**Rezime**

* Operatori služe za dodeljivanje vrednosti, poređenje dve vrednosti, pridruživanje izraza ili vršenje aritmetičkih operacija nad vrednostima;
* Postoji više vrsta operatora. Najvažniji su: operatori poređenja, aritmetički operatori, logički operatori i operatori dodeljivanja;
* Operatori poređenja su: ==, !=, <, >, <=, >=, ==, !==, operatori dodeljivanja su: =, logički operatori su &&, || i aritmetički operatori su: +, -, /, \*, % ;
* Dodeljivanje se može izvršiti po vrednosti ili po referenci;
* Prilikom primene inkrementalnog ili dekrementalnog operatora nije isto ako se operator postavi pre ili nakon promenljive;
* Prilikom upotrebe aritmetičkog operatora, PHP će, ukoliko je to moguće, sam izvršiti konverzaciju na odgovarajuće tipove.

Operatori

Jedinica: 4 od 19

Glavna tema ove lekcije će biti operatori u PHP-u. Da bismo stvorili jedan, koliko-toliko funkcionalan program u bilo kom jeziku, potrebni su nam, pored promenljivih, i operatori. I ovde postoji generalna klasifikacija na: operatore dodeljivanja, aritmetičke operatore, operatore poređenja i logičke operatore. Postoji još nekoliko vrsta operatora (string operatori, inkrement-dekrement operatori, array operatori), ali se oni (kao što ćete u nastavku i videti) generalno baziraju na pomenuta četiri osnovna operatora.

**Operatori dodeljivanja** podrazumevaju dodelu vrednosti određenoj varijabli.

* code
* source

1. $a=10;
2. $x='Hello world!';
3. $z=3;

Jasno je da je osnovni operator dodeljivanja znak =

Ipak, postoje i određene kombinacije znakova koje takođe imaju mogućnost dodeljivanja, ali se za njih pre može reći da su skraćenice nego operatori (pa će o njima biti više reči kasnije):

$x+=5; //is same like $x=$x+5;  
$x-=1;  //is same like $x=$x-1;

Podrazumevano dodeljivanje vrednosti promenljivama jeste **po vrednosti**. To znači da će, u trenutku kada dodelimo promenljivoj vrednost, ta vrednost stvarno i biti u toj promenljivoj:

Kada uradimo sledeće:

* code
* source

1. $x=3;

Naš program je uzeo jedan delić memorije računara i u taj delić memorije smestio je broj 3. Takođe, rekao je da će se taj delić memorije u programu odazivati na „ime“ x.

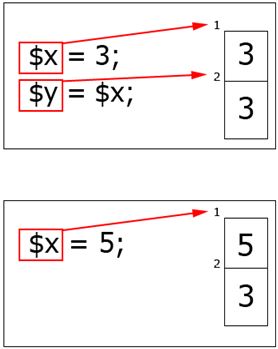
To znači da ako napišemo

* code
* source

1. $x=3;
2. $y=$x;
3. $x=5;

promenljiva $y će nakon izvršene sve tri linije imati vrednost 3.

Ovo je veoma logično. Napravili smo promenljivu x, dodelili joj vrednost 3, zatim smo napravili promenljivu y i dodelili joj vrednost promenljive x (3), a zatim smo promenili vrednost promenljive x u 5, ali je vrednost promenljive y i dalje 3, jer je u liniji $y=$x; izvršeno dodeljivanje po vrednosti.



*4.1. Stanje u memoriji prilikom dodele vrednosti*

Pogledajmo sada drugi primer. Zapravo, isti primer, sa dodatim jednim znakom **&** u drugom redu. Ovaj znak u programiranju predstavlja memorijsku adresu.

* code
* source

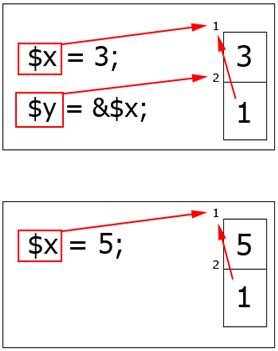
1. $x=3;
2. $y=&$x;
3. $x=5;

Nakon izvršavanja ovog koda, vrednost promenljive y biće 5, što ćemo upravo objasniti. Malopre smo opisali proces dodeljivanja po vrednosti, kada je vrednost bila direktno smeštena u memoriju ali, u slučaju dodeljivanja po referenci (što je slučaj u drugom primeru), u memoriju se smešta samo adresa.

To znači da, ako uradimo sledeće:

$y=&$x;

naš program opet uzima delić memorije, ali ovaj put u njega ne smešta broj tri, nego broj koji predstavlja memorijsku adresu promenljive x.



*4.2. Stanje u memoriji nakon dodele reference*

Ukoliko vam ovo i dalje zvuči komplikovano, pomislite na preusmeravanje poziva na vašem telefonu. Ovo je upravo to.

