir1" e mai mare decat "sir2"  
20.**strcmp("sir1", "sir2")**  
- Similar cu "strcasecmp()", diferenta fiind ca face o comparatie ce tine cont de majuscule si litere mici.  
21.**strcspn("sir", 'car')**  
- Returneaza lungimea primului segment din "sir" ce nu contine nici un caracter specificat la 'car'  
22.**strip\_tags("sir")**  
- Sterge tag-urile HTML si PHP din "sir".  
23.**stripslashes("sir")**  
- Returneaza "sir" fara caracterele backslashes adaugate. Inversul lui "addslashes()"  
24.**stripos("sir", "sub-sir")**  
- Returneaza pozitia de unde incepe "sub-sir" in "sir" (*incepand de la 0*), fara diferenta intre litere mici si majuscule. Daca "sub-sir" nu e gasit, returneaza FALSE.  
25.**strpos("sir", "sub-sir")**  
- Returneaza pozitia de unde incepe "sub-sir" in "sir" (*incepand de la 0*), case-sensitive, tinand cont de litere mici si majuscule. Daca "sub-sir" nu e gasit, returneaza FALSE.  
26.**stristr("sir", 'car')**  
- Returneaza tot sub-sirul din "sir", care incepe de la prima aparitie a lui 'car', fara diferenta intre majuscule si litere mici. Daca nu e gasit 'car', returneaza FALSE.  
27.**strstr("sir", 'car')**  
- Returneaza tot sub-sirul din "sir", care incepe de la prima aparitie a lui 'car', case-sensitive (cu deosebire intre majuscule si litere mici). Daca nu e gasit 'car', returneaza FALSE.  
28.**strlen("sir")**  
- Returneaza lungimea lui "sir" (numarul de caractere).  
29.**strrchr("sir", "sub-sir")**  
- Returneaza bucata ramasa din "sir" incepand de la ultimul caracter din "sub-sir" gasit in "sir". Functia e case-sensitive. Daca "sub-sir" nu se gaseste in "sir", returneaza FALSE.  
30.**strrev("sir")**  
- Inverseaza sir-ul.  
31.**strtolower("sir")**  
- Transforma toate literele din sir in litere mici.  
32.**strtoupper("sir")**  
- Transforma toate literele din sir in majuscule (litere mici).  
33.**substr\_count("sir", "sub\_sir")**  
- Returneaza numarul de cate ori "sub\_sir" se gaseste in "sir". Functia este case-sensitive.  
34.**substr("sir", start, lungime)**  
- Returneaza portiunea din "sir" care incepe de la "start", cu lungimea de caractere "lungime". Daca parametrul "lungime" nu e specificat, se considera pana la sfarsitul sirului.  
35.**trim("sir", 'del')**  
- Sterge din marginile sirului caracter-ul (caracterele) specificate la "del". Daca al doilea parametru, "del", nu e adaugat, sterge spatiile libere din marginile sirului.  
36.**ucfirst("sir")**  
- Transforma primul caracter din "sir" in litera mare.  
37.**ucwords("sir")**  
- Transforma primul caracter al fiecarui cuvant din "sir" in litera mare.  
38.**nl2br("sir")**  
- Inlocuieste caracterele de rand nou (*\r\n si \n*) cu tagul HTML BR. *Functie utila cand sirul e preluat dintr-un textarea.*  
39.**wordwrap("sir", lungime, "sep", true)**  
- Imparte "sir" la numarul de caractere specificat de "lungime", separand prin "sep" partile astfel obtinute. Functie utila cand se doreste scurtarea liniilor sau cuvintelor prea lungi. Daca "true" nu e specificat, va imparti doar randurile, lasand cuvintele neafectate.

**Funcții pentru Data si Timp**

Aici sunt cateva din cele mai utile **functii PHP** pentru lucrul cu **Data** si formate de **Timp**.  
1.**checkdate(nr\_luna, nr\_zi, nr\_an)**  
- Verifică validitatea datei formate din argumenții din paranteza. O dată este considerată validă dacă fiecare parametru este definit corespunzător. "nr\_luna" este numarul lunii, intre 1 și 12 inclusiv, "nr\_zi" este numarul zilei cuprinse în domeniul zilelor permise pentru luna respectivă. Anii bisecți sunt luați în considerare, "nr\_an" este anul intre 1 și 32767 inclusiv. Returneaza TRUE sau FALSE.  
2.**date\_create('sir\_data')**  
- Stil obiect-orientat: **DateTime::\_\_construct('sir\_data')**  
Creaza o instanta de timp dupa data din "sir\_data", ce poate fi folosit in alte functii pt. data si timp care necesita ca parametru o instanta de timp. In caz de esec returneaza FALSE.  
3.**date\_format(obj\_date, 'data\_form')**  
- Stil obiect-orientat: **DateTime::format('data\_form')**  
Returneaza data comform formatului din "data\_form". "obj\_date" e o instanta de timp returnata de functia "date\_create()". In caz de esec returneaza FALSE.  
4.**date\_create\_from\_format('data\_form', 'sir\_data')**  
- Stil obiect-orientat: **DateTime::createFromFormat(data\_form, sir\_data)**  
Creaza o instanta de timp a datei din "sir\_data" care e recunoscuta si trebuie sa corespunda cu formatul de data din "data\_form". Instanta obtinuta poate fi folosita pentru a afisa data din "sir\_data" cu un alt format, cu functia "date\_format()". In caz de esec returneaza FALSE.  
5.**date\_date\_set(obj\_date\_create, nr\_an, nr\_luna, nr\_zi)**  
- Stil obiect-orientat: **DateTime::setDate(nr\_an, nr\_luna, nr\_zi)**  
Seteaza o instanta de timp, dupa parametri primiti, care apoi poate fi afisata intr-un format de data, cu functia "date\_format()". In caz de esec returneaza FALSE.  
6.**date\_default\_timezone\_set('zona\_timp')**  
- Seteaza timpul folosit in script dupa zona specificata in "zona\_timp", identificator din [Lista zonelor orare susținute](http://www.php.net/manual/ro/timezones.php). Returneaza TRUE sau FALSE.  
7.   **date\_diff(obj\_date1, obj\_date2)**  
- Stil obiect-orientat: **DateTime::diff(obj\_date2)**  
Returneaza un obiect cu diferenta dintre doua instante de timp sau FALSE.  
8.strong>date\_modify(obj\_date, 'sir\_modificare'  
- Stil obiect-orientat: **DateTime::modify(sir\_modificare)**  
Modifica instanta de timp din "obj\_date" cu valoarea specificata in "sir\_modificare", sau FALSE in caz de esec.  
9.**date\_parse\_from\_format('data\_form', 'sir\_data')**  
- Returneaza un Array asociativ cu informatii detaliate despre data din "sir\_data", aceasta trebuind sa aibe formatul din "data\_form".  
10.**date\_parse('sir\_data')**  
- Returneaza un Array asociativ cu informatii detaliate despre data din "sir\_data" (care sa fie intr-un format acceptat de "strtotime()").  
11.**date\_sun\_info(timp\_UNIX, Latitudine, Longitudine)**  
- Returneaza un Array asociativ cu informatii despre rasaritul si apusul soarelui, sfarsitul si inceputul crepusculului din ziua ce se gaseste in timpul UNIX din parametrul "timp\_UNIX", din zona data de parametri "Latitudine" si "Longitudine", in grade.  
12.**date\_time\_set(obj\_date, nr\_ora, nr\_minut, nr\_secunde)**  
- Stil obiect-orientat: **DateTime::setTime(nr\_ora, nr\_minut, nr\_secunde)**  
Reseteaza instanta de timp (din "obj\_date"), definindu-i si ora, minutul, si optional secndele, specificate in ceilalti parametri. In caz de esec, FALSE..  
13.**date\_timestamp\_get(obj\_date)**  
- Stil obiect-orientat: **DateTime::getTimestamp()**  
Obtine timpul UNIX din instanta de timp "obj\_date".  
14.**date\_timestamp\_set(obj\_date, timp\_UNIX)**  
- Stil obiect-orientat: **DateTime::setTimestamp(timp\_UNIX)**  
Modifica instanta de timp "obj\_date" cu timpul UNIX din "timp\_UNIX".  
15.**date('data\_form', timp\_UNIX)**  
- Returneaza un sir cu data, avand formatul din "data\_form", al timpului UNIX din parametru "timp\_UNIX". Daca "timp\_UNIX" nu e mentionat, va returna data curenta.  
16.**getdate(timp\_UNIX)**  
- Returneaza un Array asociativ conținînd informația despre dată și oră a parametrului "timp\_UNIX", sau a datei/orei curente dacă parametrul "timp\_UNIX" nu este furnizat.  
17.**localtime(timp\_UNIX, bool)**  
- Returneaza un Array cu informatii despre ora locala (a serverului) la timpul specificat in parametrul "timp\_UNIX" (sau timpul curent, daca acesta nu e adaugat).  
"bool" poate fi true sau false, si e optional. Daca e specificat true, va returna un Array asociativ, in rest, un Array secvential.  
18.strong>microtime(bool)  
- Returneaza timpul curent Unix cu microsecunde. Dacă parametrul opțional "bool" este stabilit în TRUE, atunci este returnata o valoare **Float** (cu virgula, în secunde), altfel, returneaza sirul "**msec. sec.**", unde '**sec.**' este timpul curent măsurat în numărul de secunde de la Epoca Unix (0:00:00 January 1, 1970 GMT) și '**msec.**' este partea cu microsecunde.  
19.**mktime(ora, minut, secunde, luna, zi, an)**  
- Obtine timpul Unix al argumentelor furnizate, care sunt numere intregi ce reprezinta: "ora", "minutul", "secundele", "luna", "ziua" si "anul" din data ce o formeaza. Dacă argumentele nu sunt valide, funcția întoarce FALSE.  
20.**strtotime('data\_form')**  
- Returneaza timpul UNIX a aproape oricarei descrieri textuale în limba engleză a datei și orei specificate in sirul "data\_form".  
*Dacă anul este specificat cu două cifre, valorile între 00-69 sunt interpretate ca 2000-2069, iar valorile între 70-99 ca 1970-1999.*  
21.**time()**  
- Returneaza timpul curent, in format UNIX, si anume ***numarul de secunde de la 1 Ianuarie 1970 ora 0 (1 Ianuarie 1970 00:00:00 GMT)***. Numarul returnat poate fi folosit cu functia "date()" pentru a obtine data intr-un format cunoscut sau parti din data (ziua, ora, luna, etc.).

**Functii pentru fișiere și directoare**

Aici gasiti cateva din cele mai utile functii PHP pentru lucru cu sistemul de fisiere si directoare.  
1.**basename(adresa, "sufix")**  
- Returneaza partea cu numele fisierului dintr-o adresa URL. "sufix" e optional, daca e specificat, functia elimina si pe acesta de la sfarsitul numelui.  
2.**chmod(file, mod)**  
- Modifica prmisunile CHMOD a lui "file" (*cale si nume fiser*) cu cele specificate la "mod" (*0777, 0644, ...*). Daca modificarea reuseste returneaza TRUE, in caz contrar, FALSE.  
3.**copy(sursa, destinatie)**  
- Copie fisierul specifical la "sursa", in locatia "destinatie" (aici poate fi specificata si denumirea fisierului la dstinatie). Aca fisierul deja exista, va fi inlocuit de cel copiat. Funtctia returneaza TRUE in caz de reusita sau FALSE.  
4.**dirname(adresa)**  
- Returneaza numele directorului din "adresa", fara numele fisierului sau alte extensii.  
5.**disk\_free\_space(cale\_dir)**  
- Returneaza spatiul liber al sistemului in directorul specificat la "cale\_dir".  
6.**disk\_total\_space(cale\_dir)**  
- Returneaza spatiul total al sistemului in directorul specificat la "cale\_dir". Returneaza numarul total de bytes sau FALSE daca functia esueaza.  
7.**fopen("fisier", "mod")**  
- Deschide un pointer la "fisier", in modul de lucru "mod". "mod" poate fi (*"b" e indicat sa fie adaugat pentru sistemele windows*):

* **r** sau **rb** - pentru citire.
* **r+** sau **r+b** - citire si scriere.
* **w** sau **wb** - creare si scriere fisier de la 0.
* **w+** sau **w+b** - citire si scriere fisier de la 0, il creaza daca nu exista.
* **a** sau **ab** - scriere, adauga datele dupa cele deja existente. Creaza fisierul daca nu exista.
* **a+** sau **a+b** - citire si scriere, adauga datele dupa cele deja existente. Creaza fisierul daca nu exista
* **x** sau **xb** - creare si scriere, daca fisierul exista deja, returneaza eroare.
* **x** sau **x+b** - creare pentru citire si scriere, daca fisierul exista deja, returneaza eroare.

8.**fclose(file\_pointer)**  
- Inchide deschiderea unui fisier cu "fopen()" (*transmisa la "file\_pointer"*), eliberand memoria folosita de acea deschidere. Returneaza TRUE daca functia reuseste, in caz contrar, FALSE.  
9.**feof(file\_pointer)**  
- Testeaza sfarsitul unnui fisier cu deschiderea specificata la "file\_pointer". Functie utila cand fisierul e citit linie cu linie. Returneaza TRUE daca pointer-ul e la sfarsitul fisierului, altfel, returneaza FALSE.  
10.**fgets("pointer", lungime")**  
- Citeste numarul de caractere specificat la "lungime" dintr-un fisier deschis si specificat la "pointer". Citirea se face din locul unde a ajuns pointer-ul in fisier, daca "lungime" nu e specificat citirea se va face pana la sfarsitul randului curent (*daca lungimea liniei e mai mare de 8 KB, e indicata specificarea ei*).  
11.**file\_exists("fisier")**  
- Returneaza TRUE daca "fisier" exista, altfel, returneaza FALSE.  
12.**file\_get\_contents("fisier")**  
- Returneaza tot continutul din "fisier" intr-un sir. Daca "fisier" nu poate fi citit, returneaza FALSE.  
13.**file\_put\_contents("fisier", "continut")**  
- Scrie in "fisier" sirul specificat la "continut". Daca fisierul exista, suprascrie datele existente, daca nu exista, il creaza. Daca nu poate scrie in "fisier", returneaza FALSE.  
14.**file("fisier", *flag*)**  
- Preia continutul fisierului intr-un array, fiecare linie din "fisier" va fi un element din matrice. "flag" e optional, in locul lui se poate adauga: **FILE\_IGNORE\_NEW\_LINES** (nu va prelua si caracterele de linie noua),**FILE\_SKIP\_EMPTY\_LINES** (va sari peste randurile goale).  
15.**fileatime("file")**  
- Returneaza timestamp-ul (timpul UNIX) cand a fost accesat ultima data "file", sau in caz de nereusita, FALSE.  
16.**filectime("file")**  
- Returneaza timestamp-ul (timpul UNIX) cand a fost modificat ultima data "file", sau in caz de nereusita, FALSE. Similara cu aceasta e functia **filemtime()**  
17.**fileperms("file")**  
- Returneaza permisiunile CHMOD ale "file", sau FALSE.  
18.**filesize("file")**  
- Returneaza marimea lui "file", in bytes, sau FALSE.  
19.**flock(file\_pointer, flag)**  
- Blocheaza / deblocheaza accesul de citire sau scriere la un fisier deschis cu "fopen()", specificat la "file\_pointer". Returneaza TRUE, sau daca functia esueaza, FALSE. Modul de blocare, sau deblocarea, se specifica la "flag", acesta putand fi:

* **LOCK\_SH**- blocare acces citire.
* **LOCK\_EX**- blocare acces scriere.
* **LOCK\_UN**- deblocare

20.**fseek(file\_pointer, pozitie)**  
- Muta pozitia pointer-ului dintr-un fisier deschis cu "fopen()" la numarul de caractere specificat de "pozitie" (0 e primul caracter). In caz de succes returneaza 0, altfel, -1.  
21.**ftell(file\_pointer)**  
- Returneaza pozitia curenta a indicatorului (pointer) dintr-un fisier deschis cu "fopen()" (specificat la "file\_pointer"). In caz de esec returneaza FALSE.  
22.**fwrite(file\_pointer, 'sir')**  
- Scrie intr-un fisier deschis cu "fopen()" (specificat la "file\_pointer") continutul din "sir".  
23.**is\_dir("nume")**  
- Returneaza TRUE daca "nume" exista si este un director, altfel returneaza FALSE.  
24.**is\_file("nume")**  
- Returneaza TRUE daca "nume" exista si este un fisier, altfel returneaza FALSE.  
25.**is\_readable("nume")**  
- Returneaza TRUE daca "nume" (fisier sau director) exista si poate fi citit, altfel returneaza FALSE.  
26.**is\_uploaded\_file("fisier\_up")**  
- Returneaza TRUE daca "fisier\_up" a fost incarcat pe server, prin upload si primit prin $\_POST, altfel returneaza FALSE. Parametrul functiei trebuie sa fie de forma "$\_FILES['userfile']['tmp\_name']".  
27.**is\_writable("nume")**  
- Returneaza TRUE daca "nume" (fisier sau director) exista si poate sa scrie in el, altfel returneaza FALSE.  
28.**mkdir("dir", chmod)**  
- Returneaza TRUE daca a putut crea directorul "dir" cu permisiunile specificate la "chmod", altfel returneaza FALSE.  
29.**pathinfo("adresa\_file")**  
- Returneaza o matrice cu informatiile: ale "adresa\_file".  
30.**readfile("fisier")**  
- Citeste si returneaza la browser (afiseaza) continutul din "fisier". In caz de esec returneaza FALSE.  
31.**realpath("cale\_adresa")**  
- Returneaza calea reala in sistem a unei adrese din "cale\_adresa", rezolvand referinte ca .. / ./ ../.  
32.**rename("nume", "nume\_nou"**  
- Redenumeste un director sau fisier specificat la "nume", cu "nume\_nou". In caz de succes returneaza TRUE, altfel FALSE.  
33.**rewind(file\_pointer)**  
- Readuce pozitia pointer-ului dintr-un fisier deschis cu "fopen()" (specificat la "file\_pointer") la inceputul fisierului (0, primul caracter). In caz de succes returneaza TRUE, altfel FALSE.  
34.**rmdir("dir")**  
- Sterge directorul specificat la "dir". Acesta trebuie sa fie gol. In caz de succes returneaza TRUE, altfel FALSE.  
35.**tempnam("dir", "prefix")**  
- Creaza un fisier cu nume unic, avand la inceput in nume sirul specificat la "prefix", in directorul "dir". Ii acorda CHMOD 0600. Daca directorul specificat nu exista, creaza fisierul in directorul "temp" al sistemului si returneaza numele fisierului. In caz de nereusita returneaza FALSE.  
36.**tmpfile()**  
- Creaza un fisier temporar, cu nume unic. Returneaza un pointer de fisier, ca si "fopen()", cu modul de deschidere "w+". In caz de nereusita returneaza FALSE. Cu "fclose()" fisierul temporar va fi sters.  
37.**unlink("fisier")**  
- Sterge fisierul specificat. In caz de succes returneaza TRUE, altfel FALSE.

