3. ELEMENTE FUNDAMENTALE ALE LIMBAJULUI PHP (II)

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

3.1 STRUCTURI DE CONTROL AL EXECUTIEI

3.1.1 Descriere generala

Daca nu ar exista structuri de control, instructiunile unui program s-ar executa secvential (in ordinea in care apar in codul sursa) pana la terminarea acestuia. In programare este nevoie insa de posibilitatea de a lua decizii in functie de valori ale variabilelor, de a executa repetat portiuni de cod (de exemplu, pentru parcurgerea si prelucrarea unei succesiuni de valori) etc. , de unde si necesitatea instructiunilor de control al executiei.

PHP dispune de mai multe instructiuni de control al executiei, dintre care in acest material vor fi detaliate:

- **instructiuni decizionale** – permit executarea unor portiuni de cod diferite in functie de conditii stabilite de catre programator
- **instructiuni repetitive** – ofera modalitati de a executa o secventa de instructiuni in mod repetat, cu numar de repetitii controlat
- **instructiuni pentru includerea unui fisier intr-un script PHP** – ofera posibilitatea refolosirii codului si a structurarii unei aplicatii PHP formate din mai multe fisiere

In plus fata de acestea, PHP dispune si de un sistem de exceptii (propagarea automata si tratarea exceptiilor), care tine tot de controlul executiei, dar care va fi discutat la lectia despre clase si obiecte.

3.1.2 Instructiuni decizionale

3.1.2.1 Prezentare generala

Instructiunile decizionale permit executia conditionata a uneia sau mai multor sectiuni de cod in program in functie de valoarea unei expresii in care participa date alese de catre programator. De exemplu, daca intr-un formular web utilizatorul trebuie sa-si introduca CNP-ul, programul care prelucreaza datele primite de la utilizator va verifica validitatea CNP-ului receptionat si va lua o decizie: daca cel introdus este invalid, se va executa o sectiune de cod care afiseaza utilizatorului o eroare; daca este corect, se va executa o alta sectiune de cod care realizeaza prelucrarea restului datelor din formular.

3.1.2.2 Instructiunea decizionala simpla

Permite executia conditionata a una sau doua sectiuni de cod in functie de valoarea unei expresii ce are tipul de date boolean (si deci doar doua valori posibile: true sau false).

Iata formele posibile ale instructiunii:

- executia conditionata a unei portiuni de cod:

```
if(expresie_boolean){
      set_instructiuni_executat_numai_daca_expresia_are_valoarea_true;
}
```

Exemplu:

```
if( $a%2 == 0 ){  // daca restul impartirii lui $a la 2 este 0
      echo "Numarul $a este par!";
      echo "Felicitari!";
}
```

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

- executia a doua portiuni de cod diferite in functie de valoarea expresiei:

```
if(expresie_boolean){
      set_instructiuni_executat_in_cazul_in_care_expresia_are_valoarea_TRUE;
}else{
      set_instructiuni_executat_in_cazul_in_care_expresia_are_valoarea_FALSE;
}
```

Exemplu:

```
if( $a%2 == 0 ){                    // daca restul impartirii lui $a la 2 este 0
      echo "Numarul $a este par!";
      if($a%4 == 0){                // putem avea instructiuni if imbricate
            echo "Mai mult, numarul $a este divizibil cu 4!";
      }
}else{                              // in caz contrar inseamna ca $a este impar
      echo "Numarul $a este impar!";
}
```

- inlantuirea mai multor instructiuni decizionale:

```
if(expresie_boolean_1){
      instructiuni_executate_in_cazul_in_care_expresia_1_are_valoarea_TRUE;
} elseif (expresie_boolean_2){
      instructiuni_executate_daca _expresia_1_rezulta_FALSE_si_expresia_2_rezulta_TRUE;
}else{
      instructiuni_executate_daca _expresia_1_rezulta_FALSE_si_expresia_2_rezulta_FALSE;
}
```

Exemplu:

```
if( $optiune_user == 1 ){
      echo "Ati ales optiunea 1";
}elseif( $optiune_user == 2){
      echo "Ati ales optiunea 2";
}else echo "Optiune invalida";
```

