**Rezime**

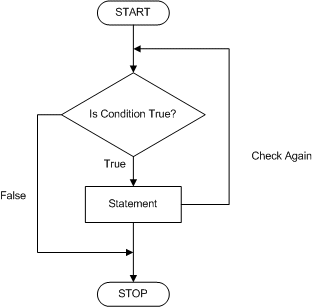
* While i do while vrše ponavljanje jednog istog zadatka određeni broj puta;
* While i do while su petlje koje kreiramo kada ne znamo koliko će tačno puta neki proces morati da bude izvršen;
* While petlja podrazumeva stupanje u telo petlje i njegovo izvršavanje samo ukoliko je uslov petlje ispunjen;
* Do while petlja podrazumeva obavezno izvršavanje bar jedne iteracije petlje, pre nego što do provere uopšte i dođe;
* While i do while petlje je moguće prekinuti ključnom rečju break;
* Moguće je preći na sledeću iteraciju while i do while petlje ključnom rečju continue;
* While i do while petlja podrazumeva uslovni izraz, čija vrednost mora biti tipa boolean;
* Izvršenje while i do while petlje se vrši ručno, unutar petlje i zato treba paziti da se ne upadne u mrtvu petlju.

while petlja

Jedinica: 8 od 19

U PHP-u while petlja predstavlja najjednostavniju petlju. Kao i if naredba, koju smo već obradili u prethodnim lekcijama, while petlja zavisi od uslova. Potrebno je napraviti paralelu između while i for. Da biste znali koja je naredba ispravnija za korišćenje, pođite od naredbe koja treba da se izvrši po zadovoljavanju uslova. Osnovna razlika je upravo u tome što if naredba izvršava blok naredbi koji sledi po zadovoljenju uslova *samo jednom,*dok while petlja izvršava naredbe *dokle god* je uslov ispunjen.

Pogledajmo kako bi se rad ove petlje mogao predstaviti grafički:



*8.1. while šema*

While petlja spada u kategoriju u kojoj nije potrebno unapred znati broj iteracija. Ona se, jednostavno, izvršava onoliko puta koliko joj to dozvoli uslov koji se pod njom izvršava. While petlja zahteva određeni uslov, koji se unosi kao parametar pri njenoj inicijalizaciji i izvršava se sve dok je i taj uslov ispunjen. Da bismo omogućili izvršavanje koda unutar while petlje, potrebno je da bude ispunjen uslov, koji je definisan na početku while naredbe. Uslov koji definišemo pri kreiranju while petlje se ne razlikuje od uslova koje smo koristili prilikom definisanja if naredbi. Uslov ima veoma značajnu ulogu, jer moramo nekada da prekinemo izvršavanje petlje kako ne bismo zapalu u, već pominjanu, **mrtvu petlju**.

Razmotrimo sada sintaksu:

while (*uslov*) {  
         *//Blok koda koji će se izvršiti ako je uslov ispunjen*}

Pogledajte sledeći primer:

* code
* source

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

Promenljiva $*number* na početku koda dobija vrednost 5. Zatim se proverava uslov *($number >= 2)*. S obzirom na to da je ovaj uslov tačan, program počinje da izvršava blok naredbi unutar *while* petlje. Unutar ovog bloka ispisuje se tekući broj, a zatim se smanjuje za 1. Nakon prvog prolaska kroz blok naredbi, ponovo se proverava uslov. Ponoviće se izvršavanje naredbi unutar petlje, sve dok je uslov tačan. Nakon izlaska iz petlje, rezultat na izlazu će biti sledeći:

*5  
          4  
          3  
          2*

Vidimo da u inicijalizaciji petlje ne postoji nešto poput inkrementacije u for petlji. Zapravo, ova petlja čak i ne mora da rukuje bilo kakvim brojačem. Jedino što joj je potrebno to je da, prilikom svake sledeće iteracije, vrednost uslova bude tačna. Ovo može da predstavlja i problem, jer ovakvu petlju moramo kontrolisati ručno.

Na primer, šta bi se dogodilo ukoliko bismo iz prethodnog primera izostavili deo:

$number -= 1;

Petlja bi se pretvorila u mrtvu petlju i verovatno zablokirala aplikaciju.

While petlja se često koristi u aplikacijama u realnom vremenu. Recimo, verovatno ne postoji igra koja u svom osnovnom mehanizmu ne sadrži baš ovakvu petlju. Ali, za razliku od igara i ostalih programa u realnom vremenu, PHP nema direktan kontakt sa korisnikom, i nije u stanju da prepozna korisnički ulaz (pritisak tastera ili miša), pa se u PHP-u njeno korišćenje svodi obično na iteraciju kroz podatke sa nekog izvora, njihovo parsiranje i slično. Na primer, čitanje fajlova, kada je potrebno da se fajl čita red po red, sve dok redovi postoje...