**Functii MySQL**

Aici gasiti cateva din cele mai utilizate **functii PHP** pentru **MySQL**.  
1.**mysql\_affected\_rows($conn)**  
- Returneaza numarul de linii afectate in ultima operatie PHP-MySQL (INSERT, UPDATE, REPLACE ori DELETE) sau -1 daca acel "query" esueaza. "$conn" este optional si reprezinta conexiunea de lucru la MySQL.  
Specificatie:*Cand se foloseste REPLACE, aceasta intai sterge inregistrarile cu aceeasi cheie primara si dupa aia adauga noile inregistrari. Functia "mysql\_affected\_rows()" va returna numarul de inregistrari sterse plus cele adauggate.*  
2.**mysql\_client\_encoding($conn)**  
- Returneaza tipul de codare a formatului de caractere folosit in conexiunea curenta ($conn).  
3.**mysql\_connect('server\_mysql', 'nume\_utilizator', 'parola\_myswql')**  
- Deschide o conexiune la MySQL. La argumentul "server\_mysql" se poate adauga si portul de conectare, sintaxa devine astfel: 'server\_mysql:port'.  
4.**mysql\_close($conn)**  
- Inchide conexiunea la MySQL "$conn", daca nu e precizat "$conn", va inchide conexiunea curenta.  
5.**mysql\_list\_dbs($conn)**  
- Returneaza o lista cu bazele de date accesibile in conexiuea curenta ($conn), altfel, FALSE.  
6.**mysql\_errno($conn)**  
- Returneaza numarul erorii de la ultima functie MySQL, sau 0 (*zero*) daca nu e vreo eroare.  
7.**mysql\_error($conn)**  
- Returneaza mesajul de eroare (sir text) dat de MySQL la ultima functie executata, sau un sir gol daca nu sunt erori.  
8.**mysql\_query("Comanda\_mysql", $conn)**  
- Trimite o interogare SQL scrisa in "Comanda\_mysql" la serverul MySQL. Returneaza un identificator in care e stocat raspunsul de la server, in functie de comanda SQL trimisa, sau in caz de esec (eroare), FALSE.  
9.**mysql\_fetch\_array(sql\_rezult, tip)**  
- Returneaza un Array ce contine rezultatul dat de MySQL in urma unei interogari facute cu "mysql\_query()" si transmis in "sql\_rezult". In caz de esec, FALSE.  
Daca "tip" are valoarea **MYSQL\_ASSOC**, va returna un Array asociativ in care cheile sunt numele coloanelor din tabelul MySQL (ca si "mysql\_fetch\_assoc()"). Daca "tip" are valoarea **MYSQL\_NUM** se va obtine o matrice secventiala, cu cheile numere consecutive (ca si "mysql\_fetch\_row()"). Iar daca "tip" are valoarea **MYSQL\_BOTH** (care e default, daca tip nu e precizat) se va obtine un Array cu ambele tipuri.  
10.**mysql\_fetch\_assoc(sql\_rezult)**  
- Returneaza un Array asociativ (in care cheile sunt numele coloanelor din tabelul MySQL) cu rezultatul dat de MySQL in urma unei interogari facute cu "mysql\_query()" si continut in "sql\_rezult". In caz de esec, FALSE.  
- Muta si pointerul intern din tabelul MySQL deasupra.  
11.**mysql\_fetch\_row(sql\_rezult)**  
- Returneaza un Array secvential (in care cheile sunt numere consecutive, de la 0) cu rezultatul dat de MySQL in urma unei interogari facute cu "mysql\_query()" si continut in "sql\_rezult". In caz de esec, FALSE.  
- Muta si pointerul intern din tabelul MySQL deasupra.  
12.**mysql\_fetch\_object(sql\_rezult)**  
- Returneaza un obiect cu rezultatul dat de MySQL in urma unei interogari facute cu "mysql\_query()" si continut in "sql\_rezult".Proprietatile acestui obiect sunt numele coloanelor. In caz de esec, FALSE.  
- Muta si pointerul intern din tabelul MySQL deasupra.  
13.**mysql\_fetch\_field(sql\_rezult)**  
- Obtine un obiect cu o lista de informatii despre coloanele dintr-o interogare cu "mysql\_query()".  
14.**mysql\_field\_len(sql\_rezult, nr\_camp)**  
- Returneaza lungimea campului de la "nr\_camp", acest parametru este numarul de ordine al coloanelor dintr-o interogare cu "mysql\_query()" (a carui raspuns e continut in "sql\_rezult"). Prima coloana are numarul 0. In caz de esec, FALSE.  
15.**mysql\_field\_name(sql\_rezult, nr\_camp)**  
- Obtine numele campului de la "nr\_camp", acest parametru este numarul de ordine al coloanelor dintr-o interogare cu "mysql\_query()" (a carui raspuns e continut in "sql\_rezult"). Prima coloana are numarul 0. In caz de esec, FALSE.  
16.**mysql\_field\_type(sql\_rezult, nr\_camp)**  
- Obtine tipul campului de la "nr\_camp" ("string", "int", "blob", sau altele ...), acest parametru este numarul de ordine al coloanelor dintr-o interogare cu "mysql\_query()" (a carui raspuns e continut in "sql\_rezult"). Prima coloana are numarul 0.  
17.**mysql\_free\_result(sql\_rezult)**  
- Elibereaza memoria serverului dupa prelucrarea informatiilor obtinute cu o interogare "mysql\_query()" (stocata in "sql\_rezult").  
18.**mysql\_get\_client\_info()**  
- Returneaza un sir cu date despre versiunea MySQL activa pe server. Nu necesita conectare la MySQL.  
19.**mysql\_get\_host\_info($conn)**  
- Da informatii despre gazda serverului MySQL din conexiunea $conn.  
20.**mysql\_insert\_id($conn)**  
- Obtine ultimul ID generat de o coloana "AUTO\_INCREMENT" in ultima interog

**Prelucrarea datei și orei**

Limbajul PHP oferă soluții simple pentru prelucrarea datei și orei. În versiunile mai noi sunt introduse facilități avansate cum ar fi DST (Daylight Saving Time), Timezones, Date Intervals, s.a. Înainte de acestea însă lucrul cu timpul se realiza folosind Unix timestamp. Conceptul de Unix timestamp desemnează numarul de secunde scurse de la 1 ianuarie 1970 (cand se considera ca a început era Unix).

Fiind vorba de un număr pozitiv ce are mereu aceeași referință, este ușor să se realizeze operații cum ar fi scăderea, adunarea sau compararea a două date reprezentate printr-un Unix timestamp. Mai jos sunt câteva aplicații menite să evidențieze cât de ușor se pot manipula date și ore în PHP.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?php  // data curenta  print date( 'd.m.Y H.i.s' )."<br>";    // data in format Unix (UNIX timestamp)  print time()."<br>";    // alegerea unui fus orar  date\_default\_timezone\_set( 'Europe/Bucharest' );    // parsarea unei date  print strtotime("now")."<br>";  print strtotime('1st January 2004');  ?> |

**Cum se formatează data sau ora?**

Formatarea datei sau a orei se poate realiza ușor folosind funcția date. Cu ajutorul acesteia se pot "extrage" doar acele părți din data care sunt necesare și se poate specifica formatul dorit. Spre exemplu, folosind funcția **date** putem afișa data în format lung (de genul: "marti, 30 septembrie 2017") sau scurt (30.09.2017) sau cu aceeași funcție putem afișa ora sau oricare combinație a celor două. Formatul se specifică print-un text format din caractere ce au o semnificație anume, transmis ca parametru funcției **date**. Pentru detalii complete despre fiecare caracter în parte accesați http://www.php.net/manual/ro/function.date.php.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <?php  # se preia data/ora curenta  $acum = time();    # formatarea se realizeaza cu functia date  # daca nu se specifica altfel, functia date va formata data/ora curenta  echo date("H:i:s") . "<br>";  echo date("Y-m-d") . "<br>";  echo date("F j, Y, g:i a") . "<br>";  echo date("l") . "<br>";    # se poate specifica un al doilea parametru ce reprezinta data/ora care se formateaza  echo date("d-m-Y H:i:s", $acum) . "<br>";  echo date("D, M d, Y", 1072915200) . "<br>";    # combinat cu mktime, se pot obtine informatii despre zile din trecut sau viitor  # de exemplu, ce zi a fost pe 4 iulie 2000  $iulie4 = mktime( 0, 0, 0, 7, 4, 2000 ); # aflu timestamp-ul  echo date( 'l', $iulie4 ); # formatez acel timestamp - extrag doar ziua  // afiseaza Tuesday    ?> |

**Cum determin ce zi și oră va fi peste 480 ore?**

Pentru a efectua ușor operații cu zile și ore, se va folosi formatul de dată Unix (UNIX timestamp), returnat de funcția **time()**. Practic, data este afișată ca numarul de secunde scurs de la 1 ianuarie 1970, ora 00:00. Având această nouă perspectivă (de a privi datele ca un număr de secunde), operațiile cu date (de genul 'cât va fi peste 2 zile', 'ce oră a fost acum xxx minute', etc.) devin extrem de simple luând forma unor simple scăderi și adunări de secunde.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?php  // cerinta este: "cat va fi ora peste 480 de ore, adica peste (480 \* 60 \*60) secunde    $acum = time();  $maiTarziu = 480 /\*ore\*/ \* 60 /\*minute\*/ \* 60 /\*secunde\*/;    print "Data ceruta este: " . date( 'd.m.Y H.i.s', $acum + $maiTarziu );  // Afiseaza Data ceruta  ?> |

**Cum afișez luna curentă și toate cele 12 luni în limba română?**

**Nota:** această secvență de cod depinde de configurările serverului pe care este executată. Limba română trebuie instalată pe server pentru a avea rezultatele scontate.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?php  setlocale( LC\_TIME , 'ROM\_ROM' );    // afiseaza numele celor 12 luni  for( $m = 1; $m <= 12; $m++ ) {      print strftime( "%B", mktime( 0,0,0, $m, 1, 2008 ) ) . "<br>";  }    // afiseaza luna curenta  print 'Luna curenta este: ' . strftime( "%B", time() );  ?> |

Rezultatul este posibil să nu fie în limba română!

**Clase și obiecte în PHP**

O clasă este o colecție de variabile și funcții care operează asupra variabilelor respective. Sintaxa folosită pentru declararea unei clase în PHP este:

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?php   class nume\_clasa   {   //date membre   var nume\_variabila\_1   ...   var nume\_variabila\_m   //metode   function nume\_functie\_1(parametri)   {   ...//definitia functiei   }   ...   function nume\_functie\_n (parametri)   {   ...//definirea functiei   }  }   ?> |

Pentru numele unei clase poate fi utilizat orice identificator permis în PHP cu o singură excepție: ***sdtclass***; acest identificator este folosit de PHP în scopuri interne.  
În PHP funcțiile ale căror identificatori încep cu **"\_\_"(două caractere '\_')** sunt considerate funcții magice și utilizarea acestora nu este recomandată.  
În PHP, datele membre nu pot fi inițializate decît cu valori constante. Pentru a inițializa variabilele cu valori care nu sunt constante trebuie folosit un constructor. Mai jos aveti un exemplu de clasă în care initializările ***nu sunt corecte***:

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?php  class Nepermis  {  var $data = date("Y-m-d");  var $nume =$prenume;  var $dest ='Mihai'.'Claudiu';  var $obiecte =array("minge","pantof");  }  ?> |

**Obiecte**

În PHP clasele sunt considerate a fi tipuri de date; ele pot fi privite ca fiind "amprentele" variabilelor propriu-zise. Pentru a crea o variabilă al cărei tip este o clasă trebuie utilizat operatorul **new**. În continuare vom defini o clasă ***Aritmetica*** cu două date membre x și y care sunt numere întregi și două metode care realizează adunarea, respectiv înmulțirea lor.

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?php  class Aritmetica  {  var $x=2;  var $y=3;  function Suma()  {  return $this -> x+$this->y;  }  function Produs()  {  return $this ->x\*$this->y;  }  }  ?> |

Pentru a crea un obiect de tipul ***Aritmetica*** vom utiliza o instrucțiune de tipul:

|  |  |
| --- | --- |
| 1 | $aritm = new Aritmetica; |

Acum putem utiliza metodele clasei; pentru a afișa suma sau produsul celor două numere vom putea apela cele două metode astfel:

|  |  |
| --- | --- |
| 1  2 | echo $aritm -> Suma ( );  echo $aritm -> Produs ( ); |

Vom obține rezultatele 5, respectiv 6. Valorile datelor membre pot fi și ele modificate prin instrucțiuni de tipul:

|  |  |
| --- | --- |
| 1  2 | $aritm -> x = 5;  $aritm -> y = 4; |

Dacă, în urma modificării apelam din nou metodele ***Suma( )*** și ***Produs( )***, rezultatele vor fi 9, respectiv 20.

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?php  class PHP4  {  var $salut="Salut PHP4!";  function Salut()  {  return $this->salut;  }  }  $salutare =new PHP4;  echo $salutare ->Salut()."<br>";  ?> |

Rezultatul codului de mai sus este:

|  |  |
| --- | --- |
| 1 | Salut PHP4! |

În acest exemplu a fost utilizată pseudo-variabila ***$this***. Aceasta este folosită pentru a indica faptul că se operează asupra unei date membre a obiectului curent.

**Extinderea claselor**

Deseori este necesară definirea unor clase cu proprietăți (date membre) și metode asemănătoare. Pentru a usura definirea unor astfel de clase a fost introdus conceptul de **extindere** (derivare) a claselor.

O clasă derivată va păstra toate proprietățile și metodele clasei pe care o extinde și poate conține diferite proprietăți și metode noi. Nu există nici o posibilitate de a elimina din clasa derivată anumite proprietăți sau metode ale clasei de bază. O anumită clasă poate avea o singură clasă **părinte**; așadar, în PHP nu este permisă**moștenirea** multiplă. Pentru a extinde o anumită clasă se utilizează cuvîntul cheie ***extends***.

ή următorul exemplu voi extinde clasa *Aritmetica*; o să adaug încă o variabilă și o să crez două noi funcții: una pentru calculul sumei celor trei variabile și una pentru calcularea produsului lor:

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?php  class Aritmetica3 extends Aritmetica  {  var z=4;  function Suma3()  {  return $this->x+$this->y+$this->z;  }  function Produs3()  {  return $this->x\*$this->y\*$this->z;  }  }  ?> |

Dacă definim un obiect prin intermediul unei instrucțiuni de genul:

|  |  |
| --- | --- |
| 1 | $aritm3 = new Aritmetica3; |

Atunci pentru acest obiect vom putea utiliza atît metodele definite în cadrul clasei *Aritmetica3: Suma3( )*și *Produs3( )*, cît și metodele definite în cadrul clasei de bază *Aritmetica: Suma( )*și *Produs( )*.

În continuare aveți un exemplu care ilustrează modul în care pot fi create și utilizate clasele derivate.

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | <?php  class Aritmetica  {  var $x=2;  var $y=3;  function Suma()  {  return $this->x+$this->y;  }  function Produs()  {  return $this->x\*$this->y;  }  }  class Aritmetica3 extends Aritmetica  {  var $z=4;  function Suma3()  {  return $this->x+$this->y+$this->z;  }  function Produs3()  {  return $this ->x\*$this->y\*$this->z;  }  }  $aritm3 = new Aritmetica3;  echo "<b>Inainte de modificare.</b>";  echo "<br>";  echo "Suma primelor doua numere: ";  echo $aritm3->Suma()."<br>";  echo "Produsul primelor doua numere: ";  echo $aritm3->Produs()."<br>";  echo "Suma celor trei numere:";  echo $aritm3->Suma()."<br>";  echo "Produsul celor trei numere:";  echo $aritm3->Produs()."<br>";  $aritm3->x=5;  $aritm3->y=4;  $aritm3->z=3;  echo "<br><br>";  echo "<b>Dupa modificare.</b><br>";  echo "<br>";  echo "Suma primelor doua numere:";  echo $aritm3 -> Suma3()."<br>";  echo "Produsul primelor doua numere:";  echo $aritm3->Produs3()."<br>";  echo "Suma celor trei numere: ";  echo $aritm3->Suma3()."<br>";  echo "Produsul celor trei numere:";  echo $aritm3->Produs3()."<br>";  ?> |

Rezultatul codului de mai sus este:

**Inainte de modificare.**  
Suma primelor doua numere: 5  
Produsul primelor doua numere: 6  
Suma celor trei numere: 5  
Produsul celor trei numere: 6

**Dupa modificare.**  
Suma primelor doua numere: 12  
Produsul primelor doua numere: 60  
Suma celor trei numere: 12  
Produsul celor trei numere: 60

În PHP clasele trebuie definite înaintea utilizării lor; așadar clasa părinte va fi definită întotdeauna înaintea clasei **fiu**.

**Constructori**

Un constructor este o metodă (funcție) a unei clase care este apelată automat în momentul în care este creată o nouă instanță a clasei (cu ajutorul operatorului **new**). În PHP, este considerată ca fiind un constructor, orice funcție care are același nume cu clasa în interiorul căreia este definită.  
Constructorii pot fi folosiți pentru inițializarea datelor membre cu valori care nu sunt constante. Ei pot avea argumente, iar acestea pot fi optionale. Pentru a putea utiliza clasa fără a specifica nici un parametru în momentul creării unui obiect, se recomandă stabilirea unor valori implicite pentru toate argumentele constructorului. În cazul în care nu este definit un constructor pentru o anumită clasă, se utilizează constructorul clasei de bază, dacă aceasta există. De exemplu, pentru următoarea secvență de cod, în momentul creării obiectului corespunzător variabilei ***$b***, va fi apelat constructorul clasei ***A***.

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <?php  class A  {  function A()  {  echo "Constructorul clasei A<br>";  }  function B()  {  echo "O functie obisnuita a clasei A.<br>";  }  }  class B extends A  {  function C()  {  echo "O functie obisnuita a clasei B.<br>";  }  }  ?> |

În PHP apelul constructorului clasei de bază trebuie să fie explicit dacă este necesară executarea operațiilor corespunzătoare. În majoritatea limbajelor de programare există funcții speciale numite destructori care sunt apelate automat în momentul "distrugerii" unui obiect. În PHP nu există destructori.

**Operatorul ::**

Uneori este utilă folosirea unor metode sau variabile ale clasei de bază sau ale unei clase care nu a fost instanțiată încă. În acest scop a fost introdus operatorul**::**  
Pentru a descrie modul de utilizare al acestui operator voi prezenta mai întîi un exemplu:

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <?php  class A  {  function exemplu()  {  echo "Functia clasei de baza.<br>";  }  }  class B extends A  {  function exemplu()  {  echo "Functia redefinita<br>\n";  A::exemplu();  }  }  A::exemplu();  $b=new B;  $b->exemplu();  $b=new B;  ?> |

Prin intermediul instrucțiunii

|  |  |
| --- | --- |
| 1 | A :: exemplu( ); |

este apelată metoda *exemplu( )* a clasei A, așadar se afișează mesajul '*Functia clasei de baza*' cu toate că nu există nici un obiect care este o instanță a acestei clase, deci nu putem scrie o instrucțiune de tipul *$a -> exemplu( );*  
În schimb apelăm metoda:

|  |  |
| --- | --- |
| 1 | $b -> exemplu( ); |

ca "o funcție a clasei" și nu ca "o funcție a unui obiect". Putem avea funcții ale claselor, dar nu putem avea variabile ale claselor. De fapt, în momentul unui astfel de apel nu se creeaza nici un obiect care este instanță a clasei respective. Ca urmare, o funcție a unei clase nu poate opera asupra unor proprietăți ale clasei, dar poate utiliza variabile locale sau globale. În plus, o astfel de funcție nu poate utiliza pseudo-variabila **$this**.

În exemplul anterior, în cadrul clasei ***B*** este redefinită funcția *exemplu( )*. Așadar, definiția "originală" (din cadrul clasei ***A***) nu poate fi accesată în interiorul clasei B decît dacă ne referim la ea explicit prin intermediul operatorului **::**.

**Accesarea clasei de baza**

În exemplul anterior am utilizat o funcție a clasei de bază. În locul utilizării denumirii clasei de baza poate fi folosită denumirea specială ***parent*** care este o referință la clasa de bază definită în cadrul construcției *extends*. Folosirea denumirii speciale este utilă în cazul în care ierarhia de clase se modifică. În acest caz este suficientă o singură modificare în cadrul construcției *extends*, fără a mai fi necesare modificări în interiorul clasei derivate. Așadar, definiția clasei ***B*** poate fi rescrisă astfel:

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php  class B extends A  {  function exemplu()  {  "Functia redefinita<br>\n";  parent :: exemplu();  }  }  ?> |

**Serializarea obiectelor**

Prin serializare se înțelege crearea unui șir de octeți care conțin reprezentarea internă (binară) a variabilei respective. Așadar, serializarea permite "salvarea" valorilor unei variabile. Dacă este serializat un obiect sunt salvate doar proprietățile acestuia (variabilele membre) și numele clasei din care face parte, nu și metodele, deoarece funcțiile nu reprezintă valori.  
Pentru a serializa un obiect este utilizată funcția ***serialize( )*** care returnează șirul de octeți care conține reprezentarea binară.  
Pentru a deserializa un obiect se folosește funcția pereche ***unserialize( )***.  
Pentru ca o astfel de operație să funcționeze corect este necesară definirea clasei din care face parte obiectul respectiv. Funcția returnează valoarea variabilei serializate. În exemplul următor aveți prezentat modul în care poate fi serializat și deserializat un obiect. Şirul de octeți obținut în urma serializării va fi scris într-un fișier și va fi citit din fișierul respectiv pentru efectuarea deserializării. De obicei serializarea și deserializarea sunt realizate în documente php diferite deoarece aeste operații nu au aproape nici o utilitate dacă sunt folosite în cadrul aceluiași document. Primul document în care se realizează serializarea trebuie să conțină o secvență asemănătoare cu următoarea:

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <?php  class A  {  var $msg ="Salutare lume";  function scrie()  {  echo $this->msg;  }  }  $a=new A;  $s=serialize($a);  //salvarea sirului intr-un fisier  $fp=fopen("fisier","w");  fputs($fp,$s);  fclose($fp);  ?> |

Pentru deserializare al doilea document va conține următoarea secvență:

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?php  class A   {  var $msg="Salutare lume";  function scrie()  {  echo $this->msg;  }  }  //citirea sirului din fisier  $s=implode("",@file("fisier"));  $a=unserialize($s);  //dupa deserializare obiectul poate fi folosit  $a->show\_one();  ?> |

Referințele pot fi utilizate pentru a accesa conținutul unei variabile folosind mai multe nume. Spre deosebire de limbajul C, în PHP referințele nu sunt pointeri, ci alias-uri într-o tabela de simboluri. În PHP denumirile variabilelor și conținutul acestora nu sunt unul și același lucru. Așadar este posibil ca același conținut să aibă denumiri diferite.