Atentie! *Expresia in functie de care se ia decizia este convertita la boolean, conform regulilor din tabelul de conversie!*

```
$a = "php"; $b = 4;     // 2 tipuri de date diferite: string, int, boolean
if($a){     // conversia la boolean a unui string care nu e nici "" nici "0" da true
      echo 'if($a) se evalueaza true!';
}
if($b){     // conversia la boolean a unui int nenul da true
      echo 'if($b) se evalueaza true!';
}
```

Expresia in baza careia se ia decizia poate fi una oricat de complexa, creata folosind operatori logici si de comparare:

```
$a = 4; $b = 6; $c = 8;
if( ($a < $c) && ((($c-$b)==($b-$a)) || ($c<$a) ) echo "TRUE!";
```

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

3.1.2.3 Instructiunea decizionala multipla

Este folosita atunci cand vrem ca pentru diferite valori ale unei expresii sa executam sectiuni diferite de cod. Comparativ cu instructiunea decizionala simpla, aici expresia nu mai este fortata sa ia doar doua valori posibile (true/false), ci putem trata separat cazul fiecarei valori de interes.

```
switch(expresie){
      case valoare1:    set_instructiuni_1;
                        break;
      case valoare2:    set_instructiuni_2;
                        // am omis break - se vor executa instructiunile pana la aparitia
                        // urmatorului break
      case valoare3:    set_instructiuni_3;
                        break;
      default:          set_instructiuni_default; // aceasta sectiune "prinde" toate
                        break;// celelalte valori ale expresiei care nu apar in case-uri
}
```

Atentie!

- *expresia este comparata cu valorile folosind operatorul ==, asadar ea este subiect de conversie conform tabelului!*
- *valorile din case pot fi numai scalar! (nu sunt permise tablouri, obiecte etc)*

Exemplu:

```
$a = "text";
switch ($a){
      case "alt text":  echo "Aceasta instructiune nu este executata";
                        break;
      case 0:           echo '$a == 0, asadar aceasta instructiune se executa!!';
                        break;
      case 'text':      echo 'Aceasta instructiune nu se mai executa, din cauza case-ului
precedent.';
                        break;
}
```

3.1.3 Instructiuni repetitive

3.1.3.1 Generalitati

Instructiunile repetitive sunt folosite atunci cand dorim executia repetata a unei portiuni de cod, in mod controlat:

- pana cand o anumita conditie este indeplinita sau pana cand inceteaza sa mai fie adevarata (ex: interogam o baza de date si obtinem un set de inregistrari; cat timp mai sunt inregistrari disponibile, extragem urmatoarea inregistrare din set)
- de un anumit numar de ori (ex: dorim sa calculam datele urmatoarelor 8 zile care pica joi)

3.1.3.2 Instructiunea while

Instructiunea while executa in mod repetat un set de instructiuni atata timp cat valoarea unei expresii este true. Inainte de fiecare noua executie este verificata valoarea expresiei.

```
while(expresie_boolean){      // ATENTIE! expresia este convertita la boolean!
      set_instructiuni_executat_cat_timp_expresia_este_TRUE;
}
```

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

Exemplu:

```
$i = 1;
while($i < 3 )          // Se executa 2 repetitii (pentru $i = 1 si $i = 2)
      echo $i * 3;      // La prima repetitie va afisa 3, la urmatoarea 6
      $i++;             // Fara aceasta linie, repetarea se facea la infinit, pentru ca
}                       // $i ar fi ramas permanent <3
```

3.1.3.3 Instructiunea do…while

Instructiunea do…while actioneaza asemanator cu while, cu diferenta ca verificarea valorii expresiei se face **dupa** fiecare executie a setului de instructiuni:

```
do{
      set_instructiuni_executat_cat_timp_expresia_este_TRUE;
} while(expresie_boolean); // ATENTIE! expresia este convertita la boolean!
```

In consecinta, do….while va executa intotdeauna cel putin o iterare (repetitie), pe cand while este posibil sa nu execute nici una daca valoarea expresiei este false de la bun inceput.

