**for petlja**

**00:26:18**

Jedinica: 7 od 19

**Rezime**

* U PHP-u postoji nekoliko vrsta petlji: for, while, foreach i njihove varijacije;
* for petlju upotrebljavamo kada znamo njene inicijalne vrednosti (koliko će se puta izvršiti);
* For petlja podrazumeva inicijalizaciju iz tri dela: inicijalizacija brojača, uslov brojača i inkrement/dekrement brojača;
* For petlja ne mora imati sva tri inicijalna parametra;
* Foreach petlja se koristi za čitanje nizova;
* Foreach petlja može čitati vrednosti ili ključeve i vrednosti iz niza;
* Petlja se može bezuslovno prekinuti ključnom rečju **break;**
* Iteracija petlje se može preskočiti ključnom rečju **continue.**

Tema lekcije koja je pred vama jeste for petlja sa njenim varijacijama. Nepisana pravila dobre prakse programiranja podrazumevaju da isti blokovi koda ne bi trebalo da se  ponavljaju, odnosno kopiraju više puta. Ipak, veoma često imamo potrebu da određene blokove koda iznova izvršavamo. Razmotrimo ovo na slučaju iz prakse. Kreirali smo blok koda kojim se na strani prikazuje jedna vest. Onog trenutka kada odlučimo da na strani prikažemo 10 vesti, dolazimo u situaciju da isti ili veoma sličan blok koda kopiramo 10 puta. I ovaj slučaj bi donekle mogao biti prihvatljiv, ali šta ukoliko dođemo u situaciju da neki blok koda ponavljamo 1000 puta, ili čak ako nismo sigurni koliko puta blok koda treba da se ponovi? Tada nam u pomoć pristižu petlje. U petlju možemo postaviti određeni blok koda i petlja će taj blok koda izvršavati onoliko puta koliko joj je to naznačeno. Svaki prolazak kroz blok koda naziva se iteracija.

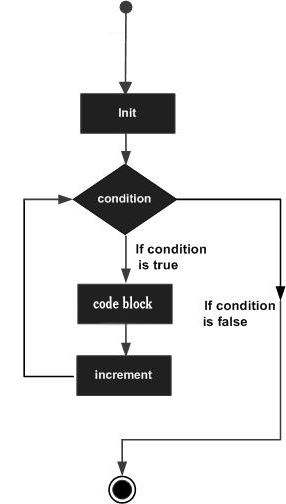
PHP ima nekoliko tipova petlji, ali se, kada se izuzmu razne varijacije, broj petlji svodi na dve **for** i **while** petlje. U ovoj lekciji reći ćemo nešto više o **for** petlji i njenoj varijaciji **foreach**.

**For petlja**

*For* petlja se koristi kada se unapred zna (ili se može izračunati) tačan broj iteracija, odnosno prolazaka kroz petlju. For petlja ima sledeću sintaksu:

      for (inicijalizacija; uslov; povećanje) {  
           blok naredbi;  
     }

Prvi parametar se koristi da kreira i postavi početnu vrednost brojaču. Drugi parametar sadrži uslov za brojač, a treći određuje na koji način se brojač povećava.



*7.1. for šema*

Na samom početku petlja inicijalizuje promenljivu koja će biti brojač (mada se ovde mogu inicijalizovati i druge promenljive). Nakon toga se proverava uslov. Ukoliko je uslov ispunjen, izvršava se blok koda definisan u telu petlje. Nakon izvršenja koda, vrši se inkrementacija brojača (ipak, ovo ne mora biti samo inkrementacija već bilo koja operacija nad brojačem), nakon čega se uslov ponovo proverava. Jedan ovakav prolaz kroz petlju naziva se iteracija. Kada testiranje uslova vrati vrednost false, izvršavanje petlje se prekida.

**Detaljnije o sintaksi:**

for (*inicijalizacija*; *uslov*; *završna\_izjava*){  
*//Blok koda*}

**inicijalizacija**– Osnovna i najčešća forma for petlje podrazumeva da se u delu koda gore predstavljenom kao [inicijalizacija](https://www.link-elearning.com/linkdl/opisPojma.php?id=139110) definiše jedna promenljiva (najčešće promenljiva **i**) koja će služiti kao brojač. U najvećem broju slučajeva je korisno da inicijalna vrednost ove promenljive bude 0, jer se često pomoću ove petlje prolazi kroz niz čije indeksiranje počinje nulom. Pored inicijalizacije promenljive koja će biti brojač, ovde možemo definisati i druge promenljive ukoliko su potrebne.