Zato je veoma bitno da u petlji uvek imamo neko sredstvo da tom petljom rukujemo (osim, naravno, break i continue naredbi). I zato, ova petlja prihvata isključivo boolean tip, kao rezultat uslovnog izraza, iako i neki drugi zapis ne bi prijavio sintaksnu grešku. Na primer, sledeći primer će se izvršiti, ali je potpuno neupotrebljiv i dovodi do mrtve petlje:

while( 1 + 3 )  
   echo "izvrsava se";

*While* petlju koristimo obično kada unapred ne znamo koliko će iteracija biti (koliko je puta potrebno da se izvrši utvrđena naredba). Kada je broj ciklusa poznat, koristićemo *for* petlju.

While petlje se takođe mogu ugnezditi. Svakako, potrebno je da imate kompletnu kontrolu koda kako ne bi došlo do propusta i zabune, i naravno, grešaka nakon izvršenja programa. Preporučljivo je, naravno, da koristite vitičaste zagrade kada vršite ugnežđenje koda. While naredbe (u ovom slučaju) pisane su jedna u drugoj sa uslovima za ispunjenje i naredbama za izvršenje.

Pogledajmo sledeći primer:

* code
* source

1. $i = 0;
2. while ($i++ < 3){
3. echo "first <br/>";
4. while (1){
5. echo "second <br/>";
6. while (1){
7. echo "third <br/>";
8. continue 3;
9. }
10. echo "This line will never be executed.<br/>";
11. }
12. echo "This line either.<br/>";
13. }

Nakon izvršenja programa, na izlazu ćemo dobiti sledeći ispis:

*first  
second  
third  
first  
second  
third  
first  
second  
third*

While petlja se može podeliti na dve vrste: **while** i **do while**. Odnosno, petlju koja se ne izvršava nijednom ukoliko uslov nije ispunjen (while) i petlju koja se izvršava bar jednom, čak i ako uslov nije ispunjen (do while). Prvu vrstu smo već videli. Ništa unutar petlje iz primera se neće izvršiti ukoliko promenljiva $number pre ulaska u petlju ne bude veća ili jednaka broju dva.

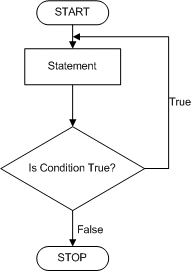
Ali, šta ako bismo želeli da telo ipak bude izvršeno bar jednom?

***Do...while* petlja**

Ova petlja je slična prethodnoj, s tom razlikom što se kod *do...while* petlje uslov proverava nakon izvršavanja bloka naredbi, a ne pre. To znači da će se blok naredbi unutar *do...while* petlje izvršiti barem jednom, bez obzira na tačnost uslova.

Do while nam omogućava da se prvo jednom izvrši blok naredbi u okviru **do** naredbe, a zatim da se prolazi kroz uslov. Dalja izvršavanja naredbi, odnosno to da li će se deo u **do** bloku ponovo izvršiti, zavisi od uslova.

Razmotrimo najpre grafički prikaz:



*8.2. do...while šema*

Pogledom na šemu zaključujemo da će blok koda petlje u svakom slučaju jednom biti izvršen. Tek nakon izvršenja koda, testitra se uslov. Ukoliko uslov nije ispunjen, petlja se napušta. Sa druge strane, ukoliko je uslov ispunjen, ponovo se izvršava blok koda, nakon čega se uslov ponovo testira. Ovakvo kruženje se nastavlja sve dok je uslov ispunjen ili dok se naredbom **break** ne napusti do…while petlja.

Pozabavimo se sada sintaksom:

do{  
  //Blok koda  
}while(*uslov koji se testira*);

Kod koji sledi veoma je sličan jednom od prethodnih, ali ovoga puta se blok koda jednom izvršava bez obzira na uslov. Tek se za eventualnu svaku sledeću iteraciju vrši provera uslova.

* code
* source

1. $number = 5;
2. do{
3. echo $number . "<br/>";
4. $number -= 1;
5. }
6. while($number >= 2);

U while petlji važe ista sintaksna pravila kao i u ostalim strukturama toka, break, continue i pravilo obaveznog stavljanja u vitičaste zagrade ukoliko je blok duži od jedne linije.