Exemplu: incercam sa generam un multiplu de 13, ales aleator intre 0 si 100:

```
do{
      $i = rand(1,100); // se genereaza un nou numar aleator intre 0 si 100 inclusiv
} while($i%13);         // repetam operatiunea cat timp $i nu se divide exact cu 13
echo $i;                // afisam numarul gasit
```

3.1.3.4 Instructiunea for

Instructiunea for este o instructiune mai complexa, ce cuprinde 3 sectiuni (vezi desenul de mai jos):

- sectiunea 1 - se executa o singura data, la intrarea in instructiunea for. Este folosita in general pentru initializarea variabilelor folosite in cadrul buclei for
- sectiunea 2 – contine o expresie care se evalueaza inaintea fiecarei repetitii. Rezultatul ei este de tip boolean; daca valoarea sa este true, se mai efectueaza inca o repetitie, in caz contrar se iese din bucla for
- sectiunea 3 – se executa la sfarsitul fiecarei repetitii, dupa executarea corpului buclei for (setul de instructiuni continut intre acolade). Este folosita in general pentru modificarea variabilelor ce participa in bucla for.

```
          sectiunea 1          sectiunea 2                    sectiunea 3
for(instructiuni_start;expresie_boolean; set_instructiuni_executat_dupa_fiecare_iterare){
      set_instructiuni_executat_in_mod_repetat;
}
```

Instructiunea for este folosita de obicei atunci cand cunoastem numarul de repetii dorit:

```
for($i = 1; $i<20; $i++){
      echo "Valoarea curenta a lui \$i este: $i";
}
```

In exemplul de mai sus, se executa intai prima sectiune din for, efectul fiind crearea variabilei $i de tip int si initializarea ei cu valoarea 1. In continuare se evalueaza expresia $i<20 (rezultat: true), iar apoi se executa

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

instructiunile din interiorul buclei for – in cazul nostru, echo. Dupa echo este executata sectiunea a 3-a din for, care are ca efect incrementarea lui $i, iar de aici lucrurile se repeta in acelasi fel incepand cu evaluarea expresiei $i<20, executia lui echo, incrementarea lui $i etc pana cand $i ajunge la 20, moment in care $i<20 se va evalua false, si executia buclei for se incheie, trecandu-se la instructiunea imediat urmatoare for-ului.

3.1.3.5 Instructiunea foreach (parcurgere de tablouri)

In plus fata de constructiile repetitive traditionale, in PHP mai exista una dedicata iterarii prin elementele unui tablou – **foreach**:

```
$a = array('a','b','c','d');
foreach ( $a as $litera) { echo "$litera "; } // a b c d
// SAU
foreach ($a as $index => $litera) { echo "$index=>$litera"; } // 0=>a 1=>b 2=>c 3=>d
```

Detalierea acestei instructiuni va fi facuta la lectia despre tablouri.

3.1.3.6 Incheierea prematura (fortata) a executiei unei instructiuni repetitive

Exista posibilitatea ca iesirea dintr-o bucla de tip while, do…while sau for sa se faca fortat (inainte ca valoarea expresiei testate la fiecare iterare sa devina false) folosind instructiunea **break**. In cazul instructiunilor repetitive imbricate, aceasta poate primi ca argument un numar care indica numarul de bucle din care se iese:

```
while(true){
      for($i = 0; $i < 10; $i ++){
            $j = rand(1,10);      // generam un numar aleator intre 1 si 10
            if($i == $j) break 2; // se iese din ambele bucle si se afiseaza "Gata while"
            if($i == ($j-2 )) break;// se iese din bucla for si se afiseaza "Gata for"
      }                             // dupa care bucla while continua sa itereze
      echo "Gata for";
}
echo "Gata while";
```

3.1.3.7 Incheierea fortata a iterarii curente

Pentru cazurile in care dorim ca executia sa "sara" direct la sfarsitul unei repetitii si sa continue cu urmatoarea, exista instructiunea **continue**. Ca si break, aceasta instructiune poate primi un argument ce specifica, in cazul buclelor imbricate, care este cea care se doreste continuata.