**uslov**– U ovom delu se definiše izraz čiji rezultat izvršavanja treba da vrati boolean vrednost. Uslov se najčešće konstruiše tako da se neka vrednost poredi sa promenljivom definisanom u brojaču. U slučaju da je ovaj uslov ispunjen nastavlja se sa izvršavanjem for petlje, u suprotnom se izvršavanje petlje prekida.

**završna\_izjava** - Kod definisan u ovom delu se izvršava na kraju svake iteracije. Najčešće se koristi za uvećanje (ili drugu izmenu) brojača.

|  |
| --- |
| **Napomena:**    Inicijalizacija, uslov i završna\_izjava se odvajaju znakom tačka sa zapetom (;) i to je obavezno. U specijalnim slučajevima je dozvoljeno potpuno izostaviti bilo kakav kod u nekom od ovih delova (ili čak u sva tri dela), ali dva znaka ; unutar zagrada ipak moraju postojati. Primer:    for ( ; ; ){     //Blok koda  }    Nemojte pisati ovakav kod ukoliko niste sasvim sigurni šta želite da uradite. |

Sledeći primer koda ispisuje pozdravnu poruku četiri puta:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <?php  for($counter=1; $counter < 5; $counter++) {      echo "Welcome! <br/>";  }  ?> |

Prvo se vrednost promenljive $*counter* postavlja na 1. Drugi parametar sadrži uslov. Blok naredbi će se izvršavati dok god je ovaj uslov ispunjen. Treći parametar povećava promenljivu $*counter* za jedan.

*For* petlje se često koriste za prolazak kroz nizove. Na primer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <?php  // creates a new array of five elements  $colors = array('red', 'green', 'blue', 'yellow','white');    for ($i = 0; $i < sizeof($colors); $i++)  {      $br = $i + 1;      echo "Value of element $br is $colors[$i].";  }  ?> |

|  |
| --- |
| **Napomena**    Kako sintaksu za kreiranje i manipulaciju nizovima još uvek nismo detaljno obradili (što ćemo uraditi u narednim lekcijama), ovde ćemo samo skrenuti pažnju na osnovne karakteristike nizova:     * Nizovi predstavljaju kolekcije podataka. To znači da jedna promenljiva može sadržati više različitih vrednosti, * Podaci smešteni u niz razlikuju se po svojim indeksima (podatak smešten na prvoj poziciji ima indeks 0, podatak smešten na drugoj poziciji ima indeks 1 i tako dalje...), * Indeksiranje elemenata u nizu gotovo uvek počinje nulom. |

Gorekreirani niz ima pet elemenata; pritom su indeksi elemenata u rasponu od 0 do 4. Zbog toga se u uslovu koristi stroga nejednakost, da program ne bi pokušao da pristupi elementu sa indeksom 5, koji ne postoji.

Znamo da postoje tri dela inicijalizacije for petlje. U prethodnim primerima je svaki deo imao po jedan parametar, ali ovo nije i obavezna struktura. Ponekad možemo u jednoj inicijalizaciji definisati više parametara.

Sledeća [petlja](https://www.link-elearning.com/linkdl/opisPojma.php?id=139121) će inicijalizovati promenljive $i i $a i dodeliti im različite vrednosti (0 i 5). Zatim, postavlja neki uslov (da $i bude manje od 10), a zatim interveniše na obe vrednosti, povećavajući ih za jedan.

|  |  |
| --- | --- |
| 1  2  3  4 | <?php  for($i=0,$a=5;$i<10;$i++,$a++)      echo $i . ":" . $a . "<br>";  ?> |

|  |
| --- |
| **Napomena**    Iz prethodnog pimera se vidi da, kao i kod if naredbe, nije neophodno postavljati vitičaste zagrade ukoliko želimo da od petlje zavisi izvršenje samo prve linije koda koji se pod petljom nalazi.    Pogledajmo to na istom primeru:    <?php  for($i=0,$a=5;$i<10;$i++,$a++)      echo $i . ":" . $a . "<br>";  echo "This line do not depends of loop.";  ?>      Nakon izvršenja ovog koda, na ekranu ćemo imati sledeći rezultat:    0:5  1:6  2:7  3:8  4:9  5:10  6:11  7:12  8:13  9:14    This line do not depends of loop. |