Konačno, nemojte se opterećivati dodeljivanjem po referenci. Ovo ćete u PHP-u retko koristiti samoinicijativno. Pored toga što je komplikovaniji, ovaj pristup je u PHP-u i sporiji od dodeljivanja po vrednosti (zapamtite da je PHP jedan od retkih u kome je prosleđivanje po referenci sporije od prosleđivanja po vrednosti).

Sa druge strane, u nekim situacijama dodeljivaćete vrednosti po referenci, a da toga niste ni svesni, na primer, kada budete upotrebljavali objekte ili nizove, koji jedino i mogu funkcionisati na ovaj način.

**Kreiranje dinamičkih promenljivih**

Ponekad, na primer, kada radite sa bazom podataka i želite da imena vaših promenljivih imaju ista imena kao i polja u bazi podataka, ili jednostavno, kada ne znate kako se neka promenljiva zove, pre nego što počnete da je koristite, koristite dinamičke promenljive. Dinamičke promenljive u PHP-u se kreiraju tako što se ispred imena promenljive čija je vrednost jednaka imenu promenljive stavi prefiks $.

U konkretnom primeru, to izgleda ovako:

* code
* source

1. <?php
2. $car = 'opel';
3. $opel = 23;
4. print $$car;
5. ?>

Nakon startovanja biće ispisan broj 23, jer smo promenljivu $opel pozvali dinamički.

Šta se dešava u PHP-u u trenutku dinamičkog navođenja? Kada je prefiks $$, php se navodi da uzima vrednost sa desne strane i to ime koristi kao ime „prave” promenljive. Ovo je zapravo vrlo logično, jer, ako imamo promenljivu koja se zove $car, a pritom imamo i string čija je vrednost „opel", treba samo na neki način da sastavimo ova dva i napišemo nešto poput: $"opel", što je upravo ono što postižemo dinamičkim promenljivama.

**Aritmetički operatori**

Svaka aritmetička operacija zahteva i aritmetički operator (+, -, \*, /, %). Oznake za sabiranje i oduzimanje su jasne, dok su ostala tri operatora: \* za množenje, / za deljenje i % za ostatak.

Operator za ostatak predstavlja celobrojni ostatak deljenja dva broja.

10%3 daje kao rezultat vrednost 1  
10%4 daje vrednost 2

Prilikom izvođenja aritmetičkih operacija, treba voditi računa o redosledu njihovog izvršavanja. Matematičke operacije u PHP-u imaju isti prioritet izvršavanja kao i u matematici. Množenje i deljenje imaju prednost nad sabiranjem i oduzimanjem. Redosled izvršavanja operacija može da se promeni upotrebom zagrada.

Kada pokušate da primenite neki od navedenih operatora na promenljive, rezultat operacije će zavisiti od tipa podatka koji je smešten u promenljivama. Na primer, nakon izvršavanja sledećih naredbi:

* code
* source

1. $number1 = 1;
2. $number2 = 2;
3. $result = $number1 + $number2;

promenljiva result će sadržati vrednost 3. Promenljive $number1 i $number2 su definisane kao brojčane promenljive, pa je rezultat njihovog sabiranja zbir brojeva koje sadrže. Međutim, situacija je drugačija ako jednu od promenljivih kreirate kao string promenljivu. Na primer, nakon izvršavanja naredbi:

* code
* source

1. $number1 = "1";
2. $number2 = 2;
3. $result = $number1 + $number2;

promenljiva $number1 je kreirana kao string promenljiva. Pre sabiranja, PHP će automatski konvertovati tekstualnu vrednost u broj. U ovom slučaju, vrednost "1" će biti konvertovana u broj 1, pa će rezultat sabiranja opet biti 3, kao i u prethodnom slučaju. Međutim, u sledećim naredbama:

* code
* source

1. $number1 = "x";
2. $number2 = 2;
3. $result = $number1 + $number2;

s obzirom da vrednost prve promenljive nije broj, PHP će promenljivu $number1 konvertovati u broj 0, pa će vrednost promenljive $result biti 2. U većini slučajeva, PHP će konvertovati string vrednost u neki neodgovarajući broj. Pogledajte sledeći primer:

* code
* source

1. $number1 = "2,000";
2. $number2 = 2;
3. $result = $number1 + $number2;

Vrednost prve promenljive bi trebalo da bude 2000, barem je tako ljudi razumeju, ali PHP zarez doživljava kao kraj broja, pa će promenljiva $number1 biti konvertovana u broj 2, a rezultat sabiranja će biti broj 4. Zbog toga se ne treba oslanjati na automatsko konvertovanje, nego treba voditi računa da je svakoj promenljivoj dodeljena prava vrednost prilikom kreiranja, kako bi mogle da se izvrše sve operacije nad njom.

|  |
| --- |
| **Napomena**    Od verzije PHP 5.6 uključen je još jedan aritmetički operator. U pitanju je operator za eksponencijalni izraz odnosno stepenovanje. Stepenovanje je matematička binarna operacija u zapisu *ab* gde *a*predstavlja osnovu, dok *b*predstavlja eksponent.    Oznaka operatora: \*\*    Primena: Sa leve strane operatora se postavlja osnova, a sa desne eksponent.    Primer:    $x = 10 \*\* 2;    Ovakav zapis bi dao isti rezultat kao 10 \* 10, odnosno 100.    Treba imati u vidu da ovaj operator neće biti dostupan ukoliko je korišćen WampServer koji podržava verziju PHP-a koja je starija od 5.6. |