**Utilizarea referințelor**

Referințele PHP permit unor variabile cu denumiri diferite să corespundă unui același conținut. Cu alte cuvinte, instrucțiunea

|  |  |
| --- | --- |
| 1 | $a = &$b |

are ca efect faptul că ***$a*** și ***$b*** referă aceeași variabilă. În această situație ***$a*** și ***$b***au același statut. Nu se poate spune că ***$a*** referă ***$b*** sau invers. O altă posibilitate de utilizare a referințelor este transmiterea prin referință a parametrilor unei funcții. Efectul unei astfel de transmisii este crearea unei variabile locale care referă spre același conținut ca variabila din contextul apelant. Să luăm în considerare următorul exemplu:

**Cod:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php    function inc (&$var)    {    $var++;    }    $a=5;    inc($a);  ?> |

Inițial valoarea variabilei **$a** este 5. După apel variabila locală *$var* și variabila din contextul apelant **$a** indică spre același conținut. Valoarea păstrată în locația de memorie respectivă este incrementată (devine 6) prin intermediul instrucțiunii *$var++;*.

Datorită faptului că cele două variabile au același conținut, valoarea variabilei **$a** va fi 6 după executarea funcției.  
Un parametru transmis prin referință poate fi:  
- o variabilă;  
- o instrucțiune **new**;  
- o referință returnată de o funcție.

Dacă unei astfel de funcții i se transmite ca parametru un alt tip de expresie rezultatul este nedefinit. Așadar, pentru o funcție care are un parametru transmis prin referință nu se poate folosi o constantă în momentul apelului. De exemplu, pentru funcția*inc( )* prezentată anterior nu este permis un apel de forma*inc(5)*.

**Referințe globale**

În momentul declarării unei variabile globale (printr-o instrucțiune de tipul **global** *$var*) se creează de fapt o referință spre o variabilă globală. Cu alte cuvinte, această instrucțiune este echivalentă cu *$var = &$GLOBALS ["var"];*.

**Referința $this**

În cadrul unei metode a unui obiect **$this** este întotdeauna o referință spre obiectul care utilizează funcția (obiectul curent).

**Construiește un site web cu PHP**

|  |  |
| --- | --- |
| ANTET | |
| MENU Pag A Pag B Pag C | CONTINUT |
| SUBSOL | |

index.php

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <?php  include("antet.html");  ?>  <tr>    <?php  include("menu.html");  $k=$\_GET['k'];  if (isset($k)) {include($k);}  else{  include("continut.html");  }  ?>  </tr>    <?php  include("subsol.html");  ?> |

antet.html

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <html>  <head>  <title>Site în PHP cu tabele</title>  </head>    <body>  <table width="600" border="l" cellspacing="0" align="center">  <tr>  <td height="78"  colspan="2" align="center" bgcolor="#00CC00" > ANTET </td> </tr> |

meniu.html

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <td width="60" height="270" align="center" bgcolor="#0099FF">  <p><strong>M E N U</p>  <hr color="blue">  <p><a href="index.php">Home</a></p>  <p><a href="index.php?k=p1.html">Pag A</a></p>  <p><a href="index.php?k=p2.html">Pag B</a></p>  <p><a href="index.php?k=p3.html">Pag C</a></p></strong>  </td> |

continut.html

|  |  |
| --- | --- |
| 1 | <td  width="325" align="center" bgcolor="#FFCC33"> CONTINUT </td> |

subsol.html

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <tr>  <td  height="58" colspan="2" align="center" bgcolor="#FF6633">SUBSOL</td>  </tr>  </table>  </body>  </html> |

p1.html

|  |  |
| --- | --- |
| 1 | <td width="325" align="center" bgcolor="#FFCC33">  PAGINA A </td> |

p2.html

|  |  |
| --- | --- |
| 1 | <td width="325" align="center" bgcolor="#FFCC33">  PAGINA B </td> |

p3.html

|  |  |
| --- | --- |
| 1 | <td width="325" align="center" bgcolor="#FFCC33">  PAGINA C </td> |

[**Vezi exemplul**](http://web.ceiti.md/ex-site-php/)

**Cîteva idei utile pentru securitatea paginilor web**

Regula numărul unu a securității online este: nu vă încredeți niciodată în utilizator. Întotdeauna verificați datele trimise către server și "curățați-le" înainte de a le utiliza. Pentru aceasta trebuie să luați în considerare cîteva posibile "găuri" de securitate. Menționez că cele scrise aici sunt doar noțiuni generale, pentru o securitate ridicată ar trebui sa aveți noțiuni specializate de baze de date și de administrare a serverelor web.

**1. Variabilele globale**

Avînd variabilele globale din *php.ini* setate pe ON puteți accesa datele trimise prin formulare mai simplu: *$variabila* în loc de *$\_GET['variabila']*, la fel și pentru *POST* sau *FILES*. Dacă scriptul nu este foarte bine gîndit, variabilele globale pot prezenta un risc major de securitate. Din acest motiv *php.ini* este distribuit cu *globals=off* în ultimele versiuni.

**2. Ghilimele "magice"**

Dacă în *php.ini* *magic\_quotes\_gpc* sunt OFF folosiți funcția *addslashes* pentru a preceda ghilimelele din datele trimise de utilizatori cu caracterul \. Dacă *magic\_quotes\_gpc* sunt ON, PHP adaugă automat caracterul **\** înainte de ghilimele dar dacă ele sunt OFF, ghilimelele din input vă pot crea probleme serioase.

Ca exemplu, să presupunem că interogarea SQL de verificare a numelui și a parolei pentru înregistrarea pe site este

|  |  |
| --- | --- |
| 1 | SELECT \* FROM users WHERE nume='$nume' AND parola='$parola' |

și dacă interogarea este executată cu succes, utilizatorul este logat. În acest caz, folosind parola 'OR' 1=1 oricine poate avea acces pe site deoarece interogarea

|  |  |
| --- | --- |
| 1 | SELECT \* FROM user WHERE nume='un nume oarecare' AND parola='' OR '1=1' |

este executată cu succes și returnează un rezultat (toate înregistrarile din baza de date, de fapt). Mai mult, atunci cînd vă așteptați ca baza de date să returneze un singur rînd, verificați acest lucru și nu dacă interogarea s-a executat cu succes. Implicit în *php.ini* *magic\_quotes\_gpc* sunt ON pentru a vă proteja de astfel de atacuri, dar este bine să verificați înainte de a renunța la *addslashes*.

**3. Includere**

Încercați să evitați includerea "vizibilă" a fișierelor în forma http://site.md/fisier.php?file=cutare.html pentru a include fișiere în cadrul unei pagini. Cu puțină neatenție din partea voastră atacatorul ar putea accesa astfel de informații sensibile din cadrul sistemului. Nu includeți fișiere străine. PHP poate "include" fișiere aflate pe alte servere decît cel care rulează dacă setarea URL *fopen wrappers* este activată în *php.ini*. În exemplul de mai sus, un atacator ar fi putut accesa adresa:

http://site.md/fisier.php?file=http://www.rau.md/scriptultau.php

pentru a include în fisier un script localizat pe alt server și astfel obține acces către toate resursele sistemului la care are acces PHP, putînd rula comenzi de sistem, afișa informații confidențiale sau sterge baza de date. Nu permiteți includerea fișierelor din altă parte decît de pe serverul vostru. Setați *allow\_url\_fopen=OFF* în *php.ini*.

**4. Formulare**

Folosiți metoda *POST* în formulare atunci cînd informația din acestea urmează să fie introdusă în baza de date. Dacă variabilele globale sunt OFF în *php.ini* sau metoda de transmitere a formularului este *GET*, un utilizator rău intenționat ar putea accesa adresa dvs. și ar introduce comentariul lui (exemplu "blabla") în baza voastră de date fără să treacă propriuzis prin site. Așa vă puteți trezi că nu mai aveți spațiu pe server iar în baza de date sunt cîteva milioane de comentarii care spun același lucru "blabla".

Verificarea provenienței cererilor către server este foarte importantă și în alt caz: formularele de loghin. Cineva care știe numele de utilizator ar putea încerca să vă găsească parola foarte usor. În acest caz va trebui să puneți o protecție suplimentară care să nu permită mai mult de X încercări consecutive eșuate de logare pentru un nume de utilizator. Această problemă se rezolva ușor folosind sesiunile. În momentul cînd cineva încearcă să trimita numele și parola din formular putem seta o variabilă de sesiune *$\_SESSION['login\_count']* care să țină minte numărul de încercări. Cînd valoarea acesteia trece de 3 (încercări nereușite) nici măcar nu mai interogați baza de date pentru a verifica datele trimise. Variabila de sesiune va rămîne în memorie cît timp browserul este deschis și sesiunea activă (opțiunea implicită a PHP de menținere a sesiunilor active este de o oră). Dacă atacatorul așteaptă o ora sau își închide browserul, sesiunea va fi închisă și va putea de alte X ori să se logheze. De cele mai multe ori acestă măsură de siguranta este suficientă pentru a preveni încercările de aflare a parolelor.

**5. Extensii**

O practică obișnuită este de a acorda extensia *.inc* fișierelor care conțin biblioteci de funcții ce urmează a fi incluse și folosite în cod. PHP nu parsează fișierele cu extensia *.inc* și dacă acestea sunt apelate direct ele sunt trimise plain text către browser. Nu puneți informații sensibile (precum nume și parola) în fișiere cu extensia *.inc*, *.txt*, sau *.html* care pot fi accesate și văzute. Folosiți pentru aceste fișiere extensia *.php* care, dacă sunt accesate direct, vor fi rulate fără să afișeze informațiile conținute în ele.

**6. Comanda CHMOD- setarea atributelor**

Comanda CHMOD (abreviere pentru **ch**ange **mod**e permissions of a file) este folosită pentru a schimba modul de acces (de permitere) a fișierelor și directoarelor de pe server. Setarea poate afecta modul cum poate fi citit, sau cum se poate executa un fișier pe server. De exemplu, dacă aveți un fișier *.php* care trebuie să execute o comandă de scriere pe server într-un fișier *.txt* trebuie să îi dați dreptul să poata fi executat iar la fișierul *.txt* trebuie să îi dați dreptul de a se putea scrie în el. Aveti 3 tipuri de acces - *OWNER* - *GROUP* - *WORLD* - fiecare cu cîte 3 setări - *READ* - *WRITE* - *EXECUTE* -  
Este bine ca la *WORLD* să nu dați decît acces la citire (pentru a putea accesa paginile de pe site) dar nu și drepturi de scriere sau execuție (pentru că nu doriți ca oricine să vă scrie ce vrea în respectivul fișier *.txt*)  
În programul TOTAL COMMANDER (pe care eu îl folosesc și pentru FTP) setarea atributelor pe server se face din meniul FILES - CHANGES ATTRIBUTES. Fiecare program de FTP are în meniul lui setare pentru atributele fișierelor de pe server.

**7. Sesiuni**

Dacă aveți o secțiune de administrare pe site, unde accesul este restrictionat doar la membrii de exemplu, este bine ca în toate paginile din această secțiune să încludă o pagină de verificare a accesului. Această pagina va verifica la fiecare accesare dacă utilizatorul este înregistrat și are acces pe pagina respectivă și permite rularea paginii doar dacă utilizatorul este înregistrat. Fără această verificare, un utilizator ar putea accesa paginile din secțiunea de administrare fără să treacă prin formularul de înregistrare. După autentificarea propriu-zisă, vom folosi variabile de sesiune pentru a păstra în memorie cîteva informații despre autentificare, pentru a le verifica mai tîrziu, atunci cînd accesăm alte pagini din cadrul secțiunii de administrare. Pornim întîi sesiunea după care trecem la salvarea informațiilor în ea ca în exemplul de mai jos:

|  |  |
| --- | --- |
| 1  2  3 | session\_start();  $\_SESSION ['nume\_admin'] = $\_POST  ['nume'];  $\_SESSION ['parola\_encriptata'] =  $parolaEncriptata; |

Pe lîngă acestea, pentru o și mai mare siguranță, vom salva id-ul sesiunii în altă variabilă. Toate sesiunile au un id unic, un string care seamănă cu rezultatul unei criptări MD5:

|  |  |
| --- | --- |
| 1 | $\_SESSION ['key\_admin'] = session\_id(); |

Cu autentificare făcută, spunem scriptului să încarce prima pagină din secțiunea de administrare:

|  |  |
| --- | --- |
| 1 | header ("location: admin.php"); |

În continuare pentru a împiedica accesul neautorizat la paginile din această secțiune, scriem un mic script de verificare a datelor sesiunii înainte de a încărca orice pagină din secțiunea de administrare:

|  |  |
| --- | --- |
| 1  2  3  4  5 | session\_start();   if ($\_SESSION ['key\_admin'] != session\_id  ()) {   print 'Acces neautorizat!';   exit;  } |

**8. Loguri de acces**

Pentru o verificare ulterioară a persoanelor care încearcă să intre în paginile restrictionate de pe site, este bine să salvăm în baza de date (sau într-un fișier text) numele, parola criptata, ora, data, semnătura browserului și ip-ul. Așa veți știi cine a încercat să vă spargă site-ul și puteți crea un script care sa verifice de exemplu ip-ul și dacă corespunde cu unul din cele care sunt blocate să nu aibă acces la nici una din pagini (chiar dacă găsește userul și parola corecte)

**9. Criptarea parolelor cu MD5**

Parolele care sunt salvate în baza de date este recomandat să le țineți criptate, astfel dacă cineva încearcă să citeasca parolele le va vedea criptate.

|  |  |
| --- | --- |
| 1 | INSERT INTO admin VALUES  ("administrator", md5("parola")); |

Criptarea folosind md5 nu este reversibila (și astfel nici dvs., nici altcineva nu o va putea afla chiar dacă are acces la baza de date).

**10. Pagina index.php în subdirectoare**

Dacă aveți mai multe directoare și subdirectoare pe server este recomandat pentru a preveni accesul la datele aflate în aceste directoare să introduceți în fiecare director o pagină numită *index.html* sau *index.php* care să facă redirectarea automat către pagina principală a site-ului. Astfel minimizati riscul să intre cineva și să vadă tot ce este în aceste subdirectoare.

**11. Fișierul .htaccess**

Pentru a bloca accesul la un anumit director (sau chiar la tot site-ul) puteti crea un fișier numit *.htaccess* (cu punct înainte) în care să introduceți ip-ul care doriți să îl blocați (sau care să aibă acces)  
*Order Deny,Allow*  
*Allow from all*

În exemplul de mai sus, toată lumea are acces. Dacă de exemplu ip-ul 192.168.1.1 nu doriți să aibă acces la acel director scrieți  
*Deny from 192.168.1.1*

Atenție, să nu vă blocați singuri ip-ul la site, că altfel trebuie să luați legătura cu administratorul serverului să steargă fișierul ca să puteți intra din nou pe site, sau sa scrieti prin FTP un nou fișier .htaccess. (la subdirectoare ștergeți directorul sau salvați un alt fișier .htaccess) În cazul în care după de ați pus fișierul pe server nu îl vedeți, el este acolo dar serverul seteaza acest nume de fișier ca fișier ascuns.

**12. Roboții de căutare**

Motoarele de căutare pot indexa tot ce se află la voi pe site. Dacă doriți ca un fișier sau director să nu fie indexat de către motoarele de căutare, trebuie să creați un fișier numit *robots.txt* care să se afle în directorul principal de pe site (nu în subdirectoare). În el scrieți următoarele comenzi:  
**# robots.txt for http://web.ceiti.md**  
**User-agent: \***  
**Disallow: /cache/**  
**Disallow: /admin**  
**disallow: /search**  
unde pentru fiecare director sau fișier se specifică locația și numele. Pentru mai multă siguranță, am găsit pe web un mic script care verifică tipul de browser și dacă nu este un browser cunoscut nu permite accesul (dacă cineva încearcă să citească paginile cu programe spion sau gen web offline).

Cum am specificat și la începutul acestui articol, cele spuse mai sus nunt doar cateva indicii și idei pentru a vă putea proteja mai bine paginile.

**Preluarea datelor din formulare**

Înainte de a vorbi despre preluarea datelor dintr-un formular, trebuie în primul rând să cunoașteți sintaxa HTML pentru crearea unui formular.

Un formular este delimitat de elementul **<form>** care conține alte câteva elemente numite “controale”, care au o varietate de metode de a prelua informații. Fiecare element din formular are un nume cât și o valoare, astfel încât datele transferate pentru procesare să fie sub forma unor perechi nume/valoare.

**Elementul <form>**

|  |  |
| --- | --- |
| 1  2  3  4  5 | <form [action=url] [method=get/post] [name=nume] >    Elementele formularului    </form> |

**Elementul <input>**

|  |  |
| --- | --- |
| 1  2  3  4  5 | <input [type=text|password|checkbox|radio|submit|image|reset|button|hidden|file] [name=nume]  [value=valoare] [checked] [disabled] [readonly] [size=latime] [maxlength=cuvinte\_maxime] [src=url]  [alt=altText] [usemap=url] [align=left|center|right|justify] [tabindex=numar] [accesskey=keyCombo]  [onfocus=script] [onblur=script] [onselect=script] [onchange=script] [accept=set\_caractere]  [events]> |

Acest element **<input>** este cel mai important în utilizarea formularelor.

**Explicarea valorilor atributului *type* ale elementului <input>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Descriere** | **Valoare** | **Exemplu** | **Rezultat** |
| Caseta text | text | |  |  | | --- | --- | | 1 | <input type="text" name="nume" value="valoare"> | |  |
| Casete de introducere a parolei | password | |  |  | | --- | --- | | 1 | <input type="password" name="nume" value="valoare"> | |  |
| Elemente ascunse | hidden | |  |  | | --- | --- | | 1 | <input type="hidden" name="nume" value="valoare"> | |  |
| Casete de validare | checkbox | |  |  | | --- | --- | | 1 | <input type="checkbox" name="nume" value="valoare"> | |  |
| Butoane radio | radio | |  |  | | --- | --- | | 1 | <input type="radio" name="nume" value="valoare"> | |  |
| Fisiere incluse | file | |  |  | | --- | --- | | 1 | <input type="file" name="nume" value="valoare"> | |  |
| Buton de trimitere | submit | |  |  | | --- | --- | | 1 | <input type="subbbmit" name="Trimite" value="Trimite"> | |  |
| Butoane personale | button | |  |  | | --- | --- | | 1 | <input type="button" name="Buton" value="Buton"> | |  |
| Imagini | image | |  |  | | --- | --- | | 1 | <input type="image" name="buton" src="poza\_buton.gif"> | |  |
| Butoane de resetare | reset | |  |  | | --- | --- | | 1 | <input type="reset" name="Reseteaza" value="Reseteaza"> | |  |

**Elementul <select>**

Acest element este folosit pentru crearea unei liste de opțiuni, fie ca un meniu care se desfășoară, fie ca o casetă cu listă.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <select [name=nume] [size=latime] [multiple] [disabled] [tabindex=numar] [onfocus=script] [onblur=script]  [onchange=script] [events]>    Elementele din select    </select> |

Fiecare din opțiunile din listă reprezintă un element **<option>**.

|  |  |
| --- | --- |
| 1 | <option [selected] [disabled] [value=valoare] [core] [international] [events]>Nume</option> |

**Exemplul 1:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <select name="test">  <option value="optiune 1">Optiune 1</option>  <option value="optiune 2">Optiune 2</option>  <option value="optiune 3">Optiune 3</option>  <option value="optiune 4">Optiune 4</option>  </select> |

Top of Form



Bottom of Form

**Exemplul 2:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <select name="test" multiple size="3">  <option value="optiune 1">Optiune 1</option>  <option value="optiune 2">Optiune 2</option>  <option value="optiune 3">Optiune 3</option>  <option value="optiune 4">Optiune 4</option>  <option value="optiune 5">Optiune 5</option>  <option value="optiune 6">Optiune 6</option>  </select> |

Top of Form



Bottom of Form

**Elementul <textarea>**

Acest element este asemanator cu cel text numai că aici se poate tasta într-o secțiune mult mai mare decat în cazul text.

|  |  |
| --- | --- |
| 1  2 | <textarea [name=nume] [rows=nr\_randuri] [cols=nr\_coloane] [disabled] [readonly] [tabindex=numar]  [onfocus=script] [onblur=script] [onselect=script] [onchange=script] [events]> </textarea> |

**Exemplu textarea:**

|  |  |
| --- | --- |
| 1 | <textarea name="nume" cols="40" rows="10">Text initial</textarea> |

Top of Form



Bottom of Form

**Exemplu 1:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <form action="formular.php" method="post">    Nume: <input type="text" name="nume" value="" size="30" maxLength="15"> <br><br>  Prenume: <input type="text" name="prenume" value=""> <br><br>  Varsta:  sub 16 <input type="radio" name="varsta" value="sub16"> | peste 16                  <input type="radio" name="varsta" value="peste16"> <br><br>  Parola dorita: <input type="password" name="parola" value=""> <br><br>    <input type="submit" name="Trimite" value="Trimite">  <input type="reset" name="Reseteaza" value="Reseteaza">    </form> |

Top of Form

|  |  |
| --- | --- |
| Nume: |  |
| Prenume: |  |

Vârsta:  
       sub 16 ani  
       peste 16 ani

|  |  |
| --- | --- |
| Parola dorită: |  |

Bottom of Form

Puneți codul într-un fișier **formular.html**, salvați și apoi vizualizați în browser accesând http://localhost/formular.html

Am vorbit până acum despre formulare, cum să le creăm și ce elemente au.