Ali, sintaksni preduslov ne podrazumeva čak ni unos sva tri parametra. Sledeći primer će takođe biti validan:

|  |  |
| --- | --- |
| 1  2  3  4 | for($i=0;$i<10;){      echo $i;      $i++;  } |

U primeru treći parametar nije definisan. On zapravo nije izostavljen, samo je potpuno prazan, što pokazuje oznaka ; iza broja 10.

Nakon ovakve inicijalizacije, inkrementiranje smo izvršili u samom bloku, što može biti veoma opasno, jer, ukoliko dođe do izostanka inkrementa, možemo pasti u takozvanu **mrtvu petlju** (petlja se izvršava beskrajno) koja će, u najboljem slučaju, blokirati okruženje.

Mrtve petlje se generalno treba paziti, jer postoje razni načini da se ona proizvede. Jednostavno, pazite da vaša petlja ne izgleda tako da njeni uslovi za izvršavanje mogu biti uvek ispunjeni.

Na primer:

|  |  |
| --- | --- |
| 1 | for($i=1;$i>0;)  echo $i++; |

Primećujete da je prethodni primer napisan u jednom redu. Podsećamo da je to zbog istog pravila koje važi i za sve ostale strukture za kontrolu toka. Nakon uslova, možete napisati samo jedan red koda bez zagrada. Ukoliko hoćete da napišete neki blok (dve linije i više), morate ceo blok staviti u vitičaste zagrade.

Videli smo da je moguće izostaviti treći parametar i rešiti njegovu funkciju novom linijom koda u telu same petlje. Pogledajmo primer kojim prikazujemo kako sličnu stvar možemo uraditi i sa preostala dva parametra:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | $i = 0;  for(;;){      if($i > 9) break;      echo $i;      $i++;  } |

Vodite računa o tome da će vrednosti petlje ostati i nakon izvršenja petlje, tako da, ako napišete:

|  |  |
| --- | --- |
| 1  2 | for($i=0;$i<10;$i++)      echo $i; |

vrednost $i nakon izvršenja će biti 10. Zato je najbolje da se držite nekih standardnih naziva promenljivih, koje ćete koristiti samo u tu svrhu. Na primer, baš slovo **i**. Tako ćete uvek znati da vam je to slovo rezervisano za petlju i nećete ga upotrebljavati za aktuelne promenljive. Naravno, ovo je samo savet. Za nazive promenljivih u petljama važe ista pravila kao i za nazive svih ostalih promenljivih u PHP-u.

For petlje možete gnezditi jednu u drugu. Kroz praksu, ćete čak videti da se veći programi upravo i sastoje iz petlji koje se nalaze u nekim većim petljama, a koje se, opet, nalaze u nekim još većim petljama. Kada budete i sami počeli da pravite neke kompleksnije strukture (sa ugnežđenim petljama), pazite na redosled izvršavanja petlji. Ne zaboravite da se vaš program, ma koliko bio kompleksan, izvršava sekvencijalno i da će, ukoliko u bilo kom trenutku započnete neki proces, svi ostali procesi biti zaustavljeni sve dok se taj proces ne završi (naravno, ovo pravilo ne važi za programiranje u realnom vremenu i multitasking programiranje).

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | for($i=0;$i<10;$i++){      for($u=0;$u<10;$u++){          echo $u . "<br>";      }      echo "---$i<br>";  } |

Ovaj primer će jasno ispisati promenljivu $u 100 puta. Ali, kolika će biti ta promenljiva $u u svakom trenutku izvršavanja?

Pošto je petlja $i spoljašnja petlja, i petlja $u se izvršava unutar nje, svaki put kada se dogodi jedna [iteracija](https://www.link-elearning.com/linkdl/opisPojma.php?id=139099) petlje $i, izvršiće se kompletan ciklus petlje $u.