Pored osnovnih matematičkih operatora, PHP obezbeđuje i operatore povećanja i smanjenja vrednosti (inkrement i dekrement operatori).

|  |  |  |
| --- | --- | --- |
| **Operator** | **Upotreba** | **Značenje** |
| **++** | $broj++; | $broj=$broj+1; |
| **--** | $broj--; | $broj=$broj-1; |
| **+=** | $broj+=2; | $broj=$broj+2; |
| **-=** | $broj-=2; | $broj=$broj-2; |
| **\*=** | $broj\*=2; | $broj=$broj\*2; |
| **/=** | $broj/=2; | $broj=$broj/2; |

*Tabela 4.1 - Tabela skraćenih operatora*

Kao što se iz tabele može videti, neki od ovih operatora su unarni. Odnosno, potrebna im je samo jedna vrednost da bi funkcionisali. U nekim slučajevima, ako želimo da inkrementaciju vršimo po eksplicitnoj vrednosti (podrazumevana vrednost je 1), onda koristimo i drugu vrednost u operatoru.

U inkrement-dekrement operatorima je veoma važna pozicija samog operatora. Iako su sledeća dva izraza identična po krajnju vrednost promenljive $a,

$a++  
++$a

„život” ove promenljive tokom izvršavanja operatora nije isti. To možemo lako utvrditi sledećim primerom:

* code
* source

1. $a=1;
2. $x=$a++;

Nakon izvršenih linija koda, promenljiva x imaće vrednost 1 ali, ako drugu liniju izmenimo sa:

* code
* source

1. $x=++$a;

promenljiva x će imati vrednost 2.

Dakle, jasno je šta se dogodilo. U prvom primeru, inkrementacija je izvršena nakon dodele vrednosti, a u drugom, dodela vrednosti se odigrala nakon inkrementacije.

Naravno, suvišno je govoriti o tome kakve posledice može imati nepažljivo rukovanje inkrement operatorima.

Sem operatora, za rad sa brojevima moguće je koristiti i ugrađene funkcije PHP-a, kao što je funkcija **sqrt()** za izračunavanje kvadratnog korena nekog broja, funkcija **abs()** za izračunavanje apsolutne vrednosti broja, funkcija **ceil()**, koja zaokružuje broj na prvi veći ceo broj, funkcija **floor()**, koja zaokružuje broj na prvi manji ceo, funkcije **max()** i **min()** koje izračunavaju maksimum i minimum i druge. Sećate li se problema sa konverzijom decimalnog broja u ceo (kada je broj 7.999999 davao rezultat 7)? Ovakvi problemi se upravo rešavaju matematičkim funkcijama.

Spisak matematičkih funkcija možete naći na sledećoj adresi:   
<http://php.net/manual/en/ref.math.php>

**Operatori za konkatenaciju stringova**

Kako ćemo se radom sa stringovima više baviti u narednim lekcijama, ovde operatorima za konkatenaciju stringova nećemo posvetiti previše pažnje, ali ćemo ipak napraviti sažeti uvod koji nam je neophodan za dalji rad.

[Konkatenacija](https://www.link-elearning.com/linkdl/opisPojma.php?id=139101) (nadovezivanje stringova jedan na drugi) se vrši oznakom **.**(tačka)

Sledeća linija:

* code
* source

1. $name = "Link" . " Group";

dodeliće promenljivoj**name** vrednost "Link Group”

Takođe, možete i ovde upotrebiti neku vrstu inkrementalnih operatora, pa će (pod uslovom da **name** već ima vrednost iz prethodnog primera (“Link Group”), nakon sledeće linije vrednost promenljive **name** biti “Link Group d.o.o.”.

* code
* source

1. $name .= " d.o.o.";

i naravno, produžena verzija iste linije:

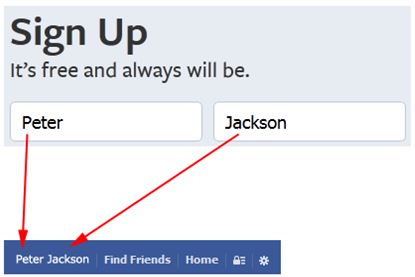
* code
* source

1. $name = $name . " d.o.o.";

Zbog čega bismo koristili ovakav operator?

Pogledajmo sledeću ilustraciju. Na njoj se nalaze dva dela Facebook sajta. Jedan je Sign Up - [forma](https://www.link-elearning.com/linkdl/opisPojma.php?id=139105) putem koje se možemo registrovati za Facebook članstvo, a druga - status korisnika, panel prikazuje status trenutno ulogovanog korisnika.