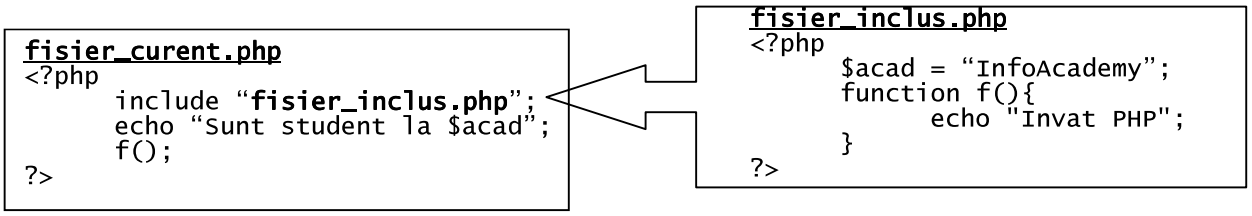
```
for($i = 0; $i < 5; $i++)
      if ( ( $i % 2 ) == 0 ) continue;    // daca $i este par se sare instructiunea echo
      echo "$i! ";
}                  // bucla for va afisa 1! 3! (0,2 si 4 se sar din cauza lui continue)
```

Atentie! *Cand se folosesc break si continue cu parametru, instructiunea switch este numarata ca una dintre bucle!*

3.1.4 Includerea unui alt fisier in fisierul PHP curent

PHP ofera instructiuni care permit includerea unui fisier (indiferent de ce contine acesta) in fisierul PHP curent, in momentul executiei sale. Acest lucru este foarte util, de exemplu, atunci cand programatorul a definit un set de functii si doreste sa le aiba disponibile in mai multe fisiere PHP; functiile pot fi plasate intr-un fisier PHP separat, care mai apoi este inclus in toate celelalte fisiere care necesita functiile in cauza.

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.



Exista 4 instructiuni folosite pentru astfel de incluziuni:

- **include** – include fisierul specificat in cel curent, in punctul in care este plasata instructiunea include:
  - daca fisierul inclus contine (si) cod PHP delimitat corect, acesta este executat
  - daca fisierul inclus nu exista sau nu poate fi citit, este emis un warning si executia scriptului curent continua cu instructiunea de dupa include
- **require** – identic cu include, cu o singura exceptie: daca fisierul nu poate fi inclus, require genereaza eroare fatala si opreste executia scriptului curent
- **include_once** – ca include, insa include fisierul doar daca acesta nu a fost inclus anterior
- **require_once** – ca include_once, insa cu eroare fatala daca fisierul nu poate fi citit

Aceste instructiuni nu sunt functii, ci constructii de limbaj, de aceea la apelarea lor argumentul poate fi sau nu inclus intre paranteze:

```
include "functii.php";
include ("functii.php");
```

Atentie! *Interpretorul iese din mod PHP atunci cand include un fisier, de aceea este imperativ ca, daca fisierul inclus contine cod PHP, acesta sa fie inclus intre taguri de delimitare.*

Instructiunile prezentate pot fi folosite pentru a include si fisiere care nu contin cod PHP; interpretorul PHP nu va incerca sa interpreteze continutul fisierului inclus decat in masura in care intalneste taguri de delimitare PHP valide. Exemplu: programatorul doreste ca paginile unui site sa aiba un header HTML predefinit, si pentru aceasta salveaza codul HTML corespunzator headerului intr-un fisier separat pe care il include in fisierele PHP ale site-ului:

index.php

```
<?php
include "header.html";
$a = rand(1,49);
echo "Numarul dvs norocos este $a!";
?>
```

header.html

```
<html><body>
<h1> Welcome to our site! </h1>
<hr>
```

3.2 VARIABILE – NOTIUNI AVANSATE

3.2.1 Domeniul de vizibilitate al unei variabile

O variabila poate fi definita:

- in interiorul unei functii – o astfel de variabila este o variabila locala si este vizibila numai intre acoladele functiei ce-o cuprinde
- in contextul global - variabila este definita in "radacina" scriptului PHP , in afara oricarei functii sau definitii de clasa. Astfel de variabile NU sunt automat vizibile in interiorul functiilor, iar incercarea de a folosi acelasi nume de variabila in interiorul unei functii va duce la crearea si modificarea unei variabile

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

locale, fara a afecta valoarea variabilei globale cu acelasi nume. Pentru ca o variabila globala sa devina vizibila in interiorul unei functii si sa poata fi citita/modificata din acest context, este necesara "importarea" ei folosind cuvantul cheie global:

```
$a = 5; $b = 10;  // crearea a doua variabile in contextul global
function f(){
      global $a;  // "importarea" variabilei globale $a
      $a++;       // este modificata variabila globala $a!
      $b = 15;    // este modificata variabila LOCALA $b
}
echo $a;          // 6 - variabila globala $a a fost modificata in interiorul functiei
echo $b;          // 10 - variabila globala $b nu a fost modificata
```