*prva iteracija petlje $i  
          prva iteracija petlje $u  
          druga iteracija petlje $u  
          ...        
          deseta iteracija petlje $u  
druga iteracija petlje $i  
....*

i tako, sve do poslednje iteracije petlje $i, kada će i ceo njen ciklus biti završen.

Na početku lekcije smo pomenuli da se brojač petlje povećava. Ovo nije obavezno pravilo. Brojač se može i smanjivati, pa čak (što smo videli u jednom od prethodnih primera) ne mora ni postojati.

|  |  |
| --- | --- |
| 1 | for($i=10;$i>0;$i--) echo $i; |

***Foreach* petlja**

Ova petlja je varijacija *for* petlje, namenjena isključivo radu sa nizovima. Naročito [asocijativnim](https://www.link-elearning.com/linkdl/opisPojma.php?id=139103). Upravo je to razlog što nećemo previše detaljno ulaziti u opis ove petlje, već ćemo samo reći par reči o osnovama njenog funkcionisanja, jer još uvek nismo obradili nizove. Ipak, u nekim od prethodnih lekcija smo spominjali nizove i ovde treba podsetiti da su nizovi kolekcije podataka koje su označene indeksima. Indeksiranje elemenata u nizu počinje nulom. Evo primera niza koji sadrži tri stringa:

|  |  |
| --- | --- |
| 1 | $colors = array('red', 'yellow', 'green'); |

Ova petlja ima dva oblika. Jedan podrazumeva rukovanje vrednostima, a drugi rukovanje ključevima i vrednostima.

Foreach funkcioniše veoma jednostavno (sa strane korisnika) jer, praktično, nije potrebno ništa da znamo o nizu kroz koji hoćemo da prođemo, osim njegovog imena. Pri inicijalizaciji unosimo naziv niza i naziv promenljive za koju želimo da preuzima aktuelnu vrednost niza prilikom svake iteracije. Ili, u drugom obliku, aktuelne vrednosti ključa i vrednosti:

Prvi oblik podrazumeva:

      foreach($niz as $vrednost){  
                 blok naredbi  
           }

Kod prvog oblika prolazi se kroz niz i prilikom svake iteracije, vrednost tekućeg elementa se smešta u promenljivu $vrednost. U bloku naredbi je, zatim, moguće koristiti ovu promenljivu, a da pritom originalna vrednost sadržana u tekućem elementu niza ne bude promenjena.

Pogledajte sledeći primer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | $colors = "";  $myArray = array('red','green','blue');    echo "Colors contained in array: ";    foreach($myArray as $value){      $colors .= $value . " ";      }  echo $colors; |

Tokom svakog prolaska kroz petlju, vrednost tekućeg elementa se dodeljuje promenljivoj $*value*. Ova vrednost se zatim dopisuje na promenljivu $*colors*, zajedno sa jednim praznim karakterom. Na kraju, rezultat će biti:

*Colors contained in array: red green blue.*

Drugi oblik *foreach* petlje je:

|  |  |
| --- | --- |
| 1  2  3 | foreach (niz as $indeks => $vrednost){          blok naredbi      } |

On ima istu funkcionalnost kao i prvi oblik, s tom razlikom što, sem vrednosti samog elementa, koristi i vrednost indeksa i smešta ga u promenljivu $*indeks*.

Kao što vidite, foreach radi prilično automatski. To je nekada dobro, ali nekada i loše, jer kada radimo sa foreachom, ne rukujemo članovima niza eksplicitno kroz njihove indekse, već samo dobijamo njegove vrednosti, pa se tako za foreach može reći da je dobar kada želimo brzo i jednostavno da izvršimo čitanje nekog niza, ali ne i neke ozbiljnije intervencije na njemu.