Hai să ne aducem aminte ca în interiorul etichetei **<form>** avem puse atributele ***action*** și ***method*** (care poate fi POST sau GET). În atributul ***action*** se pune calea către scriptul PHP care prelucrează datele iar în ***method*** se pune metoda prin care se vor transmite datele atunci când butonul **“Trimite”** este apăsat.

**POST** - această metodă face ca datele trimise prin formular să nu fie vizibile utilizatorului, să fie trimise în spatele paginii web.

**GET** - prin această metodă, datele trimise prin formular sunt vizibile în URL (URL este adresa ce este afișată în browser).

Acum că am creat formularul, avem pagina **formular.html**, haideți să cream și pagina prin care arătăm datele trimise prin formular.

Creați o pagină **formular.php** și introduceți codul următor:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <?php    echo $\_POST['nume'];    ?> |

Salvați și testați în browser accesând http://localhost/formular.html, completați câmpul “Nume”, și apoi apăsați butonul “Trimite”.

Observați că în pagina **formular.php** ne este afișat numele introdus în câmpul “Nume” din pagina anterioară, și anume **formular.html**.

Adăugăm în continuare variabilele **$\_POST** corespunzătoare formularului nostru, și apoi testăm din nou.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php    echo 'Nume:      '.$\_POST['nume'].'      <br>        Prenume:   '.$\_POST['prenume'].'   <br>        Varsta:    '.$\_POST['varsta'].'    <br>        Parola:    '.$\_POST['parola'].'    <br>';    ?> |

După cum vedeți, ne folosim de valoarea **POST**, iar datele nu sunt afișate decât dacă le definim noi **$\_POST['nume']** în pagina **formular.php**.

Numele din interiorul variabilei **$\_POST** este numele din interiorul elementelor din formular.

De exemplu, în formularul nostru avem input-ul de la nume așa:

**<input type="text" name="nume" value="" size="30" maxLength="15">**

Observați că în câmpul **name** valoarea acestuia este ***nume***. Acesta se plasează în variabila **$\_POST**, în cazul nostru **$\_POST['nume']**.

În input-ul pentru prenume avem **<input type="text" name="prenume" value="">** iar acesta se plasează în **$\_POST**așa: **$\_POST['prenume']**, ... și tot așa.

Observați câmpul **value** care nu are nimic definit. Nu este nici o greseala, veți învăța mai târziu de ce am lăsat gol acel câmp.

Am cam terminat cu metoda **POST**, acum haideți să lucrăm și să ne familiarizăm și cu metoda **GET**.

În același formular din pagina **formular.html**, numai că în eticheta **<form>**, în loc de **method="post"**, vom pune **method="get"**.

Apoi în pagina **formular.php**, în loc de variabilele **$\_POST**, vom pune variabile **$\_GET**.

În caz că nu ați înteles, deschideți pagina **formular.html**, modificați linia

**<form action="formular.php" method="post">** în **<form action="formular.php" method="get">**

Apoi deschideți pagina **formular.php** și modificați în:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php    echo 'Nume:     '.$\_GET['nume'].'     <br>        Prenume:  '.$\_GET['prenume'].'  <br>        Varsta:   '.$\_GET['varsta'].'   <br>        Parola:   '.$\_GET['parola'].'   <br>';    ?> |

Observați că rezultatul este același în pagina web, numai că informațiile formularului sunt postate și în adresa din browser (url) sub forma: http://localhost/formular.php?nume=ciobanu&prenume=andrei&varsta=peste16& parola=altaparola&Trimite=Trimite

De recomandat este să folosiți metoda **POST**, pentru că este mult mai sigură.

**Exemplu 2:**  
Formularul (**form.html**)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Formular</title>  </head>  <body bgcolor="#006699" text="#fff">  <form  method="post" action="form.php">    <p>Nume: <input type="text" name="nume">    </p>    <p>Prenume: <input type="text" name="prenume">    </p>    <p>Virsta: <input type="text" name="virsta">    </p>    <p>Sex:<br>      <input type="radio" name="sex" value="Masculin"> Masculin<br>      <input type="radio" name="sex" value="Feminin"> Feminin</p>    <p>Limbi cunoscute:<br />      <input name="limba1" type="checkbox" value="L.Romana">   L.Româna<br>      <input name="limba2" type="checkbox" value="L.Rusa">     L.Rus ă<br>      <input name="limba3" type="checkbox" value="L.Engleza">  L.Engleza<br>      <input name="limba4" type="checkbox" value="L.Franceza"> L.Franceza    </p>    <p>     Universitatea absolvită:      <select name="universitate" size="1">        <option value="USM">Universitatea de Stat din Moldova</option>        <option value="UST">Universitatea de Stat din Tiraspol</option>        <option value="UTM">Universitatea Tehnică din Moldova</option>        <option value="ASEM">Academia de Studii Economice</option>      </select>    </p>    <p>Comentarii:<br />    <textarea name="comentarii" cols="45" rows="8"></textarea>    </p>     <p>      <input type="submit" name="submit" value="Expediază">    </p>  </form>  </body>  </html> |

Top of Form

|  |  |
| --- | --- |
| Nume: |  |
| Prenume: |  |
| Virsta: |  |

Sex:  
       Masculin  
       Feminin

Limbi cunoscute:  
       L.Română  
       L.Rusă  
       L.Engleza  
       L.Franceza

Universitatea absolvită:                           

Comentarii:  
      



Bottom of Form

Scriptul (**form.php**)care preia datele din formular pentru afișare:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?php  echo "Numele: ".$\_POST['nume'];  echo "<br />Prenumele: ".$\_POST['prenume'];  echo "<br />Vîrsta: ".$\_POST['virsta'];  echo "<br />Sex: ".$\_POST['sex'];  echo "<p>Limbi cunoscute: ";  echo "<br />Limba1: ".$\_POST['limba1'];  echo "<br />Limba2: ".$\_POST['limba2'];  echo "<br />Limba3: ".$\_POST['limba3'];  echo "<br />Limba4: ".$\_POST['limba4'];  echo "</p> <p>";  echo "Universitatea absolvită: ".$\_POST['universitate'];  echo "</p>";  ?> |

Nu orice dată introdusă în formular poate fi acceptată și prelucrată de scriptul apelat din pagina Web cum ar fi un cîmp necompletat, sau informații care nu respectă formatul specificat (de exemplu cel al poștei electronice). De aceea, este necesară validarea datelor înainte ca acestea să fie prelucrate și eventual stocate în tabelele componente ale bazelor de date aflate pe server. Validarea datelor care sunt trimise serverului Web se poate face în două moduri:

* ***Validare pe calculatorul client,*** înainte ca datele să fie trimise serverului; validarea se poate realiza prin intermediul unui  script scris într-un limbaj de programare pentru client (JavaScript și VBScript);
* ***Validare pe calculatorul server,*** în acest caz, validarea se realizează după ce datele au ajuns pe server, dar înainte de prelucrarea și utilizarea lor efectivă (introducerea lor într-o bază de date, de exemplu), și se face de către o secvență de cod care poate să facă parte chiar din scriptul apelat.

Validarea la client are avantajul că reduce încărcarea serverului. Pe de altă parte, dacă validarea presupune compararea datelor introduse de client cu datele stocate într-o bază de date, operațiune necesară la logare de exemplu, este clar că aceasta nu poate fi făcută decît pe server.

**Exerciții:**

1. Scrieți codul PHP care va crea calculatorul simplu ce va îndeplini funcțiile de bază, cum ar fi adunarea, scăderea, împărțirea, înmulțirea și de asemenea pătratul unui număr și rădăcina pătrată dintr-un număr.

2. Scrieți codul PHP care va prelua datele dintr-un formular și le va salva într-un fișier text.

**Validarea datelor din formular trimise serverului**

Pentru a fi siguri ca utilizatorul a completat cîmpurile trebuie să adăugăm script-ului partea de verificare.

**Exemplu:** folosim același formular dar adăugăm partea de verificare

|  |  |
| --- | --- |
| 1  2  3  4 | <?php  $nume=$\_POST["nume"];  if ($nume=="") { echo "Cîmpul nume nu este completat"; }  else { echo $nume; }  ?>; |

Partea de verificare poate avea mai multe condiții simultane.

**Exemplu:** Același formular dar partea de verificare are doua condiții legate prin operatorul **OR** sau **||**

|  |  |
| --- | --- |
| 1  2  3  4  5 | <?php  $nume=$\_POST["nume"];  if (($nume=="") || (strlen($nume)>5)) {echo "Campul nume nu  este completat corect";}  else {echo $nume;}  ?> |

Validarea selectării elementelor **input** de tip buton *radio* și *checkbox*, precum și a elementului *select* se face prin intermediul utilizării funcției **isset().**Aceastăfuncție întoarce valoarea **True** dacă variabila transmisă ca argument există.

**Exemplu:** Preluarea valorii unui control radio

**radio.html**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <form  action="radio.php" method="GET"><br />  Ce tip de muzică vă place?  Pop <input type="radio" name="muzica"  value="pop"><br />  Rock <input type="radio" name="muzica"  value="rock"><br />  <input type="submit" value="Trimite">  </form> |

**radio.php**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?php  if (isset($\_GET['muzica']))  echo "Va place muzica ".$\_GET['muzica'];  else  echo "<a href="radio.html">Selectati o optiune</a>";  ?> |

**Exemplu:** Preluarea valorii unui control checkbox

**checkbox.html**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <form  action="checkbox.php" method="GET">  Ce tip de muzica  va place?  Pop <input type="checkbox" name="pop" value="pop"><br>  Rock <input type="checkbox" name="rock"  value="rock"><br>  <input type="submit" value="Trimite">  </form> |

**checkbox.php**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php  if (isset($\_GET['pop']) or isset($\_GET['rock'])) {  echo "Eu prefer muzica: "  if (isset($\_GET['pop'])) echo $\_GET['pop'];  if (isset($\_GET['rock'])) echo  $\_GET['rock'];  }  else echo "<a href="checkbox.html">Nu ați selectat nici o optiune</a>";  ?> |

**Funcția empty() -** verifică dacă o anumită variabilă are o valoare diferita de **0**, respective de un șir fără conținut. Dacă returnează TRUE, variabila nu are valoare, în caz contrar va returna **FALSE**. **Funcția empty() –**reprezintă opusul lui **isset()**.

Acesta este scriptul pentru**verificarea completării tuturor cîmpurilor unui formular**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <html>  <head>  <title>Trimiterea datelor</title>  </head>  <body>  <?php  $nume = $\_POST['nume'];  $prenume = $\_POST['prenume'];  if (empty($nume)) {  echo "Va rugam introduceti numele";  }  if empty($prenume)) {echo "Va rugam introduceti prenumele"; }  ?>  </body>  </html> |

Functia **is\_numeric()** returnează **TRUE** dacă variabila trimisă are o valoare numerică validă, **FALSE** în caz contrar.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | <html>  <head>  <title>Trimiterea datelor</title>  </head>  <body>  <?php  if ($\_POST['submit']) {  if (!$\_POST['nume']) {  echo "Nu ati completat numele!<br/>";  echo "<a href='text.php'>Inapoi</a>";  }  else  {$nume = $\_POST['nume']; echo  "Numele este $nume <br>";}  if (!$\_POST['virsta'])  {  echo "Nu ati completat virsta!<br/>";  echo "<a href='text.php'>Inapoi</a>";  }  else {  if (!is\_numeric($\_POST['virsta'])) {  echo "Virsta trebuie sa fie o valoare numerica!<br/>";  echo "<a href='text.php'>Inapoi</a>";}  else  {$virsta = $\_POST['virsta'];  echo "virsta  este $virsta";}  }  }  else {  ?>  <form method="post" action="<?php echo $\_SERVER['PHP\_SELF']; ?>">  Numele <input type="text" name="nume" ><br>  Virsta <input type="text" name="virsta"  ><br>  <input type="submit" name="submit" value="Trimite">  </form>  <?php  }  ?>  </body>  </html> |

Forma simplă a funcției ***ereg()*** preia două argumente: un șir care conține o expresie regulată și un șir subiect. Funcția returnează *true* dacă expresia regulată corespunde unui sub-șir al șirului subiect; în caz contrar, returnează *false*.

**Semnificația caracterelor ce se pot afla în șablon**

^ - Început de șir  
$ - Sfîrșit de șir  
\* - zero sau mai multe caractere  
+ - unul sau mai multe caractere  
? - zero sau un caracter  
{x,y}, unde x și y sunt doi întregi reprezentînd limita inferioară și superioară a numărului de caractere necesare. x trebuie întotdeauna definit.  
( ) - Secvențe de caractere  
**|** - operatorul ŞI (OR)  
**.** (punct) - reprezintă orice caracter  
[ ] - orice caracter din intervalul specificat  
^ în [ ] - orice altceva decît caracterele din interval  
^.[$()|\*+?{\ - nu trebuiesc precedate de backslash (\) în expresii regulate dacă sunt în interiorul parantezelor pătrate [ ]

Un exemplu mai complicat:  
^.[a-z]{1,2}\_?[0-9]\*([1-6]|[a-f])[^1-9]a+$

Exemplul de mai sus se traduce prin:  
Un șir care începe cu oricare 2 caractere... ^.  
urmat de 1 sau 2 litere minuscule... [a-z]{1,2}  
urmat de un underscore opțional... \_?  
urmat de zero sau mai multe cifre... [0-9]  
urmat de fie un număr între 1 și 6 sau o literă minusculă cuprinsă între a și f... \*([1-6]|[a-f])  
urmat de 2 caractere care nu sunt cifre... [^1-9]  
urmat de unul sau mai multe litere "a" la sfîrșitul șirului... a+$

**Echivalențe:**

\* - {0,}  
+ - {1,}  
? - {0,1}

***Exemplu:*** Validarea adresei de e-mail introduse în formular utilizînd funcția ***ereg()***.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Validarea adresei de e-mail</title>  </head>  <body>  <?php  if (isset($\_POST['submit'])) {  if(!ereg("/^[\_\.0-9a-zA-Z-]+@([0-9a-zA-Z][0-9a-zA-Z-]+\.)+[a-zA-Z]{2,6}$/i",$\_POST['mail']))  echo "Adresa nu este  corecta<br />";  else  echo "Adresa a fost  acceptata";  echo "<br /><a  href=\"email.php\">Inapoi</a>";  }  else{  ?>  <form name="form1" method="post" action="<?php  echo $\_SERVER['PHP\_SELF']; ?>">  Introdu parola  <input type="text"  name="mail"> <br>  <input type="submit"  name="submit">  </form>  <?php  }  ?>    </body>    </html> |

Dacă pe server este instalată o versiune mai nouă decît PHP 5.3.0 atunci scriptul de validare va returna eroarea "**Function ereg() is deprecated**". Funcția ***ereg()*** este anulată iar în locul ei poate fi folosită funcția ***preg\_match()****.*

***Exemplu:*** Validarea adresei de e-mail introduse în formular utilizînd funcția ***preg\_match()***.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Validarea adresei de e-mail</title>  </head>  <body>  <?php  if (isset($\_POST['submit'])) {  if(!preg\_match("/^[\_\.0-9a-zA-Z-]+@([0-9a-zA-Z][0-9a-zA-Z-]+\.)+[a-zA-Z]{2,6}$/i",$\_POST['mail']))  echo "Adresa de e-mail nu este corectă<br />";  else  echo "Adresa de e-mail a fost  acceptata";  echo "<br /><a  href=\"email.php\">Inapoi</a>";  }  else{  ?>  <form name="form1" method="post" action="<?php  echo $\_SERVER['PHP\_SELF']; ?>">  Introdu parola  <input type="text"  name="mail"> <br>  <input type="submit"  name="submit">  </form>  <?php  }  ?>  </body>  </html> |

**Baze de date în aplicații web**

Bazele de date sunt folosite pentru stocarea informațiilor în vederea furnizării ulterioare în funcție de solicitarea primită.

Cel mai popular sistem de baze de date folosit cu PHP este **MySQL**.

**MySQL** este un sistem de baze de date funcțional independent.

În **PHP** există funcții pentru toate operațiile executate asupra bazelor de date MySQL.

**Administrarea bazelor de date**

Administrarea MySQL se poate face din linie de comandă sau folosind browser-ul și accesînd aplicația numită **PHPMyAdmin** scrisă în PHP.

Cele mai uzuale operații cu bazele de date sunt:

|  |  |
| --- | --- |
| **Comanda** | **Semnificatie** |
| CREATE | crează o bază de date sau un tabel |
| DROP | sterge o baza de date sau un tabel |
| INSERT | adauga inregistrari intr-un tabel |
| DELETE | sterge inregistrari dintr-un tabel |
| UPDATE | updateaza inregistrarile dintr-un tabel |
| SELECT | selecteaza un tabel |
| ALTER | alterarea unui tabel |

În MySQL spatiul alocat pe discul serverului este functie de tipul de date. Cateva din tipurile de date folosite in bazele de date MySQL sunt:

|  |  |  |
| --- | --- | --- |
| **Tip** | **Mărime** | **Descriere** |
| CHAR(*lungime*) | lungime bytes | Un câmp cu lungime fixă de la 0 la 255 de caractere. Acceptă o valoare inițială. |
| VARCHAR(*lungime*) | lungime + 1 or 2 bytes | Câmp cu lungime variabilă de la 0 la 65,535 caractere. Acceptă o valoare inițială. |
| TINYTEXT | lungime + 1 bytes | Un șir cu lungime maximă de 255 characters |
| TEXT | lungime + 2 bytes | Un șir cu o lungime maximă de 65,535 caractere. Nu poate defini o valoare inițială. |
| MEDIUMTEXT | lungime + 3 bytes | Un șir cu o lungime maximă de 16,777,215 caractere |
| LONGTEXT | lungime + 4 bytes | Un șir cu o lungime maximă de 4,294,967,295 caractere |
| **Stocare numere** | **Aceste tipuri de date reprezintă numere în diferite forme.** | |
| TINYINT(*lungime*) | 1 byte | Interval de la –128 la 127. Sau de la 0 la 255 unsigned |
| SMALLINT(*lungime*) | 2 bytes | Interval de la –32,768 la 32,767. Sau de la 0 la 65,535 unsigned |
| MEDIUMINT(*lungime*) | 3 bytes | Interval de la –8,388,608 la 8,388,607. Sau de la 0 la 16,777,215 unsigned |
| INT(*lungime*) | 4 bytes | Interval de la –2,147,483,648 la 2,147,483,647. Dacă coloana este declarată UNSIGNED, intervalul este de la 0 la 4,294,967,295 |
| BIGINT(*lungime*) | 8 bytes | Interval de la –9,223,372,036,854,775,808 la 9,223,372,036,854,775,807. Sau de la 0 la 18,446,744,073,709,551,615 unsigned |
| FLOAT(*lungime, decimals*) | 4 bytes | Număr mic cu virgulă (cu zecimale). Deoarece PHP va formata numerele în funcție de calcule, se recomandă să folosești FLOAT fără parametri optionali. |
| DOUBLE(*lungime, decimals*) | 8 bytes | Număr mare cu zecimale |
| DECIMAL(*lungime, decimals*) | lungime + 1 or 2 bytes | Un tip DOUBLE care permite un număr fix de zecimale. Înainte de MySQL 5.0.3, tipul de date DECIMAL a fost stocat ca un șir, ca să nu poată fi folosit cu funcții SQL, cum ar fi SUM() |
| **Stocare dată și timp** | **MySQL stochează data într-un singur format: YYYY-MM-DD** | |
| DATE | 3 bytes | O dată stocată ca YYYY-MM-DD. Intervalul este de la 1000-01-01 la 9999-12-31 |
| DATETIME | 8 bytes | Data și ora, afișat în format YYYY-MM-DD HH:MM:SS |
| TIMESTAMP | 4 bytes | Un *timestamp* (de obicei generat automat de calculator). Poate conține valori ce reprezintă date de la începutul lui 1970 pana in Januarie 2038. Coloanele TIMESTAMP sunt afișate în același format ca și coloana DATETIME. |
| TIME | 3 bytes | Ora în format HH:MM:SS |
| **Stocare liste predefinite** | **MySQL te lasă să stochezi două tipuri de liste predefinite (pot fi vazute ca echivalente pentru radio butonul și caseta de verificare).** | |
| ENUM | 1 or 2 bytes | Acest tip de coloană stochează o singură alegere din lista predefinită, fiecare coloana poate avea una din mai multe valori posibile |
| SET | 1, 2, 3, 4, or 8 bytes | Aceast tip de coloană stochează zero sau mai multe înregistrări din lista predefinită. Ca si ENUM, cu excepția că fiecare coloană poate avea mai mult de o valoare din mai multe valori posibile, sau mai multe înregistrari dintr-o listă predefinită. Lista poate conține cel mult 64 de înregistrări. |
| **Stocare date binare** | **Stochează date binare, ca imaginile.** | |
| TINYBLOB |  | Pâna la 255 bytes |
| BLOB |  | Pâna la 64kB |
| MEDIUMBLOB |  | Pâna la 16MB |
| LONGBLOB |  | Pâna la 4GB |

- Parametri (*lungime, decimal*) sunt opționali, dar pentru motive de performanță, ar trebui specificați pentru a restrictiona câtă informatie sa fie stocata in oricare din coloane. Bazele de date sunt de obicei mai rapide cand lucreaza cu coloane care au marimea fixata.  
- Foloseste VARCHAR pentru elemente de text scurte si TEXT pentru cele mai lungi.  
- Foloseste DECIMAL pentru a stoca valori pt. bani.  
- Pentru data si ora, este indicat sa folositi TIMESTAMP daca doriti sa inregistrati in acel camp un punct fix in timp (*de exemplu data cand o inregistrare a fost introdusa in baza de date*). Folositi DATETIME daca data/ora pot fi setate si schimbate arbitrar.  
- La tipul TIMESTAMP data si ora este setata automat cu timpul curent cand se face INSERT sau UPDATE, daca nu se specifica o valoare pentru acel camp.  
- Nu este o buna idee sa folosesti date binare. Solicita prea mult baza de date, si nu se pot afisa imagini direct din baza de date.