- predefinita, de tip "superglobal" – este automat vizibila in toate contextele programului (ex: $_SERVER, $_POST, $_GET etc)

3.2.2 "Variable variables"

PHP da programatorului posibilitatea de a tine numele unei variabile (sau al unei functii) intr-o alta variabila:

```
$var = "curs"; $curs="PHP";
echo "Invat ".$$var; // Invat PHP (echivalent cu a scrie "Invat $curs")
```

3.2.3 Referinte

3.2.3.1 Asignarea prin valoare

O variabila reprezinta o zona de memorie care poate fi accesata prin inetrmediul unui nume. Atunci cand folosim operatorul de asignare pentru a memora intr-o variabila valoarea alteia, are loc o copiere a valorii dintr-o zona de memorie in cealalta:

```
$a = 4;     // se creeaza o zona de memorie cu numele $a si care stocheaza valoarea 4
$b = $a;    // se creeaza o ALTA zona de memorie si se copiaza in ea valoarea din prima
$b++;       // se modifica cea de-a doua zona de memorie; acum avem aici valoarea 5
echo $a;    // prima zona de memorie (cea a lui $a) ramane nemodificata
```

3.2.3.2 Asignarea prin referinta

In PHP exista si posibilitatea **asignarii prin referinta** – efectul fiind ca cele doua variabile devin "conectate": cele doua nume acceseaza aceeasi locatie din memorie. In aceste conditii, orice modificare adusa uneia dintre variabile o va afecta si pe cealalta (de fapt, zona de memorie care sufera modificari este una singura, dar este referita prin doua nume):

```
$a = 4;     // se creeaza o zona de memorie in care se stocheaza valoarea intreaga 4
$b =& $a;   // se creeaza un al doilea nume pentru acea zona de memorie
$b++;       // este folosit al doilea nume pentru a modifica continutul zonei de memorie
echo $a;    // 5 - continutul zonei de memorie a fost modificat
echo $b;    // 5 - nu conteaza cu ce nume accesam zona de memorie, ea este una singura
```

Pentru o explicatie detaliata (dar de nivel avansat) a mecanismului referintelor in PHP, puteti consulta http://www.derickrethans.nl/files/phparch-php-variables-article.pdf

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

3.3 FUNCTII

3.3.1 Ce este o functie

O functie reprezinta un bloc de instructiuni caruia i se da un nume, avand posibilitatea de a primi date de intrare, de a le prelucra si de a produce date de iesire. Odata definita, functia poate fi apelata ori de cate ori este nevoie folosind numele sau, pasandu-i-se date de intrare diferite si putand produce, in consecinta, rezultate de iesire diferite.

Functiile sunt una dintre modalitatile fundamentale de a refolosi cod: in loc sa scriem aceeasi secventa de instructiuni de mai multe ori in cadrul programului nostru, vom defini o functie ce contine acea secventa de instructiuni si vom apela functia in mod repetat folosind numele sau.

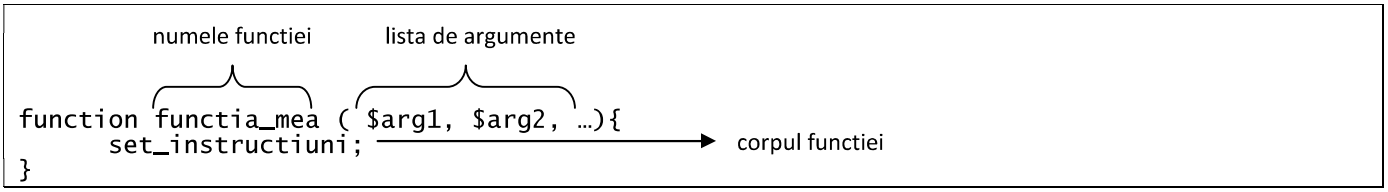
3.3.2 Tipuri de functii PHP

Din punct de vedere al provenientei lor, distingem doua tipuri de functii in PHP:

- **functii predefinite** – sunt functii scrise de catre creatorii interpretorului sau ai extensiilor PHP si care sunt deja disponibile programatorului PHP, fara ca acesta sa ia vreo masura speciala. Documentatia acestor functii se gaseste in PHP Manual, grupata pe sectiuni (ex: functii pentru lucrul cu tablouri, functii pentru MySQL etc).
- **functii definite de catre programatorul PHP** – sintaxa de definire a unei functii este prezentata in continuare

3.3.3 Definirea unei functii

O functie se defineste cu urmatoarea sintaxa:



Observatie: *spre deosebire de alte limbaje, la definirea unei functii PHP nu se mai specifica tipul de date al valorii returnate.*

Numele functiei urmeaza reguli asemanatoare cu cele pentru numele de variabile: incepe cu litera sau underscore si poate contine doar litere, cifre si underscore.