Iako u primerima svaka foreach petlja prethodi bloku u vitičastim zagradama, i ovde važi pravilo da ne moraju postojati vitičaste zagrade za samo jednu liniju koda.

|  |  |
| --- | --- |
| 1 | foreach($arr as $key => $value) echo $value; |

**Kontrola toka petlje**

Kontrolu toka petlje možemo vršiti na nekoliko načina. Prvi način je, svakako, ručnom modifikacijom kontrolne promenljive (brojača).

|  |  |
| --- | --- |
| 1  2  3  4 | for($i=0;$i<10;$i++){          if($i>5) $i=10;          echo $i . "<br>";      } |

U primeru smo rekli petlji da se izvršava sve dok je promenljiva i manja od deset uz inkrement 1. A zatim smo se, u bloku, predomislili i rekli da, ukoliko je brojač veći od 5 (dakle 6), brojač dobije vrednost 10. Pošto je brojač dobio vrednost kojom petlja ne ispunjava uslove za izvršavanje, petlja biva napuštena. Ovo rešenje će funkcionisati, ali je nezgrapno, jer, u tom slučaju, moramo poznavati ciljnu vrednost brojača, što često neće biti moguće (npr. *for($i=0;$i<$u;$i++)).*

Drugi (pravi) način je upotreba ključne reči **break**. Ovu reč smo upoznali u prethodnoj lekciji, kada smo rekli da ona bezuslovno prekida blok koda.

|  |  |
| --- | --- |
| 1  2  3  4  5 | for($i=0;$i<10;$i++)      {          if($i>5) break;          echo $i;      } |

Upotrebom ove ključne reči naša petlja nije previše izgubila na veličini, ali je zato dosta dobila na dinamici, jer sada više ne moramo da znamo krajnju vrednost petlje.

Naredba break može imati i parametar. Ukoliko ga unesemo, break će napustiti blokove u dubini naznačenoj u parametru.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | for($i=0;$i<10;$i++){      for($u=0;$u<10;$u++){          if($u==5)          break 2;      }      echo $i;  } |

Ako pokrenemo ovaj primer, na ekranu neće biti ispisano ništa, iako se break naredba nalazi u unutrašnjoj petlji, koja nam nije neophodna za izvršavanje spoljašnje petlje. Ali, s obzirom da u trenutku ispunjenja uslova $u=5 u unutrašnjoj petlji, spoljašnja petlja nije završila još nijednu iteraciju i nije stigla da ispiše poruku na ekranu, a s obzirom na to da uslov aktivira naredbu break, i da ona ima parametar 2 (znači, prekida dva bloka po dubini), prekinuće izvršenje sopstvene petlje i petlje iznad nje.

Ali, ponekad će se dogoditi da želimo da postavimo neki uslov u toku trajanja petlje, za koji, kada se ispuni, ne želimo da izvršimo iteraciju petlje. Na primer, zamislimo neku ogromnu listu ljudi (nekoliko miliona). Svi ti ljudi se nalaze u bazi podataka i nama je potrebno da izdvojimo određenu grupaciju u kojoj su samo prosvetni radnici.

Recimo da čitanje podataka o zanimanju svakog člana liste traje 1 sekund, a čitanje kompletnih podataka o članu 5 sekundi.

Recimo sada da procedura čitanja podrazumeva: čitanje podataka o zanimanju, mestu stanovanja, čitanje imena, čitanje prezimena... (što kompletno traje 5 sekundi). Ako budemo prolazili kroz kompletnu proceduru za svakog člana, izgubićemo 5 sekundi po članu, što za sve članove iznosi mnogo sekundi. Sa druge strane, možemo odmah, na početku, pogledati da li je član prosvetni radnik (za šta nam treba samo jedan sekund), i, ako nije, odmah preuzeti podatke sledećeg člana sa liste i na taj način uštedeti 4 sekunde. Što je na pomenutu cifru od nekoliko miliona priličan broj sekundi.

Ovaj jednostavan primer kod ljudi funkcioniše automatski, ali, programu je, da bi razmišljao na taj način, potrebno to eksplicitno naglasiti. Ovo naglašavanje vrši se ključnom rečju **continue**.

Ovako bi izgledao pseudokod pomenutog primera uz upotrebu ključne reči continue:

for($clanListe = 0; $clanListe < 1000000; $clanListe ++)  
   {  
       if($clanListe != "prosvetniRadnik")  
           continue;  
       //OGROMNI KOD KOJI CITA SVE OSTALE PODATKE O CLANU LISTE  
       //......  
   }

**Foreach petlja je specijalizovana za rad sa kojim (nama poznatim) tipom podatka?**

 Nizom

 Integerom

 Floatom

 Resursom

**Vežba 1**

***Problem:***

Potrebno je napraviti program koji će iscrtati sledeću strukturu:

XXXXXXXXXX  
XXXXXXXXXX

Ovo iscrtavanje mora biti realizovano putem for petlje.