Pentru ca baza de date să fucționeze mai bine coloanelor li s-au adaugat modificatori de coloana.

Tipul de date intregi incep de la valori negative la pozitive. Daca se adauga optiunea **UNSIGNED**, care este un modificator de coloana, nu vor mai fi valori negative ci vor incepe de la 0.

Alti modificatori sunt:  
**AUTO\_INCREMENT** functioneaza cu orice tip intreg. La fiecare rand nou adaugat in baza de date numarul asociat va fi incrementat.  
**NULL** înseamna fără valoare (diferit de spatiu sau zero).  
**NOT NULL** inseamna ca orice inregistrare va fi considerata ceva.  
**PRIMARY KEY** este rolul primei coloane din tabel, totodata reprezentand elementul de referinta pentru fiecare linie.

**Utilizarea extensiei MySQLi**

PHP a anunțat în 2011 că extensia **MySQL** va fi depreciată începând cu versiunea PHP 5.5, urmând să fie scoasă din funcție mai apoi. În locul acestei extensii ni se oferă 2 alternative:

1. **MySQLi** (‘i’-ul din coadă vine de la ‘**improved**’)
2. **PDO** (PHP Data Objects).

Modificarea unui proiect vechi din **MySQL** în **MySQLi** sau **PDO** poate fi destul de greu și deci nu trebuie să mai scriem proiecte noi folosind vechea extensie.

În urma acestei modificări avem și anumite avantaje. De exemplu, **PDO** este orientat pe obiecte, iar **MySQLi** vine cu suport atât pentru stilul procedural de programare cât și pentru stilul orientat pe obiecte. Ambele vin cu suport pentru prepared statements, tranzacții, și așa mai departe. Și o să vă întrebați “Dar ce îmi trebuie mie prepared statemenets?”, împortant este că aceste prepared statements vă protejează aplicația împotriva bine cunoscutului procedeu **SQL Injection**.

Și acum mă veți întreba “Până la urmă eu ce trebuie să folosesc? **MySQLi** sau **PDO**?”. Răspunsul scurt ar fi acesta: “Pe care vrei”. Mie îmi place să folosesc **MySQLi**, dar dacă aveți nevoie de suport pentru mai multe tipuri de baze de date cum ar fi: **PostgreSQL, SQLite, MySQL, Oracle, ODBC, MS SQLServer & Azure, Firebird, Informix, IBM DB2, Sybase, Cubrid, 4D**- **PDO** este soluție ideală.

Documentația **PHP** descrie **MySQLi** ca fiind opțiunea preferată recomandată de **MySQL** pentru proiecte noi.

În continuare am să arăt ambele stiluri de programare utilizînd extensia MySQLi:

[MySQLi procedural](https://web.ceiti.md/lesson.php?id=7#s1)

[MySQLi obiect orientat](https://web.ceiti.md/lesson.php?id=7#s2)

**1. MySQLi procedural**

**Conectarea la MySQL**

De cîte ori vom lucra cu MySQL va trebui să ne conectam la baza de date folosind o anume sintaxă. Vom salva această secvență de cod cu numele **conexiune.php** putand fi folosita ulterior in mai multe scripturi utilizand functia *include()*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | /\* urmeaza fiesierul conexiune.php \*/   <?php   $hostname="localhost";   $username="root";   $password="";   $database="colegiu";   $conexiune=mysqli\_connect($hostname, $username, $password) or die ("Nu mă pot conecta la baza de date");   mysqli\_select\_db($conexiune, $database) or die ("Nu găsesc baza de date");   ?> |

**mysqli\_connect()** este funcția prin care ne conectam la baza de date cu urmatorii paramatrii: numele serverului, nume utilizator si parola pentru conectarea la baza de date.

Variabila *$conexiune* va avea o valoarea TRUE sau FALSE functie de rezultatul conectarii la serverul MySQL folosind functia *mysqli\_connect()*, iar in cazul esuarii va afisa mesajul continut de constructia *die()*.

**mysqli\_select\_db** este functia care stabileste baza de date la care ne vom conecta, avand ca parametrii numele bazei de date si identificatorul de acces la conexiunea catre serverul MySQL.

In mod similar variabila *$bazadate* folosind functia *mysqli\_select\_db* va afisa in cazul negasirii bazei de date mesajul continut de *die*.

Puteti modifica variabilele din fisierul *conexiune.php* in functie de configurarile propriei baze de date:  
*$hostname*=adresa serverului, de cele mai multe ori este *localhost* dar pentru siguranta intrebati administratorul serverului  
*$username*=username-ul de conectare la baza de date  
*$password*= parola de conectare la baza de date  
*$database*=numele bazei de date

Dupa ce operatiile dorite au fost executate cu baza de date va trebui sa inchidem conexiunea folosind functia **mysqli\_close()** avand ca argument datele de acces la serverul MySQL.

|  |  |
| --- | --- |
| 1 | mysqli\_close($conexiune); |

Anterior am salvat intr-un fisier php numele utilizator si parola care poate parea un lucru riscant. Codul php este interpretat de server si doar rezultatul este afisat, deci datele continute nu vor fi afisate in browser vizitatorului.

**Crearea tabelelor**

Bazele de date contin **tabele**. Aceste tabele sunt similare cu cele din HTML continand rinduri si coloane. Fiecare diviziune se numeste **celula** sau **camp** si are anumite propritati.  
Fiecare tabel din baza de date are un nume sau titlu.  
Fiecare coloana din tabel are un nume sau titlu.

Sintaxa pentru crearea tabelului este:

|  |  |
| --- | --- |
| 1 | CREATE TABLE nume\_tabel (coloana\_1, coloana\_2,..., coloana\_n); |

Coloanele vor fi separate prin virgula iar fiecare coloana va avea un nume si tip.

Pentru a crea un tabel mai intai includem in script fisierul anterior *conexiune.php* de conectare la baza de date si apoi folosim functia **CREATE TABLE**.

**NOTA**: In urmatoarele exemple nu voi mai mentiona ca includ acest fisier dar se va subintelege acest lucru de cate ori voi face o operatie cu baza de date.

**mysqli\_query** este o functie folosita pentru interogarea unei baze de date. Foloseste parametrii: cererea emisa serverului MySQL si identificatorul de acces la baza de date.

**Exemplu**: cream tabelul **elev** cu 3 coloane. Prima coloana este **id** de tip *PRIMARY KEY* de 3 caractere, a doua **nume** de 20 caractere tip *varchar* iar a treia **prenume** de 40 caractere tot *varchar*. Inainte de a crea tabelul il stergem daca el exista deja.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\* urmeaza fisierul create.php \*/   <?php   include ("conexiune.php");   $delete="DROP TABLE IF EXISTS elev";   mysqli\_query($conexiune, $delete);   $table="CREATE TABLE elev (   id int(3) NOT NULL AUTO\_INCREMENT,   nume varchar(20) NOT NULL ,   prenume varchar(40) NOT NULL ,   PRIMARY KEY (id)) TYPE=MyISAM";   if(mysqli\_query($conexiune, $table)){   echo "Tabelul a fost creat";   }   else{   echo "Tabelul nu a fost creat";   }   mysqli\_close($conexiune);   ?> |

Functie de valoarea *mysqli\_query* este afisat succesul sau insuccesul crearii tabelului.

**Inserarea datelor**

**INSERT** este comanda pentru introducerea datelor in baza de date. Cel mai frecvent mod de introducere a datelor este preluarea lor dintr-un formular adecvat structurii bazei de date.

Sintaxa pentru introducerea datelor in tabelul bazei de date este:

|  |  |
| --- | --- |
| 1 | INSERT INTO nume\_tabel (coloana\_1, coloana\_2,..., coloana\_n) values ('valoare\_1','valoare\_2',...,'valoare\_n'); |

**Exemplu**: introducem date intr-un formular cu 2 cîmpuri *nume* si *prenume* şi apoi cu scriptul *insert.php* datele se introduc in baza de date.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <!-- urmează fişierul adăugare.html -->   <html>   <head><title>Formular</title>   </head>   <body>   <b>Adaugare inregistrari</b>   <form method="POST" action="insert.php">   Nume: <input type="text" name="nume"><br>   Prenume: <input type="text" name="prenume"><br>   <input type="submit" value="Trimite">   </form>   </body>   </html> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | /\* urmeaza fisierul insert.php \*/  <?php  include ("conexiune.php");  $nume=$\_POST['nume'];  $prenume=$\_POST['prenume'];  $query="INSERT INTO elev (nume, prenume) VALUES ('$nume','$prenume')";  if (!mysqli\_query($conexiune, $query)) {  die(mysqli\_error());  } else {  echo "datele au fost introduse";  }  mysqli\_close($conexiune);  ?> |

In functie de valoarea *mysqli\_query* este afisat un mesaj privind introducerea datelor in tabel.

In tutorialul de PHP am discutat ca asociem un nume (*name*) fiecarui camp din formular. La apasarea butonului de trimitere datele sunt preluate de un script care poate identifica datele functie de campul in care au fost introduse.

In exemplul de mai sus datele introduse in campul *nume* din formular sunt preluate folosind variabila **$\_POST** (pentru ca metoda folosita in formular a fost POST).

**Selectarea datelor**

Proprietatea **SELECT** poate fi utilizata intr-o comanda **mysqli\_query** pentru a alege anumite informatii din tabelul bazei de date.

Sintaxa pentru selectarea datelor din tabelul bazei de date este:

|  |  |
| --- | --- |
| 1 | $sql=mysqli\_query("SELECT \* FROM nume\_tabel"); |

Pentru a afisa fiecare rand din tabel se foloseste o bucla *while* si comanda **mysqli\_fetch\_row**.

**Exemplu:** Afișarea tuturor întregistrărilor din tabelul elev

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /\* urmeaza fisierul select.php \*/    <?php    include ("conexiune.php");    $sql=mysqli\_query($conexiune, "SELECT \* FROM elev");    echo "<table border=1>";    echo  "<tr><td>ID</td><td>Nume</td><td>Prenume</td></tr>";    while ($row=mysqli\_fetch\_row($sql)) {    echo  "<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td></tr>";    }    echo "</table>";    mysqli\_close($conexiune);    ?> |

Datele au fost afisate fiecare intr-o celula a unui tabel.

Folosind functia **mysqli\_num\_rows($sql)** putem afla numarul de linii continute de baza de date.

**Exemplu:** Acelasi exemplu plus numarul de linii al bazei de date

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\* urmeaza fisierul select1.php \*/    <?php    include ("conexiune.php");    $sql=mysqli\_query($conexiune, "SELECT \* FROM elev");    $rows=mysqli\_num\_rows($sql);    echo "<b>$rows</b> inregistrari în baza de date<p>";    echo "<table border=1>";    echo  "<tr><td>ID</td><td>Nume</td><td>Prenume</td></tr>";    while ($row=mysqli\_fetch\_row($sql)) {    echo  "<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td></tr>";    }    echo "</table>";    mysqli\_close($conexiune);    ?> |

**Exemplu:** Cautarea in baza de date dupa o anumita inregistrare folosind conditia WHERE.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <!-- urmeaza fisierul cautare.html -->    <html>    <head><title>Cautare</title>    </head>    <body>    <b>Cautare inregistrari</b>    <form method="POST" action="where.php">    Numele cautat: <input type="text"  name="nume1"><br>    <input type="submit" value="Trimite">    </form>    </body>    </html> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | /\* urmeaza fisierul where.php \*/   <?php   include ("conexiune.php");   $nume1=$\_POST['nume1'];   $sql=mysqli\_query($conexiune, "SELECT \* FROM elev WHERE nume='$nume1'");   echo "<table border=1>";   echo  "<tr><td>ID</td><td>Nume</td><td>Prenume</td></tr>";   while ($row=mysqli\_fetch\_row($sql)) {   echo  "<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td></tr>";   }   echo "</table>";   mysqli\_close($conexiune);   ?> |

**Conditionale**

Conditionalele sunt operatori care ne permit sa particularizam interogarea bazei de date.

|  |  |
| --- | --- |
| **Operator** | **Semnificatie** |
| = | egalitate |
| < | mai mic |
| > | mai mare |
| <= | mai mic sau egal |
| >= | mai mare sau egal |
| != | diferit de |
| IS NOT NULL | are o valoare |
| IS NULL | nu are valoare |
| BETWEEN | in interiorul unui domeniu dat |
| NOT BETWEEN | in afara unui domeniu dat |
| OR | una din cele doua ecuatii este adevarata |
| || | una din cele doua ecuatii este adevarata |
| AND | amandoua ecuatiile sunt adevarate |
| && | amandoua ecuatiile sunt adevarate |
| NOT | amandoua ecuatiile sunt false |
| ! | amandoua ecuatiile sunt false |

**Exemplu:** Cautam in baza de date folosind doua campuri obligatorii (AND, &&)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <!-- urmeaza fisierul cautare1.html -->   <html>   <head><title>Cautare 1</title>   </head>   <body>   <b>Cautare inregistrari</b>   <form method="POST" action="where1.php">   Numele cautat: <input type="text"  name="nume1"><br>   Prenumele cautat: <input type="text"  name="prenume1"><br>   <input type="submit" value="Trimite">   </form>   </body>   </html> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | /\* urmeaza fisierul where1.php \*/  <?php  include ("conexiune.php");  $nume1=$\_POST['nume1'];  $prenume1=$\_POST['prenume1'];  $sql=mysqli\_query($conexiune, "SELECT \* FROM elev WHERE nume='$nume1' && prenume='$prenume1'");  echo "<table border=1>";  echo  "<tr><td>ID</td><td>Nume</td><td>Prenume</td></tr>";  while ($row=mysqli\_fetch\_row($sql)) {  echo  "<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td></tr>";  }  echo "</table>";  mysqli\_close($conexiune);  ?> |

**LIKE si NOT LIKE**

Daca dorim sa facem o cautare mai generala in baza de date, introducand doar cateva caractere, putem folosi simbolurile **LIKE** sau **NOT LIKE**. Caracterul **\_** anunta ca rezultatul va contine inca un caracter in plus iar **%** inlocuieste zero sau mai multe caractere. Acestea pot fi adaugate la inceput, la sfarsit sau in ambele parti.

**Exemplu:** cautare generala folosind *LIKE* si caracterul % si la inceput si la sfarsit

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /\* urmeaza fisierul cautare2.html \*/   <html>   <head><title>Căutare 2</title>   </head>   <body>   <b>Cautare inregistrari</b>   <form method="POST" action="where2.php">>   Numele cautat: <input type="text" name="nume1"><br>   <input type="submit" value="Trimite">   </form>   </body>   </html> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | /\* urmeaza fisierul where2.php \*/    <?php    include ("conexiune.php");    $nume1=$\_POST['nume1'];    $sql=mysqli\_query($conexiune, "SELECT \* FROM elev WHERE nume LIKE '%$nume1%'");    echo "<table border=1>";    echo "<tr><td>ID</td><td>Nume</td><td>Prenume</td></tr>";    while ($row=mysqli\_fetch\_row($sql)) {    echo "<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td></tr>";    }    echo "</table>";    mysqli\_close($conexiune);    ?> |

**Modificarea datelor înregistrate**

Putem modifica valoarea unei inregistrari din baza de date pentru un anume *id* cu urmatoarea sintaxa:

|  |  |
| --- | --- |
| 1  2 | UPDATE nume\_tabel SET coloana\_1='$nou\_coloana\_1',  coloana\_2='$nou\_coloana\_2',...,  coloana\_n='$nou\_coloana\_n' WHERE id='$nou\_id'; |

Cunoscand valoarea identificatorului **id** al unei inregistrari putem actualiza o inregistrare in baza de date, dar mai inainte trebuie sa afisam inregistrarile pentru a selecta care se potrivesc solicitarilor noastre.  
Primul fisier va afisa sub forma unui formular fiecare inregistrare din baza de date. Campurile formularelor sunt completate cu valorile corespunzatoare fiecarei inregistrari, ele putand fi modificate.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\* urmeaza fisierul update.php \*/     <?php    include ("conexiune.php");    $result = mysqli\_query($conexiune, "SELECT \* FROM elevi");      while($row = mysqli\_fetch\_assoc($result)) {      ?>    <form action="updated.php" method="post">      <input type="hidden" name="ud\_id" value="<?php echo  $row['id'];?>">    Nume: <input type="text" name="ud\_nume" value="<?php echo $row['nume'];?>">    Prenume: <input type="text" name="ud\_prenume" value="<?php echo $row['prenume'];?>">    <input type="Submit" value="Modifica">    </form>    <?php     }    ?> |

Al doilea fisier preia datele modificate si le suprascrie in baza de date.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | /\* urmeaza fisierul updated.php \*/  <?php    include ("conexiune.php");    $ud\_id=$\_POST['ud\_id'];    $ud\_nume=$\_POST['ud\_nume'];    $ud\_prenume=$\_POST['ud\_prenume'];    $query="UPDATE elevi SET nume='$ud\_nume', prenume='$ud\_prenume' WHERE  id='$ud\_id'";    $checkresult = mysqli\_query($conexiune, $query);    if ($checkresult) {      echo "Modificare efectuata";    } else {      echo "Modificare neefectuata";    }    mysqli\_close($conexiune);    ?> |

**Exemplu** de editare multiplă a datelor

[VIEW](https://web.ceiti.md/edit/) [DOWNLOAD](https://web.ceiti.md/edit/down/Edit.rar)

**Ștergerea elementelor dintr-o tabelă MySQL**

Stergerea poate fi facuta folosind id-ul inregistrarii dorite cu sintaxa:

|  |  |
| --- | --- |
| 1 | DELETE FROM nume\_tabel WHERE id='$id'; |

Primul fisier va contine un formular prin care introducem datele care vor fi cautate in baza de date.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <!-- urmeaza fisierul index.html -->    <html>    <head><title>Șterge înregistrări</title>    </head>    <body>    <b>Căutare înregistrări</b>    <p>    <form method="POST" action="delete.php">    Numele cautat: <input type="text"  name="nume"><br>    <input type="submit" value="Trimite">    </form>    </body>    </html> |

Al doilea fisier va afisa rezultatul cautarii si un formular in care vom completa id-ul inregistrarii care dorim s-o stergem.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\* urmeaza fisierul delete.php \*/     <?php    include ("conexiune.php");    $nume=$\_POST['nume'];    $sql=mysqli\_query($conexiune, "SELECT \* FROM elevi WHERE nume LIKE '%$nume%'");    echo "<table border=\"1\">";    echo  "<tr><td>ID</td><td>Nume</td><td>Prenume</td></tr>";    while ($row=mysqli\_fetch\_row($sql)) {    echo  "<tr><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td></tr>";    }    echo "</table>";    mysqli\_close($conexiune);    ?>    <p>    <form method="POST" action="deleted.php">    ID-ul inregistrarii ce va fi stearsă: <input type="text"  name="id" size="3"><br>    <input type="submit" value="Trimite">    </form> |

Al treilea fisier preia id-ul dorit si sterge respective inregistrare.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | /\* urmeaza fisierul deleted.php \*/     <?php    include ("conexiune.php");    $id=$\_POST['id'];    $sql=mysqli\_query($conexiune, "DELETE FROM elevi WHERE id='$id'");    if (!$sql) {    die(mysqli\_error());    } else {    echo "Datele au fost șterse";    }    mysqli\_close($conexiune);    ?> |

**Exemplu** de ștergere multiplă a datelor

[VIEW](https://web.ceiti.md/delete/) [DOWNLOAD](https://web.ceiti.md/delete/down/Delete.rar)

**Aplicaţie Web** care să permită operaţii de listare, updatare, inserare, ştergere (CRUD)

[VIEW](https://web.ceiti.md/crud/)

**2. MySQLi obiect orientat**

Fiecare sarcină – conectare, interogare, extragere, ... – reprezintă de fapt o metodă a obiectului **mysqli()**.