Ovo važi i za do while. Ali, prilikom pisanja bez vitičastih zagrada budite oprezni, iako deluje kao da je blok oivičen strukturom (poput switch-case slučajeva), u do...while petlji ne samo da se neće izvršiti više od jedne linije unutar petlje (bez vitičastih zagrada) već će i doći do greške:

Ovaj kod će se neometano izvršiti:

* code
* source

1. $a = 5;
2. do
3. echo $a ++;
4. while( $a < 5 );

Ovaj kod će prijaviti grešku

* code
* source

1. $a = 5;
2. do
3. echo $a ++;
4. echo "This line occurs error";
5. while( $a < 5);

Obratite pažnju i na to da ispred naredbe **do**nema oznake **;** ali da je ovaj znak obavezan iza linije koja sadrži while naredbu (poslednje linije).

**Vežba 1**

***Problem:***

Dat je sledeći kod:

$html = <<<HTML  
<div style="#S#"> #I# </div>  
HTML;  
$divNum = 10;  
$style = "border: 1px solid black; background: yellow; margin: 5px; padding: 4px;";

Potrebno je kroz petlju realizovati kod koji će prikazati html kod u promenljivoj $html minimum jednom i maksimum broj puta naveden u promenljivoj $divNum.

Prilikom svake iteracije u div tagu mora umesto oznake #I# biti broj iteracije, dok umesto oznake #S# mora biti stilizacija naznačena u promenljivoj $style.

***Rešenje:***

* code
* source

1. <?php
2. $html = <<<HTML
3. <div style="#S#"> #I# </div>
4. HTML;
5. $divNum = 10;
6. $style = "border: 1px solid black; background: yellow; margin: 5px; padding: 4px;";
7. $html = str\_replace("#S#", $style, $html);
8. $i = 0;
9. do
10. {
11. echo str\_replace("#I#", $i, $html);
12. $i++;
13. } while ($i < $divNum);
14. ?>

Kao vrednost promenljive definišemo string kreiran upotrebom heredoc sintakse i označavamo delove koji će kroz ostatak koda biti modifikovani (#S# i #I#). Sledeća promenljiva je $divNum, koja označava broj DIV elemenata koji će biti kreirani. Ostaje da kreiramo još jednu promenljivu ($style) kojom definišemo stilizaciju.

Kako je potrebno izvršiti editovanje stringa u vidu zamene oznake prethodno pripremljenom stilizacijom, koristimo funkciju str\_replace(). Prvi parametar funkcije je string koji će biti tražen. Drugi parametar je string koji će zameniti pronađeni, a trećim parametrom određujemo string nad kojim će se vršiti pretraga. Ova funkcija može imati i četvrti, opcioni parametar, kojim bismo odredili maksimalan broj ovakvih zamena.

Linijom:

* code
* source

1. $html = str\_replace("#S#", $style, $html);

smo rešili izmenu koja se tiče stilizacije elementa. Nakon ovoga postavljamo promenljivu $i koja inicijalno ima vrednost 0. Kako bi se ispisivanje DIV taga desilo bar jednom, odlučujemo se za do...while petlju.

Unutar tela ove petlje vršimo ispisivanje DIV elementa na strani, na sledeći način:

* code
* source

1. echo str\_replace("#I#", $i, $html);

Kao što možemo uočiti, i ovde vršimo izmenu stringa kako bismo svakom DIV tagu dodali sadržaj koji odgovara brojaču iteracija ($i).

Kako ne bismo zapali u „mrtvu“ petlju, vršimo inkrementaciju brojača $i++.

Uslov za izvršavanje iteracija je da $i bude manje od broja DIV tagova, koji smo odredili promenljivom $divNum i tako postavljamo uslov:

* code
* source

1. $i < $divNum

|  |
| --- |
| **Napomena**    U primeru se (kako bi on bio uspešno realizovan) pojavljuju i elementi koji u ovom delu kursa još uvek nisu obrađeni. Možete ih samo preskočiti i sačekati da budu objašnjeni u lekcijama, a možete se ukratko informisati o njima u narednim redovima:    Oznake <<<HTML i HTML; su deo herodoc string sintakse. Ona omogućava da se string napiše i zapamti u svom originalnom obliku, uključujući nove redove i ostala formatiranja. Ovde smo ovu sintaksu upotrebili da bismo zapamtili string sa div tagom.    Funkcija str\_replace menja sadržaj stringa. Ona ima tri parametra. Prvi je string koji se traži (u primeru su to #S# i #I#), drugi parametar je string kojim će nađeni string biti zamenjen, a treći parametar je sam string na kome će intervencija biti izvršena. |