Fokus je na imenu i prezimenu. Vidimo da je u Sign Up formi ime i prezime razdvojeno. Ovo znači da će ovi podaci biti razdvojeni i u bazi podataka, a i u PHP skripti. Oni će biti reprezentovani kroz dve promenljive. Na primer: $ime i $prezime. U statusu korisnika, ovi podaci su spojeni: Peter Jackson. Da bi podaci bili spojeni, najverovatnije je autor skripte iskoristio upravo operator za konkatenaciju stringa.



*4.3. Prikaz Facebook korisničkog panela i dela Sign Up forme*

**Operatori poređenja**

Vrše poređenje određenih vrednosti i kao rezultat daju vrednost tipa boolean. Oznake su:

* + == jednako je (operandi imaju istu vrednost, ali ne moraju biti istog tipa);
  + != nije jednako;
  + < manje od;
  + > veće od;
  + >= veće ili jednako od;
  + <= manje ili jednako od;
  + === identično je (operandi imaju istu vrednost i istog su tipa);
  + !== ili nije jednaka vrednost ili nije jednak tip.

Pogledajmo nekoliko primera:

Ako bismo imali dve promenljive, $x i $y, pri čemu je $x=10, a $y=20, mogli bismo da uporedimo njihove vrednosti operatorom poređenja:

Ako bismo hteli da pitamo da li je $x jednako $y, napisali bismo:

*$x==$y*

Ovako napisan kod vratiće kao rezultat boolean tip. I to, u ovom slučaju, vrednost false. Jer $x i $y nemaju istu vrednost.

Naravno, ako bismo napisali samo poređenje dve vrednosti, kod ne bi imao ni neki naročit smisao. Zato se operatori poređenja najčešće koriste u kombinaciji sa uslovnim izrazima.

Na primer, ako hoćemo da napišemo poruku „Hello!" ukoliko je $x jednako $y, napisaćemo:

* code
* source

1. if($x==$y) echo "Hello!";

Ako startujemo prethodno napisani kod, nećemo dobiti poruku Hello, jer $x nije jednak $y.

Mogli bismo zato promeniti uslov i napisati: ako $x nije jednako $y napiši poruku. Tada bismo mogli iskoristiti negaciju i napisati:

* code
* source

1. if($x!=$y) echo "Hello!";

Zatim bi po istom principu mogli koristiti i sve ostale pomenute operatore.

Poslednja dva operatora su malo teža za razumevanje u ovom trenutku, pa je dobro da ih dodatno praktično objasnimo.

Uzmimo sledeći blok:

* code
* source

1. $x = 1;
2. $y = "1";
3. $z = $x == $y;

Vrednost promenljive $z bi nakon ovog koda bila **true**, jer promenljive $x i $y imaju istu vrednost.

Ali nakon izvršenja sledećeg koda:

* code
* source

1. $x = 1;
2. $y = "1";
3. $z = $x === $y;

Vrednost promenljive $z će biti **false**, jer, iako promenljive $x i $y imaju istu vrednost, njihovi tipovi se razlikuju.

**Logički operatori**

Logički operatori se koriste za logičke algebarske operacije. Najčešće ih možemo videti u kombinaciji sa operatorima poređenja, gde se koriste u svrhu izvođenja složenih poređenja sa više promenljivih. Logički operatori kao rezultat vraćaju Boolean vrednost.

Ovi operatori spajaju više izraza u određenom uslovu rezultirajući takođe konačnom vrednošću tipa boolean:

* + prvi izraz && drugi izraz – ako su i prvi i drugi izraz tačni, rezultat je true (tačno),
  + prvi izraz || drugi izraz – ukoliko je prvi ili drugi izraz tačan, rezultat je true,
  + znak negacije ! vrednost ili izraz - konvertuje operand u Boolean vrednost i vrši njegovu negaciju.

**Logička konjunkcija*(&&)***

Operator && (logičko AND) obavlja logičku konjunkciju dva operanda. Ovo znači da će vraćena vrednost biti true **ako i samo ako** oba operanda imaju vrednost true. Ipak, treba dodatno napomenuti da je dovoljno da samo prvi operand ima vrednost false da bi rezultat bio false. Pogledajmo sledeći primer:

* code
* source

1. $a = 1;
2. $b = 2;
3. if($a==0 && ++$b==3)
4. {
5. echo "It can not be seen.";
6. }
7. echo "The second condition wasn't checked, so b is still $b";

Opšti primeri:

$a1 =  true && true;     //Promenljiva a1 dobija vrednost true  
$a2 =  true && false;    //Promenljiva a2 dobija vrednost false  
$a3 = false && true;     //Promenljiva a3 dobija vrednost false  
$a4 = false && false;    //Promenljiva a4 dobija vrednost false

Umesto logičkih vrednosti možemo koristiti i izraze:

* code
* source

1. $a1 = (3 > 0) && (3 < 5); *// Variable a1 gets value true*

Na primer, ako bismo hteli da proverimo da li korisnik ima više od 13 godina i manje od 25 godina, napisali bismo:

* code
* source

1. $user\_age = 20;
2. if($user\_age>13 && $user\_age<25)
3. echo "User is younger than 25 and older than 13 years";

U ovom slučaju bi tekst: "User is younger than 25 and older than 13 years" bio ispisan na strani. Provere radi, možete promeniti vrednost promenljive user\_age i proveriti ponašanje programa.