Argumentele sunt variabile care pot fi folosite in interiorul functiei si care memoreaza datele sale de intrare. Valorile argumentelor sunt specificate la apelarea functiei. Functia poate fi apelata de mai multe ori cu valori diferite ale argumentelor, ceea ce inseamna ca acelasi set de instructiuni (corpul functiei) va prelucra date diferite.

Exemplu:

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

```
// o functie care afla daca un numar se divide cu altul
function divizibil($a1, $a2){
      if( $a1%$a2 == 0 ) // daca restul impartirii primului numar la cel de-al doilea e 0
            echo "Numarul $a1 se divide cu $a2!";
      else
            echo "Numarul $a1 NU se divide cu $a2!";
}
divizibil(18,6);  // apelarea functiei; afiseaza: "Numarul 18 se divide cu 6!"
                  // in functie, $a1 a primit valoarea 18, iar $a2 valoarea 6
```

Setul de argumente specificat la definirea unei functii impune restrictii asupra felului in care va fi ea apelata (folosita) mai apoi: daca la definire s-au specificat doua argumente obligatorii, functia trebuie apelata folosind cel putin doua argumente (vezi mai jos sectiunea despre apelarea unei functii).

Odata definita, o functie are vizibilitate globala (in orice context al scriptului PHP).

Incepand cu PHP 4, functiile NU trebuie neaparat declarate *inaintea* folosirii lor, cu exceptia cazului in care definirea lor este conditionata (ex: functie definita in cadrul unei instructiuni conditionale if).

3.3.4 Apelarea unei functii

3.3.4.1 Sintaxa si posibilitati

Apelarea unei functii se face folosind numele acesteia si specificand intre paranteze valorile argumentelor (daca exista), care pot fi valori constante, variabile, constante sau expresii:

```
f();                    // apelarea unei functii fara argumente
divizibil(15,5);        // apelarea unei functii cu doua argumente

$a = 4; $b = 5; define('CT', 4);

divizibil($a, CT);      // apelarea functiei folosind ca argumente variabile si constante
divizibil($a+$b, 3);    // apelarea functiei folosind o expresie ca argument
divizibil($a, rand(1,5));     // folosirea rezultatului unei alte functii ca argument
```

Numarul de argumente folosite la apelarea functiei trebuie sa fie cel putin egal cu numarul de argumente specificate la definirea acesteia. In caz contrar, este emis un warning si argumentele lipsa primesc valoarea null:

```
function suma($x, $y){
      echo $x+$y;
}
suma(5);
// Warning:  Missing argument 2 for suma() […]
// Notice:  Undefined variable: y […]
/* (cele doua mesaje contineau si alte informatii insa a fost retinuta doar portiunea
relevanta pentru discutia de fata) */
```

Observatie: *chiar daca o functie este gandita sa lucreze cu numere (ca in exemplul de mai sus), la apelare ea poate primi orice fel de argumente, de aceea programatorul trebuie sa se asigure de tipul corect de date al argumentelor primite inainte de a initia prelucrarea acestora. Functii utile in acest sens sunt is_integer(), is_boolean() etc (vezi lista completa la tipuri de date)*

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

3.3.4.2 Pasarea argumentelor prin valoare

Cand la apelarea unei functii se paseaza ca argument o variabila, ceea ce primeste functia este de fapt o copie a valorii variabilei, astfel incat functia nu poate modifica variabila primita ca argument:

```
function f($a){ $a++; } // $a este o variabila locala a functiei
$b = 5;     // se aloca o zona de memorie cu numele $b si initializata cu valoarea 5
f($b);      /* valoarea lui $b este copiata in variabila locala $a din interiorul
            functiei; functia lucreaza astfel pe o copie a lui $b, fara a afecta $b. Este
            echivalent cu $a = $b (asignare prin valoare) */
echo $b;    // 5, deoarece ceea ce se incrementeaza este $a, variabila locala
```