**Vežba 2**

***Problem:***

Potrebno je napisati program koji će sabirati brojeve od 1 do 10 i ispisivati zbirove nakon svakog sabiranja broja sa sumom, a na kraju ispisati i sumu svih suma  (Total) iz predhodnog perioda sabiranja. Potrebno je koristiti do...while petlju. Nakon izvršenja, program treba da ispisuje sledeći sadržaj:

*0 + 1 je jednako: 1  
1 + 2 je jednako: 3  
3 + 3 je jednako: 6  
6 + 4 je jednako: 10  
10 + 5 je jednako: 15  
15 + 6 je jednako: 21  
21 + 7 je jednako: 28  
28 + 8 je jednako: 36  
36 + 9 je jednako: 45  
45 + 10 je jednako: 55*

*Suma, total svih brojeva od 1 do 10 je: 55*

***Rešenje:***

* code
* source

1. <?php
3. $amount = 0;
4. $number = 0;
6. do
7. {
8. $number ++;
9. echo $amount. " + ". $number. " je jednako : ";
10. $amount = $amount + $number;
11. echo $amount. "<br />";
13. }
14. while ( $number!= 10 );
16. echo "<br />";
17. echo "Amount all numbers: ". $amount;
19. ?>

Suština logike ovog zadatka biće završena u telu petlje, tako da ćemo na početku kreirati radne promenljive i dodeliti im vrednosti, zatim kreirati do-while petlju i na kraju izvršiti prikazivanje sume svih vrednosti. Naš kod bi nakon ovoga izgledao ovako:

* code
* source

1. $amount = 0;
2. $number = 0;
3. do{
4. }while ( $number!= 10 );
5. echo "<br />";
6. echo "Amount all numbers: ". $amount;

Sada je potrebno kreirati kod koji će se naći unutar tela petlje. Najpre ćemo izvršiti inkrementaciju brojača, kako ne bismo zapali u „mrtvu“ petlju i kako bismo svaki put uvećavali vrednost promenljive $number za 1, da bismo je kao takvu dalje koristili.

Nakon ovoga na strani prikazujemo izraz koji će biti izračunat:

* code
* source

1. echo $amount. " + ". $number. " je jednako : ";

Odmah nakon toga vršimo i samo izračunavanje ovakvog izraza i prikazujemo to korisniku:

* code
* source

1. $amount = $amount + $number;
2. echo $amount. "<br />";

Ovim je naš zadatak u potpunosti završen.

**Vežba 3**

***Problem:***

Potrebno je napraviti tabelu kao na prikazanoj slici. Program treba da izračunava korake cena u odnosu na „Količina \* Cena". U zavisnosti od koraka pomeranja količina, početna vrednost je 10 i kreće se do 100, sa korakom uvećanja za 10. Pomoć prilikom formatiranja tebele (HTML kod) nalazi se u rešenju obeležena crvenom bojom.

|  |  |
| --- | --- |
| ***Kolicina*** | ***Cena*** |
| *10* | *50* |
| *20* | *100* |
| *30* | *150* |
| *40* | *200* |
| *50* | *250* |
| *60* | *300* |
| *70* | *350* |
| *80* | *400* |
| *90* | *450* |
| *100* | *500* |

***Rešenje:***

<?php  
$price = 5;  
$counter= 10;  
echo "<table border = \"1\" align = \"center\" >";  
echo "<tr><th> Quantity </th>";  
echo "<th>Price </th></tr>";  
while ( $counter<= 100 ) {  
      echo "<tr> <td>";  
      echo $counter;  
      echo "</td> <td>";  
      echo $price \* $counter;  
      echo "</td> </tr>";  
      $counter = $counter + 10;  
}  
echo "</table>";  
?>

Kod koji predstavlja rešenje zadatka po svojoj strukturi veoma je sličan prethodnim, tako da ovde nećemo previše duboko zalaziti u detalje koji su već dobro poznati. Treba napomenuti da je izbegavanje specijalnih karaktera u stringu ostvareno oznakom (\). Unutar while petlje za svaku iteraciju prikazujemo vrednost za $counter kao i vrednost za $counter \* $price. Određivanje koraka vršimo u poslednjoj liniji koda za telo while petlje:

* code
* source

1. $counter = $counter + 10;

**Vežba 4**

***Problem:***

Potrebno je napisati program koji će zadavanjem željenog broja (dodeljivanje vrednosti promenljivoj pre while petlje) na izlazu ispisivati sve brojeve u opsegu većih od 200 i manjih od 400, ukoliko se željeni broj nalazi u predviđenom opsegu. Koristiti while petlju ...