**Logička disjunkcija*(*||*)***

Operator **||** (logičko OR) obavlja logičku disjunkciju dva operanda. Kao i prethodni operator (&&), i **||**je binarni operator, što znači da rukuje sa dva operanda. Vraća logičku vrednost true **ako je bar jedan**od operanada true.

$o1 =  true || true;     // Promenljiva o1 dobija vrednost true  
$o2 = false || true;     // Promenljiva o2 dobija vrednost true  
$o3 =  true || false;    // Promenljiva o3 dobija vrednost true  
$o4 = false || false;    // Promenljiva o4 dobija vrednost false

Mogli bismo proveriti da li korisnik ima 13 ili 25 godina, ali bismo tada iskoristili logički operator**ili (||):**

* code
* source

1. $user\_age = 13;
2. if($user\_age==13 || $user\_age==25)
3. echo "User is 13 or 25 years old";

Takođe, možemo praviti i kombinacije ovih operatora:

* code
* source

1. $user\_age = 13;
2. $user\_gender = "male";
3. if(($user\_age == 13 || $user\_age == 25) && ($user\_gender == "male")){
4. echo "Valid user";
5. }

**Logička negacija (!)**

Za razliku od prethodna dva, ovaj operator nije binarni, već unarni. Dakle, za svoje operacije ne koristi dve, već samo jedan operand. Operand može biti logička vrednost ili izraz. Svrha ovog operatora je inverzija logičkog operanda uz koji se nalazi.

Primer:

$n1 = !true;  // Promenljiva n1 dobija vrednost false  
$n2 = !false; // Promenljiva n2 dobija vrednost true

**Za one koji žele da saznaju više:** Ukoliko se operator ! nađe uz operand koji nije logička vrednost, on će ga najpre konvertovati u odgovarajuću logičku vrednost, a zatim izvršiti negaciju te vrednosti. Ovo nam ukazuje na to da neku nelogičku vrednost možemo konvertovati u logičku tako što nad njom izvršimo dvostruku negaciju (!!x). Na primer:

* code
* source

1. $x = 10;
2. $x = !!$x;

Promenljiva x je sada postala Boolean vrednost true.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Naziv** | **Primer** | **Rezultat** |
| **and** | And | $x and $y | True ako su i promenljiva $x i promenljiva $y true |
| **or** | Or | $x or $y | True ako je bar jedna promenljiva true |
| **xor** | Xor | $x xor $y | True ako je jedna ili druga promenljiva true, ali ne i ako su obe true |
| **&&** | And | $x && $y | True ako su i promenljiva $x i promenljiva $y true |
| **||** | Or | $x || $y | True ako je bar jedna promenljiva true |
| **!** | Not | !$x | True ako je $x false, ili false ako je promenljiva $x true |

*Tabela 4.2*

Dobro je znati da php uvek pokušava optimalno da tretira poređenje. Na primer, ako pitamo sledeće:

*if($a==10 || $b==20)...*

Ukoliko promenljiva $a ima vrednost 10, dalja poređenja neće biti vršena (jer nema potrebe).

Sa druge strane, u sledećem kodu:

*if($a==10 && $b==20)...*

ako promenljiva $a nema vrednost 10, dalja provera će takođe momentalno biti prekinuta, jer uslov nikako ne može biti ispunjen.

**Bit operatori (operacije nad bitovima)**

S obzirom na to da je najmanja jedinica kojom se u PHP-u može rukovati kroz promenljive bajt, PHP omogućava određene operatore koji mogu rukovati vrednostima na nivou bitova.

To su bit šift operatori,kao i bit OR, AND i XOR operatori.

**Šift operatori** (<< i >>) omogućavaju pomeranje bitova na levu ili desnu stranu unutar jednog bajta. Na primer:

* code
* source

1. $a=10;
2. $b=$a<<1;

Vrednost promenljive $a je 10. Ovo je u binarnom zapisu broj: 1010. Kada promenljivoj $b dodelimo vrednost $a<<1, mi smo, u stvari, pomerili sve bitove u promenljivoj $a za jedno mesto ulevo i dodelili tu novu vrednost promenljivoj $b. Ova vrednost je sada 1010**0** (ili ako Vam je lakše 00010100 ili 0000000000010100).

Vrednost promenljive $b sada će biti 10100, odnosno 20.

Ako želite da vidite binarnu reprezentaciju broja, najbolje da koristite printf funkciju i formatirate binarni izlaz vrednosti:

* code
* source

1. printf("%b",$b);

**Binarno ne (komplement)** (~) je operator koji invertuje sadržaj bajtova aktuelne promenljive.

Ako je binarna vrednost promenljive bila 1010, sada će biti 0101. Odnosno, 1111111111111111111111111111**0101**, što možete isprobati sledećim primerom:

* code
* source

1. $a=10;
2. printf("%b",~$a);

**Binarno i (&)**daje novu vrednost kao rezultat. Ova vrednost dobija se poređenjem bitova na istim pozicijama u oba operanda. Ako oba poređena bita imaju različite vrednosti, ili vrednosti 0, rezultat će biti 0. Samo ukoliko oba bita imaju vrednost 1, rezultat će biti 1.