**Conectarea la serverul MySQL**

Înainte de a putea accesa informații din baza de date, trebuie creată o conexiune cu serverul MySQL. Pentru a vă conecta la un server MySQL cu PHP și MySQLi, se creează o **instanță de obiect mysqli**, cu formula **new mysqli()**, la care se adaugă datele de conectare.

**- Sintaxa:**

|  |  |
| --- | --- |
| 1 | $mysqli = new mysqli($host, $user, $pass, $db, $port); |

- *$host* - Specifică serverul la care să se conecteze. Dacă adăugați valoarea NULL sau un șir gol "", serverul va folosi valoarea standard: "*localhost*".  
- *$user* - Specifică numele de utilizator la baza de date MySQL.  
- *$pass* - Specifică parola utilizatorului pentru conectare.  
- *$db* - Opțional. Numele bazei de date care conține tabelele.  
- *$port* - Opțional. Specifică port-ul prin care se face conectarea la serverul MySQL, standard este 3306

**Închiderea conexiunii**

Conexiunea va fi închisă automat când scriptul se termină. Se recomandă totuși să închideți conexiunea mai devreme, când scriptul nu o mai folosește. Acest lucru va elibera memoria folosită de conexiune. Pentru a închide o conexiune se folosește metoda **close()** din clasa **MySQLi**.

**- Sintaxa:**

|  |  |
| --- | --- |
| 1 | $mysqli->close(); |

***Exemplu:*** Conectarea la baza de date *colegiu* și închiderea conexiunii.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Conectare si afisare</title>  </head>  <body>  <?php     // setează variabilele de acces la serverul MySQL    $host = "localhost";    $user = "root";    $pass = "";    $db = "colegiu";      // creează obiectul mysqli    // deschide conexiunea    $mysqli = new mysqli($host, $user, $pass, $db);      // verifică dacă au apărut erori la conectare    if (mysqli\_connect\_errno()) {        die("Nu mă pot conecta la serverul MySQL!");    }        ...      // închide conexiunea    $mysqli->close();    ?>  </body>  </html> |

**Creare bazei de date cu MySQLi**

Pentru a stoca informații în MySQL, este nevoie de o bază de date.

Comanda CREATE DATABASE este folosită pentru a crea o bază de date în MySQL.

**- Sintaxa:**

|  |  |
| --- | --- |
| 1 | CREATE DATABASE nume\_bd |

Pentru a face un script PHP să execute instrucțiunile SQL, mai întâi trebuie să creați o instanța de obiect mysqli, care va conține conexiunea la server, apoi folosiți metoda query() din clasa MySQLi.

***Exemplu:*** Crearea bazei de date *colegiu*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Creare baza de date</title>  </head>  <body>   <?php      // setează variabilele de acces la serverul MySQL    $host = "localhost";    $user = "root";    $pass = "";      // creează obiectul mysqli    // deschide conexiunea    $mysqli = new mysqli($host, $user, $pass);      // verifică dacă au apărut erori la conectare    if ($mysqli ->connect\_error) {        die("Conexiunea a eșuat: " . $mysqli ->connect\_error);    }    // creează baza de date    $query = "CREATE DATABASE colegiu";      if ($result = $mysqli->query($query)) {    echo "Baza de date a fost creată cu succes";    } else {    echo "Eroare la crearea bazei de date: " . $mysqli ->error;    }    $mysqli ->close();    ?>  </body>  </html> |

**Creare tabel în MySQL**

După ce ați realizat o conexiune la o baza de date, puteți crea tabele în ea. Tabelele sunt elementele care stochează datele în baza de date. Pentru a crea un tabel în MySQL, se folosește comanda CREATE TABLE, apoi se apelează metoda query().

**- Sintaxa:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | CREATE TABLE `table\_name`  (  `nume\_coloana1` tip\_date,  `nume\_coloana2` tip\_date,  `nume\_coloana3` tip\_date,  ....  ) |

-*tip\_date* – specifică tipul de date pe care o coloană le poate conține. Pentru o listă cu tipurile de date MySQL.

După "tip\_date", se pot specifica și alte atribute opționale pentru fiecare coloană:

* NOT NULL- Fiecare rând trebuie să conțină o valoare în acea coloană, valorile null nu sunt permise.
* DEFAULT value - Setează o valoare inițială care este adăugată când nici o altă valoare nu este transferată.
* UNSIGNED - Poate fi folosit pentru numere, setează să fie folosite doar numere pozitive și zero.
* AUTO INCREMENT - MySQL va mari automat valoarea din rândul următor cu 1 de fiecare data când o nouă înregistrare este adăugată.
* PRIMARY KEY - Este folosit pentru a identifica unic rândurile dintr-un tabel. Coloana cu setarea PRIMARY KEY este de obicei un ID numeric, și în general folosită cu AUTO\_INCREMENT.

Fiecare tabel ar trebui să aibă o coloană PRIMARY KEY. Valoarea ei trebuie să fie unică pentru fiecare înregistrare din tabel.

***Exemplu:*** Crearea tabelului *elev*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Creare tabel</title>  </head>  <body>   <?php       // setează variabilele de acces la serverul MySQL    $host = "localhost";    $user = "root";    $pass = "";    $db = "colegiu";      // creează obiectul mysqli    // deschide conexiunea    $mysqli = new mysqli($host, $user, $pass, $db);      // verifică dacă au apărut erori la conectare    if ($mysqli ->connect\_error) {    die("Conexiunea a eșuat: " . $mysqli ->connect\_error);    }    // interogare pentru CREATE TABLE    $query = "CREATE TABLE `elev` (    `id` INT(11) UNSIGNED AUTO\_INCREMENT PRIMARY KEY,    `nume` VARCHAR(25) NOT NULL,    `prenume` VARCHAR(25) NOT NULL,    `grupa` VARCHAR(25) NOT NULL,    `specialitatea` VARCHAR(25) NOT NULL,    `sex` CHAR(1),    reg\_date TIMESTAMP    ) ";      // Execuă interogarea $sql query pe server pentru a crea tabelul    if ($mysqli->query($query) === TRUE) {    echo 'Table "elev" successfully created';    }    else {    echo 'Error: '. $mysqli->error;    }    $mysqli->close();    ?>  </body>  </html> |

În comenzile SQL este recomandat ca numele de tabel și coloane să fie adăugat între apostrof ` ` (caracterul de lângă butonul cu numărul 1 pe tastatură). Aceasta este sintaxa corectă dar nu este strict necesară.

**Selectarea datelor**

Instrucțiunea SELECT este utilizată pentru a selecta date din una sau mai multe tabele

**- Sintaxa:**

|  |  |
| --- | --- |
| 1 | SELECT nume\_coloană(e) FROM nume\_tabel |

sau putem folosi \* caracterul pentru a selecta toate coloanele dintr-un tabel:

|  |  |
| --- | --- |
| 1 | SELECT \* FROM nume\_tabel |

***Exemplu:*** Afișarea datelor din tabelul *elev*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Afisarea datelor</title>  </head>  <body>  <?php      // setează variabilele de acces la serverul MySQL    $host = "localhost";    $user = "root";    $pass = "";    $db = "colegiu";      // creează obiectul mysqli    // deschide conexiunea    $mysqli = new mysqli($host, $user, $pass, $db);      // verifică dacă au apărut erori la conectare    if (mysqli\_connect\_errno()) {    die("Nu mă pot conecta la serverul MySQL!");    }    // creează interogarea    $query = "SELECT \* FROM elev";      // execută interogarea    if ($result = $mysqli->query($query))  {      // verifică dacă interogarea a returnat vreun rând    if ($result->num\_rows > 0) {      // dacă da, afișează rândurile returnate unul după altul    echo "<table cellpadding=10 cellspacing=0 border=1>";    echo "<tr><td>Id</td><td>Nume</td><td>Prenume</td><td>Varsta</td><td>Grupa</td><td>Specialitatea</td>";    echo "<td>Adresa</td><td>Sex</td></tr>";    while($row = $result->fetch\_array())  {    echo "<tr>";    echo "<td>".$row[0]."</td>";    echo "<td>".$row[1]."</td>";    echo "<td>".$row[2]."</td>";    echo "<td>".$row[3]."</td>";    echo "<td>".$row[4]."</td>";    echo "<td>".$row[5]."</td>";    echo "<td>".$row[6]."</td>";    echo "<td>".$row[7]."</td>";    echo "</tr>";    }    echo "</table>";    }    else {    // dacă nu, afișează mesaj    echo "Nu a fost găsit nici un rând!";    }    // eliberează memoria    $result->close();    }    else {    // afișează mesaj de eroare    echo "Interogarea SQL conține o eroare:  $query. ".$mysqli->error;    }    // închide conexiunea    $mysqli->close();    ?>  </body>  </html> |

**Clauza WHERE și LIKE**

Pentru a căuta valori specifice, pentru a extrage doar acele înregistrari care respectă anumite criterii, se adaugă o clauză WHERE în interogarea SELECT.

Clauza WHERE este folosită pentru a filtra înregistrările. Se adaugă după numele tabelului.

**- Sintaxa:**

|  |  |
| --- | --- |
| 1 | SELECT nume\_coloană(e) FROM nume\_tabel WHERE conditie |

*conditie* - poate fi una sau mai multe expresii condiționale care specifică criteriul de selectare. Aceste expresii condiționale pot folosi operatori de comparație, aritmetici și logici.

Tabelul urmator prezinta operatorii cei mai des folosiți cu WHERE în instructiunile MySQL:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Semnificație** | **Exemplu** |
| = | egalitate | WHERE sex = f |
| < | mai mic | WHERE varsta < 45 |
| > | mai mare | WHERE varsta > 18 |
| <= | mai mic sau egal | WHERE varsta < =45 |
| >= | mai mare sau egal | WHERE varsta >= 18 |
| != | diferit de | WHERE sex != m |
| IS NOT NULL | are o valoare | WHERE adresa IS NOT NULL |
| IS NULL | nu are valoare | WHERE specialitatea IS NULL |
| BETWEEN | în interiorul unui domeniu dat | WHERE varsta BETWEEN 18 AND 45 |
| NOT BETWEEN | în afara unui domeniu dat | WHERE varsta NOT BETWEEN 18 AND 45 |
| IN | se găsește în lista de valori | WHERE grupa IN(I-1241, I-1242, I-1243, I-1244, I-1245) |
| ||(OR) | una din cele două condiții este adevărată | WHERE varsta>18 || sex=f |
| &&(AND) | amândouă condiții sunt adevarate | WHERE varsta>18 && sex=f |

***Exemplu:*** Căutarea în baza de date *colegiu* după o anumită înregistrare folosind clauza WHERE.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!-- urmeaza fisierul cautare.html -->    <!doctype html>    <html>    <head>    <meta charset="utf-8">    <title>Cautare</title>    </head>    <body>    <b>Cautare inregistrari</b>    <form method="POST" action="where.php">    Numele căutat: <input type="text"  name="nume"><br>    <input type="submit" value="Trimite">    </form>    </body>    </html> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | /\* urmează fișierul where.php \*/  <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Cautare</title>  </head>  <body>  <?php  // setează variabilele de acces la serverul MySQL  $host = "localhost";  $user = "root";  $pass = "";  $db = "colegiu";    // creează obiectul mysqli  // deschide conexiunea  $mysqli = new mysqli($host, $user, $pass, $db);    // verifică dacă au apărut erori la conectare  if ($mysqli ->connect\_error) {      die("Conexiunea a eșuat: " . $mysqli->connect\_error);  }    $nume=$\_POST['nume'];    // creează interogarea  $query = "SELECT \* FROM elev WHERE nume='$nume'";     // execută interogarea  if ($result = $mysqli->query($query)) {     // verifică dacă interogarea a returnat vreun rând      if ($result->num\_rows > 0) {          // dacă da,  afișează rândurile returnate unul după altul          echo "<table cellpadding=10 cellspacing=0 border=1>";          echo "<tr><td>Id</td><td>Nume</td><td>Prenume</td><td>Varsta</td><td>Grupa</td><td>Specialitatea</td>";          echo "<td>Adresa</td><td>Sex</td></tr>";            while($row = $result->fetch\_array()) {             echo "<tr>";             echo "<td>".$row[0]."</td>";             echo "<td>".$row[1]."</td>";             echo "<td>".$row[2]."</td>";             echo "<td>".$row[3]."</td>";             echo "<td>".$row[4]."</td>";             echo "<td>".$row[5]."</td>";             echo "<td>".$row[6]."</td>";             echo "<td>".$row[7]."</td>";             echo "</tr>";          }          echo "</table>";      }      else {          // dacă nu, afișează mesaj          echo "Nu a fost găsit nici un rând!";      }       // eliberează memoria      $result->close();  }  else {      // afișează mesaj de eroare      echo "Interogarea SQL conține o eroare: $query. ".$mysqli->error;  }   // închide conexiunea  $mysqli->close();   ?>  </body>  </html> | |

***Exemplu:*** Căutarea în baza de date după două câmpuri obligatorii

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <!-- urmeaza fisierul cautare1.html -->   <!doctype html>  <html>  <head>  <meta charset="utf-8">    <title>Cautare 1</title>    </head>    <body>    <b>Cautare inregistrari</b>    <form method="POST" action="where1.php">    Numele cautat: <input type="text"  name="nume"><br>    Prenumele cautat: <input type="text"  name="prenume"><br>    <input type="submit" value="Trimite">    </form>    </body>    </html> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | /\* urmează fișierul where1.php \*/  <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Cautare</title>  </head>  <body>  <?php  // setează variabilele de acces la serverul MySQL  $host = "localhost";  $user = "root";  $pass = "";  $db = "colegiu";    // creează obiectul mysqli  // deschide conexiunea  $mysqli = new mysqli($host, $user, $pass, $db);    // verifică dacă au apărut erori la conectare  if ($mysqli ->connect\_error) {      die("Conexiunea a eșuat: " . $mysqli->connect\_error);  }    $nume=$\_POST['nume'];  $prenume=$\_POST['prenume'];    // creează interogarea  $query = "SELECT \* FROM elev WHERE nume='$nume' && prenume='$prenume'";     // execută interogarea  if ($result = $mysqli->query($query)) {     // verifică dacă interogarea a returnat vreun rând      if ($result->num\_rows > 0) {            // dacă da,  afișează rândurile returnate unul după altul          echo "<table cellpadding=10 cellspacing=0 border=1>";          echo "<tr><td>Id</td><td>Nume</td><td>Prenume</td><td>Varsta</td><td>Grupa</td><td>Specialitatea</td>";          echo "<td>Adresa</td><td>Sex</td></tr>";            while($row = $result->fetch\_array()) {             echo "<tr>";             echo "<td>".$row[0]."</td>";             echo "<td>".$row[1]."</td>";             echo "<td>".$row[2]."</td>";             echo "<td>".$row[3]."</td>";             echo "<td>".$row[4]."</td>";             echo "<td>".$row[5]."</td>";             echo "<td>".$row[6]."</td>";             echo "<td>".$row[7]."</td>";             echo "</tr>";          }          echo "</table>";      }      else {          // dacă nu, afișează mesaj          echo "Nu a fost găsit nici un rând!";      }       // eliberează memoria      $result->close();  }  else {      // afișează mesaj de eroare      echo "Interogarea SQL conține o eroare: $query. ".$mysqli->error;  }   // închide conexiunea  $mysqli->close();   ?>  </body>  </html> | |

***Exemplu:*** Căutare generală folosind *LIKE* și caracterul % la început și la sfârșit

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | /\* urmează fișierul cautare2.html \*/  <!doctype html>  <html>  <head>  <meta charset="utf-8">    <title>Căutare 2</title>    </head>    <body>    <b>Căutare înregistrări</b>    <form method="POST" action="where2.php">>    Numele căutat: <input type="text" name="nume"><br>    <input type="submit" value="Trimite">    </form>    </body>    </html> |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | /\* urmează fișierul where2.php \*/  <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Cautare</title>  </head>  <body>  <?php  // setează variabilele de acces la serverul MySQL  $host = "localhost";  $user = "root";  $pass = "";  $db = "colegiu";    // creează obiectul mysqli  // deschide conexiunea  $mysqli = new mysqli($host, $user, $pass, $db);    // verifică dacă au apărut erori la conectare  if ($mysqli ->connect\_error) {      die("Conexiunea a eșuat: " . $mysqli->connect\_error);  }    $nume=$\_POST['nume'];    // creează interogarea  $query = "SELECT \* FROM elev WHERE nume LIKE '%$nume%'";     // execută interogarea  if ($result = $mysqli->query($query)) {     // verifică dacă interogarea a returnat vreun rând      if ($result->num\_rows > 0) {            // dacă da,  afișează rândurile returnate unul după altul          echo "<table cellpadding=10 cellspacing=0 border=1>";          echo "<tr><td>Id</td><td>Nume</td><td>Prenume</td><td>Varsta</td><td>Grupa</td><td>Specialitatea</td>";          echo "<td>Adresa</td><td>Sex</td></tr>";            while($row = $result->fetch\_array()) {             echo "<tr>";             echo "<td>".$row[0]."</td>";             echo "<td>".$row[1]."</td>";             echo "<td>".$row[2]."</td>";             echo "<td>".$row[3]."</td>";             echo "<td>".$row[4]."</td>";             echo "<td>".$row[5]."</td>";             echo "<td>".$row[6]."</td>";             echo "<td>".$row[7]."</td>";             echo "</tr>";          }          echo "</table>";      }      else {          // dacă nu, afișează mesaj          echo "Nu a fost găsit nici un rând!";      }       // eliberează memoria      $result->close();  }  else {      // afișează mesaj de eroare      echo "Interogarea SQL conține o eroare: $query. ".$mysqli->error;  }   // închide conexiunea  $mysqli->close();   ?>  </body>  </html> | |

**Inserarea datelor**

Pentru a adaugă înregistrări noi într-un tabel MySQL, se folosește comanda INSERT INTO.

Există două metode prin care o interogare INSERT poate fi scrisă:

1) Prima metodă nu specifică numele coloanelor unde datele for fi introduse, ci doar valorile lor.

|  |  |
| --- | --- |
| 1  2 | INSERT INTO nume\_tabel  VALUES (valoare1, valoare2, valoare3, ...) |

În această metoda, trebuie să specificați o valoare pentru fiecare coloana, chiar dacă este NULL. Dacă sunt cinci coloane în tabel, trebuie să adăugați cinci valori, separate prin virgula.

2) Pentru a doua metodă se specifică atât numele coloanei /coloanelor cât și valorile care vor fi introduse:

|  |  |
| --- | --- |
| 1  2 | INSERT INTO nume\_tabel (coloana1, coloana2, coloana3,...)  VALUES (valoare1, valoare2, valoare3,...) |

Aceasta este în general metoda preferată, deoarece puteți adaugă doar coloanele care vor primi valori. Orice coloană căruia nu îi este atribuită o valoare va fi tratată ca NULL (sau îi va fi dată valoarea default, dacă aceasta este definită). Dacă o coloană nu poate avea valoarea NULL (a fost definită ca NOT NULL) și nu are o valoare default, nespecificând o valoare va genera eroare. Dacă o coloană este AUTO\_INCREMENT sau TIMESTAMP, nu este nevoie să fie specificată în interogarea SQL, serverul MySQL va seta și adăuga valoarea automat. Pentru ca PHP să execute comenzile SQL, trebuie să folosești metoda mysqli **query()**.

***Exemplu:*** Inserare date în tabelul *elev*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Inserare</title>  </head>  <body>  <?php      if (!isset($\_POST['submit']))  {    // formularul  nu a fost transmis    ?>    <form  action="<?=$\_SERVER['PHP\_SELF']?>" method="post">    Nume: <input type="text" name="nume">    Prenume: <input type="text" name="prenume">    Grupa: <input type="text" name="grupa">    <input type="submit" name="submit" value="Trimite">    </form>    <?php    }    else {    // formularul nu a fost transmis    // setează variabilele de acces la serverul  MySQL    $host = "localhost";    $user = "root";    $pass = "";    $db = "colegiu";      // preia informațiile din formular    // verificare că toate câmpurile au fost  completate      $nume = empty($\_POST['nume']) ? die ("EROARE: Completați Nume!") : $\_POST['nume'];    $prenume = empty($\_POST['prenume']) ? die ("EROARE: Completați Prenumele") : $\_POST['prenume'];    $grupa = empty($\_POST['grupa']) ? die ("EROARE: Completați Grupa") : $\_POST['grupa'];      // deschide conexiunea    $mysqli = new  mysqli($host, $user, $pass, $db);      // verifică  dacă au apărut erori la conectare    if  (mysqli\_connect\_errno()) {    die("Nu mă pot conecta la serverul  MySQL!");    }    // creează interogare    $query = "INSERT INTO elev (nume,  prenume, grupa) VALUES ('$nume', '$prenume', '$grupa')";      // Trimite interogarea $query la MySQL pentru  a introduce valorile    if  ($mysqli->query($query) === TRUE) {    echo "A  fost inserată înregistrarea cu "; }    else {    echo 'Error: '. $mysqli->error;    }    $mysqli->close();    }    ?>  </body>  </html> |

Odată ce formularul a fost transmis, informațiile din acesta sunt folosite pentru a crea o interogare **INSERT**, care este apoi trimisă către baza de date. Din moment ce **query()** returnează o valoare de tip Boolean, indicând dacă interogarea s-a desfășurat cu succes sau nu, este posibilă verificarea rezultatului instrucțiunii **INSERT** și returnarea unui mesaj adecvat.