Ispis na izlazu biće, na primer, ukoliko je vrednost željenog broja jednaka 380:

*Number: 380  
Number: 381  
Number: 382  
Number: 383  
Number: 384  
Number: 385  
Number: 386  
Number: 387  
Number: 388  
Number: 389  
Number: 390  
Number: 391  
Number: 392  
Number: 393  
Number: 394  
Number: 395  
Number: 396  
Number: 397  
Number: 398  
Number: 399*

***Rešenje:***

* code
* source

1. <?php
2. $number = 380;
3. do{
4. echo "Number: $number <br>";
5. $number++;
6. }while($number > 200 && $number < 400);
7. ?>

Prateći zahteve zadatka i vodeći računa o tome da ne zapadnemo u petlju koja će se neprekidno izvršavati, kreiramo naredni kod:

* code
* source

1. $number = 380;
2. do{
3. echo "Number: $number <br>";
4. $number++;
5. }while);

Ostaje potreba za definisanjem uslova od kojeg će ponavljanje bloka koda petlje zavisiti. Kao što je to postavkom zadatka i naglašeno, potrebno je da broj bude veći od 200 i manji od 400, te zato definišemo sledeći izraz:

* code
* source

1. $number > 200 && $number < 400

**Vežba 5**

***Problem:***

Potrebno je napisati program za aukcijsku prodaju koji će početi sa izvršavanjem po dobijanju početne vrednosti (cene), dodeljivanjem vrednosti promenljivoj pre ulaza u petlju. Program ispisuje vrednosti uvećavajući ih za jedan i izdaje obaveštenje o odobrenju artikla za prodaju; kada cena dostigne vrednost preko 1000, status artikla je obaveštenje o mogućnosti prodaje. Ispis na izlazu (ukoliko je početna vrednost postavljena na 999), na primer, može izgledati ovako:

Current price: 999. Still not for sale.   
Current price: 1000. Still not for sale.   
Current price: 1001. YOU CAN START WITH SALE !

***Rešenje:***

* code
* source

1. <?php
2. $price = 999;
3. do{
4. echo "Current price: " . $price . " Still not for sale. </br>";
5. $price = $price + 1;
6. }while($price <= 1000);
7. echo "Current price: " . $price . " YOU CAN START WITH SALE !";
8. ?>

Rešenje zadataka predstavlja klasičan primer upotrebe do...while petlje. Najpre je kreirana promenljiva koja se kroz telo petlje prikazuje na dokumentu i uvećava za 1. Čitava petlja podleže očekivanom uslovu:

* code
* source

1. $price <= 1000

Onog trenutka kada se promenljiva $price promeni tako da uslov više nije ispunjen, napušta se petlja i izvršava se poslednja linija koda:

* code
* source

1. echo "Current price: " . $price . " YOU CAN START WITH SALE !";

**Vežba 6**

***Problem:***

Potrebno je napisati program koji će za zadatu temperaturu vršiti konverziju iz celzijusa (C) u kelvine (K) i farenhajte (F). Potrebno je rešiti problematiku u okviru while petlje i uslov za izvršenje naredbe je da zadata temperatura bude manja od 115 stepeni.

***Rešenje:***

* code
* source

1. <?php
2. $ctemp = -10;
3. while ($ctemp < 115) {
4. print ("$ctemp degrees C converts to ");
5. print (32 + $ctemp / 5 \* 9 );
6. print (" degrees F and to ");
7. print ($ctemp + 273.1 );
8. print (" degrees K<BR>");
9. $ctemp = $ctemp + 20;
10. }
11. ?>

Početna vrednost promenljive $ctemp je -10. Kroz while petlju se vrši konvertovanje stepena celzijusovih u farenhajte i kelvine. Za konvertovanje celzijusa u farenhajte koristmo formulu:

* code
* source

1. 32 + $ctemp / 5 \* 9

, dok za konvertovanje celzijusa u kelvine koristimo formulu:

* code
* source

1. $ctemp + 273.1

Na kraju je potrebno da odredimo neki korak po kojem će se promenljiva $ctemp menjati. Kako ne bismo imali previše izlaza na strani, postavljamo:

* code
* source

1. $ctemp = $ctemp + 20;

**U odnosu na if naredbu, kada se koristi while petlja?**

 Kada treba izvršiti blok naredbi više puta

 Kada blok naredbi treba izvršiti samo jednom

 Nikada. Uvek se koristi samo if uslov