Takođe, promenljivama mora biti određeno to koliko će oznaka X biti u jednom redu i koliko će kompletna struktura imati oznaka X.

***Pomoć:***

Novi red je prilikom emitovanja u HTML moguće ostvariti sledećom naredbom:

echo "<br>";

***Rešenje:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?php  $numElements = 20;  $rowEl = 10;    for($i=0;$i<$numElements;$i++)      {          if($i%$rowEl==0 && $i>0)              echo "<br/>";              echo "X";      }  ?> |

Objašnjenje rešenja:

Najpre smo kreirali promenljive koje sadrže ukupan broj elemenata (numElements) i broj elemenata u jednom redu (rowEl). Nakon toga kreiramo spoljašnju for petlju koja će izvršiti po jednu iteraciju za svaki element. Unutar ove petlje postavljamo if uslovno grananje i ispitujemo da li je promenljiva $i koja predstavlja brojač deljiva sa brojem elemenata u redu. Pored ovoga ispitujemo i da li je $i veće od nula. Ukoliko su ovi uslovi ispunjeni, potrebno je preći u novi red, zbog čega ispisujemo html tag „<br/>“. Nakon ovog ispitivanja pristupamo štampanju elementa, tj. karaktera „X“.

**Vežba 2**

***Problem:***

U aplikaciju ulazi određeni broj koji se nalazi u promenljivoj $selectedNumber. Potrebno je napraviti petlju koja se izvršava onoliko puta kolika je vrednost promenljive $selectedNumber i prilikom svake iteracije podiže se na eksponent jednak broju trenutne iteracije petlje (prvi krug petlje ^1, drugi krug ^2...).

***Pomoć:***

PHP poznaje funkciju za eksponent: *pow(vrednost,eksponent)*

***Rešenje:***

|  |  |
| --- | --- |
| 1  2  3 | $selectedNumber = 20;  for($i=1;$i<=$selectedNumber;$i++)             echo pow($selectedNumber,$i) . "<br>"; |

Upotrebom for petlje obezbeđujemo izvršavanje koda kroz nekoliko iteracija. Kako promenljiva $i ima ulogu brojača, iteracije će se izvršavati sve dotle dok je vrednost za $i manja ili jednaka broju koji je definisan promenljivom $selectedNumber počevši od 1. Kako nakon linije:

|  |  |
| --- | --- |
| 1 | for($i=1;$i<=$selectedNumber;$i++) |

nije definisan blok koda, tako će samo izvršavanje naredne linije zavisiti od petlje. Postavkom je zahtevano da se selektovani broj u svakoj iteraciji podigne na eksponent koji odgovara broju iteracije. Kako bismo neki broj podigli na željeni eksponent, koristimo funkciju pow(). Ova funkcija radi sa dva obavezna parametra. Prvi parametar je osnova, a drugi eksponent.

**Vežba 3**

***Problem:***

Na osnovu sledećih promenljivih:

$a = 5;  
$b = 8;

i ugnežđenih petlji, potrebno je napraviti sledeći izlaz:

012345678  
100000008  
200000008  
300000008  
412345678

***Rešenje:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <?php    $a = 5;    $b = 8;    for($i = 0; $i < $a; $i++)    {      echo $i;      for($u = 1; $u <= $b; $u ++)          {              if($u == $b || $i == 0 || $i == $a-1)                  echo $u;              else                  echo "0";          }          echo "<br>";    }  ?> |