**Modificarea datelor înregistrate**

După ce ați adăugat rânduri în tabelul MySQL, puteți să modificați datele înregistrate, folosind comanda UPDATE.

Instrucțiunea **UPDATE** este folosită pentru a modifica sau actualiza informațiile existente într-un tabel.

**- Sintaxa:**

|  |  |
| --- | --- |
| 1  2 | UPDATE  nume\_tabel SET coloana\_1='$nou\_coloana\_1', coloana\_2='$nou\_coloana\_2',..., coloana\_n='$nou\_coloana\_n'       WHERE id='$nou\_id'; |

- Clauza WHERE este importantă într-o interogare UPDATE , specifică serverului MySQL care înregistrare sau înregistrări trebuie actualizate. Daca omiteți clauza WHERE, vor fi afectate toate rândurile!

- Comanda UPDATE este trimisa la serverul MySQL cu metoda **query()**a obiectului mysqli.

Cunoscând valoarea identificatorului **id** al unei înregistrări putem actualiza o înregistrare în baza de date, dar mai înainte trebuie să afișăm înregistrările pentru a selecta care se potrivesc solicitărilor noastre.

Primul fișier va afișa sub forma unui formular fiecare înregistrare din baza de date. Câmpurile formularelor sunt completate cu valorile corespunzătoare fiecărei înregistrări, ele putând fi modificate:

***Exemplu:*** Modificare date din tabelul *elev*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Afișare pentru modificare</title>  </head>  <body>   <?php      // setează variabilele de acces la serverul MySQL    $host =  "localhost";    $user =  "root";    $pass =  "";    $db =  "colegiu";     // creează  obiectul mysqli    // deschide  conexiunea    $mysqli = new  mysqli($host, $user, $pass, $db);      // verifică  dacă au apărut erori la conectare    if ($mysqli  ->connect\_error) {       die("Conexiunea a eșuat: " .  $mysqli->connect\_error);    }    // creează  interogarea    $query =  "SELECT \* FROM elev";      // execută  interogarea    if ($result =  $mysqli->query($query)) {      while($row = $result->fetch\_assoc()) {      $id=$row['id'];    $nume=$row['nume'];    $prenume=$row['prenume'];    ?>    <form action="updated.php"  method="post">    <input type="text" name="ud\_id" value="<?php echo "$id";?>">    Nume: <input type="text" name="ud\_nume" value="<?php echo "$nume";?>">    Prenume: <input type="text" name="ud\_prenume" value="<?php echo "$prenume";?>">    <input type="Submit" value="Modifica">    </form>    <?php    }    }    $mysqli ->close();    ?>  </body>  </html> |

Al doilea fișier (udated.php) preia datele modificate și le suprascrie în baza de date:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Inserare</title>  </head>  <body>    <?php      // setează  variabilele de acces la serverul MySQL    $host =  "localhost";    $user =  "root";    $pass =  "";    $db =  "colegiu";      // creează  obiectul mysqli    // deschide  conexiunea    $mysqli = new  mysqli($host, $user, $pass, $db);      $ud\_id=$\_POST['ud\_id'];    $ud\_nume=$\_POST['ud\_nume'];    $ud\_prenume=$\_POST['ud\_prenume'];      // creează interogare    $query="UPDATE elev SET nume='$ud\_nume', prenume='$ud\_prenume' WHERE id='$ud\_id'";      // Trimite interogarea $query la MySQL pentru  a introduce valorile    if  ($mysqli->query($query) === TRUE) {    echo  "Modificare efectuata "; }    else {    echo 'Modificare neefectuata. Error: '.  $mysqli->error;    }    $mysqli->close();    ?>      </body>  </html> |

**Exemplu**de editare multiplă a datelor

[VIEW](https://web.ceiti.md/edit/)

**Ștergerea elementelor dintr-o tabelă MySQL**

Evident, același lucru poate fi îndeplinit și în cazul altor instrucțiuni de manipulare a datelor. Următorul exemplu îți demonstrează cum să folosești instrucțiunea **DELETE** pentru a elimina selectiv elemente din tabel.

***Exemplu:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Selectare si eliminare</title>  </head>  <body>     <?php     // setează variabilele de acces la serverul MySQL   $host = "localhost";   $user = "root";   $pass = "";   $db = "colegiu";     // creează obiectul  mysqli   // deschide conexiunea   $mysqli = new mysqli($host, $user, $pass, $db);     // verifică dacă au apărut erori la conectare   if (mysqli\_connect\_errno())  {   die("Nu mă pot conecta la serverul MySQL!");   }     // dacă am primit  valoarea id-ului prin GET, o ștergem   if (isset($\_GET['id']))  {     // creează interogarea pentru ștergere   $query = "DELETE FROM elev WHERE id = ".$\_GET['id'];     // execută interogarea   if ($mysqli->query($query)) {     // afișează numărul de rânduri care au fost șterse   echo $mysqli->affected\_rows." row(s) affected";   }   else {     // afișează mesaj de eroare   echo "Interogarea conține o eroare:  $query. ".$mysqli->error;   }   }     // creează interogarea pentru a citi informațiile   $query = "SELECT  \* FROM elev";     // execută interogarea   if ($result = $mysqli->query($query))  {     // verifică dacă a fost returnat vreun rând   if ($result->num\_rows > 0) {     // dacă da   // afișează rândurile returnate unul după  altul   echo "<table cellpadding=10 border=1>";     while($row = $result->fetch\_array())  {   echo "<tr>";   echo "<td>".$row[0]."</td>";   echo "<td>".$row[1]."</td>";   echo "<td>".$row[2]."</td>";   echo "<td><a href=".$\_SERVER['PHP\_SELF']."?id=".$row[0].">Șterge</a></td>";   echo "</tr>";   }   }   // eliberează memoria   $result->close();   }   else {   // afișează mesaj de eroare   echo "Interogarea conține o eroare: $query. ".$mysqli->error;   }   // închide conexiunea   $mysqli->close();   ?> |

De remarcat faptul că am folosit proprietatea **affected\_rows** a obiectului **mysqli**, care returnează numărul total de rânduri ce au fost afectate de către ultima operație.

Arborele de coduri extensia **MySQLi** include două funcții adiționale pentru erorile de conexiune, anume **mysqli\_connect\_errno()** și **mysqli\_connect\_error()**, care conțin numai informații asupra erorilor de conexiune (nu și asupra celor de interogare). Folosește-le pentru a depana erorile din conexiunile tale MySQL, ca în exemplul de mai jos.

***Exemplu:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Conectare si inserare</title>  </head>  <body>     <?php     // creează obiectul mysqli   // deschide conexiunea   $mysqli = new mysqli("localhost", "root", "", "colegiu");     // verifică dacă au aparut erori   if (mysqli\_connect\_errno())  {   die("Nu mă pot conecta la serverul MySQL:  ".mysqli\_connect\_error());   }     // creează interogarea   $query = "SELECT  \* FROM elev";     // execută interogarea   $result = $mysqli->query($query);     // afișează erorile   if(!$result) {   $error\_number = $mysqli->errno;   $error\_msg = $mysqli->error;   echo "Eroare MySQL $error\_number: $error\_msg";   }   // închide conexiunea   $mysqli->close();   ?> |

Și, în caz că te întrebai de ce nu am folosit sintaxa pe obiecte pentru cele doua funcții utilizate în script-ul de mai sus, răspunsul este unul simplu: **nu se poate**. Dacă intervine o eroare în conectarea la server, obiectul **mysqli()**nu va fi creat, și astfel metodele și proprietățile ce au legătura cu acel obiect nu vor exista. Din acest motiv, pentru a depăna erorile de conexiune din extensia **MySQLi**, trebuie ca întotdeauna să folosești notații procedurale în detrimentul celor folosite în programarea orientată pe obiecte.

**Exemplu** de ștergere multiplă a datelor

[VIEW](https://web.ceiti.md/delete/)

**Aplicaţie Web** care să permită operaţii de listare, updatare, inserare, ştergere (CRUD)

[VIEW](https://web.ceiti.md/crud/)

**Citirea conținutului unui director**

PHP furnizează trei funcții care vă permite să citiți conținutul unui catalog, ca și cum catalogul ar fi un fisier. Aceste functii sunt:  
**1. opendir()** - deschide directorul pentru citire  
**2. readdir ()** - citeste o intrare dintr-un catalog  
**3. closedir()** - închide catalogul, eliberînd resursele alocate de functia **opendir()**

Rezultatul apelarii functiei **readdir()** este un sir care contine numele urmatorului fisier sau sub-director din catalogul deschis. Functia returneaza FALSE atunci cînd au fost citite toate intrarile din catalog.  
Iata un exemplu care prezinta modul de utilizare a acestor functii pentru citirea continutului unui catalog:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?php    $dc = @opendir("home/plomar/curs-php");    if (!$dc) {   echo "Directorul nu a putut fi deschis";    }    else {    while ($ok = readdir($dc)) {    echo "<br /> $ok";    }    closedir($dc);    }    ?> |

Datele de iesire ale exemplului pot fi similare cu cele prezentate mai jos :  
style  
imgs  
lectia  
scripts  
index.php

- Sau puteti folosi functia **scandir($dir)**. Aceasta preia intr-un Array numele tuturor directoarelor si fisierelor din directorul "$dir". In caz de nereusita returneaza False.

**Crearea unui director**

Pentru a crea un director, utilizati functia **mkdir()**, care are urmatoarea forma:  
            **mkdir(nume\_dir, mod)**

- Unde "nume\_dir" este calea sau numele directorului care va fi creat, iar "mod" reprezinta privilegiile care trebuie acordate noului catalog.  
In mod normal, prima cifra a argumentului mod trebuie sa fie 0, astfel încît PHP sa-l perceapa ca pe o valoare scrisa în octal. Functia returneaza TRUE daca creeaza catalogul; în caz contrar, returneaza FALSE.

De exemplu, pentru a crea un catalog denumit "lucru" si pentru a-i atribui privilegiile "rwxr-x--x", utilizati functia **mkdir()** dupa cum urmeaza:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?php    $dc = @opendir("home/plomar/curs-php");    if (!$dc) {   echo "Directorul nu a putut fi deschis";    }    else {    while ($ok = readdir($dc)) {    echo "<br /> $ok";    }    closedir($dc);    }    ?> |

Pentru a modifica numele unui director, puteti folosi functia **rename()**, care are următoarea formă:  
            **rename(nume\_vechi, nume\_nou)**  
Unde "nume\_vechi" este numele si calea originala a directorului, iar "nume\_nou" este numele nou dorit.

**Lucrul cu fișiere**

Folosind PHP-ul se pot executa anumite operații asupra fișierelor aflate pe server cum ar fi:  
**fopen()** = deschide fișierul indicat  
**fclose()** = inchide fișierul  
**fread()** = citește conținutul fișierului  
**fwrite()** = scrie în fișier  
**filesize()** = indică dimensiunea fișierului

**Deschiderea fișierelor**

Sintaxa functiei **fopen()** este:

|  |  |
| --- | --- |
| 1 | fopen(param1, param2); |

param1 = fisierul, calea catre fisier sau adresa fisierului care va fi deschis  
param2 = modul in care va fi deschis fisierul, si poate avea valorile:

r = fisier deschis doar pentru citire  
r+ = fisier deschis doar pentru citire si scriere  
w = fisier deschis doar pentru scriere  
w+ = fisier deschis pentru citire si scriere iar daca nu exista fisierul il creza  
a = fisier deschis pentru adaugare la sfarsit  
a+ = fisier deschis adaugare la sfarsit iar daca nu exista fisierul il creza  
t = fisier deschis in mod text  
b = fisier deschis in mod binar  
sau combinatii ale acestora.

**Citirea fisierelor**

Sintaxa functiei **fread()** este:

|  |  |
| --- | --- |
| 1  2  3 | $filename="proba.txt";  $handle=fopen($filename, "r");  $contents=fread($handle, filesize($filename)); |

variabila *$handle* va fi folosita ulterior si in alte functii.

**Scrierea in fisiere**

Sintaxa functiei **fwrite()** este:

|  |  |
| --- | --- |
| 1 | fwrite($handle, "continut ce va fi scris in  fisier"); |

**Verificarea drepturilor de acces asupra fisierului**

Sintaxa functiei **is\_writable** este:

|  |  |
| --- | --- |
| 1 | is\_writable($filename); |

returneaza **TRUE** daca se poate si **FALSE** daca nu se poate scrie in fisier

**Inchiderea fisierului**

Sintaxa functiei **fclose()** este:

|  |  |
| --- | --- |
| 1 | fclose($handle); |

Folosind aceste functii de baza putem incerca cateva exemple simple.

Exemplu: se verifica dreptul de scriere in fisier. Primul pas este crearea fisierului proba.txt apoi se verifica exemplul care va returna mesajul *Acces interzis scrierii in fisier*. Se modifica drepturile asupra fisierului. Se verifica iar exemplul care va returna mesajul *Acces permis scrierii in fisier*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?php  $filename="proba.txt";  if(is\_writable($filename)) { //se verifica dreptul de scriere in fisier  echo "Acces permis scrierii in fisier";  }  else {  echo "Acces interzis scrierii in fisier";  }  ?> |

Exemplu: adaugam exemplului anterior verificarea deschiderii fisierului.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?php    $filename="proba.txt";    if(is\_writable($filename)) { //se verifica dreptul de scriere in fisier    if(!$handle=fopen($filename, 'a')) { //se verifica deschiderea fisierului    echo "Fisierul <b>".$filename."</b> nu poate fi  deschis";    exit; //se incheie executia scriptului    }    echo "Fisierul <b>".$filename."</b> poate fi  deschis";    } else {    echo "Acces interzis pentru scriere in fisier";    }  ?> |

Exemplu: adaugam exemplului anterior scrierea in fisier si tiparirea continutului sau

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?php  $filename="proba.txt";  if(is\_writable($filename)) { //se verifica dreptul de scriere in fisier  if(!$handle=fopen($filename, 'a')) { //se verifica deschiderea fisierului  echo "Fisierul <b>".$filename."</b> nu poate fi  deschis";  exit; //se incheie executia scriptului  }  if(fwrite($handle, date("j-n-Y g:ia")."\r\n")===FALSE) {  //testeaza daca se poate scrie in fisier  die ("Nu se poate scrie in ".$filename.""); //iesire  }  fclose($handle);  $matrice=file($filename); //se citeste continutul fisierului si se stocheaza  intr-o matrice  foreach ($matrice as $cheie =>$valoare) {  echo $valoare."<br>"; //printeaza cate o valoare pe un rand  }  } else {  echo "Acces interzis pentru scriere in fisier";  }  ?> |

**Afișarea unui colaj de imagini**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <?php  $dir\_fotografii="d:/imagini/";  if (is\_dir($dir\_fotografii)){  if($dh=opendir($dir\_fotografii)){  while  (($file=readdir($dh))!==FALSE){  $tablou[]=$file;  }  closedir($dh);}  }  $dimensiune=count($tablou);  $nr\_foto=$dimensiune-2;  /\*pentru a  elimina subdirectoarele speciale . si ..\*/  if ($nr\_foto>0) {  echo  "<center>Directorul contine: ".$nr\_foto."  fotografii</center>  <table  align=\"center\"cellpadding=\"5\">";  for ($i=2; $i<$dimensiune;$i++){  $fisier=$dir\_fotografii.$tablou[$i];  if($i%2==0)  echo "<tr><td><img  src=\"$fisier\" /></td>";  else  echo "<td><img  src=\"$fisier\" /></td></tr>";  }  echo  "</table>";  }  else echo "Nu sunt  fotografii!";    ?> |

**Exerciţii:**

1. Creati un folder pe discul D:\ , in el creati un fisier cu numele exemplu.txt , deschideti fisierul dat si scrieti un text in el , salvati si afisati continutul din fisier.
2. Creati un fisier care contine un text, afisati numarul de cuvinte din acest text.
3. Creati un fisier exemplu.txt cu ajutorul parametrului w+ si inscrieti in el un text dupa care inchideti fisierul si ii atribuiti parametrul readonly (r).
4. Creati 2 fisiere exemplu1.txt si exemplu2.txt in care sunt inscrise cate un sir de caractere, creati al 3-lea fisier exemplu3.txt in care concatenati continutul din fisierele create anterior. si afisati continutul din exemplu3.txt.
5. Creati un fisier exemplu1.txt in care scrieti un vector din 5 elemente, sortati vectorul si inscrieti intr-un alt fisier exemplu2.txt
6. Creati un fisier in care sa contina un text , afisati in browser cate litere "a","b","c",miniscule si majuscule se contin in fisierul dat.
7. Creati un fisier exemplu1.txt in care inserati un tablou bidimensional 3x3, salvati si afisati elementele de pe diagonala din acest tablou.
8. Creati un fisier care contine un text , creati un al doilea fisier in care cuvintele separate prin spatiu din primul fisier apar ca cuvinte din rand nou in al 2-lea fisier. exemplu (Ion merge la scoala )->primu fisier.  
   Ion  
   merge  
   la  
   scoala -> al doilea fisier.
9. Creati un fisier care contine un text si intr-un al doilea fisier inscrieti acelasi text crescator dupa marimea cuvintelor .

**Creare imagini cu PHP**

Pe langa crearea de cod de tip text (HTML, XML, ...), PHP poate fi folosit si pentru crearea si manipularea de imagini cu diferite formate (GIF, PNG, JPEG, WBMP si XPM).

PHP are implemetat cateva functii pentru lucru cu informatii legate de imagini, cum ar fi lungimea, latimea sau extensia imaginii. Pentru crearea de noi imagini sau manipularea celor existente este necesar sa aveti instalat in PHP o librarie GD cu functii pentru imagini.

**- Daca folositi WampServer sau Xampp, acestea au deja incluse librarii GD**.

Puteti afla informatii despre libraria GD instalata folosind functia **gd\_info()**, aceasta returneaza o matrice cu informatii despre libraria GD inclusa.

Exemplu:

|  |  |
| --- | --- |
| 1  2  3 | <?php  var\_dump(gd\_info());  ?> |

**Crearea imaginilor**

Pentru crearea unei imagini cu PHP sunt esentiali urmatorii pasi:

* Definirea identificatorului de imagine si suprafata de baza a imaginii.
* Stabilirea culorilor fundalului, formei sau a textului din imagine.
* Trimiterea datelor de iesire cu imaginea creata.
* Eliberarea memoriei asociata creeri imaginii.

Cam asta e, pe scurt, partea teoretica, acum sa trecem la practica, voi prezenta pas cu pas un script PHP simplu care va afisa in browser o imagine PNG formata dintr-un dreptunghi cu fundal albastru si in interior un text.

Scriptul este urmatorul:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?php    $text = 'Textul din imagine';   // setare dimensiuni suprafetei de baza    $height = 40;    $width = 170;    // definirea imaginii    $im = imagecreate($width, $height);   // setare culori    $fundal = imagecolorallocate($im, 10, 10, 250);    $text\_color = imagecolorallocate($im, 255, 255, 255);   // finalizarea imaginii    imagefill($im, 0, 0, $fundal);    imagestring($im, 4, 10, 15, $text, $text\_color);   // trimiterea datelor de iesire    header('Content-type: image/png');    imagepng($im);   // curatarea memoriei    imagedestroy($im);    ?> |

- Acest script va afisa urmatorul rezultat

|  |
| --- |
| **Textul din imagine** |

Sa studiem acest script:  
- Pentru inceput am setat variabila "*$text*" ce contine textul care va fi adaugat in imagine, si variabilele "*$height*" si "*$width*" care reprezinta dimensiunile imaginii.  
- In continuare avem "*$im = imagecreate($width, $height)*", functia **imagecreate()** returneaza un identificator de imagine ce creaza suprafata de baza a imaginii, prelund ca argumente dimensiunile acesteia (in pixeli).