Na primer:

0101 &  
1011 je  
0001

**Binarno ili** (|) daje takođe novu vrednost kao rezultat. Ali se u ovom slučaju ta vrednost sastoji od svih bitova, gde bar jedna od poređenih vrednosti ima vrednost jedan.

Na primer:

0101 |  
1011 je  
1111

Konačno, **binarno XOR (exclusive or)** (^). Ovaj operator uzima u obzir samo bitove u kojima ili jedna ili druga promenljiva imaju vrednost 1, ali ne i ukoliko obe imaju ovu vrednost.

1010 ^  
1000 je  
0010

**Error suppression operator (supresor grešaka)**

Jednostavan operator, koji onemogućava emitovanje greške usled naredbe kojoj prethodi. Pokušajte da u vašem kodu izvršite sledeću naredbu (pod uslovom da nemate fajl pod nazivom „adcd“ u folderu gde se php fajl koji izvršavate nalazi):

include "abcd";

Ako pokušate da izvršite ovu naredbu, PHP će prijaviti grešku, ali, ukoliko naredbi dodate operator @

@include "abcd";

Greška neće biti emitovana.

Koliko je ovo dobro, zavisi od situacije u kojoj se nalazite. Ponekad neemitovanje greške dosta znači, ali nekada može i da smeta.

**Backtick operator  - kosi navodnik**

Omogućava izvršenje sistemske komande i emitovanje njenog izlaza u neku promenljivu.

echo `dir`;

Ako pokušate da izvršite ovaj primer, na strani će biti emitovan sadržaj foldera koji sadrži Vašu PHP aplikaciju.

Kao što je ranije napomenuto sa krajem 2015. godine stiže nova verzija PHP-a pod oznakom PHP7. Upravo ova verzija PHP-a donosi i dva nova operatora. U pitanju su takozvani **spaceship** (ili operator kombinovanog poređenja) i **null coalesce** operator.

**Spaceship operator**

Oznaka za spaceship operator izgleda ovako:

<=>

Radi se o operatoru koji koristi dva operanda i služi za njihovo poređenje. Ukoliko bismo želeli da promenljive $a i $b uporedimo ovim operatorom, napisali bismo to ovako:

$a <=> $b

Da bi primer bio očigledniji, napisaćemo sledeće:

* code
* source

1. $a = 5;
2. $b = 10;
3. $z = $a <=> $b;

U ovom slučaju bi vrednost promenljive $z bila: -1. Spaceship operator vraća vrednost -1 ako je prvi operand manji od drugog. U slučaju da oba operanda imaju istu vrednost, operator bi vratio broj 0. Na kraju, da je prvi operand veći od drugog, bio bi vraćen broj 1.

|  |
| --- |
| **Za one koji žele da saznaju više**    Spaceship operator zapravo predstavlja skraćeni oblik sledećeg izraza:    ($a < $b) ? -1 : (($a > $b) ? 1 : 0)    Kako je za razumevanje ovog izraza neophodno poznavati ternarni operator o kojem ćemo malo kasnije govoriti, ovde ćemo dati samo kraće objašnjenje. Naime, dati primer predstavlja kod koji se može napisati u verzijama PHP-a pre sedmice i koji će uredno funkcionisati. Ovaj kod najpre proverava da li je prvi operand manji od drugog, i ako jeste, odmah vraća vrednost -1 (što i spaceship operator radi). Ukoliko ovo nije slučaj, prelazi na alternativni deo koda, gde se proverava da li je prvi operand veći. Ukoliko je ovaj uslov ispunjen, biva vraćena vrednost 1. Na kraju, ako ni ovaj uslov ipak nije ispunjen, jasno je da su operandi jednaki i biva vraćen broj 0. |

**Null coalesce operator**

Oznaka za null coalesce operator izgleda ovako:

??

Svrha ovog operatora je da proveri da li promenljiva poseduje neku „korisnu“ vrednost i da na osnovu tog testa nosivoj promenljivoj dodeli upravo tu „korisnu vrednost“ ili neku drugu vrednost koja je podrazumevana.

Kako bismo bliže objasnili ovaj operator, zamislimo sledeću situaciju. Želimo promenljivoj $a da dodelimo vrednost koju ima promenljiva $b, ali pritom nismo sigurni da li promenljiva $b uopšte postoji. Ako postoji, mi želimo njenu vrednost, ali ako ne postoji, želimo da naša promenljiva $a dobije neku podrazumevanu vrednost, recimo 10.