Ako pažljivo pogledamo strukturu tabele brojeva koja je data u postavci zadatka, uočavamo da se ona sastoji od 5 redova i 9 kolona. To znači da će nam verovatno biti potrebne dve petlje. Prebrojavanje iteracija u petljama možemo početi od kog god broja želimo, te ćemo tako for petlju koja će kontrolisati prikaz redova započeti od 0, a petlju koja se tiče broja kolona ćemo započeti od 1. Tako ćemo odmah kreirati dve promenljive, koje će predstavljati broj redova i kolona u skladu sa ovakvom logikom za brojače petlji:

|  |  |
| --- | --- |
| 1  2 | $a = 5;  $b = 8; |

Broj redova se definiše petljom koja kao brojač koristi promenljivu $i, a uslovljavanje vrši u odnosu na promenljivu $a. Dakle, sada imamo sledeći kod:

|  |  |
| --- | --- |
| 1  2  3 | $a = 5;    $b = 8;    for($i = 0; $i < $a; $i++){} |

Ostaje da rešimo stukturu tela ove petlje. Kako ćemo na prvom mestu u svakom redu upisivati broj koji odgovara iteraciji, (podsetimo da se iteracije prebrojavaju promenljivom $i), odmah postavljamo kod za takvo ispisivanje:

|  |  |
| --- | --- |
| 1 | echo $i; |

Sada definišemo drugu petlju kod koje prebrojavanje počinje od 1, a traje sve dok je brojač ($u) manji ili jednak vrednosti promenljive $b. Kada se ova petlja završi, znači da smo završili sa tim redom i da treba da pređemo u sledeći, te zato postavljamo BR tag.

Sada naš kod izgleda ovako:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | $a = 5;  $b = 8;  for($i = 0; $i < $a; $i++)  {    echo $i;    for($u = 1; $u <= $b; $u ++){}        echo "<br>";  } |

Svaka iteracija unutrašnje petlje imaće za cilj ispisivanje jednog karaktera u redu. Da bismo ispoštovali formu uzorka koji je dat postavkom zadatka, moramo izvršiti uslovljavanje kojim se određuje da li će na strani biti prikazan broj koji odgovara brojaču $u ili broj 0. Ukoliko vrednost brojača unutrašnje petlje odgovara vrednosti promenljive $b, potrebno je prikazati vrednost tog brojača i zato je naša prva provera:

|  |  |
| --- | --- |
| 1 | $u == $b |

Takođe, u prvom redu treba redom prikazivati brojeve od 0 do 8, a to će se desiti kada se ispuni sledeći uslov:

|  |  |
| --- | --- |
| 1 | $i == 0 |

Na kraju treba proveriti i da li je $i == $a-1 kako bismo utvrdili da li se radi o poslednjem redu (podsetimo da $a treba umanjiti za 1 jer prebrojavanje vršimo od 1, a brojač $i počinje nulom).

Ovim je naš primer završen.

**Vežba 4**

***Problem:***

U aplikaciju ulaze dve promenljive:

$numberOfCharacters = 50;  
$characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

Potrebno je napraviti aplikaciju koja na osnovu prosleđenog seta karaktera i broja generiše šifru. Šifra mora sadržati mala i velika slova iz liste karaktera $characters. Šifra takođe mora imati onoliko karaktera koliko je naznačeno u promenljivoj $numberOfCharacters.

Šifru je potrebno emitovati na izlazu.

***Rešenje:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <?php  $numberOfCharacters = 50;  $characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  $allCharacters = $characters . strtolower($characters);  $pass = "";  for($i=0;$i<$numberOfCharacters;$i++)      $pass.=$allCharacters[rand(0,strlen($allCharacters)-1)];  echo $pass;  ?> |

Objašnjenje rešenja:

Kreiramo promenljivu kojom određujemo dužinu šifre koju ćemo generisati ($numberOfCharacters). Zatim kreiramo bazu karaktera kroz definisanje promenljive $characters. Kako vrednost ove promenljive čine samo velika slova, potrebno je pridodati i sva mala slova. Da ne bismo pisali sve te karaktere, koristimo funkciju strtolower() koja prebacuje sva slova iz stringa u mala slova i rezultat rada ove funkcije pridružujemo ostalim karakterima, nakon čega konačno dobijamo promenljivu $allCharacters. Buduća šifra će biti smeštena u promenljivu $pass te zato kreiramo ovu promenljivu, u početku kao prazan string. Prolaskom kroz for petlju u svakoj iteraciji pristupamo nasumičnom karakteru stringa $allCharacters, što postižemo upotrebom funkcije rand(). Nakon što petlja prođe i poslednju iteraciju, pristupamo štampanju promenljive $pass.