3.3.4.3 Pasarea argumentelor prin referinta

Atunci cand dorim ca o variabila pasata ca argument unei functii sa fie modificata in interiorul functiei, putem defini functia sa faca pasarea argumentelor prin referinta:

```
function modifica( &$a){ $a += 10; }
$b = 5;
modifica($b); /* variabila locala nu mai primeste o copie a valorii lui $b, ci i se
                 asigneaza prin referinta variabila $b.Este ca si cum am scrie $a =& $b; */
echo $b;      /* 15 - zona de memorie cu numele $b a fost modificata prin intermediul
              celui de-al doilea nume primit in interiorul functiei ($a) */
```

Atentie! *Daca functia accepta un argument prin referinta, la apelarea ei nu se poate folosi pe pozitia acelui argument o constanta, expresie etc, ci doar o variabila!*

```
function g(&$y) {
      $y++;
}
g($a);      // Functioneaza, argumentul e nume de variabila
g(5);       // Nu este permis - argumentul poate fi doar variabila,
g($a+17);   //    nu expresie sau constanta
```

3.3.5 Accesarea variabilelor globale in interiorul unei functii

Variabilele globale nu sunt vizibile automat in interiorul unei functii. Daca se incearca in interiorul functiei folosirea unei variabile cu acelasi nume ca una globala, va fi creata o variabila locala si toate operatiile se vor efectua asupra acestei variabile, fara a o afecta pe cea globala. Exista doua modalitati de a actiona asupra variabilelor globale din interiorul functiei:

- prin intermediul variabilei predefinite $GLOBALS, care este un tablou asociativ cu variabilele globale definite (vezi tablouri)
- "importand" variabila dorita in interiorul functiei

```
$a = "sunt o variabila globala";
function f(){
      global $a;                    // echivalent cu $a = &$GLOBALS['a'];
      $a = "incalzirea globala";    // este modificata variabila globala
}
function g(){
      $GLOBALS['a'] = 45;
}
f(); echo $a;     // "incalzirea globala" - $a a fost modificat in cadrul functiei f()
g(); echo $a;     // 45 - $a a fost modificat si in interiorul functiei g()
```

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

3.3.6 Argumentele unei functii

3.3.6.1 Argumente cu valori default

Programatorul poate specifica valori default pentru argumente, moment in care ele devin optionale la apelarea functiei. Valoarea default trebuie sa fie o expresie constanta.

```
function suma($a, $b, $c=0){  // daca la apelare functia are doar 2 arg, $c devine 0
      echo $a+$b+$c;
}
$x = suma(1,2,3);             // 6; in functie, $c are valoarea 3, furnizata la apelare
$y = suma(1,2);               // 3; in functie, $c are valoarea default (0)
```

Incepand din PHP 5, este posibil ca si argumentele pasate prin referinta sa aiba valori default. Daca, in acest caz, este pasata o variabila inexistenta ca argument la apelarea functiei, variabila in cauza va fi creata:

```
function f(&$x=4){…}
f($a);
echo $a;    // 4 – variabila $a a fost creata automat
```

3.3.6.2 Functii cu numar variabil de argumente

O functie poate fi apelata folosind un numar variabil de argumente (ex: la o prima apelare folosim un argument, la.urmatoarea apelare 3 argumente, apoi din nou unul etc). PHP pune la dispozitia programatorului cateva functii predefinite cu ajutorul carora acesta poate determina, in corpul unei functii, cu cate argumente a fost apelata functia si care sunt acelea:

- **func_num_args()** – returneaza numarul de argumente pasate functiei
- **func_get_args()** – returneaza lista de argumente pasate functiei
- **func_get_arg($pozitie)** – returneaza argumentul de pe pozitia specificata

Exemplu de folosire a conceptului: scrierea unei functii care realizeaza media unei liste de numere cu lungime variabila, primita sub forma de argumente:

```
function medie(){ // nu impunem apelarea cu argumente, insa verificam inauntrul functiei
      $nr_args = func_num_args();
      if($nr_args){
            $suma = 0;  // variabila in care construim suma numerelor
            for($i=0; $i<$nr_args; $i++){
                  $suma+=func_get_arg($i); // adaugam numarul (argumentul) curent la suma
            }
            return $suma/$nr_args;         // intoarcem rezultatul
      }else
            return FALSE;                  // semnalizam faptul ca lista de argumente
}                                          // este incorecta
```

3.3.7 Returnarea unei valori: instructiunea *return*

3.3.7.1 Utilizare

Instructiunea return poate fi folosita pentru a iesi din functia curenta, cu posibilitatea returnarii unei valori. Daca o functie nu contine return, iesirea din ea va avea loc dupa incheierea executiei instructiunilor ce constituie corpul functiei.