Upotrebom coalesce operatora to bismo mogli napisati ovako:

* code
* source

1. $a = $b ?? 10;

|  |
| --- |
| **Za oni koji žele da saznaju više**    I ovaj operator, kao i spaceship o kojem smo govorili, zapravo predstavlja skraćenu sintaksu koja je i ranije postojala. Naime, posao ovog operatora bismo mogli da uradimo i na tradicionalan način:    $a = isset($b) ? $b : 10;    I ovde vidimo upotrebu ternarnog operatora, o kojem ćemo govoriti kasnije. |

**Formatiranje različitih prikaza brojeva**

Često je potrebno neke brojčane vrednosti prikazati u određenom formatu. Na primer, ukoliko te brojčane vrednosti predstavljaju novčane iznose, tada je potrebno da budu prikazane sa dva decimalna mesta i da hiljade budu razdvojene zarezom.

Da biste prikazali neki broj u određenom formatu, koristite funkciju:

*number\_format(broj,brojDecimala,"decSeparator","hiljSeparator")*

Argumenti ove funkcije su redom:

* + *broj* - broj koji se formatira. Ovaj argument je obavezan;
  + *brojDecimala* - broj decimalnih mesta. Ukoliko se izostavi, podrazumeva se da je broj decimalnih mesta 0. Mora da se navede ukoliko se koriste argumenti decSeparator i hiljSeparator;
  + *decSeparator* - karakter koji se koristi za odvajanje decimalnog dela broja. Podrazumevani separator je tačka;
  + *hiljSeparator* - karakter koji se koristi za odvajanje hiljada. Podrazumevani separator je zarez.

Na primer: echo number\_format("1000000",2,",",".");

Nakon formatiranja, broj se konvertuje u string. Zbog toga je neophodno sve aritmetičke operacije nad brojem izvršiti pre njegovog formatiranja.

Za komplikovanija formatiranja brojeva moguće je koristiti funkcije printf() i sprintf(). Funkcija printf() samo prikazuje broj u određenom formatu. Funkcija sprintf() formatira broj i smešta ga u promenljivu. Ove funkcije se koriste i za formatiranje string vrednosti.

**Primeri**

Napravimo jedan program koji sabira dva broja:

**Primer 1**

* code
* source

1. <?php
2. *// the addition of two numbers*
3. *//assigning a value to a variable x*
4. $x=1;
5. *// assigning a value to a variable y*
6. $y=2;
7. *// assigning a value to a variable z*
8. $z=$x+$y;
9. *//printing result*
10. echo $z;
11. ?>

Rezultat u pretraživaču trebalo bi da bude broj 3. Komentari jasno objašnjavaju šta se u ovom primeru događa.

Ne treba zaboraviti da konvencije PHP jezika važe samo unutar PHP taga. Stoga, ne treba pokušavati izvršavanje komandi ili ispis komentara u delu koji je van tih [tagova](https://www.link-elearning.com/linkdl/opisPojma.php?id=139127).

**Primer 2**

U ovom primeru izračunaćemo površinu kruga.

Znamo da nam je za to potrebna samo promenljiva r. Takođe, tokom izračunavanja, biće nam potreban i broj PI. Sa obzirom na to da je u pitanju fiksna vrednost, možemo upotrebiti konstantu.

* code
* source

1. <?php
2. *// calculating the area of a circle*
3. *// defining constants PI*
4. define("PI", 3.14);
5. *// defining variable r*
6. $r=10;
7. *//calculating p*
8. $p = ($r\*$r)\*PI;
9. echo $p;
10. ?>

Ovaj program, iako kratak, sadrži veoma zanimljive pojave. Pre svega, definisanje konstante. Procedura je dosta jasna, funkcija **define** prihvata dva parametra (naziv konstante, vrednost konstante). Tako dobijamo definiciju konstante:

define("PI", 3.14);

Umesto PI smo mogli da upotrebimo bilo koji drugi naziv (u konvencijama naziva varijabli). Svakako, pri definiciji konstante, nismo ograničeni samo na brojčane vrednosti. Konstanta može nositi bilo koji tip podatka.

* code
* source

1. define("nameSurname", "Peter Andersen"); *// string*
2. define("age", 1980); *//integer*
3. define("male", true); *//boolean*

U nastavku programa, nakon inicijalizacije promenljive r, primećujemo da je pri inicijalizaciji promljive p jedan deo iskaza odvojen zagradama, kao i da konstanta PI nema znak $ ispred imena. Ovo je normalno jer konstante i ne treba da imaju oznaku $ ispred naziva, dok se zagrade koriste za logičko razdvajanje celina pri aritmetičkim operacijama. Na primer, sledeće dve linije koda neće dati dva ista rezultata:

* code
* source

1. echo 2-(3\*3);
2. echo (2-3)\*3;

Na kraju, u poslednjoj liniji, program jednostavno emituje rezultat na izlazu.

Treba samo napomenuti da kvadrat u ovom programu nismo morali računati ručno, već smo mogli u tu svrhu upotrebiti jednu iz kolekcije funkcija za rad sa matematičkim operacijama:

* code
* source

1. pow($r,2);

pri čemu je prvi broj naša ulazna vrednost, a drugi broj eksponent.