* Ca alternativa se poate folosi o imagine deja existenta, peste care sa adaugati testul, pentru aceasta se foloseste functia **imagecreatefrompng("nume.png")** (sau "imagecreatefromjpeg()", "imagecreatefromgif()"; in functie de tipul imaginii), exemplu: *$im = imagecreatefrompng("img.png")*

- Urmatorul pas este definirea culorii pentru fundal si pentru text, pentru aceasta se foloseste functia**imagecolorallocate()**, aceasta returneaza un identificator de culoare care va fi folosit pentru accesarea culorii si preia 4 argumente: primul argument este identificatorul de imagine creat cu "imagecreate()" (*in script este in variabila "$im"*), urmatoarele trei argumente reprezinta valorile RGB (Red, Green, Blue) pentru stabilirea culorii.  
- Pentru adaugarea culorii de fundal se foloseste functia **imagefill()**, care preia 4 argumente (*in script avem "imagefill($im, 0, 0, $fundal);"*), argumentele sunt:

* identificatorul de imagine
* urmatoarele doua argumente definesc coordonatele X si Y de start pentru umplerea culorii (*0, 0*)
* ultimul argument reprezinta identificatorul de culoare folosit (*aici cel din variabila $fundal*)

- Pentru adaugarea textului in cadrul imaginii se foloseste functia **imagestring()**, aceasta preia 6 argumente:

1. identificatorul de imagine
2. tipul fondului folosit, care poate fi un numar intre 1 si 5, reprezentand fonturi prestabilite ca alternativa se poate folosi functia *imageloadfont()*, pentru includerea de alte fonturi
3. al treilea argument reprezinta distanta, in pixeli, fata de marginea din stanga a imaginii
4. al patrulea argument reprezinta distanta, in pixeli, fata de marginea de sus a imaginii
5. al cincilea argument reprezinta sirul introdus
6. ultimul argument este culoarea textului

- Urmatotul pas este trimiterea datelor de iesire pentru afisarea in browser.

* Pentru aceasta intai scriptul spune browser-ului ca trimite o imagine si nu test sau HTML, se face acest lucru folosind functia **header()** cu tipul MIME al imaginii "*header('Content-type: image/png')*". Daca doriti ca imaginea sa fie de tip JPEG (sau GIF) inlocuiti "png" cu "jpeg" (sau "gif"), astfel '*Content-type: image/jpeg*'.
* dupa ce s-au trimis datele pentru "header", se trimit datele imaginii utilizand functia **imagepng()** (sau "imagejpeg()", "imagegif(); in functie de tipul imaginii") care preia ca argument identificatorul de imagine (in script avem "*imagepng($im)*")
  + Ca alternativa, imaginea poate fi scrisa intr-un fisier, in loc sa fie afisata in browser o salvati pe server, pentru aceasta se adauga un al doilea parametru la "imagepng()" reprezentand numele fisierului, exemplu "*imagepng($im, "img.png")*" si nu se mai scrie functia "header()". (trebuie sa aveti permisiuni de scriere pe server)

In final s-a folosit functia **imagedestroy()** (care foloseste ca argument identificatorul de imagine) pentru eliberarea resurselor de memorie folosite la crearea imaginii.

**Utilizarea imaginilor create cu PHP**

Deoarece un header poate fi trimis numai o data pentru o pagina, si asta este singura cale de a spune browser-ului ca sunt trimise date de imagine, poate deveni mai dificil cand se doreste crearea si afisarea mai multor imagini cu PHP.

In acest caz:  
a) Puteti folosi scriptul pentru a salva imaginea pe server (dupa cum a fost explicat mai sus) si apoi folositi etichete <img> pentru afisarea imaginii  
- Exemplu:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <?php    $text = 'Textul din imagine';    // setare dimensiuni    $height = 50;    $width = 170;    $im = imagecreate($width, $height);    // setare culori    $fundal = imagecolorallocate($im, 10, 10, 250);    $text\_color = imagecolorallocate($im, 255, 255, 255);    // finalizarea imaginii    imagestring($im, 4, 10, 15, $text, $text\_color);    // salvarea imaginii pe server    imagepng($im, "img.png");    // curatarea memoriei    imagedestroy($im);    ?>    <img src="img.png" alt="Afisare 1" />    --- <img src="img.png" alt="Afisare 2" /> |

- Acest script va afisa urmatorul rezultat

|  |  |  |
| --- | --- | --- |
| **Textul din imagine** | **\_ \_ \_** | **Textul din imagine** |

b) O alta metoda este scrierea intr-un fisier php extern a scriptului care genereaza si afiseaza imaginea, apoi apelati scriptul PHP in atributul **src**, ca in exemplul urmator:

|  |  |
| --- | --- |
| 1  2 | <img src="script\_img.php" alt="Afisare  1" />  --- <img src="script\_img.php" alt="Afisare 2" /> |

- Unde "script\_img.php" este fisierul PHP care genereaza si afiseaza imaginea

**PHP - GD Library**

Functiile GD sunt folosite pentru crearea imaginilor. Verificati folosind *phpinfo()*, ca serverul de gazduire suporta aceste functii, inainte de a le utiliza.  
Pentru a insera in paginile web o imagine creata cu functiile din php folositi eticheta *img*

|  |  |
| --- | --- |
| 1 | <img src="imagine.php" alt="Imagine  creata cu GD Library"> |

In urmatorul exemplu vom genera o imagine in format JPEG, cu dimensiunile 200px x 100px, fond verde.

|  |  |  |
| --- | --- | --- |
| <?php header ("Content-type: image/jpeg"); $img = ImageCreate (200,100); $fond = ImageColorAllocate ($img,0,200,0); ImageJPEG ($img, '', 100); ?> | |  | | --- | |  | |

*header*- declara tipul imaginii  
*Content-type* poate avea valorile:  
image/jpeg - pentru JPEG  
image/gif - pentru GIF  
image/png - pentru PNG  
Cu *ImageCreate* sunt declarate dimensiunile imaginii  
Culoarea fondului imaginii este declarata in *ImageColorAllocate*, prin cele 3 culori de baza R-G-B, in format zecimal (0-255)  
*ImageJPEG*- afiseaza imaginea in browser (pentru GIF se foloseste ImageGIF, iar pentru PNG ImagePNG). Calitatea imaginii generate va fi "best quality" (100).

Adaugam exemplului anterior textul "Proba text", cu caractere albe.

|  |  |  |
| --- | --- | --- |
| <?php header ("Content-type: image/jpeg"); $img = ImageCreate (200,100); $fond = ImageColorAllocate ($img,0,200,0); $text = ImageColorAllocate ($img,255,255,255); ImageString ($img,5,10,10,"Proba text",$text); ImageJPEG ($img, '', 100); ?> | |  | | --- | | **Proba text** | |

Culoarea fontului este declarata tot cu *ImageColorAllocate*, similar culorii fondului *ImageString*contine 3 valori numerice si textul continut de imagine. Prima este marimea fontului, urmatoarele 2 sunt distantele in pixeli (coordonatele pe X si pe Y) fata de coltul stanga sus, stabilind pozitia textului in interiorul imaginii.

Cream o linie din punctul (10,90) in punctul (190,90). Referinta coordonatelor este coltul stanga sus.

|  |  |  |
| --- | --- | --- |
| <?php header ("Content-type: image/jpeg"); $img = ImageCreate (200,100); $fond = ImageColorAllocate ($img,0,200,0); $col = ImageColorAllocate ($img,255,255,255); ImageLine ($img,10,90,190,90,$col); ImageJPEG ($img, '', 100); ?> | |  | | --- | | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | |

Cu *ImageLine* creăm linia. Cele 4 valori numerice continute reprezinta coordonatele x si y pentru punctul de inceput si de sfarsit al liniei, iar ultima este declarata culoarea.

**Ce este World Wide Web?**

**Inter-rețele și Internet**

În lume există milioane de calculatoare. În aceste calculatoare sunt stocate informații. Pentru a putea face schimb de informații, calculatoarele sunt interconectate, formînd rețele de calculatoare.Multe dintre aceste rețele de calculatoare sunt conectate între ele, formînd inter-rețele (rețele de rețele de calculatoare). O rețea de rețele se numește internet („ net" înseamnă în limba engleză „rețea").  
Cea mai mare inter-rețea publică (rețea de rețele de calculatoare cu acces public) este rețeaua Internet.

**World Wide Web**

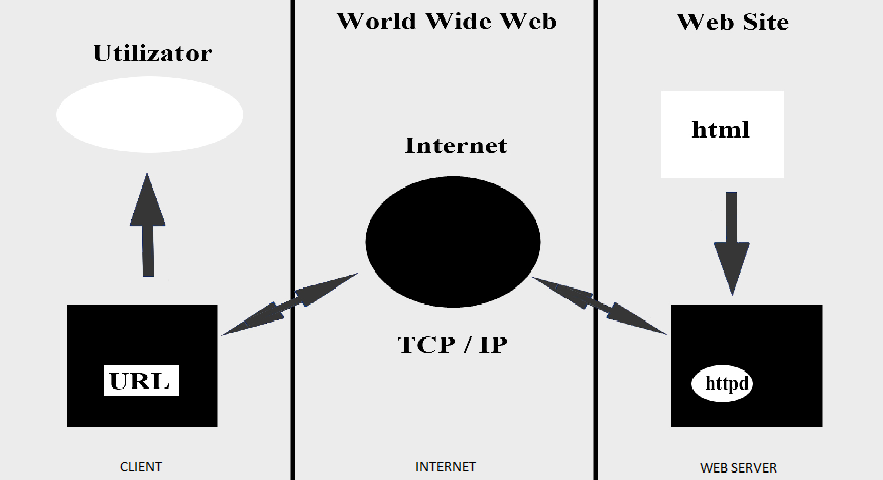
Există mai multe metode de acces (servicii) la informația stocată pe un calculator prin intermediul rețelei Internet:

1. FTP (File Transfer Protocol) este serviciul pentru transferul fișierelor;
2. Telnet este serviciul pentru accesul de la distanță la resursele unui calculator;
3. Electronic Mail este serviciul de mesagerie electronică;
4. News este serviciul de știri;

World Wide Web este o altă metodă (serviciu) de acces la informația stocată pe un calculator aflat oriunde în lume.  
World Wide Web (pe scurt - WWW) înseamnă în traducere ad-litteram „pînză de păianjen întinsă în lumea întreagă„.  
Denumirea evidențiază caracterul planetar al schimbului de informații prin intermediul rețelei Internet.

**Tehnologia client-server**

World Wide Web-ul utilizează tehnologia client-server, care constă în (vezi figura următoare):  
1. Un calculator (clientul) formulează o cerere.  
2. Cererea este expediată unui server.  
3. Cererea parcurge un mediu pînă la server.  
4. Serverul analizează cererea, o execută,formulează răspunsul și îl expediază clientului.  
5. Răspunsul parcurge mediul înapoi către client.  
6. Clientul recepționează răspunsul la cererea solicitată.

  
Fig:   Tehnologia client-server

**HTTP**

Pentru a comunica intre ele, două calculataore folosesc un sistem de reguli ce formează un *protocol*Serviciul WWW utilizează ca protocol de comunicare între client și server *HiperText Transfer Protocol* (HTTP), adică Protocolul de Transfer al Hipertextului.

**Hipertext**

Hipertextul este un text îmbogățit. El conține:*text obișnuit;* *etichete* pentru formatarea textului și încapsularea altor tipuri de informații (salturi rapide către alte resurse de informații, sunete, imagini, filme etc). Hipertextul este stocat în fișiere avînd o extensie specială: .htm sau .html

**HTML**

Un fișier ce conține hipertext este scris într-un limbaj specific numit HiperText Markup Language (HTML), adică Limbajul de Marcare a Hipertextului. HTML este un limbaj care permite inserarea de:

1. text;
2. sunete, imagini și filme;
3. indicatori de prezentare a informației;
4. legături (link-uri) către alte pagini Web aflate oriunde în lume;
5. aplicații (programe JavaScript, Java, VRML etc).

**Host**

Un calculator din Internet se numește host (gazdă).  
Pentru a fi identificat în mod unic, calculatorul primește un nume (o adresă), de exemplu, mishu.cnmv.ro

**Pachete**

Informația care circulă între calculatoare interconectate este încapsulată în pachete. Un pachet conține:

1. adresa expeditorului și adresa destinatarului;
2. informația;
3. numele aplicației client care a formulat cererea și numele aplicației de pe server care va primi cererea spre rezolvare. Pachetul este lansat în rețeaua Internet.

**TCP/IP**

Rețeaua Internet dispune de mijloace de dirijare a pachetelor astfel încît acestea să ajungă la destinație.

- Un astfel de mijloc de dirijare a pachetelor este Internet Protocol (IP), adică Protocolul Internet. Rețeaua Internet dispune de mijloace de corecție a erorilor de transmitere a pachetelor.  
-Un astfel de mijloc de corecție este Transfer Control Protocol (TCP), adică Protocolul de control al transferului.

**Pagina Web**

în orice calculator informația este stocată în unități numite *fișiere*.  
Aceste fișiere conțin;

1. text;
2. programe;
3. imagini;
4. filme;
5. sunete etc.

Pentru WWW sunt importante anumite fișiere speciale, numite și pagini Web.Acestea au extensia .htm sau .html.

**Browser**

Paginile Web sunt vizualizate pe calculatorul client prin intermediul unei aplicații speciale numite browser. Browser-ul permite schimbarea paginii vizualizate prin intermediul:

1. unei ferestre speciale, numită Addresss;
2. meniului File, Open, Browse;
3. legăturilor afișate în pagina Web. Cele mai utilizate browsere sunt:
4. Netscape Communicator;
5. Internet Explorer;
6. HotJava.

**Site Web**

O mulțime organizată de pagini Web formează un site Web. în WWW, clientul solicită pagini Web de la un site Web.

**httpd**

*HiperText Transfer Protocol Demon* (httpd) este o aplicație care se execută pe serverul Web pentru a prelucra cererile de pagini Web recepționate de acesta de la clienți.

**Home Page**

Home Page (pagina gazdă) este o pagină din site-ul unei organizații care:  
\* este în mod uzual primă pagină accesată din site;  
\* este o pagină de prezentare a organizației;  
\* oferă modul cel mai eficient de explorare a informațiilor aflate în site.

**Server Web**

Un server Web este un calculator care adăpostește un Site Web și care este capabil să răspundă la cereri de pagini Web din partea unui client.  
Pentru a putea răspunde permanent cererilor Web, un calculator trebuie să ruleze permanent o aplicație specială: httpd. Cele mai întîlnite servere Web sunt:  
-Apache Server;  
-Microsoft Web Server;  
-Oracle Web Server.

**URL**

Un utilizator al serviciului WWW poate solicita vizualizarea prin intermediul browse-rului a unei pagini Web care poate fi localizată pe un calculator aflat oriunde în lume. O pagină Web este unic» deteiminată prin adresa URL asociată.  
Uniform Resource Locator (URL) înseamnă Identificator Universal al Resurselor în Internet. Un exemplu de URL este: http: //www.cnmv.ro/index.html

1. Adresa URL a paginii curente vizualizate de browser apare în fereastra Address.
2. Un URL precizează:
3. metoda de acces la pagină - de exemplu: http;
4. calculatorul pe care se află - de exemplu: www.cnmv.ro;
5. numele paginii - de exemplu: index.html.

**Avantajele serviciului World Wide Web**

Paginile Web au următoarele caracteristici:

1. sunt multimedia, adică ele conțin informații sub formă de text, imagini, sunete, filme etc;
2. sunt interactive, adică răspund la cererile utilizatorului;
3. sunt independente de platforma hardware și software, adică se văd la fel pe orice calculator, avînd instalat orice sistem de operare și utilizînd orice browser.

**Crearea unei pagini Web**

Crearea unei pagini Web presupune:  
1. Editarea fișierului HTML utilizînd:

1. un editor de texte obișnuit (Notepad, Wordpad, Word, WordPerfect etc.)
2. un editor de texte dedicat (Netscape Composer, HotMetal etc.)

2. Salvarea paginii Web cu extensia .html sau .htm înir-un site Web;  
3. Rezolvarea referințelor conținute în pagina Web (legături, imagini, sunete, filme, aplicații Java etc).

Pagina Web astfel creată poate fi vizualizată prin intermediul unui browser.

**Greșeli de web design**

În aceasta temă vom vorbi despre greșelile uzuale pe care le fac designerii web, fară a atinge însă problema folosirii stilurilor CSS in locul tabelelor.  
Am putea face chiar un top 10 al celor mai frecvente greșeli, fară a da însă exemple sau a aminti firme de web design de renume care fac subiectul acestei teme.

**1.Crearea de pagini web aglomerate**

O pagină web trebuie să ofere un balans între spațiul ocupat de informații (text, imagini, meniuri) si spațiul alb sau spațiul negativ (fundal, distantele dintre paragrafe si alte elemente ale paginii, locurile goale). O pagina web trebuie sa arate aerisit, ușor de parcurs cu vederea, simpla. Vizitatorul trebuie sa poată citi printre rînduri si sa găsească ușor informațiile care îl interesează. Soluția pentru a evita această greșeală este să folosiți "padding" si "margin" cînd creați stilul paginii, pentru a spatia coloanele, paragrafele, imaginile de text, etc.  
Se pare însă că tendința în web design-ul romanesc este exact inversa: paginile sunt aglomerate, textul este înghesuit în imagini pentru a face loc publicității online. Sunt greu de parcurs, obositoare si lungi.

**2.Informațiile sunt greu de găsit in pagina web**

Aceasta este o greșeală frecvent întîlnită pe multe pagini. Vizitatorul intra pe un site cu un scop: să se informeze. Daca în primele secunde după intrarea pe site el nu va găsi informațiile promise (prin titlu, prin URL, prin descrierea din pagină care face referință la site-ul respectiv) va părăsi site-ul. Simplu.  
Elementele importante ale unei pagini web trebuiesc scoase în evidență. Porțiunile din text care oferă esența informațiilor trebuiesc de asemenea scoase în evidentă.

**3.Linkurile sunt greu de găsit în pagina web**

Linkurile sunt cele mai importante elemente ale unei pagini din punctul de vedere al unui vizitator. Daca acestea nu pot fi găsite ușor, vizitatorul poate renunța la navigarea pe site-ul respectiv. Exista doua principii de care trebuie sa ținem cont atunci cînd realizam o pagina web:

1. linkurile trebuiesc deosebite de restul textului prin folosirea unei alte culori sau a sublinierii;
2. trebuie evitat ca textul care nu este link sa arate ca un link (aceeași culoare sau subliniere), altfel vizitatorul se va simți păcălit.

**4.Folosirea unui format de imagine incorect**

Formatele standard pentru imagini folosite pe site-urile web sunt JPEG, PNG sau GIF. Folosirea altor formate pot duce la încărcări grele ale paginii web. Există însă și exemple de pagini care deși folosesc aceste formate corecte, timpii de încărcare sunt foarte mari pentru imagini. Problemele sunt diverse, însă cele mai comune sunt imaginile nepregătite web (redimensionate și salvate pentru web) sau folosirea de imagini de dimensiuni improprii ca fundal pentru elementele paginii.

**5.Titlul paginii este lasat cel implicit**

In Dreamweaver titlul implicit al unei pagini este "Untitled Document". Omiterea schimbării titlului este o greșeală de neiertat chiar și pentru cel mai neexperimentat web designer. Credeți că nu există designeri care să facă această greșeală? Dați o căutare pe Google după cuvîntul cheie "Untitled Document" si veți rămîne surprinși să găsiți aproape 30 de milioane de rezultate.

**6.Meniu greu de folosit sau de găsit**

Există o tendință în web design de a ascunde meniul în elemente grafice din pagină sau în elemente din cadrul unei imagini. Deși aceasta tendință este interesantă din punct de vedere artistic, din punct de vedere funcțional nu este potrivită. Meniul trebuie sa fie ușor accesibil vizitatorului, singura opțiune care se acceptă la un număr mare de pagini fiind meniurile de tip "drop-down".

**7.Textul are mărimea fixă**

Vizitatorul ar trebui să poată redimensiona textul după cum dorește. Definirea textului ca mărime în pixeli va păstra dimensiunea acestuia fixă. Soluția ar fi ca definirea să se facă utilizînd atributul EM.

**8.Scopul site-ului nu este clar**

în primele secunde după încărcarea paginii web vizitatorul trebuie să își dea seama despre ce este vorba în pagina pe care a ajuns. Acest lucru poate fi obținut prin folosirea unor titluri și subtitluri sugestive pentru paragrafe. Există însă destule pagini web pentru care trebuie să pierzi timp ca vizitator doar pentru a afla despre ce sunt acestea.

**9.Folosirea unor culori stringente**

Culorile stringente, sau combinațiile de culori nepotrivite pot obosi ochiul uman, sau pot distrage atenția de la informațiile cu valoare din pagina web. Doua lucruri importante trebuie avute în vedere la alegerea setului de culori pentru site: combinația trebuie sa fie plăcuta ochiului si sa nu utilizeze mai mult de 3 culori de baza (restul fiind doar nuanțe ale acestora).

**10.Integrarea reclamelor în conținutul paginii**

Reclamele și publicitatea online trebuie menținute separat de conținutul informativ al paginii. Vizitatorul trebuie să își dea seama ușor care sunt banerele sau zonele cu publicitate online, si care sunt imaginile sau linkurile care fac parte din conținutul paginii. Amestecarea reclamelor cu conținutul va deruta vizitatorul si îl va face să părăsească site-ul.

Acestea ar fi principalele greșeli făcute de web designeri. Exista însă și cazuri speciale, excepții, și acestea nu pot fi considerate greșeli. Spre exemplu, site-urile de fotografie pot încărca imagini in format TIFF sau BMP din diverse motive: exemple, teste, etc. Bineînțeles ca nu putem considera ca designerul a făcut o greșeală ce trebuie corectată.

© REALIZARE ANDREI CIOBANU 2009-2020