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

Instructiunea return poate fi folosita in cadrul functiei in doua moduri:

- fara argumente – cauzeaza iesirea fortata din functie si continuarea executiei cu linia imediat urmatoare apelului de functie
- cu argument – argumentul poate fi o variabila sau expresie si reprezinta valoarea returnata de catre functie, care poate fi mai apoi asignata unei variabile sau folosita intr-o expresie

```
function e_par($a){
      if($a%2) return false;
      // urmatoarele doua linii nu se mai executa daca $a e impar
      echo "$a este numar par!;
      return true;
}
e_par(13);  // nu se afiseaza nimic
e_par(20);  // 20 este un numar par!
```

3.3.7.2 Returnarea prin valoare

O functie poate returna o valoare prin folosirea instructiunii **return** urmata de o expresie a carei valoare poate avea orice tip de date (scalar, tablou, obiect etc).

Atentie! Daca, in cadrul functiei, instructiunea return nu este folosita sau nu i se specifica argument, functia va intoarce automat NULL.

```
function suma($x1, $x2){
      return $x1+$x2;
}
$s = suma (3,4);  // in $s se memoreaza valoarea expresiei pasate ca argument lui return
```

Cand o functie returneaza o valoare, aceasta poate fi memorata intr-o variabila, poate fi folosita ca parte a unei expresii sau poate fi pur si simplu ignorata:

```
$s = suma(56,3);  // memorarea valorii returnate intr-o variabila
echo "16 adunat cu 67 da ".suma(16,67);   // concatenarea unui sir cu valoarea returnata
suma(56,3);       // simpla apelare a functiei; rezultatul returnat se pierde
```

Ceea ce se returneaza este o copie a valorii variabilei/expresiei primite ca parametru de return:

```
function peste_zece(&$a){     // pasarea argumentului se face prin referinta
      if($a < 10)
            $a += 10;         // este modificata chiar valoarea variabilei "originale"
      return $a;              // se returneaza o copie a valorii lui $a
}

$b = 2;
$c = peste_zece($b);          // $c NU este o referinta la $b, ci o copie a valorii lui
echo "\$b: $b, \$c: $c";      // $b: 12, $c: 12 ($b a fost modificat in functie)
$b++;                         // modificam $b; $c nu ar trebui sa fie afectat
echo "\$b: $b, \$c: $c";      // $b: 13, $c: 12 ($b si $c ocupa zone de memorie diferite)
```

3.3.7.3 Returnarea prin referinta

Functia poate returna prin referinta – in acest caz, **trebuie folosit & si in numele functiei, si la apelarea ei:**

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

```
function &increment($x) {      // & indica faptul ca functia returneaza prin referinta
      $x++;
      return $x;               // sintaxa lui return e neschimbata
}
$a = 5;
$b =& increment($a);    // asignarea valorii returnate se face de asemenea prin referinta
```

Folosim returnarea prin referinta atunci cand dorim ca o functie sa decida catre ce variabila sa cream o referinta:

```
function &interogheaza_sql($interogare){
      // instructiuni care verifica conexiunea cu baza de date, validitatea interogarii
      [...]
      // daca verificarile s-au incheiat cu succes, se face interogarea efectiva
      $rezultat = mysql_query($interogare);
      return $rezultat;
}

$rezultat_sql =& interogheaza_sql($query);      // presupunand ca $query exista deja
```

3.4 BIBLIOGRAFIE

- PHP manual
  - structuri de control: http://ro.php.net/manual/en/language.control-structures.php
  - functii: http://ro.php.net/manual/en/language.functions.php

- Zend PHP 5 Certification Study Guide:
  - capitolul 1: "PHP Basics", pag 26-31 (Conditional Structures)
  - capitolul 2: “Functions”, pag 37-45

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.