**Koja je uloga Error suppression operatora?**

 Onemogućava emitovanje greške usled naredbe kojoj prethodi

 Omogućava emitovanje greške usled naredbe kojoj prethodi

 Prikazuje poruku o grešci kada se postavi iza naredbe echo

**Vežba 1**

***Problem:***

Potrebno je preraditi sledeću aplikaciju tako da na izlazu bude broj sa dve decimale.

* code
* source

1. <?php
2. $price = 527.356;
3. $discount = 15;
4. $pricewithdiscount = $price - (($discount/100)\*$price);
5. echo $pricewithdiscount;
6. ?>

Trenutni izlaz aplikacije je:

448.2526

***Rešenje:***

* code
* source

1. <?php
2. $price = 527.356;
3. $discount = 15;
4. $pricewithdiscount = $price - (($discount/100)\*$price);
5. echo number\_format($pricewithdiscount,2)
6. ?>

Obratite pažnju na to da u postavci zadatka na 5. liniji stoji sledeće:

echo $pricewithdiscount;

Ovakav izraz bi nam na dokumentu prikazao vrednost promenljive: $pricewithdiscount, bez ikakvog formatiranja, odnosno sa ukupnim brojem decimala kojih u našem slučaju ima 4. Kako mi na izlazu želimo da prikažemo broj sa dve decimale, potrebno je da najpre izvršimo formatiranje broja. U tu svrhu koristimo funkciju number\_format(). Ova funkcija može prihvatiti najviše 4 parametra. Prvi parametar je broj koji se formatira, drugim parametrom se određuje broj decimala, dok je treći parametar string koji će biti korišćen za odvajanje decimalnog dela broja. Na kraju imamo mogućnosti definisanja i 4. parametra, koji predstavlja string za odvajanje hiljada. Prvi parametar je obavezan, dok su ostala tri opciona. Za potrebe našeg zadatka dovoljno je definisati dva parametra, odnosno broj koji se formatira i broj decimala. Zato menjamo 5. liniju i pre ispisa vršimo formatiranje na sledeći način:

echo number\_format($pricewithdiscount,2)

**Vežba 2**

***Problem:***

Aplikacija prolazi kroz brojeve od 0 do 1000. Potrebno je prikazati samo pozitivne brojeve koji su deljivi sa 3 bez ostatka. Postojeći kod izgleda ovako:

* code
* source

1. <?php
2. for($i=0;$i<1000;$i++)
3. if( OVDE JE POTREBNO UNETI USLOV )
4. echo $i . "<br>";
5. ?>

***Rešenje:***

Može se upotrebiti modulo operator:

* code
* source

1. <?php
2. for($i=0;$i<1000;$i++)
3. if($i%3 == 0 && $i != 0)
4. echo $i . "<br>";
5. ?>

Ova vežba je prvenstveno namenjena polaznicima koji imaju manjeg predznanja u oblasti programiranja u bilo kom jeziku. Ovde ćemo zato samo ukratko pojasniti drugu i treću liniju koda, ali ćemo se sa ovakvom sintaksom veoma detaljno upoznati u sledećem modulu, te se na ovaj primer možete vratiti i nakon obrade lekcija iz modula 2.

Iako je akcenat ovog zadatka postavljen na definisanje uslova uz pomoć operatora koji se obrađuju lekcijom, u par rečenica ćemo obraditi for i if, odnosno linije 2 i 3. Na liniji dva se nalazi for petlja kojom se obezbeđuje izvršavanje naredne linije koda (linija koda 3) 1000 puta. Takođe, u svakom od ovih izvršavanja može se koristiti promenljiva $i koja će svaki put imati vrednost uvećanu za 1. Ovo znači da će u prvom izvršavanju petlje promenljiva $i imati vrednost 0, zatim vrednost 1, 2 ... itd. Prilikom svakog izvršavanja petlje proverava se uslov koji je definisan u zagradi nakon if u liniji 3. Ovde je potrebno postaviti izraz koji će predstavljati uslov definisan upotrebom operatora koji su obrađivani lekcijom. Ukoliko je uslov ispunjen, odnosno ukoliko je vraćena vrednost true, izvršiće se i sledeća linija koda (linija 4.). Četvrtom linijom se obezbeđuje prikaz broja koji odgovara vrednosti promenljive $i.

Sada možemo pojasniti uslov koji je u zagradama definisan. Najpre proveravamo da li je vrednost koja odgovara promenljivoj $i deljiva brojem 3. U tu svrhu koristimo modulo operator koji vraća brojčanu vrednost koja predstavlja ostatak nakon deljenja. Ukoliko je nakon deljenja ostatak 0, znači da je broj deljiv sa 3, a to je upravo ono što se i traži zadatkom. Kako u nizu rešenja ne bi bila prikazana i nula, vršimo još jedno testiranje, odnosno proveravamo da li je broj $i različit od 0. Bitno nam je da oba ova uslova budu ispunjena, te iz tog razloga između dva uslovljavanja postavljamo logičko AND (&&). Time su svi naši uslovi provereni i zadatak je završen.